Free-form Flows: Make Any Architecture a Normalizing Flow

Felix Draxler*, **Peter Sorrenson***, **Lea Zimmermann**, **Armand Rousselot**, **Ullrich Köthe**
Computer Vision and Learning Lab, Heidelberg University
*Equal contribution

Abstract

Normalizing Flows are generative models that directly maximize the likelihood. Previously, the design of normalizing flows was largely constrained by the need for analytical invertibility. We overcome this constraint by a training procedure that uses an efficient estimator for the gradient of the change of variables formula. This enables any dimension-preserving neural network to serve as a generative model through maximum likelihood training. Our approach allows placing the emphasis on tailoring inductive biases precisely to the task at hand. Specifically, we achieve excellent results in molecule generation benchmarks utilizing $E(n)$-equivariant networks at greatly improved sampling speed. Moreover, our method is competitive in an inverse problem benchmark, while employing off-the-shelf ResNet architectures. We publish our code at `https://github.com/vislearn/FFF`.

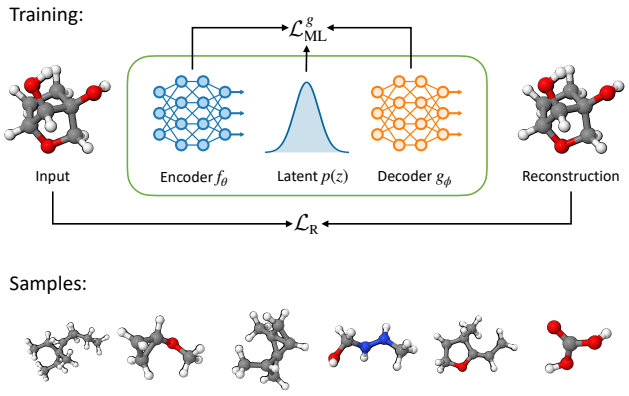

Figure 1: Free-form flows (FFF) train a pair of encoder and decoder neural networks with a fast maximum likelihood estimator $\mathcal{L}^g_{\mathrm{ML}}$ and reconstruction loss $\mathcal{L}_{\mathrm{R}}$. This enables training any dimension-preserving architecture as a one-step generative model. For example, an equivariant graph neural network can be trained on the QM9 dataset to generate molecules by predicting atom positions and properties in a single decoder evaluation. *(Bottom)* Stable molecules sampled from our $E(3)$-FFF trained on the QM9 dataset for several molecule sizes.

1 INTRODUCTION

Generative models have actively demonstrated their utility across diverse applications, successfully scaling to high-dimensional data distributions in scenarios ranging from image synthesis to molecule generation (Rombach et al., 2022; Hoogeboom et al., 2022). Normalizing flows (Dinh et al., 2015; Rezende and Mohamed, 2015) have helped propel this advancement, particularly in scientific domains, enabling practitioners to optimize data likelihood directly and thereby facilitating a statistically rigorous approach to learning complex data distributions. A major factor that has held normalizing flows back as other generative models (notably diffusion models) increase in power and popularity has been that their expressivity is greatly limited by architectural constraints, namely those necessary to ensure bijectivity and compute Jacobian determinants.

In this work, we contribute an approach that frees normalizing flows from their conventional architectural confines, thereby introducing a flexible new class of maximum likelihood models. For model builders, this shifts the focus away from meeting invertibility requirements towards incorporating the best inductive biases to solve the problem at hand. Our aim is that the methods introduced in this paper will allow practitioners to spend more time incorporating domain knowledge into their models, and allow more problems

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

to be solved via maximum likelihood estimation.

The key methodological innovation is the adaptation of a recently proposed method for training autoencoders (Sorrenson et al., 2024) to dimension-preserving models. The trick is to estimate the gradient of the encoder's Jacobian log-determinant by a cheap function of the encoder and decoder Jacobians. We show that in the full-dimensional context many of the theoretical difficulties that plagued the interpretation of the bottlenecked autoencoder model disappear, and the optimization can be interpreted as a relaxation of normalizing flow training, which is tight at the original solutions.

In molecule generation, where rotational equivariance has proven to be a crucial inductive bias, our approach outperforms traditional normalizing flows and generates valid samples more than an order of magnitude faster than previous approaches. Further, experiments in simulation-based inference (SBI) underscore the model's versatility. We find that our training method achieves competitive performance with minimal fine-tuning requirements.

In summary our contributions are as follows:

- We remove all architectural constraints from normalizing flows by introducing maximum-likelihood training for free-form architectures. We call our model the free-form flow (FFF), see fig. 1 and section 3.
- We prove that the training has the same minima as traditional normalizing flow optimization, provided that the reconstruction loss is minimal, see section 4.
- We demonstrate competitive performance with minimal fine-tuning on inverse problems and molecule generation benchmarks, outperforming ODE-based models in the latter. Compared to a diffusion model, our model produces stable molecules more than two orders of magnitude faster. See section 5.

2 RELATED WORK

Normalizing flows traditionally rely on specialized architectures that are invertible and have a manageable Jacobian determinant (see section 3.1). See Papamakarios et al. (2021); Kobyzev et al. (2021) for an overview.

One body of work builds invertible architectures by concatenating simple layers (coupling blocks) which are easy to invert and have a triangular Jacobian, which makes computing determinants easy (Dinh et al., 2015). Expressive power is obtained by stacking many layers and their universality has been confirmed theoretically (Huang et al., 2020; Teshima et al., 2020; Koehler et al., 2021; Draxler et al., 2022, 2023). Many choices for coupling blocks have been proposed such as MAF Papamakarios et al. (2017), RealNVP (Dinh et al., 2017), Glow (Kingma and Dhariwal, 2018), Neural Spline Flows (Durkan et al., 2019), see Kobyzev et al. (2021) for an overview. Instead of analytical invertibility, our model relies on the reconstruction loss to enforce approximate invertibility.

Another line of work ensures invertibility by using a ResNet structure and limiting the Lipschitz constant of each residual layer (Behrmann et al., 2019; Chen et al., 2019). Somewhat similarly, neural ODEs (Chen et al., 2018; Grathwohl et al., 2019) take the continuous limit of ResNets, guaranteeing invertibility under mild conditions. Each of these models requires evaluating multiple steps during training and thus become quite expensive. In addition, the Jacobian determinant must be estimated, adding overhead. Like these methods, we must estimate the gradient of the Jacobian determinant, but can do so more efficiently. Flow Matching Lipman et al. (2023); Liu et al. (2023); Albergo and Vanden-Eijnden (2023) improves training of these continuous normalizing flows in speed and quality, but still involves an expensive multi-step sampling process. By construction, our approach consists of a single model evaluation, and we put no constraints on the architecture apart from inductive biases indicated by the task at hand.

Two interesting methods (Gresele et al., 2020; Keller et al., 2021) compute or estimate gradients of the Jacobian determinant but are severely limited to architectures with exclusively square weight matrices and no residual blocks. We have no architectural limitations besides preserving dimension. Intermediate activations and weight matrices may have any dimension and any network topology is permitted.

3 METHOD

3.1 Normalizing Flows

Normalizing flows (Rezende and Mohamed, 2015) are generative models that learn an invertible function $f_\theta(x) : \mathbb{R}^D \to \mathbb{R}^D$ mapping samples x from a given data distribution $q(x)$ to latent codes z. The aim is that z follows a simple target distribution, typically the multivariate standard normal.

Samples from the resulting generative model $p_\theta(x)$ are obtained by mapping samples of the simple target distribution $p(z)$ through the inverse of the learned func-

tion:

$$x = f_\theta^{-1}(z) \sim p_\theta(x) \text{ for } z \sim p(z).$$

This requires a tractable inverse. Traditionally, this was achieved via invertible layers such as coupling blocks (Dinh et al., 2015) or by otherwise restricting the function class. We replace this constraint via a simple reconstruction loss, and learn a second function $g_\phi \approx f_\theta^{-1}$ as an approximation to the exact inverse.

A tractable determinant of the Jacobian of the learned function is required to account for the change in density. As a result, the value of the model likelihood is given by the change of variables formula for invertible functions:

$$p_\theta(x) = p(Z = f_\theta(x))|J_\theta(x)|. \tag{1}$$

Here, $J_\theta(x)$ denotes the Jacobian of f_θ at x, and $|\cdot|$ the absolute value of its determinant.

Normalizing Flows are trained by minimizing the Kullback-Leibler (KL) divergence between the true and learned distribution. This is equivalent to maximizing the likelihood of the training data:

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(q(x)\|p_\theta(x)) &= \mathbb{E}_{x\sim q(x)}[\log q(x) - \log p_\theta(x)] \\ &= \mathbb{E}_x[-\log p(f_\theta(x)) - \log|J_\theta(x)|] + \text{const}\,. \end{aligned} \tag{2}$$

By eq. (1), this requires evaluating the determinant of the Jacobian $|J_\theta(x)|$ of f_θ at x. If we want to compute this exactly, we need to compute the full Jacobian matrix, requiring D backpropagations through f_θ. This linear scaling with dimension is prohibitive for most modern applications. The bulk of the normalizing flow literature is therefore concerned with building invertible architectures that are expressive and allow computing the determinant of the Jacobian more efficiently. We circumvent this via a trick that allows efficient estimation of the gradient $\nabla_\theta \log|J_\theta(x)|$, noting that this quantity is sufficient to perform gradient descent.

3.2 Gradient trick

The results of this section are an adaptation of results in Caterini et al. (2021) and Sorrenson et al. (2024).

Here, we derive how to efficiently estimate the gradient of the maximum-likelihood loss in eq. (2), even if the architecture does not yield an efficient way to compute the change of variables term $\log|J_\theta(x)|$. We avoid this computation by estimating the gradient of $\log|J_\theta(x)|$ via a pair of vector-Jacobian and Jacobian-vector products, which are readily available in standard automatic differentiation software libraries.

Gradient via trace estimator

Theorem 3.1. *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ be a C^1 invertible function parameterized by θ. Then, for all $x \in \mathbb{R}^D$:*

$$\nabla_{\theta_i} \log|J_\theta(x)| = \mathrm{tr}\big((\nabla_{\theta_i} J_\theta(x))(J_\theta(x))^{-1}\big)\,. \tag{3}$$

The proof is by direct application of Jacobi's formula, see appendix A.1. This is not a simplification per se, given that the RHS of eq. (3) now involves the computation of both the Jacobian as well as its inverse. However, we can estimate it via the Hutchinson trace estimator (where we omit dependence on x for simplicity):

$$\begin{aligned}\mathrm{tr}\big((\nabla_{\theta_i} J_\theta)J_\theta^{-1}\big) &= \mathbb{E}_v[v^T(\nabla_{\theta_i} J_\theta)J_\theta^{-1}v] \\ &\approx \frac{1}{K}\sum_{k=1}^{K} v_k^T(\nabla_{\theta_i} J_\theta)J_\theta^{-1}v_k.\end{aligned}$$

Now all we require is computing the dot products $v^T(\nabla_{\theta_i} J_\theta)$ and $J_\theta^{-1}v$, where the random vector $v \in \mathbb{R}^D$ must have unit covariance.

Matrix inverse via function inverse To compute $J_\theta^{-1}v$ we note that, when f_θ is invertible, the matrix inverse of the Jacobian of f_θ is the Jacobian of the inverse function f_θ^{-1}:

$$J_\theta^{-1}(x) = (\nabla_x f_\theta(x))^{-1} = \nabla_z f_\theta^{-1}\big(z = f_\theta(x)\big).$$

This means that the product $J_\theta^{-1}v$ is simply the dot product of the Jacobian of f_θ^{-1} with the vector v. This Jacobian-vector product is readily available via forward automatic differentiation.

Use of stop-gradient We are left with computing the dot product $v^T(\nabla_{\theta_i} J_\theta)$. Since v^T is independent of θ, we can draw it into the gradient $v^T(\nabla_{\theta_i} J_\theta) = \nabla_{\theta_i}(v^T J_\theta)$. This vector-Jacobian product can be again readily computed, this time with backward automatic differentiation.

