Latent Space Refinement for Deep Generative Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep generative models are becoming widely used across science and industry for a variety of purposes. A common challenge is achieving a precise implicit or explicit representation of the data probability density. Recent proposals have suggested using classifier weights to refine the learned density of deep generative models. We extend this idea to all types of generative models and show how latent space refinement via iterated generative modeling can circumvent topological obstructions and improve precision. This methodology also applies to cases were the target model is non-differentiable and has many internal latent dimensions which must be marginalized over before refinement. We demonstrate our Latent Space Refinement (LaSeR) protocol on a variety of examples, focusing on the combinations of Normalizing Flows and Generative Adversarial Networks. We make all codes publicly available.[1]

1 Introduction

Generative models are essential tools for many aspects of scientific and engineering workflows. First-principles simulations encode physical laws and then samples from these simulations can be used for designing, performing, and interpreting a measurement. However, these physics-based simulations can be too slow or not precise enough for a growing number of studies. Deep learning-based techniques such as Generative Adversarial Networks (GAN) [1, 2], Variational Autoencoders (VAE) [3, 4], and Normalizing Flows (NF) [5, 6] are powerful surrogates that can accelerate slow simulations and model complex datasets that would otherwise be intractable to describe from first principles. For example, a growing number of studies are exploring these tools for high energy physics (HEP) applications [7–71].

A key difference in generative modeling between many scientific applications, including HEP, and typical industrial applications is that individual samples are often not useful. Inference is performed on a statistical basis and so it is essential to model the probability density precisely and not just match its support. Existing deep generative models have shown great promise in qualitatively modeling complex probability densities, but it is often challenging to achieve precision.

A useful strategy to improve the precision of generative models is to refine their predictions. For example, Ref. [72] showed how the classification part of a GAN can be used to reweight and resample the input probability distribution of the random noise. A similar idea was introduced in Ref. [41] that is not specific to GANs, whereby a classifier network is trained on the generated samples to produce weights that improve the precision of the generated probability distribution. We combine and extent these approaches by introducing the **La**tent **S**pac**e** **R**efinement (LaSeR) protocol, illustrated in Fig. 1. Like DCTRGAN [41, 73], LaSeR starts with any generative model $g(z)$ and trains a post-hoc classifier to distinguish the generated samples from the target data. A challenge with DCTRGAN is

[1] https://anonymous.4open.science/r/LaSeR

Submitted to NeurIPS 2021 Workshop on DGMs and Downstream Applications. Do not distribute.

that the results are weighted, which reduces the statistical power of a generated sample. The energy-based framework of Discriminator Driven Latent Sampling (DDLS) [72] produces unweighted samples by transferring the weights to the latent space and then performing Langevin Markov Chain Monte Carlo (MCMC) to produce unweighted samples.

We develop a more general protocol that works for any generative model and is more efficient than the Langevin MCMC approach of Ref. [72]. We begin by pulling back the weights from our post-hoc classifier to the latent space, either directly or via the OMNIFOLD method [74] when the generator is not surjective. We then propose to learn a second generative model $\Phi(y)$ that maps an auxiliary latent space onto the refined latent space. Generating from the refined model amounts to sampling from the auxiliary latent space and applying $g(\Phi(y))$. We focus on the case where g is a normalizing flow and Φ is a GAN because GANs need not to be invertible ; however, the method can be applied to any pair of generative models. Furthermore, the procedure can be iterated for further refinement. Learning a post-hoc generative model in a latent space was also studied by the authors of Refs. [75–77], where a standard autoencoder becomes probabilistic via a second model such as a NF trained on the bottleneck layer. Similarly, the authors of Ref. [78] proposed an iterative VAE setup to bring the latent space closer to a target multidimensional Gaussian random variable.

This paper is organized as follows. We review related work and describe the statistical properties and challenges associated with existing generative models in Section 2. We introduce various ways of implementing the LASER protocol in Sec. 3. Illustrative numerical examples are provided in Sec. 4 and the paper ends with conclusions and outlook in Sec. 5.

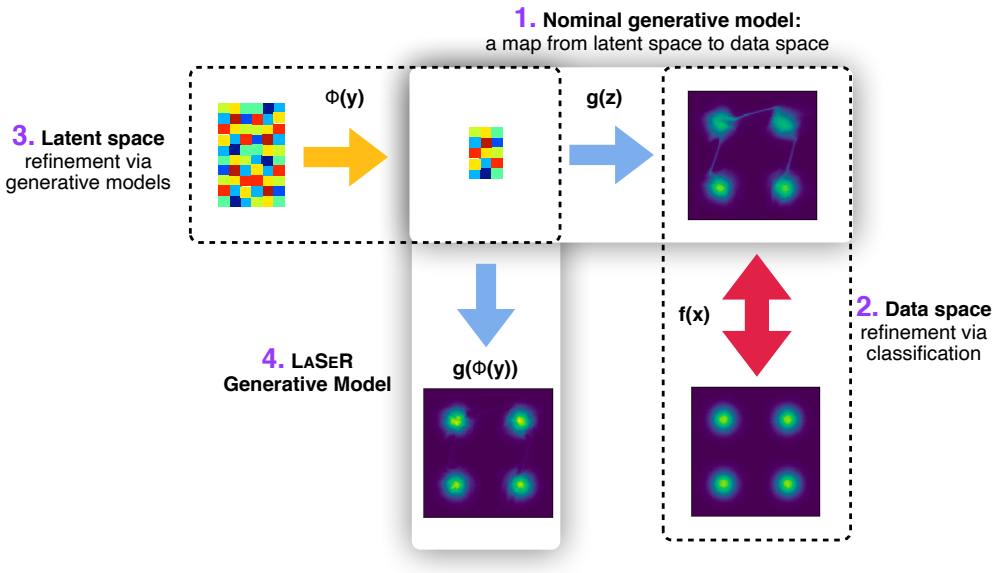

Figure 1: A schematic diagram illustrating the LASER protocol. **1**. The upper right part of the diagram represents a given generative model $g : \mathbb{R}^N \to \mathbb{R}^M$ that maps features from a latent space to a data space. **2**. A classifier network $f : \mathbb{R}^M \to \mathbb{R}$ is trained to distinguish between the generated samples and real samples from data. The output of this classifier is interpreted as a likelihood ratio between the generated samples and the real samples and is pulled back to the latent space. If f is trained with binary cross entropy, the weights are approximated as $w(x) = f(x)/(1 - f(x))$ and then the weight of a latent space point z is $w(g(z))$. **3**. Next, a second generative model $\Phi : \mathbb{R}^L \to \mathbb{R}^N$ is trained with these weights to transform the original latent space into a new latent space. **4**. The LASER model is then given by $g(\Phi(y))$.