In order to implement the final gradient with respect to the flow parameters θ, we draw the derivative with respect to parameters out of the trace, making sure to prevent gradient from flowing to J_θ^{-1} by wrapping it in a stop-gradient operation $\mathtt{SG}$:

$$\begin{aligned}\mathrm{tr}\big((\nabla_{\theta_i} J_\theta)J_\theta^{-1}\big) &= \nabla_{\theta_i}\,\mathrm{tr}(J_\theta \mathtt{SG}(J_\theta^{-1})) \\ &\approx \nabla_{\theta_i}\frac{1}{K}\sum_{k=1}^{K} v_k^T J_\theta \mathtt{SG}(J_\theta^{-1}v_k).\end{aligned}$$

Summary The above argument shows that

$$\nabla_{\theta_i} \log|J_\theta(x)| \approx \nabla_{\theta_i}\frac{1}{K}\sum_{k=1}^{K} v_k^T J_\theta \mathtt{SG}(J_\theta^{-1}v_k).$$

Algorithm 1 FFF loss function. Vector-Jacobian product = `vjp`; Jacobian-vector product = `jvp`. Time and space complexity are $\mathcal{O}(D)$.

function LOSS($x, f_\theta, g_\phi, \beta$)
  $v \sim p(v)$ ▷ $\mathbb{E}[vv^T] = I$
  $z, v_1 \leftarrow \texttt{vjp}(f_\theta, x, v)$ ▷ $z = f_\theta(x), v_1^T = v^T \frac{\partial z}{\partial x}$
  $\hat{x}, v_2 \leftarrow \texttt{jvp}(g_\phi, z, v)$ ▷ $\hat{x} = g_\phi(z), v_2 = \frac{\partial \hat{x}}{\partial z} v$
  $\mathcal{L}^g_{\text{ML}} \leftarrow \frac{1}{2}\|z\|^2 - \texttt{SG}(v_2^T)v_1$ ▷ stop gradient to v_2
  $\mathcal{L}_{\text{R}} \leftarrow \|\hat{x} - x\|^2$
  $\mathcal{L}^g \leftarrow \mathcal{L}^g_{\text{ML}} + \beta\mathcal{L}_{\text{R}}$
  return $\mathcal{L}^g$
end function

Instead of computing the full Jacobian $J_\theta(x)$, which involved as many backpropagation steps as dimensions, we are left with computing just one vector-Jacobian product and one Jacobian-vector product for each k. In practice, we find that setting $K = 1$ is sufficient and we drop the summation over k for the remainder of this paper. We provide an ablation study on the effect of K in appendix C.3.

This yields the following maximum likelihood training objective, whose gradients are an unbiased estimator for the true gradients from exact maximum likelihood as in eq. (2):

$$\mathcal{L}^{f^{-1}}_{\text{ML}} = \mathbb{E}_{x,v}[-\log p(f_\theta(x)) - v^T J_\theta \texttt{SG}(J_\theta^{-1} v)]. \quad (4)$$

This result enables training normalizing flow architectures with a tractable inverse function, but whose Jacobian determinant is not easily accessible. We now move on to show how this gradient estimator can be adapted for free-form dimension-preserving neural networks.

3.3 Free-form Flows (FFF)

The previous section assumed that we have access to both f_θ and its analytic inverse f_θ^{-1}. Typically an analytic inverse is obtained by explicitly constructing invertible neural networks (INNs) or defining the flow as a differential equation with a known reverse time process (Neural ODEs). In contrast, we drop the assumption of an analytic inverse and replace f_θ^{-1} with a learned inverse $g_\phi \approx f_\theta^{-1}$. We ensure that (i) f_θ is invertible and that (ii) $g_\phi \approx f_\theta^{-1}$ through a reconstruction loss:

$$\mathcal{L}_{\text{R}} = \tfrac{1}{2}\mathbb{E}_x[\|x - g_\phi(f_\theta(x))\|^2]. \quad (5)$$

This removes all architectural constraints from f_θ and g_ϕ except from preserving the dimension.

Similarly to Sorrenson et al. (2024), the replacement $g_\phi \approx f_\theta^{-1}$ leads to a modification of $\mathcal{L}^{f^{-1}}_{\text{ML}}$, where we replace J_θ^{-1} by J_ϕ, where J_ϕ is shorthand for the Jacobian of g_ϕ evaluated at $f_\theta(x)$:

$$\mathcal{L}^g_{\text{ML}} = \mathbb{E}_{x,v}[-\log p(f_\theta(x)) - v^T J_\theta \texttt{SG}(J_\phi v)] \quad (6)$$

Combining the maximum likelihood (eqs. (4) and (6)) and reconstruction (eq. (5)) components of the loss leads to the following losses:

$$\mathcal{L}^{f^{-1}} = \mathcal{L}^{f^{-1}}_{\text{ML}} + \beta\mathcal{L}_{\text{R}} \quad \text{and} \quad \mathcal{L}^g = \mathcal{L}^g_{\text{ML}} + \beta\mathcal{L}_{\text{R}} \quad (7)$$

where the two terms are traded off by a hyperparameter β. We optimize $\mathcal{L}^g$ with the justification that it has the same critical points as $\mathcal{L}^{f^{-1}}$ (plus additional ones which aren't a problem in practice, see section 4.3).

3.3.1 Likelihood Calculation

Once training is completed, our generative model involves sampling from the latent distribution and passing the samples through the decoder g_ϕ.

In order to calculate the likelihoods induced by g_ϕ, we can use the change of variables formula:

$$\begin{aligned} p_\phi(X = x) &= p(Z = g_\phi^{-1}(x))|J_\phi(g_\phi^{-1}(x))| \\ &\approx p(Z = f_\theta(x))|J_\phi(f_\theta(x))| \end{aligned}$$

where the approximation is due to $g_\phi^{-1} \approx f_\theta$.

In the next section, we theoretically justify the use of free-form architectures and the combination of maximum likelihood with a reconstruction loss.

Algorithm 2 FFF likelihood calculation: returns an approximation of $\log p_\phi(x)$. Time complexity is $\mathcal{O}(D^3)$ and space complexity is $\mathcal{O}(D^2)$.

function LOGLIKELIHOOD(x, f_θ, g_ϕ)
  $z \leftarrow f_\theta(x)$
  $J_\phi \leftarrow \texttt{jacobian}(g_\phi, z)$ ▷ $J_\phi = \frac{\partial g_\phi(z)}{\partial z}$
  $\ell \leftarrow -\frac{1}{2}\|z\|^2 - \frac{D}{2}\log(2\pi) + \log|J_\phi|$
  return ℓ
end function

4 THEORY

In this section we provide three theorems which emphasize the validity of our method. Firstly, we show that optimizing the loss function $\mathcal{L}^{f^{-1}}$ (using an exact inverse) is a bound on the spread divergence between the data and generating distributions. Secondly, we show under what conditions the gradients of the relaxation $\mathcal{L}^g$ (loss using a non-exact inverse) equal those of $\mathcal{L}^{f^{-1}}$. Finally and most importantly, we show that solutions to $\mathcal{L}^{f^{-1}}$ exactly learn the data distribution. In addition, every critical point of $\mathcal{L}^{f^{-1}}$ is a critical point

of $\mathcal{L}^g$, meaning that optimizing $\mathcal{L}^g$ is equivalent, except for some additional critical points which we argue do not matter in practice. Please refer to appendix A for detailed derivations and proofs of the results in this section.

4.1 Loss Derivation

In addition to the intuitive development given in the previous sections, $\mathcal{L}^{f^{-1}}$ (eq. (7)) can be rigorously derived as a bound on the KL divergence between a noisy version of the data and a noisy version of the model, known as a spread divergence (Zhang et al., 2020). The bound is a type of evidence lower bound (ELBO) as employed in VAEs (Kingma and Welling, 2014).

Theorem 4.1. *Let f_θ and g_ϕ be C^1 and let f_θ be globally invertible. Define the spread KL divergence $\tilde{\mathcal{D}}_{KL}$ as the KL divergence between distributions convolved with isotropic Gaussian noise of variance σ^2. Let $\beta = 1/2\sigma^2$. Then there exists a function D of θ and ϕ such that*

$$\nabla_\theta \mathcal{L}^{f^{-1}} = \nabla_\theta D \quad \textit{and} \quad \nabla_\phi \mathcal{L}^{f^{-1}} = \nabla_\phi D$$

and

$$D \geq \tilde{\mathcal{D}}_{KL}(q(x) \parallel p_\phi(x))$$

As a result, minimizing $\mathcal{L}^{f^{-1}}$ is equivalent to minimizing an upper bound on $\tilde{\mathcal{D}}_{KL}(q(x) \parallel p_\phi(x))$.

The fact that we optimize a bound on a spread KL divergence is beneficial in cases where $q(x)$ is degenerate, for example when $q(x)$ is an empirical data distribution (essentially a mixture of Dirac delta distributions). (Non-spread) KL divergences with $q(x)$ in this case will be almost always infinite. In addition, by taking σ very small (and hence using large β), the difference between the standard and spread KL divergence is so small as to be negligible in practice.

Since the above derivation resembles an ELBO, we can ask whether the FFF can be interpreted as a VAE. In appendix A.2 we provide an argument that it can, but one with a very flexible posterior distribution, in contrast to the simple distributions (such as Gaussian) typically used in VAEs posteriors. As such it does not suffer from typical VAE failure modes, such as poor reconstructions and over-regularization.

Note that the above theorem states a result in terms of the decoder distribution $p_\phi(x)$, not the encoder distribution $p_\theta(x)$ which was used to motivate the loss function. While this may seem counterproductive at first, it is in fact more useful to optimize for $p_\phi(x)$ matching the data distribution than for $p_\theta(x)$ to match the data distribution, since $p_\phi(x)$ is the model we use to generate data from. In any case, it is simple to demonstrate that $D' \geq \mathcal{D}_{\mathrm{KL}}(q(x) \parallel p_\theta(x))$ where D' has the same gradients as $\mathcal{L}^{f^{-1}}$ (see appendix A.2), so both encoder and decoder models will become more similar to the data distribution with increasing optimization.

4.2 Error Bound

The accuracy of the estimator for the gradient of the log-determinant depends on how close g_ϕ is to being an inverse of f_θ. In particular, we can bound the estimator's error by a measure of how close the product of the Jacobians is to the identity matrix. This is captured in the following result.

Theorem 4.2. *Let f_θ and g_ϕ be C^1, let J_θ be the Jacobian of f_θ at x and let J_ϕ be the Jacobian of g_ϕ at $f_\theta(x)$. Suppose that f_θ is locally invertible at x, meaning $J_\theta(x)$ is an invertible matrix. Let $\|\cdot\|_F$ be the Frobenius norm of a matrix. Then the absolute difference between $\nabla_{\theta_i} \log |J_\theta(x)|$ and the trace-based approximation is bounded:*

$$\begin{aligned} &|\mathrm{tr}((\nabla_{\theta_i} J_\theta) J_\phi) - \nabla_{\theta_i} \log |J_\theta||\\ &\leq \|(\nabla_{\theta_i} J_\theta) J_\theta^{-1}\|_F \|J_\theta J_\phi - I\|_F \end{aligned}$$

The Jacobian deviation, namely $\|J_\theta J_\phi - I\|_F$, could be minimized by adding such a term to the loss as a regularizer. We find in practice that the reconstruction loss alone is sufficient to minimize this quantity and that the two are correlated in practice. While it could be possible in principle for a dimension-preserving pair of encoder and decoder to have a low reconstruction loss while the Jacobians of encoder and decoder are not well matched, we don't observe this in practice. Such a function would have to have a very large second derivative, which is implicitly discouraged in typical neural network optimization (Rahaman et al., 2019).

In appendix A.3 we prove an additional result which quantifies the difference between the gradients of $\mathcal{L}^{f^{-1}}$ and $\mathcal{L}^g$, showing that it is bounded by $\mathbb{E}_x \left[\|J_\theta J_\phi - I\|_F^2\right]$.

4.3 Critical Points

The following theorem states our main result: that optimizing $\mathcal{L}^g$ (eq. (7)) is almost equivalent to optimizing $\mathcal{L}^{f^{-1}}$, and that the solutions to $\mathcal{L}^{f^{-1}}$ are maximum likelihood solutions where $p_\theta(x) = q(x)$. Note that this a result on the functional level: if we say f is a critical point of $\mathcal{L}^{f^{-1}}$, we mean that adding any infinitesimally small deviation δf to f does not change $\mathcal{L}^{f^{-1}}$. These optima may not be within the set of functions reachable under gradient descent with our chosen network architecture, and the particular neural network implementation may introduce local minima which are not captured in the theorem.

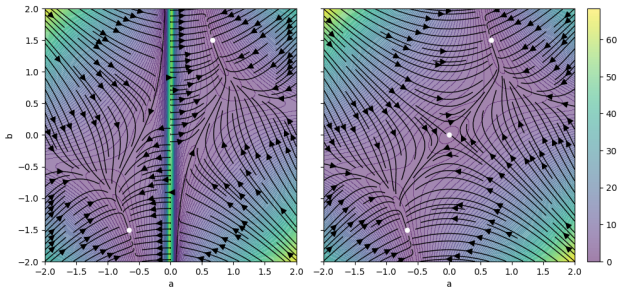

Figure 2: Gradient landscape of $\mathcal{L}^{f^{-1}}$ (*left*) and $\mathcal{L}^{g}$ (*right*) for a linear 1D model with $f(x) = ax$, $g(z) = bz$, $q(x) = \mathcal{N}(0, 1.5^2)$ and $\beta = 1$. The flow lines show the direction and the contours show the magnitude of the gradient. White dots are critical points. $\mathcal{L}^{g}$ has the same minima $(\pm 2/3, \pm 1.5)$ as $\mathcal{L}^{f^{-1}}$, with an additional critical point at $a = b = 0$. This is a saddle, so we will not converge to it in practice. Therefore optimizing $\mathcal{L}^{g}$ results in the same solutions as $\mathcal{L}^{f^{-1}}$.

Theorem 4.3. *Let f_θ and g_ϕ be C^1 and let f_θ be globally invertible. Suppose $q(x)$ is finite and has support everywhere. Then the critical points (on the functional level) of $\mathcal{L}^{f^{-1}}$ (for any $\beta > 0$) are such that*

1. *$g_\phi(z) = f_\theta^{-1}(z)$ for all z, and*
2. *$p_\theta(x) = q(x)$ for all x, and*
3. *All critical points are global minima*

Furthermore, every minimum of $\mathcal{L}^{f^{-1}}$ is a critical point of $\mathcal{L}^{g}$. If the reconstruction loss is minimal, $\mathcal{L}^{g}$ has no additional critical points.

Note that $\mathcal{L}^{g}$ may have additional critical points if the reconstruction loss is not minimal, meaning that f_θ and g_ϕ are not globally invertible. An example is when both f_θ and g_ϕ are the zero function and $q(x)$ has zero mean. We can avoid such solutions by ensuring that β is large enough to not tolerate a high reconstruction loss. In appendix B.4 we give guidelines on how to choose β in practice.

Figure 2 provides an illuminating example. Here the data and latent space are 1-dimensional and f and g are simple linear functions of a single parameter each. As such we can visualize the gradient landscape in a 2D plot. We see that the additional critical point at the origin is a saddle: there are both converging and diverging gradients. In stochastic gradient descent, it is not plausible that we converge to a saddle since the set of points which converge to it deterministically have measure zero in the parameter space. Hence in this example $\mathcal{L}^{g}$ will converge to the same solutions as $\mathcal{L}^{f^{-1}}$. In addition, it has a smoother gradient landscape (no diverging gradient at $a = 0$). While this might not be important in this simple example, in higher dimensions where the Jacobians of adjacent regions could be inconsistent (if the eigenvalues have different signs), it is useful to be able to cross regions where the Jacobian is singular without having to overcome an excessive gradient barrier.

5 EXPERIMENTS

In this section, we demonstrate the practical capabilities of free-form flows (FFF). We mainly compare the performance against normalizing flows based on architectures which are invertible by construction. First, on an inverse problem benchmark, we show that using free-form architectures offers competitive performance to recent spline-based and ODE-based normalizing flows. This is achieved despite minimal tuning of hyperparameters, demonstrating that FFFs are easy to adapt to a new task. Second, on two molecule generation benchmarks, we demonstrate that specialized networks can now be used in a normalizing flow. In particular, we employ the equivariant graph neural networks $E(n)$-GNN (Satorras et al., 2021b). This $E(n)$-FFF outperforms ODE-based equivariant normalizing flows in terms of likelihood, and generates stable molecules significantly faster than a diffusion model.

5.1 Simulation-Based Inference

One popular application of generative models is in solving inverse problems. Here, the goal is to estimate hidden parameters from an observation. As inverse problems are typically ambiguous, a probability distribution represented by a generative model is a suitable solution. From a Bayesian perspective, this probability distribution is the posterior of the parameters given the observation. We learn this posterior via a conditional generative model.

In particular, we focus on simulation based inference (SBI, Radev et al. (2022, 2021); Bieringer et al. (2021)), where we want to predict the parameters of a simulation. The training data is pairs of parameters and outputs generated from the simulation.

We train FFF models on the benchmark proposed in (Lueckmann et al., 2021), which is comprised of ten inverse problems of varying difficulty at three different simulation budgets (i.e. training-set sizes) each. The models are evaluated via a classifier 2-sample test (C2ST) (Lopez-Paz and Oquab, 2017; Friedman, 2003), where a classifier is trained to discern samples from the trained generative model and the true pa-

rameter posterior. The model performance is then reported as the classifier accuracy, where 0.5 demonstrates a distribution indistinguishable from the true posterior. We average this accuracy over ten different observations. In fig. 3, we report the C2ST of our model and compare it against the baseline based on neural spline flows (Durkan et al., 2019) and flow matching for SBI (Wildberger et al., 2023). Our method performs competitively, especially providing an improvement over existing methods in the regime of low simulation budgets. Regarding tuning of hyperparameters, we find that a simple fully-connected architecture with skip connections works across datasets with minor modifications to increase capacity for the larger datasets. We identify the reconstruction weight β large enough such that training becomes stable. We give all dataset and more training details in appendix C.1.

5.2 Molecule Generation

Free-form normalizing flows (FFF) do not make any assumptions about the underlying networks f_θ and g_ϕ, except that they preserve dimension. We can leverage this flexibility for tasks where explicit constraints *should* be built into the architecture, as opposed to constraints that originate from the need for tractable optimization (such as coupling blocks).

As a showcase, we apply FFF to molecule generation. Here, the task is to learn the joint distribution of a number of atoms $x_1, \dots, x_N \in \mathbb{R}^n$. Each prediction of the generative model should yield a physically valid position for each atom: $x = (x_1, \dots, x_N) \in \mathbb{R}^{N \times n}$.

The physical system of atoms in space have an important symmetry: if a molecule is shifted or rotated in space, its properties do not change. This means that a generative model for molecules should yield the same probability regardless of orientation and translation:

$$p_\phi(Qx + t) = p_\phi(x). \tag{8}$$

Here, the rotation $Q \in \mathbb{R}^{n \times n}$ acts on x by rotating or reflecting each atom $x_i \in \mathbb{R}^n$ about the origin, and $t \in \mathbb{R}^n$ applies the same translation to each atom. Formally, (Q, t) are realizations of the Euclidean group $E(n)$. The above eq. (8) means that the distribution $p_\phi(x)$ is invariant under the Euclidean group $E(n)$.

Köhler et al. (2020); Toth et al. (2020) showed that if the latent distribution $p(z)$ is *invariant* under a group G, and a generative model $g_\phi(z)$ is *equivariant* to G, then the resulting distribution is also invariant to G. Equivariance means that applying any group action to the input (e.g. rotation and translation) and then applying g_ϕ should give the same result as first applying g_ϕ and then applying the group. For example, for the Euclidean group:

$$Qg_\phi(z) + t = g_\phi(Qz + t). \tag{9}$$

This implies that we can learn a distribution invariant to the Euclidean group by construction by making normalizing flows equivariant to the Euclidean group as in eq. (9). Previous work has demonstrated that this inductive bias is more effective than data augmentation, where random rotations and translations are applied to each data point at train time (Köhler et al., 2020; Hoogeboom et al., 2022).

We therefore choose an $E(n)$ equivariant network as the networks $f_\theta(x)$ and $g_\phi(z)$ in our FFF. We employ the $E(n)$-GNN proposed by Satorras et al. (2021b). We call this model the $E(n)$-free-form flow ($E(n)$-FFF). We give the implementation details in appendix C.2.

The $E(n)$-GNN has also been the backbone for previous normalizing flows on molecules. However, to the best of our knowledge, all realizations of such architectures have been based on neural ODEs, where the flow is parameterized as a differential equation $\frac{\mathrm{d}x}{\mathrm{d}t} = f_\theta(x(t), t)$. While training, one can avoid solving the ODE by using the rectified flow or flow matching objective (Liu et al., 2023; Lipman et al., 2023; Albergo and Vanden-Eijnden, 2023). However, they still have the disadvantage that they require integrating the ODE for sampling. Our model, in contrast, only calls $f_\phi(z)$ once for sampling.

Boltzmann Generator We test our $E(n)$-FFFs in learning a Boltzmann distribution:

$$q(x) \propto e^{-\beta u(x)},$$

	NLL (↓)	Sampling time (↓)
	DW4	
$E(n)$-NF (Satorras et al., 2021a)	1.72 ± 0.01	**0.024** ms
OT-FM (Klein et al., 2023)	**1.70** ± 0.02	0.034 ms
E-OT-FM (Klein et al., 2023)	**1.68** ± 0.01	0.033 ms
$E(n)$-FFF (ours)	**1.68** ± 0.01	**0.026** ms
	LJ13	
$E(n)$-NF	-16.28 ± 0.04	0.27 ms
OT-FM	-16.54 ± 0.03	0.77 ms
E-OT-FM	-16.70 ± 0.12	0.72 ms
$E(n)$-FFF (ours)	**-17.09** ± 0.16	**0.11** ms
	LJ55	
OT-FM	-88.45 ± 0.04	40 ms
E-OT-FM	**-89.27** ± 0.04	40 ms
$E(n)$-FFF (ours)	-88.72 ± 0.16	**2.1** ms

Table 1: Equivariant free-form flows ($E(n)$-FFF) sample significantly faster than previous models, and achieve comparable or better negative log-likelihood (NLL, lower is better). More details in table 6.

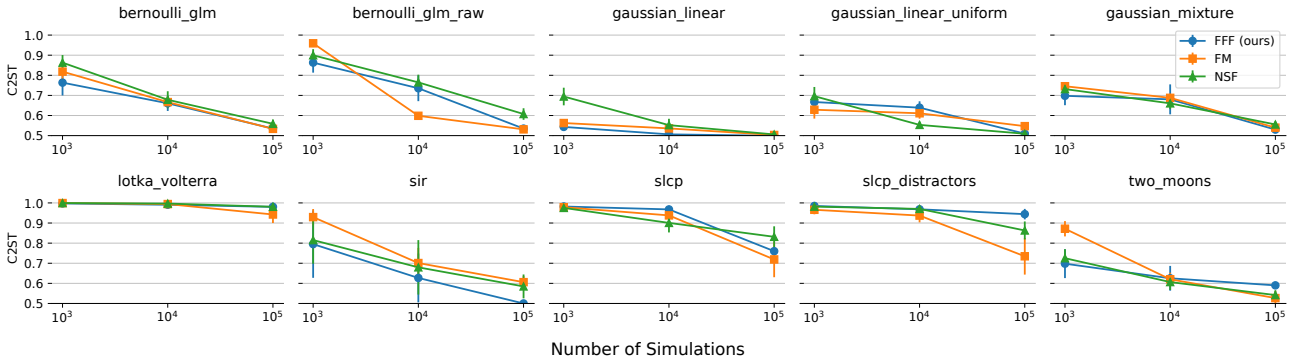

Figure 3: C2ST accuracy on the SBI benchmark datasets. We compare our method (FFF) against flow matching (FM) Wildberger et al. (2023) and the neural spline flow (NSF) baseline in the benchmark dataset Lueckmann et al. (2021). The accuracy is averaged over ten different observations, with error bars indicating the standard deviation. Our performance is comparable to the competitors across all datasets, with no model being universally better or worse.

where $u(x) \in \mathbb{R}$ is an energy function that takes the positions of atoms $x = (x_1, \dots, x_N)$ as an input. A generative model $p_\phi(x)$ that approximates $q(x)$ can be used as a Boltzmann generator (Noé et al., 2019). The idea of the Boltzmann generator is that having access to $u(x)$ allows re-weighting samples from the generator after training even if $p_\phi(x)$ is different from $q(x)$. This is necessary in order to evaluate samples from $q(x)$ in a downstream task: Re-weighting samples allows computing expectation values $\mathbb{E}_{x\sim q(x)}[O(x)] = \mathbb{E}_{x\sim p_\phi(x)}[\frac{q(x)}{p_\phi(x)} O(x)]$ from samples of the generative model $p_\phi(x)$ if $p_\phi(x)$ and $q(x)$ have the same support.

We evaluate the performance of free-form flows (FFF) as a Boltzmann generator on the benchmark tasks DW4, LJ13, and LJ55 (Köhler et al., 2020; Klein et al., 2023). Here, pairwise potentials $v(x_i, x_j)$ are summed as the total energy $u(x)$:

$$u(x) = \sum_{i,j} v(x_i, x_j).$$

DW4 uses a double-well potential $v_{\texttt{DW}}$ and considers four particles in 2D. LJ13 and LJ55 both employ a Lennard-Jones potential $v_{\texttt{LJ}}$ between 13 respectively 55 particles in 3D space (see appendix C.2.3 for details). We make use of the datasets presented by Klein et al. (2023), which obtained samples from $p(x)$ via MCMC.[1]

In table 1, we compare our model against (i) the equivariant ODE normalizing flow trained with maximum likelihood $E(n)$-NF (Satorras et al., 2021a), and (ii) two equivariant ODEs trained via optimal transport (equivariant) flow matching Klein et al. (2023). We find our model to have comparable or better negative log-likelihood than competitors. In addition, $E(n)$-FFFs sample significantly faster than competitors because our model needs to evaluate the learned network only once, as opposed to the multiple evaluations required to integrate an ODE.

[1]Datasets available at: `https://osf.io/srqg7/?view_only=28deeba0845546fb96d1b2f355db0da5`

	NLL (↓)	Stable (↑)	Sampling time (↓) Raw	Stable
$E(3)$-NF	-59.7	4.9 %	13.9 ms	309.5 ms
$E(3)$-DM	**-110.7**	**82.0** %	1580.8 ms	1970.6 ms
$E(3)$-FFF	-76.2	8.7 %	**0.6** ms	**8.1** ms
Data	-	95.2 %	-	-

Table 2: $E(3)$-FFF (ours) trained on QM9 generates a stable molecule faster than previous models because a sample is obtained via a single function evaluation. $E(3)$-DM is the $E(3)$-diffusion model (Hoogeboom et al., 2022), $E(3)$-NF the $E(3)$-normalizing flow (Satorras et al., 2021a). The latter is also trained explicitly using maximum likelihood, yet outperformed by $E(3)$-FFF in terms of negative log-likelihood (NLL) and what ratio of generated molecules is stable.

QM9 Molecules As a second molecule generation benchmark, we test the performance of $E(3)$-FFF in generating novel molecules. We therefore train on the QM9 dataset (Ruddigkeit et al., 2012; Ramakrishnan et al., 2014), which contains molecules of varying atom count, with the largest molecules counting 29 atoms. The goal of the generative model is not only to predict the positions of the atoms in each molecule $x = (x_1, \dots, x_N) \in \mathbb{R}^3$, but also each atom's prop-

erties h_i (atom type (categorical), and atom charge (ordinal)).