2 Background

A generator is a function g that maps a latent space $\mathcal{Z} \subseteq \mathbb{R}^N$ onto a target or feature space $\mathcal{X} \subseteq \mathbb{R}^M$, with underlying probability densities $p_\mathcal{Z}$ and $p_\mathcal{X}$, respectively. Typically, $p_\mathcal{Z}$ is chosen to be simple (e.g. normal or uniform) so that it is efficient to generate data $Z \sim p_\mathcal{Z}$. In some cases, the latent space

$\mathcal{Z}$ is broken into two components: one component that specifies the input features of interest and one component that specifies auxiliary latent features that are marginalized over. When this happens, the map g is viewed as a stochastic function of the first latent space component. While our examples exclusively cover the case of deterministic maps from the full latent space to the target space, our approach can accommodate both settings as we explain in more detail below.

The function g can be constructed from first-principles insights about the dynamics of a system or it can be learned from data. For example, commonly-used deep generative models include

- Generative adversarial networks (GANs) [1, 2]
- Variational autoencoders (VAEs) [3, 4]
- Normalizing flows (NFs) [5, 6]

Apart from the specific learning objective, all generative models have their intrinsic advantages and disadvantages. The relevant features to understand the LaSeR protocol are summarized in Table 1 and are further discussed in the following.

Table 1: A comparison of commonly used deep generative models.

Method	Train on data	Exact log-likelihood	Non-topology preserving
Variational Autoencoders	✓	✗	✓
Generative Adversarial Networks	✓	✗	✓
Normalizing Flows	✓	✓	✗

2.1 Generative models and coordinate transformations

While the three generative models introduced in the previous section can all be trained directly on unlabeled data, they have different strategies for estimating $p_{\mathcal{X}}$. GANs and VAEs learn this density implicitly by introducing an auxiliary task. In the case of GANs, the auxiliary task is performed by a discriminator network that tries to distinguish samples drawn from $p_{\mathcal{Z}}$ passed through g and those drawn from $p_{\mathcal{X}}$ directly. For VAEs, the generator is called the decoder and the auxiliary task requires an encoder network h to satisfy $g(h(x)) \approx x$, while regularizing the latent space probability density. Due to the structure of these networks, N need not be the same size as M.

In contrast to GANs and VAEs, NFs explicitly encode an estimate for the probability density $p_{\mathcal{X}}$. These networks rely on a coordinate transformation which maps the prior distribution $p_{\mathcal{Z}}$ into a target distribution p_g with g now being invertible. This requires $M = N$ but allows for an analytic expression for the probability density induced by g:

$$p_g(x) \equiv p_g(g(z)) = \left| \frac{\partial g(z)}{\partial z} \right|^{-1} p_{\mathcal{Z}}(z). \tag{1}$$

In order to match p_g and the data probability density $p_{\mathcal{X}}$, one can directly maximize the log-likelihood of the data without resorting to an auxiliary task:

$$\log p_g(x) = \log p_{\mathcal{Z}}(g^{-1}(x)) + \log \left| \frac{\partial g^{-1}(x)}{\partial x} \right|. \tag{2}$$

2.2 Topological obstructions

While the bijective nature of NFs allows for an explicit representation of the target probability density estimate, they inevitably suffer from a significant drawback. In order to find a mapping g which satisfies Eq. (1) and matches the data probability density, the manifolds defined by the prior distribution $p_{\mathcal{Z}}$ and the target distribution $p_{\mathcal{X}}$ need to be topologically equivalent. A common way to describe the topological properties of manifolds are the Betti numbers.

Definition 1. *The nth Betti number b_n of a topological space X is defined as the rank of the nth homology group H_n of X [79, 80].*

Informally, the nth Betti number denotes the number of n-dimensional holes of a topological space. In fact, the first three Betti numbers do have simple geometrical interpretations:

- b_0 is the number of connected components.
- b_1 is the number of one-dimensional or circular holes.
- b_2 is the number of two-dimensional voids.

For instance, in Fig. 2 we illustrate a torus which has one connected component $b_0 = 1$, two one-dimensional holes $b_1 = 2$, and a single void enclosed within the surface $b_2 = 1$.

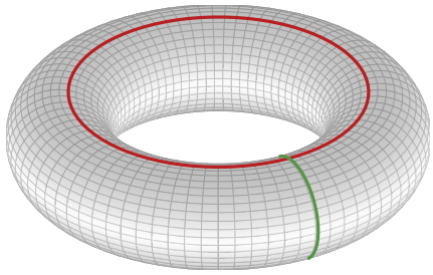

Figure 2: For a torus, the first Betti number is $b_1 = 2$, which can be intuitively understood as the number of circular holes.

It has been proven by Poincaré that these numbers are invariant under homeomorphisms. This implies the following proposition:

Proposition 1. *Any bijective mapping $g(z)$ is a homeomorphism and preserves the topological structure of the input space.*

A proof can be found in Refs. [81, 82]. This mathematical statement has a major impact on any generative model relying on bijective mappings, such as NFs. If the target space has a different topological structure than the latent space, these models cannot properly describe the target space topology as they inherently preserve the latent space topology. It is indeed possible that these inconsistencies are hidden in some highly-correlated observable which might not be investigated during or after optimization. Moreover, it has been shown in Ref. [83] that you can hide and diminish the topological issues if you increase the network complexity. However, you can achieve better and computationally cheaper results by relaxing the bijetivity constraint itself [83] or by augmenting additional dimensions [82, 84]. Further, the issues arising by non-trivial topologies have also been investigated in the context of autoencoders in Ref. [85].