We again employ the $E(3)$-GNN (Satorras et al., 2021b). The part of the network that acts on coordinates $x_i \in \mathbb{R}^3$ is equivariant to rotations, reflections and translations (Euclidean group $E(3)$). The network leaves the atom properties h invariant under these operations.

We show samples from our model in fig. 1. Because free-form flows only need one network evaluation to sample, they generate two orders of magnitude more stable molecules than the $E(3)$-diffusion model (Hoogeboom et al., 2022) and one order of magnitude more than the $E(3)$-normalizing flow (Satorras et al., 2021a) in a fixed time window, see table 2. This includes the time to generate unstable samples, which are discarded. A molecule is called stable if each atom has the correct number of bonds, where bonds are determined from inter-atomic distances. $E(3)$-FFF also outperforms $E(3)$-NF trained with maximum likelihood both in terms of likelihood and in how many of the sampled molecules are stable. See appendix C.2.4 for implementation details.

6 CONCLUSION

In this work, we present free-form flows (FFF), a new paradigm for normalizing flows that enables training arbitrary dimension-preserving neural networks with maximum likelihood. Invertibility is achieved by a reconstruction loss and the likelihood is maximized by an efficient surrogate. Previously, designing normalizing flows was constrained by the need for analytical invertibility. Free-form flows allow practitioners to focus on the data and suitable inductive biases instead.

We show that free-form flows are an exact relaxation of maximum likelihood training, converging to the same solutions provided that the reconstruction loss is minimal. We provide an interpretation of FFF training as the minimization of a lower bound on the KL divergence between noisy versions of the data and the generative distribution. Furthermore this bound is tight if f_θ and g_ϕ are true inverses.

In practice, free-form flows perform on par or better than previous normalizing flows, exhibit fast sampling by only requiring a single function evaluation, and are easy to tune. We provide a practical guide for adapting them to new problems in appendix B.

Acknowledgments

This work is supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC-2181/1 - 390900948 (the Heidelberg STRUCTURES Cluster of Excellence). It is also supported by the Vector Stiftung in the project TRINN (P2019-0092). AR acknowledges funding from the Carl-Zeiss-Stiftung. LZ acknowledges support by the German Federal Ministery of Education and Research (BMBF) (project EMUNE/031L0293A). The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG.

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. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

Albergo, M. S. and Vanden-Eijnden, E. (2023). Building normalizing flows with stochastic interpolants. In *International Conference on Learning Representations*.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. (2019). Invertible residual networks. In *International Conference on Machine Learning*.

Bieringer, S., Butter, A., Heimel, T., Höche, S., Köthe, U., Plehn, T., and Radev, S. T. (2021). Measuring QCD splittings with invertible networks. *SciPost Physics*, 10(6):126.

Caterini, A. L., Loaiza-Ganem, G., Pleiss, G., and Cunningham, J. P. (2021). Rectangular flows for manifold learning. *Advances in Neural Information Processing Systems*.

Chen, R. T., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. (2019). Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. *Advances in neural information processing systems*.

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

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using Real NVP. In *International Conference on Learning Representations*.

Draxler, F., Kühmichel, L., Rousselot, A., Müller, J., Schnoerr, C., and Koethe, U. (2023). On the Convergence Rate of Gaussianization with Random Rotations. In *International Conference on Machine Learning*.

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

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Neural Spline Flows. In *Advances in Neural Information Processing Systems*.

Falcon, W. and The PyTorch Lightning team (2019). PyTorch lightning.

Friedman, J. H. (2003). On multivariate goodness–of–fit and two–sample testing. *Statistical Problems in Particle Physics, Astrophysics, and Cosmology*, 1:311.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. (2019). Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*.

Gresele, L., Fissore, G., Javaloy, A., Schölkopf, B., and Hyvarinen, A. (2020). Relative gradient optimization of the jacobian term in unsupervised deep learning. In *Advances in Neural Information Processing Systems*.

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. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

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 *International Conference on Machine Learning*.

Hoogeboom, E., Nielsen, D., Jaini, P., Forré, P., and Welling, M. (2021). Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*.

Hoogeboom, E., Satorras, V. G., Vignac, C., and Welling, M. (2022). Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*.

Huang, C.-W., Dinh, L., and Courville, A. (2020). Augmented Normalizing Flows: Bridging the Gap Between Generative Flows and Latent Variable Models. In *International Conference on Learning Representations, Workshop Track*.

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

Keller, T. A., Peters, J. W., Jaini, P., Hoogeboom, E., Forré, P., and Welling, M. (2021). Self normalizing flows. In *International Conference on Machine Learning*.

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

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *International Conference on Learning Representations*.

Klein, L., Krämer, A., and Noé, F. (2023). Equivariant Flow Matching. In *Advances in Neural Information Processing Systems*.

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

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

Köhler, J., Klein, L., and Noé, F. (2020). Equivariant flows: Exact likelihood generative learning for symmetric densities. In *International Conference on Machine Learning*.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. (2023). Flow Matching for Generative Modeling. In *International Conference on Learning Representations*.

Liu, X., Gong, C., and Liu, Q. (2023). Learning to Generate and Transfer Data with Rectified Flow. In *International Conference on Learning Representations*.

Lopez-Paz, D. and Oquab, M. (2017). Revisiting classifier two-sample tests. In *International Conference on Learning Representations*.

Lueckmann, J.-M., Boelts, J., Greenberg, D., Goncalves, P., and Macke, J. (2021). Benchmarking simulation-based inference. In *International Conference on Artificial Intelligence and Statistics*.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In van der Walt, S. and

Jarrod Millman, editors, *9th Python in Science Conference*.

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):eaaw1147.

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*.

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

Papoulis, A. and Pillai, S. U. (2002). *Probability, Random Variables and Stochastic Processes*. McGraw-Hill.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*.

Radev, S. T., Graw, F., Chen, S., Mutters, N. T., Eichel, V. M., Bärnighausen, T., and Köthe, U. (2021). OutbreakFlow: Model-based Bayesian inference of disease outbreak dynamics with invertible neural networks and its application to the COVID-19 pandemics in Germany. *PLoS computational biology*, 17(10):e1009472.

Radev, S. T., Mertens, U. K., Voss, A., Ardizzone, L., and Köthe, U. (2022). BayesFlow: Learning Complex Stochastic Models With Invertible Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1452–1466.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. (2019). On the spectral bias of neural networks. In *International Conference on Machine Learning*.

Ramakrishnan, R., Dral, P. O., Rupp, M., and von Lilienfeld, O. A. (2014). Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1:140022.

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

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Ruddigkeit, L., van Deursen, R., Blum, L. C., and Reymond, J.-L. (2012). Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. *Journal of Chemical Information and Modeling*, 52:2864–2875.

Satorras, V. G., Hoogeboom, E., Fuchs, F., Posner, I., and Welling, M. (2021a). E (n) equivariant normalizing flows. *Advances in Neural Information Processing Systems*.

Satorras, V. G., Hoogeboom, E., and Welling, M. (2021b). E(n) equivariant graph neural networks. In *International Conference on Machine Learning*.

Sorrenson, P., Draxler, F., Rousselot, A., Hummerich, S., Zimmermann, L., and Köthe, U. (2024). Lifting architectural constraints of injective flows. In *International Conference on Learning Representations*.

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

The pandas development team (2020). Pandas-dev/pandas: Pandas.

Toth, P., Rezende, D. J., Jaegle, A., Racanière, S., Botev, A., and Higgins, I. (2020). Hamiltonian generative networks. In *International Conference on Learning Representations*.

Weinstock, R. (1974). *Calculus of Variations: With Applications to Physics and Engineering*. Courier Corporation.

Wildberger, J., Dax, M., Buchholz, S., Green, S. R., Macke, J. H., and Schölkopf, B. (2023). Flow Matching for Scalable Simulation-Based Inference. In *Advances in Neural Information Processing Systems*.

Zhang, M., Hayes, P., Bird, T., Habib, R., and Barber, D. (2020). Spread divergence. In *International Conference on Machine Learning*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Not applicable

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Yes
   (c) Clear explanations of any assumptions. Yes
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Not applicable
   (c) New assets either in the supplemental material or as a URL, if applicable. Not applicable
   (d) Information about consent from data providers/curators. Not applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not applicable
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not applicable

Supplementary Materials

OVERVIEW

The appendix is structured into three parts:

- Appendix A: A restatement and proof of all theoretical claims in the main text, along with some additional results.
  - Appendix A.1: The gradient of the log-determiant can be written as a trace.
  - Appendix A.2: A derivation of the loss as a lower bound on a KL divergence.
  - Appendix A.3: A bound on the difference between the true gradient of the log-determinant and the estimator used in this work.
  - Appendix A.4: Properties of the critical points of the loss.
  - Appendix A.5: Exploration of behavior of the loss in the low β regime, where the solution may not be globally invertible.
- Appendix B: Practical tips on how to train free-form flows and adapt them to new problems.
  - Appendix B.1: Tips on how to set up and initialize the model.
  - Appendix B.2: Code for computing the loss function.
  - Appendix B.3: Details on how to estimate likelihoods.
  - Appendix B.4: Tips on how to tune β.
- Appendix C: Details necessary to reproduce all experimental results in the main text.
  - Appendix C.1: Simulation-based inference.
  - Appendix C.2: Molecule generation.

A THEORETICAL CLAIMS

This section contains restatements and proofs of all theoretical claims in the main text.

A.1 Gradient via Trace

Theorem A.1. *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ be a C^1 invertible function parameterized by θ. Then, for all $x \in \mathbb{R}^D$:*

$$\nabla_{\theta_i} \log |J_\theta(x)| = \operatorname{tr}\left((\nabla_{\theta_i} J_\theta(x))(J_\theta(x))^{-1}\right).$$

Proof. Jacobi's formula states that, for a matrix $A(t)$ parameterized by t, the derivative of the determinant is

$$\frac{\mathrm{d}}{\mathrm{d}t}|A(t)| = |A(t)| \operatorname{tr}\left(A(t)^{-1}\frac{\mathrm{d}A(t)}{\mathrm{d}t}\right)$$

and hence

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\log|A(t)| &= |A(t)|^{-1}\frac{\mathrm{d}}{\mathrm{d}t}|A(t)| \\ &= \operatorname{tr}\left(A(t)^{-1}\frac{\mathrm{d}A(t)}{\mathrm{d}t}\right) \\ &= \operatorname{tr}\left(\frac{\mathrm{d}A(t)}{\mathrm{d}t}A(t)^{-1}\right)\end{aligned}$$

using the cyclic property of the trace in the last step. Applying this formula, with $A = J_\theta(x)$ and $t = \theta_i$ gives the result. □

A.2 Loss Derivation

Here we derive the loss function via an upper bound on a Kullback-Leibler (KL) divergence. Before doing so, let us establish some notation and motivation.

Our generative model is as follows:

$$\begin{aligned}p(z) &= \mathcal{N}(z; 0, I) \\ p_\phi(x \mid z) &= \delta(x - g_\phi(z))\end{aligned}$$

meaning that to generate data we sample from a standard normal latent distribution and pass the sample through the generator network g_ϕ. The corresponding inference model is:

$$\begin{aligned}q(x) &= \text{data distribution} \\ q_\theta(z \mid x) &= \delta(z - f_\theta(x))\end{aligned}$$

Our goal is to minimize the KL divergence

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(q(x) \parallel p_\phi(x)) &= \mathbb{E}_{q(x)}\left[\log\frac{q(x)}{p_\phi(x)}\right] \\ &= \mathbb{E}_{q(x)}\left[-\log\int p_\phi(x,z)\mathrm{d}z\right] - h(q(x))\end{aligned}$$

where h denotes the differential entropy. Unfortunately this divergence is intractable, due to the integral over z (though it would be tractable if g_ϕ^{-1} and $\log|J_{g_\phi}(z)|$ are tractable due to the change of variables formula – in this case the model would be a typical normalizing flow). The variational autoencoder (VAE) is a latent variable model which solves this problem by minimizing

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(q_\theta(x,z) \parallel p_\phi(x,z)) &= E_{q_\theta(x,z)}\left[\log\frac{q_\theta(x,z)}{p_\phi(x,z)}\right] \\ &= E_{q_\theta(x,z)}\left[\log\frac{q(x)}{p_\phi(x)} + \log\frac{q_\theta(z \mid x)}{p_\phi(z \mid x)}\right] \\ &= \mathcal{D}_{\mathrm{KL}}(q(x) \parallel p_\phi(x)) + E_{q(x)}\left[\mathcal{D}_{\mathrm{KL}}(q_\theta(z \mid x) \parallel p_\phi(z \mid x))\right] \\ &\geq \mathcal{D}_{\mathrm{KL}}(q(x) \parallel p_\phi(x))\end{aligned}$$

The inequality comes from the fact that KL divergences are always non-negative. Unfortunately this KL divergence is not well-defined due to the delta distributions, which make the joint distributions over x and z degenerate. Unless the support of $q_\theta(x, z)$ and $p_\phi(x, z)$ exactly overlap, which is very unlikely for arbitrary f_θ and g_ϕ, the divergence will be infinite. The solution is to introduce an auxiliary variable $\tilde{x}$ which is the data with some added Gaussian noise:

$$p(\tilde{x} \mid x) = q(\tilde{x} \mid x) = \mathcal{N}(\tilde{x}; x, \sigma^2 I)$$

The generative model over z and $\tilde{x}$ is therefore

$$\begin{aligned} p(z) &= \mathcal{N}(z; 0, I) \\ p_\phi(\tilde{x} \mid z) &= \mathcal{N}(\tilde{x}; g_\phi(z), \sigma^2 I) \end{aligned}$$

and the inference model is

$$\begin{aligned} q(\tilde{x}) &= \int q(x) q(\tilde{x} \mid x) \mathrm{d}x \\ q(\tilde{x} \mid x) &= \mathcal{N}(\tilde{x}; x, \sigma^2 I) \\ q_\theta(z \mid \tilde{x}) &= \frac{\int q(x) q(\tilde{x} \mid x) q_\theta(z \mid x) \mathrm{d}x}{\int q(x) q(\tilde{x} \mid x) \mathrm{d}x} \end{aligned}$$

Now the relationship between $\tilde{x}$ and z is stochastic and we can safely minimize the KL divergence which will always take on finite values:

$$\mathcal{D}_{\mathrm{KL}}(q_\theta(\tilde{x}, z) \parallel p_\phi(\tilde{x}, z)) \geq \mathcal{D}_{\mathrm{KL}}(q(\tilde{x}) \parallel p_\phi(\tilde{x}))$$

The KL divergence between the noised variables is known as a spread KL divergence $\tilde{\mathcal{D}}_{\mathrm{KL}}$ (Zhang et al., 2020):

$$\tilde{\mathcal{D}}_{\mathrm{KL}}(q(x) \parallel p_\phi(x)) = \mathcal{D}_{\mathrm{KL}}(q(\tilde{x}) \parallel p_\phi(\tilde{x}))$$

For convenience, here is the definition of $\mathcal{L}^{f^{-1}}$:

$$\mathcal{L}^{f^{-1}} = \mathbb{E}_{q(x)p(v)} \left[-\log p(Z = f_\theta(x)) - v^T J_\theta \mathtt{SG}(J_\theta^{-1} v) + \beta \|x - g_\phi(f_\theta(x))\|^2 \right] \tag{10}$$

which has the same gradients (with respect to model parameters) as

$$\mathbb{E}_{q(x)} \left[-\log p(Z = f_\theta(x)) - \log |J_\theta| + \beta \|x - g(f(x))\|^2 \right]$$

Now we restate the theorem from the main text:

Theorem A.2. *Let f_θ and g_ϕ be C^1 and let f_θ be globally invertible. Define the spread KL divergence $\tilde{\mathcal{D}}_{KL}$ as the KL divergence between distributions convolved with isotropic Gaussian noise of variance σ^2. Let $\beta = 1/2\sigma^2$. Then there exists a function D of θ and ϕ such that*

$$\nabla_\theta \mathcal{L}^{f^{-1}} = \nabla_\theta D \quad \textit{and} \quad \nabla_\phi \mathcal{L}^{f^{-1}} = \nabla_\phi D$$

and

$$D \geq \tilde{\mathcal{D}}_{KL}(q(x) \parallel p_\phi(x))$$

As a result, minimizing $\mathcal{L}^{f^{-1}}$ is equivalent to minimizing an upper bound on $\tilde{\mathcal{D}}_{KL}(q(x) \parallel p_\phi(x))$.

Proof. Let

$$D = \mathcal{D}_{\mathrm{KL}}(q_\theta(\tilde{x}, z) \parallel p_\phi(\tilde{x}, z))$$

We will use the identity (Papoulis and Pillai, 2002)

$$h(Z) = h(X) + \mathbb{E}[\log |J_f(X)|]$$

where h is the differential entropy, and the random variables are related by $Z = f(X)$ where f is invertible. As a result,

$$h(q(z \mid \tilde{x})) = h(q(x \mid \tilde{x})) + \mathbb{E}_{q(x|\tilde{x})}[\log |J_f(x)|] = \mathbb{E}_{q(x)}[\log |J_f(x)|] + \text{const.} \tag{11}$$

In the following, drop θ and ϕ subscripts for convenience. The unspecified extra terms are constant with respect to network parameters. Let ϵ be a standard normal variable. Then:

$$
\begin{aligned}
D &= \mathbb{E}_{q(\tilde{x},z)}\left[\log q(\tilde{x}) + \log q(z \mid \tilde{x}) - \log p(z) - \log p(\tilde{x} \mid z)\right] \\
&= \mathbb{E}_{q(\tilde{x})}\left[-h(q(z \mid \tilde{x}))\right] + \mathbb{E}_{q(\tilde{x},z)}\left[-\log p(z) - \log p(\tilde{x} \mid z)\right] + \text{const.} && (12)\\
&= \mathbb{E}_{q(x)}\left[-\log |J_f(x)|\right] + \mathbb{E}_{q(\tilde{x},z)}\left[-\log p(z) - \log p(\tilde{x} \mid z)\right] + \text{const.} && (13)\\
&= \mathbb{E}_{q(x)q(\epsilon)}\left[-\log |J_f(x)| - \log p(Z = f(x)) - \log p(\tilde{X} = x + \sigma\epsilon \mid Z = f(x))\right] + \text{const.} && (14)\\
&= \mathbb{E}_{q(x)q(\epsilon)}\left[-\log |J_f(x)| - \log p(Z = f(x)) + \frac{1}{2\sigma^2}\|x + \sigma\epsilon - g(f(x))\|^2\right] + \text{const.} && (15)\\
&= \mathbb{E}_{q(x)q(\epsilon)}\left[-\log |J_f(x)| - \log p(Z = f(x)) + \frac{1}{2\sigma^2}\|x - g(f(x))\|^2 + \frac{1}{\sigma}\epsilon^\top(x - g(f(x)))\right] + \text{const.} && (16)\\
&= \mathbb{E}_{q(x)}\left[-\log |J_f(x)| - \log p(Z = f(x)) + \beta\|x - g(f(x))\|^2\right] + \text{const.} && (17)
\end{aligned}
$$

Where the following steps were taken:

- Regard $\mathbb{E}_{q(\tilde{x})}[\log q(\tilde{x})] = -h(q(\tilde{x}))$ as a constant (eq. (12))
- Substitute in eq. (11) and regard $h(q(x \mid \tilde{x}))$ as constant (eq. (13))
- Make a change of variables from $\tilde{x}, z$ to x, ϵ with $\tilde{x} = x + \sigma\epsilon$ and $z = f(x)$ (eq. (14))
- Substitute the log-likelihood of the Gaussian $p(\tilde{x} \mid z)$, discard constant terms (eq. (15))
- Expand the final quadratic term, discard the constant term in $\|\epsilon\|^2$ (eq. (16))
- Evaluate the expectation over ϵ, noting that ϵ is independent of x and $\mathbb{E}[\epsilon] = 0$. Substitute $\beta = 1/2\sigma^2$ (eq. (17))
- Recognize that the final expression has the same gradient as $\mathcal{L}^{f^{-1}}$ from eq. (10)

Since the extra terms are constant with respect to θ and ϕ we have

$$\nabla_\theta \mathcal{L}^{f^{-1}} = \nabla_\theta D \quad \text{and} \quad \nabla_\phi \mathcal{L}^{f^{-1}} = \nabla_\phi D$$

and

$$D \geq \mathcal{D}_{\mathrm{KL}}(q(\tilde{x}) \parallel p_\phi(\tilde{x})) = \tilde{\mathcal{D}}_{\mathrm{KL}}(q(x) \parallel p_\phi(x))$$

was already established. As a result the gradients of $\mathcal{L}^{f^{-1}}$ are an unbiased estimate of the gradients of D and minimizing $\mathcal{L}^{f^{-1}}$ under stochastic gradient descent will converge to the same solutions as when minimizing D. □

We can also demonstrate the related bound:

$$D' \geq \mathcal{D}_{\mathrm{KL}}(q(x) \parallel p_\theta(x))$$

where $D' = \mathcal{L}^{f^{-1}} + \text{const.}$ (with a different constant to D). This is easy to see, since $\mathcal{L}^{f^{-1}} = \mathcal{L}_{\mathrm{ML}}^{f^{-1}} + \beta\mathcal{L}_{\mathrm{R}}^{f^{-1}}$ and $\mathcal{L}_{\mathrm{ML}}^{f^{-1}} = \mathcal{D}_{\mathrm{KL}}(q(x) \parallel p_\theta(x)) + \text{const.}$ and $\mathcal{L}_{\mathrm{R}}^{f^{-1}} \geq 0$.

A.3 Error Bound

Theorem A.3. *Let f_θ and g_ϕ be C^1, let J_θ be the Jacobian of f_θ at x and let J_ϕ be the Jacobian of g_ϕ at $f_\theta(x)$. Suppose that f_θ is locally invertible at x, meaning $J_\theta(x)$ is an invertible matrix. Let $\|\cdot\|_F$ be the Frobenius norm of a matrix. Then the absolute difference between $\nabla_{\theta_i} \log |J_\theta(x)|$ and the trace-based approximation is bounded:*

$$|\mathrm{tr}((\nabla_{\theta_i} J_\theta)J_\phi) - \nabla_{\theta_i} \log |J_\theta|| \leq \|J_\theta^{-1}\nabla_{\theta_i} J_\theta\|_F \|J_\phi J_\theta - I\|_F$$

Proof. The Cauchy-Schwarz inequality states that, for an inner product $\langle \cdot, \cdot \rangle$

$$|\langle u, v \rangle|^2 \leq \langle u, u \rangle \langle v, v \rangle$$

The trace forms the so-called Frobenius inner product over matrices with $\langle A, B \rangle_F = \operatorname{tr}(A^T B)$. Applying the inequality gives

$$\begin{aligned} |\operatorname{tr}(A^T B)|^2 &\leq \operatorname{tr}(A^T A) \operatorname{tr}(B^T B) \\ &= \|A\|_F^2 \|B\|_F^2 \end{aligned}$$

with $\|A\|_F = \sqrt{\operatorname{tr}(A^T A)}$ the Frobenius norm of A.

Recall from theorem A.1 that

$$\nabla_{\theta_i} \log |J_\theta| = \operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\theta^{-1})$$

Therefore

$$\begin{aligned} |\operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\phi) - \nabla_{\theta_i} \log |J_\theta|| &= \left|\operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\phi) - \operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\theta^{-1})\right| \\ &= \left|\operatorname{tr}((\nabla_{\theta_i} J_\theta)(J_\phi - J_\theta^{-1}))\right| \\ &= \left|\operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\theta^{-1} (J_\theta J_\phi - I))\right| \\ &\leq \|(\nabla_{\theta_i} J_\theta) J_\theta^{-1}\|_F \|J_\theta J_\phi - I\|_F \end{aligned}$$

where the last line is application of the Cauchy-Schwarz inequality and the cyclicity of the trace. □

Theorem A.4. *Suppose the conditions of theorem A.3 hold but extend local invertibility of f_θ to invertibility wherever $q(x)$ has support. Then the difference in gradients between $\mathcal{L}^g$ and $\mathcal{L}^{f^{-1}}$ is bounded:*

$$\left|\nabla_{\theta_i} \mathcal{L}^g - \nabla_{\theta_i} \mathcal{L}^{f^{-1}}\right| \leq \mathbb{E}_{q(x)}\left[\|(\nabla_{\theta_i} J_\theta) J_\theta^{-1}\|_F^2\right]^{\frac{1}{2}} \mathbb{E}_{q(x)}\left[\|J_\theta J_\phi - I\|_F^2\right]^{\frac{1}{2}}$$

Proof. In addition to the Cauchy-Schwarz inequality used in the proof to theorem A.3, we will also require Jensen's inequality for a convex function $\alpha : \mathbb{R} \to \mathbb{R}$

$$\alpha(\mathbb{E}_{q(x)}[x]) \leq \mathbb{E}_{q(x)}[\alpha(x)]$$

and Hölder's inequality (with $p = q = 2$) for random variables X and Y

$$\mathbb{E}[|XY|] \leq \mathbb{E}[|X|^2]^{\frac{1}{2}} \mathbb{E}[|Y|^2]^{\frac{1}{2}}$$

The only difference between $\mathcal{L}^g$ and $\mathcal{L}^{f^{-1}}$ is in the estimation of the gradient of the log-determinant. We use this fact, along with the inequalities, which we apply in the Jensen, Cauchy-Schwarz, Hölder order:

$$\begin{aligned} \left|\nabla_{\theta_i} \mathcal{L}^g - \nabla_{\theta_i} \mathcal{L}^{f^{-1}}\right| &= \left|\mathbb{E}_{q(x)}\left[\operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\phi)\right] - \mathbb{E}_{q(x)}\left[\nabla_{\theta_i} \log |J_\theta|\right]\right| \\ &= \left|\mathbb{E}_{q(x)}\left[\operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\theta^{-1} (J_\theta J_\phi - I))\right]\right| \\ &\leq \mathbb{E}_{q(x)}\left[\left|\operatorname{tr}((\nabla_{\theta_i} J_\theta) J_\theta^{-1} (J_\theta J_\phi - I))\right|\right] \\ &\leq \mathbb{E}_{q(x)}\left[\|(\nabla_{\theta_i} J_\theta) J_\theta^{-1}\|_F \|J_\theta J_\phi - I\|_F\right] \\ &\leq \mathbb{E}_{q(x)}\left[\|(\nabla_{\theta_i} J_\theta) J_\theta^{-1}\|_F^2\right]^{\frac{1}{2}} \mathbb{E}_{q(x)}\left[\|J_\theta J_\phi - I\|_F^2\right]^{\frac{1}{2}} \end{aligned}$$

□

A.4 Critical Points

Theorem A.5. *Let f_θ and g_ϕ be C^1 and let f_θ be globally invertible. Suppose $q(x)$ is finite and has support everywhere. Then the critical points of $\mathcal{L}^{f^{-1}}$ (for any $\beta > 0$) are such that*

1. *$g_\phi(z) = f_\theta^{-1}(z)$ for all z, and*

2. *$p_\theta(x) = q(x)$ for all x, and*
3. *All critical points are global minima*

Furthermore, every minimum of $\mathcal{L}^{f^{-1}}$ is a critical point of $\mathcal{L}^g$. If the reconstruction loss is minimal, $\mathcal{L}^g$ has no additional critical points.