Consequently, if we want to correctly reproduce the target space with a non-trivial topology, we either need to modify the latent space to match the topological structure, or the bijective model g has to be replaced by a non-topology preserving model such as GANs or VAEs. While these models do have more flexibility and do not suffer from the same topological obstructions, this freedom usually goes hand in hand with training instabilities and more time-consuming hyperparameter searches. In order to benefit from both worlds and to fix the topology of the latent space, we propose to use the LASER protocol which is described in the following.

3 Proposed methods

The LASER protocol is illustrated in Fig. 1. The input is a generator $g(z)$ that could either be a neural network or a black-box (BB) simulator (e.g. a physics-based Monte Carlo).

As a first step, a classifier is trained to distinguish samples drawn from g and samples drawn from the target data probability density. It is well-known [86, 87] (and forms the basis of the DCTR protocol [88]) that an ideally trained classifier f using the binary cross entropy (BCE) loss function will have the property $f/(1-f) \propto p(x|\text{target})/p(x|\text{from } g)$. The proportionality constant is $p(\text{target})/p(\text{from } g)$ and is unity when the two datasets have the same number of samples. Other loss functions such as the mean squared error have the same property. It is also possible to engineer the loss function to directly learn the likelihood ratio [89, 90]. Throughout this paper, we use BCE.

Each generated sample is assigned a weight $w(x_i) = f(x_i)/(1 - f(x_i))$. These weights are then assigned to the corresponding latent space point $g(z_i) = x_i$. When g is surjective, w is a proper function of z. However, there are many potential applications where the generator has many internal latent degrees of freedom that are marginalized over and are not part of $\mathcal{Z}$. In this case, one can use the OMNIFOLD algorithm [74] which is an iterative expectation maximization-style approach that returns weights for the z_i that maximizes the likelihood in the data space. The weights extracted with OMNIFOLD are a proper function of the latent space. This is not necessarily the case for weights directly inferred from the classifier, since the mapping g can send the same latent space point to different data space points. In other words, the marginalized latent dimensions make g a stochastic function.

The final step of LASER requires to sample from weighted latent space $(z, w(z))$. This can be for instance done directly using the following methods:

- **Rejection sampling:** We can use the weights $w(z)$ to perform rejection sampling on the latent space. This is simple, but ineffective if the weights are broadly distributed. Thus, we do not employ this method in our experiments.
- **Markov chain Monte Carlo (MCMC):** The weights $w(z)$ induce a new probability distribution
$$q_{\mathcal{Z}}(z) = p_{\mathcal{Z}}(z)\, w(z)/Z_0, \tag{3}$$
with some normalisation factor Z_0. In general, this probability distribution is non-trivial and does not allow to determine its quantile function analytically. However, if the probability distribution $q_{\mathcal{Z}}$ and its derivative $\partial q_{\mathcal{Z}}/\partial z$ are tractable a MCMC algorithm can be employed. While Ref. [72] uses a simple Langevin dynamics algorithm, we suggest to use the more advanced Hamiltonian Monte Carlo (HMC) [91, 92] algorithm as it generates significantly better samples.

However, instead of using the above approaches, we suggest for the LASER protocol to convert the weighted latent space $(z, w(z))$ into a new unweighted latent space. For this, we train a second generative model Φ which we call the refiner. The refiner maps an auxiliary latent space $\mathcal{Y} \subseteq \mathbb{R}^L$ onto the refined latent space $\mathcal{Z}' \subseteq \mathbb{R}^N$, with underlying probability densities $p_{\mathcal{Y}}$ and $p_{\mathcal{Z}'} = q_{\mathcal{Z}}$, respectively. The refiner Φ can be either a GAN or a NF, where the latter requires $L = N$. In order to train the model on the weighted latent space $(z, w(z))$ we accommodate the weights into the loss functions as it was proposed in Refs. [93, 94]. In contrast to the MCMC algorithm, this method does not require to calculate the derivative of the weights $\partial w/\partial z$.

At the end, we generate new samples from the refined generator by first sampling from the auxiliary latent space, mapping that to the refined latent space, and then passing that through g. In some cases, the entire procedure can be iterated, i.e. if g is a neural network, its weights could be updated using the refined latent space.

Table 2 illustrates some of the features of various combinations of generators g and latent space refiner networks Φ.

Table 2: Comparison of various combinations of generator g and refiner network Φ.

Generator g	Refiner Φ	Note
GAN	GAN	Possible to iterate
GAN	NF	Possible to iterate
NF	GAN	Cannot iterate (density no longer explicit)
NF	NF	Suffers topological obstructions

4 Numerical examples

We consider three different 2-dimensional examples which allows to visualize both the full output and the latent space of the primary generator. All three sets of training data consist of 480k data points. In all models and experimentes we used a batch size of 2000 points and an epoch is defined as iterating through all points in the training set, giving 240 model updates per epoch. We run all our

experiments on our internal GPU cluster, which consists of multiple Nvidia GTX 1080 TI and Nvidia RTX 2080 Ti GPUs. Depending on the model the computation times range between 30 mins and 5 hours.

4.1 Implementation details

All models and the training routines are implemented in PyTorch 1.8 [95]. In the following we specify the implementation details for the various networks and the chosen hyperparameters.

Baseline model. In all examples we use the Real-NVP [96] implementation of a NF as the baseline model. Our implementation relies partially on the FrEIA library.[2] In the first two examples, the NF consists of 12 coupling blocks, where each block consists of a fully connected multi-layer perceptron (MLP) with 3 hidden layers, 48 units per layer and leaky ReLU activation functions. In the third more complex example, we increase the number of coupling blocks to 20 and the number of units to 60. In all cases the model is trained for 100 epochs by minimizing the negative log-likelihood with an additional weight decay of 10^{-5}. The optimization is performed using Adam [97] with default β parameters and $\alpha_0 = 10^{-3}$. We further employ a exponential learning rate schedule, i.e. $\alpha_n = \alpha_0 \, \gamma^n$, with $\gamma = 0.999$ which decays the learning rate after each epoch.