Proof. In the following we will use Einstein notation, meaning that repeated indices are summed over. For example, $a_i b_i$ is shorthand for $\sum_i a_i b_i$. We will drop θ and ϕ subscripts to avoid clutter. We will also use primes to denote derivatives, for example: $f'(x) = J_f(x)$. In addition, gradients with respect to parameters should be understood as representing the gradient of a single parameter at a time, so $\nabla_\theta \mathcal{L}$ is shorthand for $(\nabla_{\theta_1}\mathcal{L}, \dots)$.

We will use the calculus of variations to find the critical points on a functional level. For a primer on calculus of variations, please see Weinstock (1974). Our loss is of the form

$$\mathcal{L}^{f^{-1}} = \int \lambda(x, f, f', g)\mathrm{d}x$$

with

$$\lambda(x, f, f', g) = q(x)\left(\frac{1}{2}\|f(x)\|^2 - \log|f'(x)| + \beta\|g(f(x)) - x\|^2\right)$$

By the Euler-Lagrange equations, critical points satisfy

$$\frac{\partial \lambda}{\partial g_i} = 0$$

for all i and

$$\frac{\partial \lambda}{\partial f_i} - \frac{\partial}{\partial x_j}\left(\frac{\partial \lambda}{\partial f'_{ij}}\right) = 0$$

for all i.

Taking the derivative with respect to g:

$$\frac{\partial \lambda}{\partial g_i} = q(x) \cdot 2\beta(g(f(x)) - x)_i = 0$$

and hence $g(f(x)) - x = 0$ for all x (since $q(x) > 0$). By a change of variables with $z = f(x)$, this means $g = f^{-1}$. Therefore we have proven statement 1.

Now differentiating with respect to f and substituting $g = f^{-1}$:

$$\begin{aligned}\frac{\partial \lambda}{\partial f_i} &= q(x)\left(f_i(x) + 2\beta(g(f(x)) - x)_j g'_{ji}(f(x))\right)\\ &= q(x) f_i(x)\end{aligned}$$

and with respect to f':

$$\begin{aligned}\frac{\partial \lambda}{\partial f'_{ij}} &= -q(x)(f'(x)^{-1})_{lk}\frac{\partial f'_{kl}}{\partial f'_{ij}}\\ &= -q(x)(f'(x)^{-1})_{ji}\end{aligned}$$

meaning

$$\begin{aligned}\frac{\partial}{\partial x_j}\left(\frac{\partial \lambda}{\partial f'_{ij}}\right) &= -\frac{\partial}{\partial x_j} q(x)(f'(x)^{-1})_{ji} - q(x)\frac{\partial}{\partial x_j}(f'(x)^{-1})_{ji}\\ &= -q(x)\left(\frac{\partial}{\partial x_j}\log q(x)(f'(x)^{-1})_{ji} + \frac{\partial}{\partial x_j}(f'(x)^{-1})_{ji}\right)\end{aligned}$$

Putting it together means

$$q(x)\left(f_i(x) + \frac{\partial}{\partial x_j}\log q(x)(f'(x)^{-1})_{ji} + \frac{\partial}{\partial x_j}(f'(x)^{-1})_{ji}\right) = 0$$

By dividing by $q(x)$ and multiplying by $f'_{ik}(x)$, we have

$$\frac{\partial}{\partial x_k}\log q(x) = -f_i(x)f'_{ik}(x) - \frac{\partial}{\partial x_j}(f'(x)^{-1})_{ji}f'_{ik}(x) \tag{18}$$

Furthermore, since $f(x)$ is invertible:

$$\frac{\partial}{\partial x_j}\left(f'(x)^{-1}f'(x)\right)_{jk} = \frac{\partial}{\partial x_j}\delta_{jk} = 0$$

Then using the product rule:

$$\frac{\partial}{\partial x_j}(f'(x)^{-1})_{ji}f'_{ik}(x) + (f'(x)^{-1})_{ji}f''_{ikj}(x) = 0 \tag{19}$$

In addition,

$$\frac{\partial}{\partial x_k}\log|f'(x)| = (f'(x)^{-1})_{ji}f''_{ijk}(x) \tag{20}$$

from Jacobi's formula, and since Hessians are symmetric in their derivatives, we can put together eq. (19) and eq. (20) to form

$$\frac{\partial}{\partial x_j}(f'(x)^{-1})_{ji}f'_{ik}(x) = -\frac{\partial}{\partial x_k}\log|f'(x)|$$

Substituting into eq. (18) and integrating, we find

$$\log q(x) = -\frac{1}{2}\|f(x)\|^2 + \log|f'(x)| + \text{const.}$$

The RHS is $\log p_\theta(x)$ by the change of variables formula, and hence $p_\theta(x) = q(x)$ for all x. This proves statement 2.

Now we will show that all critical points are global minima.

The negative log-likelihood part of $\mathcal{L}$ is bounded below by $h(q(x))$ and the reconstruction part is bounded below by zero. Hence if all critical points achieve a loss of $h(q(x))$ they are all global minima.

Since $g = f^{-1}$ for all critical points, the reconstruction loss is zero.

Since $p_\theta(x) = q(x)$ for all critical points, the negative log-likelihood loss is $h(q(x))$:

$$\mathbb{E}_{q(x)}[-\log p_\theta(x)] = \mathbb{E}_{q(x)}[-\log q(x)] = h(q(x))$$

This proves statement 3.

It now remains to show that every minimum of $\mathcal{L}^{f^{-1}}$ is a critical point of $\mathcal{L}^g$.

$\mathcal{L}^g$ is of the form:

$$\mathcal{L}^g = \int \tilde{\lambda}(x, f, f', g)\mathrm{d}x$$

with

$$\tilde{\lambda}(x, f, f', g) = q(x)\left(\frac{1}{2}\|f(x)\|^2 - \operatorname{tr}(f'(x)\mathtt{SG}(g'(f(x)))) + \beta\|g(f(x)) - x\|^2\right)$$

$$\frac{\partial\tilde{\lambda}}{\partial g_i} = q(x)\cdot 2\beta(g(f(x)) - x)_i$$

as before, and is zero with the substitution $g = f^{-1}$.

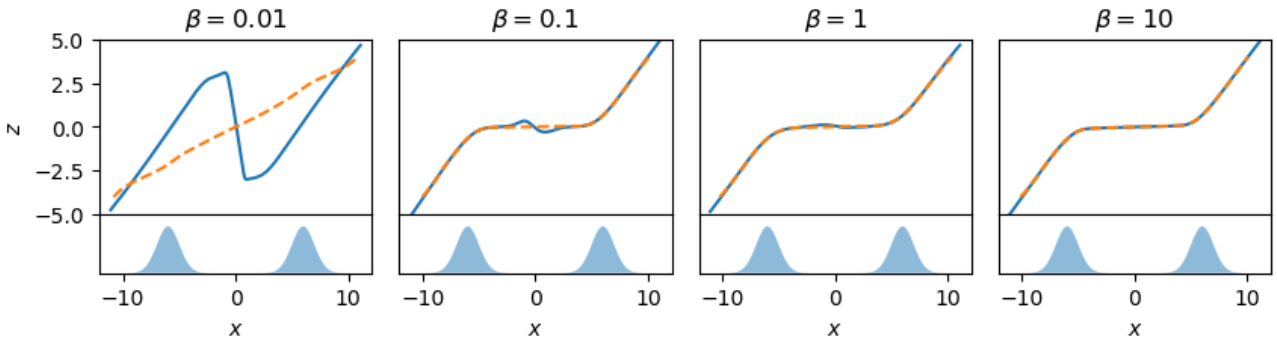

Figure 4: Solutions to $\mathcal{L}^{f^{-1}}$ for various β. The data is the two-component Gaussian mixture shown in the lower panels. Solid blue lines show f_θ and dashed orange lines show g_ϕ. Note that f_θ is not invertible between the mixtures when β is small.

$$\frac{\partial \tilde{\lambda}}{\partial f_i} = q(x) f_i(x)$$

as before and

$$\begin{aligned}\frac{\partial \tilde{\lambda}}{\partial f'_{ij}} &= -q(x) g'_{lk}(f(x)) \frac{\partial f'_{kl}}{\partial f'_{ij}} \\ &= -q(x) g'_{ji}(f(x)) \\ &= -q(x)(f'(x)^{-1})_{ji}\end{aligned}$$

with the substitution $g'(f(x)) = f'(x)^{-1}$. Since this is the same expression as before we must have

$$\frac{\partial \tilde{\lambda}}{\partial f_i} - \frac{\partial}{\partial x_j}\left(\frac{\partial \tilde{\lambda}}{\partial f'_{ij}}\right) = 0$$

meaning that f and g are critical with respect to $\tilde{\mathcal{L}}$. This shows that the critical points (and hence minima) of $\mathcal{L}^{f^{-1}}$ are critical points of $\mathcal{L}^g$.

In the case where f is not required to be globally invertible, $\mathcal{L}^g$ may have additional critical points when f is in fact not invertible (see fig. 2 in section 4.3 for an example). However, if the reconstruction loss is minimal, therefore zero, f will be invertible and the above arguments hold. If this is the case, there are no additional critical points of $\mathcal{L}^g$.

□

A.5 Ensuring Global Invertibility

Free-form flows use arbitrary neural networks f_θ and g_ϕ. Since we rely on the approximation $g_\phi \approx f_\theta^{-1}$, it is crucial that the reconstruction loss is as small as possible. We achieve this in practice by setting β large enough. In this section, we give the reasoning for this choice.

In particular, we show that when β is too small and the data is made up of multiple disconnected components, there are solutions to $\mathcal{L}^{f^{-1}}$ that are not globally invertible, even if f_θ is restricted to be locally invertible. We illustrate some of these solutions for a two-component Gaussian mixture in fig. 4. We approximate the density as zero more than 5 standard deviations away from each mean. When β is extremely low the model gives up on reconstruction and just tries to transform each component to the latent distribution individually.

Let us now analyse the behavior of this system mathematically. Our argument goes as follows: First, we assume that the data can be split into disconnected regions. Then it might be favorable that the encoder computes latent

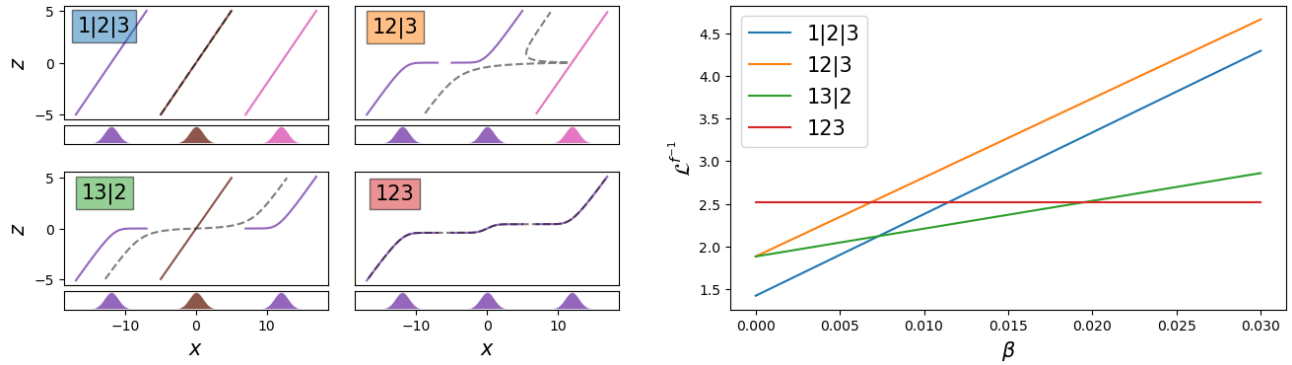

Figure 5: Intuition behind theorem theorem A.6: Comparison of invalid solutions to learning a Gaussian mixture of three modes with non-invertible encoders (blue, orange, green), compared to an invertible encoder (red). *(Left)* As the encoder is not invertible by construction, it may learn to reuse each latent code z once for each disconnected component. This reduces the negative-likelihood at each point, as the derivative $f'_\theta(x)$ is larger at each data point. The decoder (dotted gray line) then cannot reconstruct the data. *(Right)* Increasing β increases the importance of reconstruction over maximum likelihood and thus selects the best solution (red).

codes such that each region covers the full latent space. This means that each latent code z is assigned once in each region. This is a valid encoder function f_θ and we compute its loss $\mathcal{L}^{f^{-1}}$ in theorem A.6. In corollary A.6.1, we show that when $\beta < \beta_{\text{crit}}$ solutions which are not globally invertible have the lowest loss. It is thus vital that $\beta > \beta_{\text{crit}}$ or larger to ensure the solution is globally invertible.

Theorem A.6. *Let f_θ and g_ϕ be C^1. Suppose that $q(x)$ may not have support everywhere and allow f_θ to be non-invertible in the regions where $q(x) = 0$. Suppose the set $\mathcal{S} = \{x : q(x) > 0\}$ is made up of k disjoint, connected components: $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{S}_i$.*

For each partition $\mathcal{P}$ of $\{\mathcal{S}_i\}_{i=1}^k$ consider solutions of $\mathcal{L}^{f^{-1}}$ where

1. *f_θ transforms each element of the partition to $p(z)$ individually, and*
2. *g_ϕ is chosen (given f_θ) such that $R_{\min} = \mathbb{E}_x \left[\|g_\phi(f_\theta(x)) - x\|^2\right]$ is minimal*

The loss achieved is

$$\mathcal{L}^{f^{-1}} = h(q(x)) - H(\mathcal{P}) + \beta R_{\min}(\mathcal{P})$$

where $h(q(x))$ is the differential entropy of the data distribution and $H(\mathcal{P})$ is the entropy of α where $\alpha_i = \int_{\mathcal{P}_i} q(x)\mathrm{d}x$.

Note that the solutions in theorem A.6 are not necessarily minima of $\mathcal{L}^{f^{-1}}$, they just demonstrate what values it can take.

Proof. Let $\mathcal{L} = \mathbb{E}_v[\mathcal{L}^{f^{-1}}]$. The loss can be split into negative log-likelihood and reconstruction parts: $\mathcal{L} = \mathcal{L}_{\text{NLL}} + \beta\mathcal{L}_{\text{R}}$.

Consider a partition $\mathcal{P}$ of $\{\mathcal{S}_i\}_{i=1}^k$. Let $q_i(x)$ be the distribution which is proportional to $q(x)$ when $x \in \mathcal{P}_i$ but zero otherwise (weighted to integrate to 1):

$$q_i(x) = \frac{1}{\alpha_i} q(x) \mathbb{1}_{x \in \mathcal{P}_i}$$

with

$$\alpha_i = \int_{\mathcal{P}_i} q(x)\mathrm{d}x$$

The type of solution described in the theorem statement will be such that $p_\theta(x) = q_i(x)$ for $x \in \mathcal{P}_i$. This means that

$$\begin{aligned}
\mathcal{L}_{\text{NLL}} &= -\int q(x) \log p_\theta(x) \mathrm{d}x \\
&= -\sum_i \int_{\mathcal{P}_i} q(x) \log p_\theta(x) \mathrm{d}x \\
&= -\sum_i \alpha_i \int_{\mathcal{P}_i} q_i(x) \log q_i(x) \mathrm{d}x \\
&= \sum_i \alpha_i h(q_i(x))
\end{aligned}$$

We also have

$$\begin{aligned}
h(q(x)) &= -\sum_i \int_{\mathcal{P}_i} q(x) \log q(x) \mathrm{d}x \\
&= -\sum_i \alpha_i \int_{\mathcal{P}_i} q_i(x) \log(\alpha_i q_i(x)) \mathrm{d}x \\
&= \sum_i \alpha_i \left(h(q_i(x)) - \log \alpha_i\right) \\
&= \mathcal{L}_{\text{NLL}} + H(\alpha)
\end{aligned}$$

and therefore

$$\mathcal{L}_{\text{NLL}} = h(q(x)) - H(\mathcal{P})$$

Clearly $\mathcal{L}_{\text{R}} = R_{\min}(\mathcal{P})$. As a result

$$\mathcal{L} = h(q(x)) - H(\mathcal{P}) + \beta R_{\min}(\mathcal{P})$$

□

Corollary A.6.1. *Call the solution where $\mathcal{P} = \mathcal{S}$ the globally invertible solution. For this solution, $\mathcal{L}^{f^{-1}} = h(q(x))$.*

For a given partition $\mathcal{P}$, the corresponding solution described in theorem A.6 has lower loss than the globally invertible solution when $\beta < \beta_{crit}$ where

$$\beta_{crit} = \frac{H(\mathcal{P})}{R_{\min}(\mathcal{P})}$$

Proof. If $\mathcal{P} = \mathcal{S}$ then $\alpha = (1)$. Therefore $H(\mathcal{P}) = 0$. Since f is invertible in this case, $R_{\min}(\mathcal{P}) = 0$. Therefore $\mathcal{L} = h(q(x))$.

Now consider a partition $\mathcal{P} \neq \mathcal{S}$. This has loss

$$\mathcal{L} = h(q(x)) - H(\mathcal{P}) + \beta R_{\min}(\mathcal{P})$$

By solving:

$$h(q(x)) - H(\mathcal{P}) + \beta R_{\min}(\mathcal{P}) \leq h(q(x))$$

we find

$$\beta \leq \beta_{\text{crit}} = \frac{H(\mathcal{P})}{R_{\min}(\mathcal{P})}$$

□

Corollary A.6.1 tells us that β must be large enough or the minima of $\mathcal{L}^{f^{-1}}$ will favor solutions which are not globally invertible. In practice, it is difficult to compute the value of β_{crit} for a given partition, as well as finding the partitions in the first place, so β must be tuned as a hyperparameter until a suitable value is found (see

appendix B.4). Note that $\beta > \beta_{\text{crit}}$ does not guarantee that the solution will be globally invertible and globally-invertible solutions may only be the minima of $\mathcal{L}^{f^{-1}}$ in the limit $\beta \to \infty$. However, for practical purposes a large value of β will be sufficient to get close to the globally invertible solution.

Various solutions for a three-component Gaussian mixture distribution are illustrated in fig. 5, along with the loss values as a function of β. Here we approximate regions five or more standard deviations away from the mean as having zero density, in order to partition the space into three parts as per theorem A.6. We see that each solution has a region of lower loss than the globally-invertible solution when $\beta < \beta_{\text{crit}}$ and that β must at least be greater than the largest β_{crit} (and potentially larger) in order to avoid non-globally-invertible solutions.

While this analysis is for $\mathcal{L}^{f^{-1}}$, the main conclusion carries over to $\mathcal{L}^{g}$, namely that β must be sufficiently large to ensure global invertibility. When optimizing $\mathcal{L}^{g}$, large β is especially important since the loss relies on the approximation $g_\phi \approx f_\theta^{-1}$ which is only achievable if f_θ is globally invertible.

B PRACTICAL GUIDE TO FREE-FORM FLOWS

This section gives a brief overview over how to get started with adapting free-form flows to a new problem.

B.1 Model setup

The pair of encoder $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$, which represents $z = f_\theta(x)$, and decoder $g_\phi : \mathbb{R}^D \to \mathbb{R}^D$, which represents $x = g_\phi(z)$, can be any pair of dimension-preserving neural networks. Any architecture is allowed. While in principle batch-norm violates the assumptions for our theorems (because the Jacobians of each item in the batch should be independent), this works well in practice. In our experiments, we found best performance when encoder and decoder each have a global skip connection:

$$z = f_\theta(x) = x + \tilde{f}_\theta(x)$$
$$x = g_\phi(z) = z + \tilde{f}_\theta(z).$$

This has the advantage that the network is initialized close to the identity, so that training starts close to the parameters where $x \approx g_\phi(f_\theta(x))$ and the reconstruction loss is already low.

Conditional distributions If the distribution to be learned should be conditioned on some context $c \in \mathbb{R}^C$, i.e. $p(x|c)$, feed the context as an additional input to both encoder $f_\theta : \mathbb{R}^D \times \mathbb{R}^C \to \mathbb{R}^D$ and decoder $g_\phi : \mathbb{R}^D \times \mathbb{R}^C \to \mathbb{R}^D$. For networks with a skip connection:

$$z = f_\theta(x; c) = x + \tilde{f}_\theta(x; c)$$
$$x = g_\phi(z; c) = z + \tilde{g}_\phi(z; c).$$

If they are multi-layer networks, we observe training to be accelerated when not only the first layer, but also subsequent layers get the input.

B.2 Training

The PyTorch code in listing 1 computes the gradient of free-form flows using backward autodiff. The inputs `encode` and `decode` can be arbitrary PyTorch functions.

B.3 Likelihood estimation

For a trained free-form flow, we are interested in how well the learnt model captures the original distribution. We would like to ask "How likely is our model to generate this set of data?" We can answer this question via the negative log-likelihood NLL, which is smaller the more likely the model is to generate these data points:

$$\text{NLL} = -\sum_{i=1}^{N_{\text{unseen}}} \log p_\phi(X = x_i).$$

```python
import torch
from math import sqrt, prod


def change_of_variables_surrogate(x: torch.Tensor, encode, decode):
    """
    Compute the per-sample surrogate for the change of variables gradient.

    Args: see below
    Returns:
        z: Latent code. Shape: (batch_size, *z_shape)
        x1: Reconstruction. Shape: (batch_size, *x_shape)
        Per-sample surrogate. Shape: (batch_size,)
    """
    x.requires_grad_()
    z = encode(x)

    # Sample v from sphere with radius sqrt(total_dim)
    batch_size, total_dim = x.shape[0], prod(x.shape[1:])
    v = torch.randn(batch_size, total_dim, device=x.device, dtype=x.dtype)
    v *= sqrt(total_dim) / torch.sum(v ** 2, -1, keepdim=True).sqrt()
    v = v.reshape(x.shape)

    # $ g'(z) v $ via forward-mode AD
    with torch.autograd.forward_ad.dual_level():
        dual_z = torch.autograd.forward_ad.make_dual(z, v)
        dual_x1 = decode(dual_z)
        x1, v1 = torch.autograd.forward_ad.unpack_dual(dual_x1)

    # $ v^T f'(x) $ via backward-mode AD
    v2, = torch.autograd.grad(z, x, v, create_graph=True)

    # $ v^T f'(x) stop_grad(g'(z)) v $
    surrogate = torch.sum((v2 * v1.detach()).reshape(batch_size, total_dim), -1)

    return z, x1, surrogate


def fff_loss(x: torch.Tensor, encode, decode, beta: float):
    """
    Compute the per-sample MLAE loss for a latent normal distribution
    Args:
        x: Input data. Shape: (batch_size, *x_shape)
        encode: Encoder function. Takes an input `x` and returns a latent code `z` of
            shape (batch_size, *z_shape).
        decode: Decoder function. Takes a latent code `z` and returns a reconstruction
            `x1` of shape (batch_size, *x_shape).
        beta: Weight of the reconstruction error.

    Returns:
        Per-sample loss. Shape: (batch_size,)
    """
    z, x1, surrogate = change_of_variables_surrogate(x, encode, decode)
    nll = torch.sum((z ** 2).reshape(x.shape[0], -1) ** 2).sum(-1) - surrogate
    return nll + beta * ((x - x1).reshape(x.shape[0], -1) ** 2).sum(-1)
```

Listing 1: PyTorch implementation of FFF gradient computation

For normalizing flows with analytically invertible encoder f_θ and decoder g_θ, evaluating the NLL can be achieved via the change of variables of the encoder, as the encoder Jacobian determinant is exactly the inverse of the decoder Jacobian determinant:

$$\begin{aligned} -\log p_\theta(X = x_i) &= -\log p(Z = g_\theta^{-1}(x)) + \log\det g'_\theta(g_\theta^{-1}(x)) \\ &= -\log p(Z = f_\theta(x)) - \log\det f'_\theta(x). \end{aligned}$$

The FFF encoder and decoder are only coupled via the reconstruction loss, and the distribution of the decoder (the actual generative model) might be slightly different from the encoder. We therefore compute the change of variables with the decoder Jacobian. In order to get the right latent code that generated a data point, we use the encoder $f_\theta(x)$:

$$\begin{aligned} -\log p_\phi(X = x_i) &= -\log p(Z = g_\phi^{-1}(x)) + \log\det g'_\phi(g_\phi^{-1}(x)) \\ &\approx -\log p(Z = f_\theta(x)) + \log\det g'_\phi(f_\theta(x)). \end{aligned} \tag{21}$$

This approximation $f_\theta(x) \approx g_\phi^{-1}(x)$ is sufficiently valid in practice. For example, for the Boltzmann generator on DW4, we find that the average distance between an input x and its reconstruction $x' = g_\phi(f_\theta(x))$ is 0.0253. Comparing the energy $u(x)$ to the energy $u(x')$ of the reconstruction, the mean absolute difference is 0.11, which is less than 1% of the energy range $\max_{x\in\mathcal{X}_{\text{test}}} u(x) - \min_{x\in\mathcal{X}_{\text{test}}} u(x) = 13.7$.