Post-hoc classifier. As a post-hoc classifier we employ a MLP with 8 hidden layers, 96 units per layer and leaky ReLU activation functions. We train it for 50 epochs by minimizing the BCE loss between true data samples and samples from the generator g. Optimization is again performed using Adam with default β parameters and $\alpha_0 = 10^{-3}$ and an exponential learning rate schedule with $\gamma = 0.999$.

Refiner network. Finally, the refiner network is a GAN consisting of a discriminator and a generator, both MLPs with 7 hidden layers, 100 units per layer and Leaky ReLU activation functions. As a GAN does not require to have the same dimensions in the latent space as in the feature space we choose 4-dimensional auxiliary latent space to simplify the task of the generator. The standard GAN objective is replaced by the weighted BCE loss introduced in Ref. [93] to accommodate the weights of the weighted training data. The GAN is trained for 200 epochs using the Adam optimizer with $(\alpha, \beta_1, \beta_2) = (10^{-4}, 0.5, 0.9)$ and also using an exponential learning rate schedule with $\gamma = 0.999$. To compensate an imbalance in the training we update the discriminator four times for each generator update.

Hamiltonian Monte Carlo (HMC). In order to have full control of all parameters and inputs we implemented our own version of Hamiltonian Monte Carlo in PyTorch, using its automatic differentiation module to compute the necessary gradients to run the algorithm. For all examples, we initialize a total of 100 Markov Chains running in parallel to reduce computational overhead and solve the equation of motions numerically using the leapfrog algorithm [92] with a step size of $\varepsilon = 0.004$ and 50 leapfrog steps. To avoid artifacts originating from the random initialization of the chains, we start with a burn-in phase and discard the first 3000 points from each chain.

4.2 Probability distance measures

In order to quantify the performance of the base model and its refined counterparts, we implemented two measures of distance:

Jensen–Shannon divergence (JSD). A simple but commonly used distance measure between two probability distributions $p(x)$ and $q(x)$ is the Jensen–Shannon divergence

$$\mathrm{JSD}(p,q) = \frac{1}{2} \int dx \, p(x) \log \left(\frac{2p(x)}{p(x)+q(x)} \right) + q(x) \log \left(\frac{2p(x)}{p(x)+q(x)} \right) . \tag{4}$$

In contrast to the Kullback–Leibler (KL) divergence it is symmetric and always returns a finite value.

[2] Framework for Easily Invertible Architectures: `https://github.com/VLL-HD/FrEIA`

Earth mover distance (EMD). Another common distance measure between two probability distributions $p(x)$ and $q(x)$ is the earth mover's distance (EMD). The EMD is usually formulated as an optimal transport problem. Lets assume the two distributions are represented as clusters $P = \{(x_i, p(x_i)\}_{i=1}^{M}$ and $Q = \{(y_j, q(y_j)\}_{j=1}^{M'}$. Then, the EMD between two probabilities is the minimum cost required to rearrange one probability distribution p into the other q by discrete probability flows f_{ij} from one cluster point x_i to the other y_j

$$\begin{gathered}\mathrm{EMD}(p,q) = \min_{\{f_{ij}>0\}} \sum_{i=1}^{M}\sum_{i=1}^{M'} f_{ij}||x_i - y_j||_2, \\ \sum_{j=1}^{M'} f_{ij} \leq p(x_i), \quad \sum_{i=1}^{M} f_{ij} \leq q(y_j), \quad \sum_{i=1}^{M}\sum_{i=1}^{M'} f_{ij} = \min\left(\sum_{i=1}^{M} p(x_i), \sum_{j=1}^{M'} q(y_j)\right),\end{gathered} \tag{5}$$

where the optimal flow f_{ij} is found by solving this optimization problem. For an efficient implementation we used the POT library [98].

Numerical approximation. In order to calculate these quantities numerically we generate 2M data points for all models and examples and approximate the probability densities from a 2-dimensional histogram. As ground distance measure between clusters P and Q for the calculation of the EMD, we consider the Euclidean distance between the center points of each bin. In order to estimate the error by approximating the probabilities, we calculated the JSD and EMD score on equally large but independent test samples drawn from the truth distribution. The extracted error estimations are indicated in parenthesis in Table 3.

4.3 Experimental results

In the following we will simply refer to the LASER protocol using the Hamiltonian Monte Carlo as HMC method, and we will refer to the LASER protocol using a refiner network as LASER method.

In the first example, illustrated in Fig. 3, we considered a multi-modal distribution which can be topologically described by its Betti numbers $b_0 = 4$ and $b_{n\geq 1} = 0$. In contrast, our non-refined latent space is described by a 2-dimensional normal distribution with $b_0 = 1$ and $b_{n>0} = 0$. As both spaces are topologically different our baseline model fails to correctly reproduce the truth distribution. It is obvious that the baseline model cannot properly separate the individual modes and always leaves a connection between them. This is in agreement with Proposition 1 as our baseline model is a NF and thus inevitably topology preserving. After training a post-hoc classifier the reweighted feature space of the DCTR method clearly resolved the topological obstructions. Pulling back these weights onto the orginal latent space induces a weighted latent space which is now also clearly separated into four pieces and thus shows the same topological structure as the data distribution. This weighted latent space is correctly reproduced in the refined latent spaces of the HMC and LASER method. Acting with our generator g on these refined latent spaces shows a highly improved feature space distribution.

In Fig. 4, we considered a double donut distribution with Betti numbers $b_0 = 1$, $b_1 = 2$ and $b_{n\geq 2} = 0$ as a second example. This example does not have disconnected parts but contains two circular holes. Again the baseline model has troubles to properly close the loops and reproduce the holes as these topological features are not present in the non-refined latent space. After reweighting the latent space with the DCTR weights, these topological features emerge as holes in the latent space. Both refined latent spaces show the same topological structure as the reweighted latent space, resulting in an improved feature space distribution for both the HMC and the LASER method.

Finally, in the last example we combine the topological features from the previous examples to make our baseline model fail even more. In detail, we considered a set of 3 displaced rings which can be topologically described by the Betti numbers $b_0 = 3$, $b_1 = 3$ and $b_{n\geq 1} = 0$, as illustrated in Fig. 5. As with previous examples, the baseline model utterly fails to reproduce the true data distribution. As the topological structure of the data distribution is more complex the reweighted latent space in the DCTR method is considerably more involved then the weighted latent spaces of the previous examples. Owing to this more complex structure of the weighted latent space our refiner model Φ has a much harder job to properly reproduce these topological features in the refined latent space of the LASER method. While being far from optimal the feature space distribution of the LASER refined

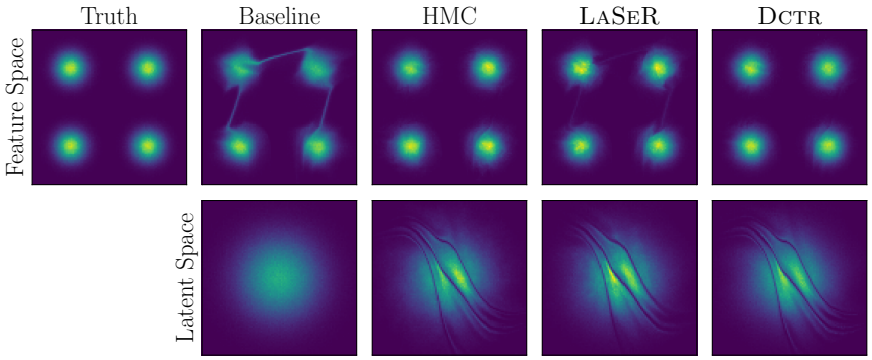

Figure 3: Comparison of the baseline model and the refined outputs by either using the HMC, LaSeR or DCTR method on a 2-dimensional multi-modal gaussian distribution with Betti numbers $b_0 = 4$, $b_1 = 0$.

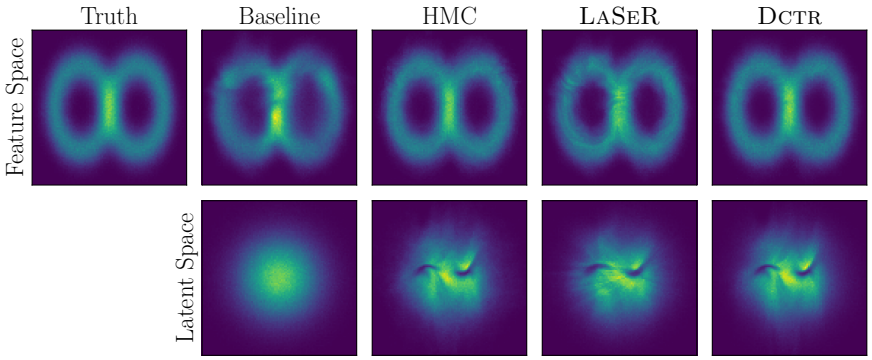

Figure 4: Comparison of the baseline generator and the refined outputs by either using the HMC, LaSeR or DCTR method on a 2-dimensional double donut distribution with Betti numbers $b_0 = 1$, $b_1 = 2$.

generator is notably improved in comparison to the baseline model. In this particular complicated example the output of the HMC method seems to show slightly better agreement with the truth distribution than the LaSeR method. However, these results can be improved by spending more time on a detailed network hyperparameter optimization.

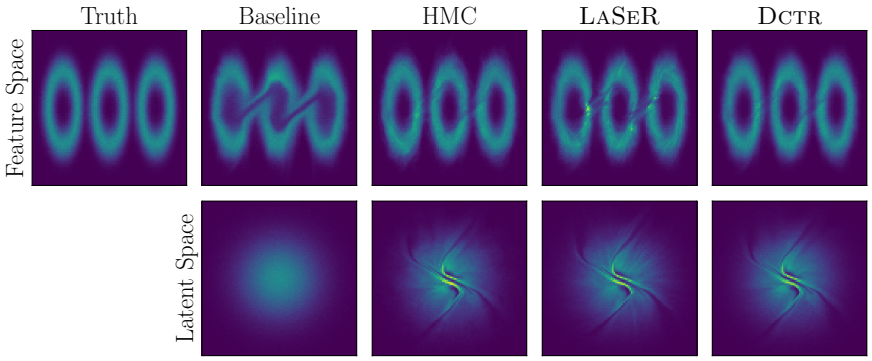

Figure 5: Comparison of the baseline generator and the refined outputs by either using the HMC, LaSeR or DCTR method on a 2-dimensional triple ring distribution with Betti numbers $b_0 = 3$, $b_1 = 3$.

Table 3: Earth mover distance (EMD) and Jensen–Shannon divergence (JSD) on test data for the baseline model and the proposed refinement methods for various two-dimensional examples. The best results are written in bold face. The errors on the approximate EMD and JSD are indicated in parenthesis.

Method	Gaussians		Double donut		Rings	
	EMD	JSD	EMD	JSD	EMD	JSD
Baseline	0.28(2)	0.66(4)	0.23(2)	0.27(4)	0.067(6)	0.21(2)
HMC	0.22(2)	**0.12(4)**	**0.06(2)**	**0.07(4)**	0.137(6)	**0.07(2)**
LaSeR	**0.21(2)**	0.24(4)	0.09(2)	0.09(4)	**0.047(6)**	0.08(2)
DCTR	0.09(2)	0.09(4)	0.11(2)	0.05(4)	0.037(6)	0.04(2)

In order to provide a quantitative comparison of performances, we calculated the EMD and JSC scores of our baseline model and its refined counterparts for all examples. These scores are summarized in Table 3 and show that the HMC method yields the best performance in most scenarios for the unweighted output. However, the LaSeR method is equally good or better for the the most complex example, showing that the HMC method has troubles to sample from the highly tangled latent space. Moreover, the HMC method cannot be used at all if the derivatives of the weights are not tractable, in which case the LaSeR method is still applicaple. The overall best performance is achieved by the DCTR method which however corresponds to a weighted feature space. As these weights are also used to obtain the refined latent space for the HMC and LaSeR method the score of the DCTR yield as a benchmark and provides lower limit. Indeed, we can ask ourselves why the EMD of both unweighted methods are smaller than the EMD of the DCTR method for the double donut example. However, owing to the the uncertainty on the numerical estimation of the scores, which are indicated in parenthesis, all scores a statistically consistent.