B.4 Determining the optimal reconstruction weight

Apart from the usual hyperparameters of neural network training such as the network architecture and training procedure, free-form flows have one additional hyperparameter, the reconstruction weight β. We cannot provide a rigorous argument for how β should be chosen at this stage.

However, we find that it is easy to tune in practice by monitoring the training negative log-likelihood over the first epoch (see eq. (21)). This involves computing the Jacobian $f'_\theta(x)$ explicitly. We can then do an exponential search on β:

1. If the negative log-likelihood is unstable (i.e. jumping values; reconstruction loss typically also jumps), increase β by a factor.
2. If the negative log-likelihood is stable, we are in the regime where training is stable but might be slow. Try decreasing β to see if that leads to training that is still stable yet faster.

For a rough search, it is useful to change β by factors of 10. We observe that there usually is a range of more than one order of magnitude for β where the optimization converges to the same quality. We find that training with larger β usually catches up with low β in late training. Higher β also ensures that the reconstruction loss is lower, so that likelihoods are more faithful, see appendix B.3.

C EXPERIMENTS

All our experiments can be reproduced via our public repository at `https://github.com/vislearn/FFF/`.

C.1 Simulation-Based Inference

Our models for the SBI benchmark use the same ResNet architecture as the neural spline flows Durkan et al. (2019) used as the baseline. It consists of 10 residual blocks of hidden width 50 and ReLU activations. Conditioning is concatenated to the input and additionally implemented via GLUs at the end of each residual block. We also define a simpler, larger architecture which consists of 2x256 linear layers followed by 4x256 residual blocks without GLU conditioning. We denote the architectures in the following as ResNet S and ResNet L. To find values for architecture size, learning rate, batch size and β we follow Wildberger et al. (2023) and perform a grid search to pick the best value for each dataset and simulation budget. As opposed to Wildberger et al. (2023) we run the full grid, but with greatly reduced search ranges, which are provided in Table 3, which amounts to a similar budget. The best hyperparameters for each setting are shown in Table 4. Notably, this table shows

	Range
Reconstruction weight β	$10, 25, 100, 500$
Learning rate	$\{1, 2, 5, 10\} \times 10^{-4}$
Batch size	$2^2, ..., 2^8$
Architecture size	S, L*

Table 3: Hyperparameter ranges for the grid search on the SBI benchmark. *We only perform the search over architecture size for the 100k simulation budget scenarios.

Dataset	Batch size	Learning rate	β	ResNet size
Bernouli glm	8/32/128	5×10^{-4}	25/25/500	S/S/L
Bernouli glm raw	16/64/32	$5/10/10 \times 10^{-4}$	25/25/50	S/S/L
Gaussian linear	8/128/128	$5/10/1 \times 10^{-4}$	25/500/500	S
Gaussian linear uniform	8/8/32	$5/2/5 \times 10^{-4}$	500/10/100	S/S/L
Gaussian mixture	4/16/32	$5/2/10 \times 10^{-4}$	10/500/25	S/S/L
Lotka Volterra	4/32/64	$10/10/5 \times 10^{-4}$	500/500/25	S
SIR	8/32/64	$10/10/5 \times 10^{-4}$	500/25/25	S
SLCP	8/32/32	5×10^{-4}	10/25/25	S/S/L
SLCP distractors	4/32/256	$5/10/5 \times 10^{-4}$	25/10/10	S
Two moons	4/16/32	$5/5/1 \times 10^{-4}$	500	S/S/L

Table 4: Hyperparamters found by the grid search for the SBI benchmark. Cells are split into the hyperparameters for all three simulation budgets, unless we use the same setting across all of them.

that our method oftentimes works well on the same datasets for a wide range different β values. The entire grid search was performed exclusively on compute nodes with "AMD Milan EPYC 7513 CPU" resources and took $\sim 14.500\text{h} \times 8$ cores total CPU time for a total of 4480 runs.

C.2 Molecule Generation

C.2.1 $E(n)$-GNN

For all experiments, we make use of the $E(n)$ equivariant graph neural network proposed by Satorras et al. (2021b) in the stabilized variant in Satorras et al. (2021a). It is a graph neural network that takes a graph (V, E) as input. Each node $v_i \in V$ is the concatenation of a vector in space $x_i \in \mathbb{R}^n$ and some additional node features $h_i \in \mathbb{R}^h$. The neural network consists of L layers, each of which performs an operation on $v^l = [x_i^l; h_i^l \to x_i^{l+1}; h_i^{l+1}]$. Spatial components are transformed *equivariant* under the Euclidean group $E(n)$ and feature dimensions are transformed *invariant* under $E(n)$.

$$
\begin{aligned}
\mathbf{m}_{ij} &= \phi_e\left(\boldsymbol{h}_i^l, \boldsymbol{h}_j^l, d_{ij}^2, a_{ij}\right), \\
\tilde{e}_{ij} &= \phi_{inf}(m_{ij}), \\
\boldsymbol{h}_i^{l+1} &= \phi_h\left(\boldsymbol{h}_i^l, \sum_{j \neq i} \tilde{e}_{ij} \mathbf{m}_{ij}\right), \\
\boldsymbol{x}_i^{l+1} &= \boldsymbol{x}_i^l + \sum_{j \neq i} \frac{\boldsymbol{x}_i^l - \boldsymbol{x}_j^l}{d_{ij} + 1} \phi_x\left(\boldsymbol{h}_i^l, \boldsymbol{h}_j^l, d_{ij}^2, a_{ij}\right)
\end{aligned}
$$

Here, $d_{ij} = \|x_i^l - x_j^l\|$ is the Euclidean distance between the spatial components, a_{ij} are optional edge features that we do not use. The $\tilde{e}_{ij}$ are normalized for the input to ϕ_h. The networks $\phi_e, \phi_{inf}, \phi_h, \phi_x$ are learnt fully-connected neural networks applied to each edge or node respectively.

C.2.2 Latent distribution

As mentioned in section 5, the latent distribution must be invariant under the Euclidean group. While rotational invariance is easy to fulfill, a normalized translation invariant distribution does not exist. Instead, we adopt the

	DW4	LJ13	LJ55
Layer count	20	8	10
Reconstruction weight β	10	200	500
Learning rate	0.001	0.001	0.001
Learning rate scheduler	One cycle	-	- / Exponential $\gamma = 0.99999$
Gradient clip	1	1	0.1
Batch size	256	256	48
Duration	50 epochs	400 epochs	300k / 450k steps

Table 5: Hyperparameters used for the Boltzmann generator tasks. The format "A / B" specifies a two-step training.

approach in Köhler et al. (2020) to consider the subspace where the mean position of all atoms is at the origin: $\sum_{i=1}^{N} x_i = 0$. We then place a normal distribution over this space. By enforcing the output of the $E(n)$-GNN to be zero-centered as well, this yields a consistent system. See Köhler et al. (2020) for more details.

C.2.3 Boltzmann Generators on DW4, LJ13 and LJ55

We consider the two potentials

$$\text{Double well (DW):} \qquad v_{\texttt{DW}}(x_1, x_2) = \frac{1}{2\tau}\left(a(d-d_0) + b(d-d_0)^2 + c(d-d_0)^4\right),$$
$$\text{Lennard-Jones (LJ):} \qquad v_{\texttt{LJ}}(x_1, x_2) = \frac{\epsilon}{2\tau}\left(\left(\frac{r_m}{d}\right)^{12} - 2\left(\frac{r_m}{d}\right)^{6}\right).$$

Here, $d = \|x_1 - x_2\|$ is the Euclidean distance between two particles. The DW parameters are chosen as $a = 0, b = -4, c = 0.9, d_0 = 4$ and $\tau = 1$. For LJ, we choose $r_m = 1, \epsilon = 1$ and $\tau = 1$. This is consistent with (Klein et al., 2023) and we use their MCMC samples as data, of which we use 400k samples for validation and 500k for testing the final model.

We give hyperparameters for training the models in table 5. We consistently use the Adam optimizer. While we use the $E(n)$-GNN as our architecture, we do not make use of the features h because the Boltzmann distributions in question only concern positional information. Apart from the varying layer count, we choose the following $E(n)$-GNN model parameters as follows: Fully connected node and edge networks (which are invariant) have one hidden layers of hidden width 64 and SiLU activations. Two such invariant blocks are executed sequentially to parameterize the equivariant update. We compute the edge weights $\tilde{e}_{ij}$ via attention. Detailed choices for building the network can be determined from the code in Hoogeboom et al. (2022).

C.2.4 QM9 Molecule Generation

For the QM9 (Ruddigkeit et al., 2012; Ramakrishnan et al., 2014) experiment, we again employ a $E(3)$-GNN. This time, the dimension of node features h is composed of a one-hot encoding for the atom type and an ordinal value for the atom charge. Like Satorras et al. (2021a), we use variational dequantization for the ordinal features (Ho et al., 2019), and argmax flows for the categorical features (Hoogeboom et al., 2021). For QM9, the number of atoms may differ depending on the input. We represent the distribution of molecule sizes as a categorical distribution.

We again employ the $E(3)$-GNN with the same settings as for the Boltzmann generators. We use 16 equivariant blocks, train with Adam with a learning rate of 10^{-4} for 700 epochs. We then decay the learning rate by a factor of $\gamma = 0.99$ per epoch for another 100 epochs. We set reconstruction weight to $\beta = 2000$. We use a batch size of 64.

For both molecule generation tasks together, we used approximately 6,000 GPU hours on an internal cluster of NVIDIA A40 and A100 GPUs. A full training run on QM9 took approximately ten days on a single such GPU.

	NLL (↓)	Sampling time (↓)	
		Raw	incl. $\log q_\theta(x)$
	DW4		
$E(n)$-NF	1.72 ± 0.01	0.024 ms	0.10 ms
OT-FM	1.70 ± 0.02	0.034 ms	0.76 ms
E-OT-FM	1.68 ± 0.01	0.033 ms	0.75 ms
FFF	1.68 ± 0.01	0.026 ms	0.74 ms
	LJ13		
$E(n)$-NF	-16.28 ± 0.04	0.27 ms	**1.2** ms
OT-FM	-16.54 ± 0.03	0.77 ms	38 ms
E-OT-FM	-16.70 ± 0.12	0.72 ms	38 ms
FFF	**-17.09 ± 0.16**	**0.11** ms	3.5 ms
	LJ55		
OT-FM	-88.45 ± 0.04	40 ms	6543 ms
E-OT-FM	**-89.27 ± 0.04**	40 ms	6543 ms
FFF	-88.72 ± 0.16	**2.1** ms	**311** ms

Table 6: Boltzmann generator negative log-likelihood and sampling times, including the time to compute the density from the network Jacobians. Note that in all cases, the log prob could be distilled by a $E(3)$-invariant network with scalar output for faster density estimation. NLLs are due to Klein et al. (2023). Errors are the standard deviations over runs. The other models are based on an ODE trained via maximum likelihood ($E(n)$-NF, Satorras et al. (2021a)), and trained via Optimal Transport Flow Matching with (OT-FM) or without (E-OT-FM) equivariance-aware matching (Klein et al., 2023). $E(n)$-NF is too memory intensive to train on LJ55 efficiently. Expands table 1 in main text.

Setup	NLL
Classical NF (eq. (2))	10.55
NF + trace estimator (eq. (4))	10.60
INN as FFF	16.42
Transformer as FFF	10.10
ResNet as FFF, $K = 1, bs = 256$	10.60
ResNet as FFF, $K = 2, bs = 256$	10.18
ResNet as FFF, $K = 1, bs = 64$	12.96
ResNet as FFF, $K = 2, bs = 64$	12.30

Table 7: Ablation in terms of NLL on MINIBOONE: We start with a classical normalizing flow trained with exact likelihood, then add the trace estimator. Next, we compare different architectures trained as free-form flows. Finally, we compare using more Hutchinson samples K and varying batch size bs.

C.3 Ablation Studies

We study the effect of different modifications to our method in an ablation study on the MINIBOONE dataset shown in Table 7. Firstly, we train a classical normalizing flow, implemented as a coupling flow Dinh et al. (2015) (Classical NF) and compare it to a model where the exact likelihood loss of eq. (2) has been replaced by the trace estimator of eq. (4), but still using the exact inverse (NF + trace estimator). The resulting NLL shows that using eq. (4) provides a good estimate for the maximum likelihood objective, with only small deterioration due to the increased stochasticity. Next we train two identical coupling flow networks as FFF, meaning we no longer use their invertibility and instead rely on the reconstruction loss of eq. (7) to learn an inverse (INN as FFF). This shows that the coupling flow architecture is suboptimal as FFF, despite a guarantee of invertibility. Finally, we show that with the right architecture (ResNet as FFF/Transformer as FFF) we can reach and even outperform coupling flows. We also include an ablation study on the effect of increasing the the number of Hutchinson samples K vs. increasing batch size bs. Both measures reduce stochasticity of the optimizer and can lead to better performance, but we find that the effect of increasing batch size is more pronounced, and can lead to more improvements than increasing K at the same cost.

C.3.1 Software libraries

We build our code upon the following python libraries: PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon and The PyTorch Lightning team, 2019), Tensorflow (Abadi et al., 2015) for FID score evaluation, Numpy (Harris et al., 2020), Matplotlib (Hunter, 2007) for plotting and Pandas (McKinney, 2010; The pandas development team, 2020) for data evaluation.