5 Conclusions and outlook

We have presented the LaSeR protocol, which provides a post-hoc method to improve the performance of any generative model. In particular, it has shown remarkable results on several numerical examples and improved the EMD (JSD) score of the baseline model by 25-61% (62-67%). It further managed to fix the topological obstructions encountered by the base model. While we have only explicitly demonstrated LaSeR for refining normalizing flows as the primary generator, the protocol is also valid for other deep generative models as well as other generative models that need not be differentiable or surjective.

While LaSeR is a promising approach for generative model refinement, it also has some limitations. First of all, there is some dependence on the efficacy of the primary generator. For example, if the primary generator is a constant, then no amount of refining can recover the data probability density. Furthermore, the refinement comes at a computational cost with an additional step on top what is required to construct the original generative model. This can be particularly important when the new latent space is sourced from a much bigger space than the original one. Finally, even though LaSeR produces unweighted examples, the statistical precision of the resulting dataset may be limited by the size of the training dataset [42]. We leave further investigations of the statistical power of refined samples and the interplay with hyperparameter choices for future research.

References

[1] I. J. Goodfellow et al., *Generative Adversarial Nets*, *Conference on Neural Information Processing Systems* **2** (2014) 2672.

[2] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta and A. A. Bharath, *Generative adversarial networks: An overview*, *IEEE Signal Processing Magazine* **35** (Jan, 2018) 53.

[3] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, `1312.6114`.

[4] D. P. Kingma and M. Welling, *An Introduction to Variational Autoencoders*, *Foundations and Trends in Machine Learning* **12** (2019) 307.

[5] D. J. Rezende and S. Mohamed, *Variational inference with normalizing flows*, *International Conference on Machine Learning* **37** (2015) 1530.

[6] I. Kobyzev, S. Prince and M. Brubaker, *Normalizing Flows: An Introduction and Review of Current Methods*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020) 1.

[7] L. de Oliveira, M. Paganini and B. Nachman, *Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis*, 1701.05927.

[8] M. Paganini, L. de Oliveira and B. Nachman, *Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters*, *Phys. Rev. Lett.* **120** (2018) 042003, [1705.02355].

[9] M. Paganini, L. de Oliveira and B. Nachman, *CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks*, *Phys. Rev.* **D97** (2018) 014021, [1712.10321].

[10] S. Alonso-Monsalve and L. H. Whitehead, *Image-based model parameter optimization using Model-Assisted Generative Adversarial Networks*, 1812.00879.

[11] A. Butter, T. Plehn and R. Winterhalder, *How to GAN Event Subtraction*, 1912.08824.

[12] J. Arjona Martinez, T. Q. Nguyen, M. Pierini, M. Spiropulu and J.-R. Vlimant, *Particle Generative Adversarial Networks for full-event simulation at the LHC and their application to pileup description*, *ACAT 2019* (2019) , [1912.02748].

[13] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn and R. Winterhalder, *How to GAN away Detector Effects*, 1912.00477.

[14] S. Vallecorsa, F. Carminati and G. Khattak, *3D convolutional GAN for fast simulation*, *Proceedings, 23rd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2018): Sofia, Bulgaria, July 9-13, 2018* **214** (2019) 02010.

[15] SHiP collaboration, C. Ahdida et al., *Fast simulation of muons produced at the SHiP experiment using Generative Adversarial Networks*, 1909.04451.

[16] S. Carrazza and F. A. Dreyer, *Lund jet images from generative and cycle-consistent adversarial networks*, *Eur. Phys. J.* **C79** (2019) 979, [1909.01359].

[17] A. Butter, T. Plehn and R. Winterhalder, *How to GAN LHC Events*, *SciPost Phys.* **7** (2019) 075, [1907.03764].

[18] J. Lin, W. Bhimji and B. Nachman, *Machine Learning Templates for QCD Factorization in the Search for Physics Beyond the Standard Model*, *JHEP* **05** (2019) 181, [1903.02556].

[19] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat and S. Palazzo, *DijetGAN: A Generative-Adversarial Network Approach for the Simulation of QCD Dijet Events at the LHC*, 1903.02433.

[20] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, *LHC analysis-specific datasets with Generative Adversarial Networks*, 1901.05282.

[21] V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin and E. Zakharov, *Generative Models for Fast Calorimeter Simulation.LHCb case*, *CHEP 2018* (2018) , [1812.01319].

[22] ATLAS collaboration, *Deep generative models for fast shower simulation in ATLAS*, *ATL-SOFT-PUB-2018-001* (Jul, 2018) .

[23] K. Zhou, G. Endrodi, L.-G. Pang and H. Stocker, *Regressive and generative neural networks for scalar field theory*, *Phys. Rev.* **D100** (2019) 011501, [1810.12879].

[24] F. Carminati, A. Gheata, G. Khattak, P. Mendez Lorenzo, S. Sharan and S. Vallecorsa, *Three dimensional Generative Adversarial Networks for fast simulation*, *Proceedings, 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017): Seattle, WA, USA, August 21-25, 2017* **1085** (2018) 032016.

[25] S. Vallecorsa, *Generative models for fast simulation*, *Proceedings, 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017): Seattle, WA, USA, August 21-25, 2017* **1085** (2018) 022005.

[26] K. Datta, D. Kar and D. Roy, *Unfolding with Generative Adversarial Networks*, 1806.00433.

[27] P. Musella and F. Pandolfi, *Fast and Accurate Simulation of Particle Detectors Using Generative Adversarial Networks*, *Comput. Softw. Big Sci.* **2** (2018) 8, [1805.00850].

[28] M. Erdmann, L. Geiger, J. Glombitza and D. Schmidt, *Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks*, *Comput. Softw. Big Sci.* **2** (2018) 4, [1802.03325].

[29] K. Deja, T. Trzcinski and u. Graczykowski, *Generative models for fast cluster simulations in the TPC for the ALICE experiment*, *Proceedings, 23rd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2018): Sofia, Bulgaria, July 9-13, 2018* **214** (2019) 06003.

[30] D. Derkach, N. Kazeev, F. Ratnikov, A. Ustyuzhanin and A. Volokhova, *RICH 2018*, 1903.11788.

[31] H. Erbin and S. Krippendorf, *GANs for generating EFT models*, 1809.02612.

[32] M. Erdmann, J. Glombitza and T. Quast, *Precise simulation of electromagnetic calorimeter showers using a Wasserstein Generative Adversarial Network*, *Comput. Softw. Big Sci.* **3** (2019) 4, [1807.01954].

[33] J. M. Urban and J. M. Pawlowski, *Reducing Autocorrelation Times in Lattice Simulations with Generative Adversarial Networks*, 1811.03533.

[34] L. de Oliveira, M. Paganini and B. Nachman, *Tips and Tricks for Training GANs with Physics Constraints*, 2017.

[35] L. de Oliveira, M. Paganini and B. Nachman, *Controlling Physical Attributes in GAN-Accelerated Simulation of Electromagnetic Calorimeters*, *J. Phys. Conf. Ser.* **1085** (2018) 042017, [1711.08813].

[36] S. Farrell, W. Bhimji, T. Kurth, M. Mustafa, D. Bard, Z. Lukic et al., *Next Generation Generative Neural Networks for HEP*, *EPJ Web Conf.* **214** (2019) 09005.

[37] B. Hooberman, A. Farbin, G. Khattak, V. Pacela, M. Pierini, J.-R. Vlimant et al., *Calorimetry with Deep Learning: Particle Classification, Energy Regression, and Simulation for High-Energy Physics*, 2017.

[38] D. Belayneh et al., *Calorimetry with Deep Learning: Particle Simulation and Reconstruction for Collider Physics*, 1912.06794.

[39] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol et al., *Getting High: High Fidelity Simulation of High Granularity Calorimeters with High Speed*, 2005.05334.

[40] Y. Alanazi et al., *AI-based Monte Carlo event generator for electron-proton scattering*, 2008.03151.

[41] S. Diefenbacher, E. Eren, G. Kasieczka, A. Korol, B. Nachman and D. Shih, *DCTRGAN: Improving the Precision of Generative Models with Reweighting*, *Journal of Instrumentation* **15** (2020) P11004, [2009.03796].

[42] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman and T. Plehn, *GANplifying Event Samples*, 2008.06545.

[43] R. Kansal, J. Duarte, B. Orzari, T. Tomei, M. Pierini, M. Touranakou et al., *Graph Generative Adversarial Networks for Sparse Data Generation in High Energy Physics*, *34th Conference on Neural Information Processing Systems* (11, 2020) , [2012.00173].

[44] A. Maevskiy, F. Ratnikov, A. Zinchenko and V. Riabov, *Simulating the Time Projection Chamber responses at the MPD detector using Generative Adversarial Networks*, 2012.04595.

[45] Y. S. Lai, D. Neill, M. Płoskoń and F. Ringer, *Explainable machine learning of the underlying physics of high-energy particle collisions*, 2012.06582.

[46] S. Choi and J. H. Lim, *A Data-driven Event Generator for Hadron Colliders using Wasserstein Generative Adversarial Network*, 2102.11524.

[47] F. Rehm, S. Vallecorsa, V. Saletore, H. Pabst, A. Chaibi, V. Codreanu et al., *Reduced Precision Strategies for Deep Learning: A High Energy Physics Generative Adversarial Network Use Case*, 2103.10142.

[48] F. Rehm, S. Vallecorsa, K. Borras and D. Krücker, *Validation of Deep Convolutional Generative Adversarial Networks for High Energy Physics Calorimeter Simulations*, 3, 2021, 2103.13698.

[49] S. Carrazza, J. Cruz-Martinez and T. R. Rabemananjara, *Compressing PDF sets using generative adversarial networks*, 2104.04535.

[50] M. S. Albergo, G. Kanwar and P. E. Shanahan, *Flow-based generative models for Markov chain Monte Carlo in lattice field theory*, *Phys. Rev.* **D100** (2019) 034515, [1904.12072].

[51] G. Kanwar, M. S. Albergo, D. Boyda, K. Cranmer, D. C. Hackett, S. Racanière et al., *Equivariant flow-based sampling for lattice gauge theory*, 2003.06413.

[52] J. Brehmer and K. Cranmer, *Flows for simultaneous manifold learning and density estimation*, 2003.13913.

[53] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale and S. Schumann, *Exploring phase space with Neural Importance Sampling*, 2001.05478.

[54] C. Gao, S. Höche, J. Isaacson, C. Krause and H. Schulz, *Event Generation with Normalizing Flows*, *Phys. Rev. D* **101** (2020) 076002, [2001.10028].

[55] C. Gao, J. Isaacson and C. Krause, *i-flow: High-Dimensional Integration and Sampling with Normalizing Flows*, 2001.05486.

[56] B. Nachman and D. Shih, *Anomaly Detection with Density Estimation*, *Phys. Rev. D* **101** (2020) 075042, [2001.04990].

[57] S. Choi, J. Lim and H. Oh, *Data-driven Estimation of Background Distribution through Neural Autoregressive Flows*, 2008.03636.

[58] Y. Lu, J. Collado, D. Whiteson and P. Baldi, *SARM: Sparse Autoregressive Model for Scalable Generation of Sparse Images in Particle Physics*, 2009.14017.

[59] S. Bieringer, A. Butter, T. Heimel, S. Höche, U. Köthe, T. Plehn et al., *Measuring QCD Splittings with Invertible Networks*, 2012.09873.

[60] J. Hollingsworth, M. Ratz, P. Tanedo and D. Whiteson, *Efficient sampling of constrained high-dimensional theoretical spaces with machine learning*, 2103.06957.

[61] J. W. Monk, *Deep Learning as a Parton Shower*, 1807.03685.

[62] T. Cheng, J.-F. Arguin, J. Leissner-Martin, J. Pilette and T. Golling, *Variational Autoencoders for Anomalous Jet Tagging*, 2007.01850.

[63] K. Dohi, *Variational Autoencoders for Jet Simulation*, 2009.04842.

[64] J. N. Howard, S. Mandt, D. Whiteson and Y. Yang, *Foundations of a Fast, Data-Driven, Machine-Learned Simulator*, 2101.08944.

[65] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol et al., *Decoding Photons: Physics in the Latent Space of a BIB-AE Generative Network*, 2102.12491.

[66] B. Bortolato, B. M. Dillon, J. F. Kamenik and A. Smolkovič, *Bump Hunting in Latent Space*, 2103.06595.

[67] K. Deja, J. Dubiński, P. Nowak, S. Wenzel and T. Trzciński, *End-to-end sinkhorn autoencoder with noise generator*, 2020.

[68] A. Hariri, D. Dyachkova and S. Gleyzer, *Graph Generative Models for Fast Detector Simulations in High Energy Physics*, 2104.01725.

[69] C. Fanelli and J. Pomponi, *DeepRICH: Learning Deeply Cherenkov Detectors*, *Mach. Learn. Sci. Tech.* **1** (11, 2019) 015010, [1911.11717].

[70] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot and R. Winterhalder, *Invertible Networks or Partons to Detector and Back Again*, 2006.06685.

[71] M. Bellagente, M. Haußmann, M. Luchmann and T. Plehn, *Understanding Event-Generation Networks via Uncertainties*, 2104.04543.

[72] T. Che, R. Zhang, J. Sohl-Dickstein, H. Larochelle, L. Paull, Y. Cao et al., *Your gan is secretly an energy-based model and you should use discriminator driven latent sampling*, 2003.06060.

[73] A. Andreassen and B. Nachman, *Neural Networks for Full Phase-space Reweighting and Parameter Tuning*, *Phys. Rev. D* **101** (2020) 091901, [1907.08209].

[74] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman and J. Thaler, *OmniFold: A Method to Simultaneously Unfold All Observables*, *Phys. Rev. Lett.* **124** (2020) 182001, [1911.09107].

[75] V. Böhm and U. Seljak, *Probabilistic auto-encoder*, 2020.

[76] Y. Li, K. Swersky and R. Zemel, *Generative moment matching networks*, in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1718–1727, PMLR, 07–09 Jul, 2015, http://proceedings.mlr.press/v37/li15.html.

[77] Z. Xiao, Q. Yan and Y. Amit, *Generative latent flow*, 2019.

[78] B. Dai and D. Wipf, *Diagnosing and enhancing VAE models*, in *International Conference on Learning Representations*, 2019, https://openreview.net/forum?id=B1e0X3C9tQ.

[79] O. Bobrowski and S. Mukherjee, *The topology of probability distributions on manifolds*, *Probability Theory and Related Fields* **161** (07, 2013) .

[80] A. Hatcher, *Algebraic Topology*. Cambridge University Press, 2002.

[81] L. Younes, *Shapes and Diffeomorphisms*. Springer Berlin Heidelberg, 2010.

[82] E. Dupont, A. Doucet and Y. W. Teh, *Augmented neural odes*, 1904.01681.

[83] R. Cornish, A. L. Caterini, G. Deligiannidis and A. Doucet, *Relaxing bijectivity constraints with continuously indexed normalising flows*, 2021.

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

[85] J. Batson, C. G. Haaf, Y. Kahn and D. A. Roberts, *Topological Obstructions to Autoencoding*, 2102.08380.

[86] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[87] M. Sugiyama, T. Suzuki and T. Kanamori, *Density Ratio Estimation in Machine Learning*. Cambridge University Press, 2012, 10.1017/CBO9781139035613.

[88] A. Andreassen and B. Nachman, *Neural Networks for Full Phase-space Reweighting and Parameter Tuning*, *Phys. Rev. D* **101** (2020) 091901(R), [1907.08209].

[89] R. T. D'Agnolo and A. Wulzer, *Learning New Physics from a Machine*, *Phys. Rev. D* **99** (2019) 015014, [1806.02350].

[90] B. Nachman and J. Thaler, *E Pluribus Unum Ex Machina: Learning from Many Collider Events at Once*, 2101.07263.

[91] S. Duane, A. Kennedy, B. J. Pendleton and D. Roweth, *Hybrid monte carlo*, *Physics Letters B* **195** (1987) 216–222.

[92] R. M. Neal, *MCMC using Hamiltonian dynamics*, 1206.1901.

[93] M. Backes, A. Butter, T. Plehn and R. Winterhalder, *How to GAN Event Unweighting*, 2012.07873.

[94] R. Verheyen and B. Stienen, *Phase Space Sampling and Inference from Weighted Events with Autoregressive Flows*, 2011.13445.

[95] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan et al., *Pytorch: An imperative style, high-performance deep learning library*, in *Advances in Neural Information Processing Systems 32*.

[96] L. Dinh, J. Sohl-Dickstein and S. Bengio, *Density estimation using real NVP*, in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017, https://openreview.net/forum?id=HkpbnH9lx.

[97] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015, http://arxiv.org/abs/1412.6980.

[98] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon et al., *Pot: Python optimal transport*, *Journal of Machine Learning Research* **22** (2021) 1–8.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] see Section 2
   (b) Did you include complete proofs of all theoretical results? [Yes] see Section 2 and the cited references.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 4 for implementation details and the URL for the code is provided as a footnote in the abstract.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]