Dissertation
submitted to the
Combined Faculties of the Natural Sciences and Mathematics
of the Ruperto-Carola-University of Heidelberg, Germany
for the degree of
Doctor of Natural Sciences

Put forward by
Marco Bellagente
born in Desio
Oral examination: 26.01.2022

Go with the Flow

Normalizing Flow applications for High Energy Physics

Referees: Prof. Dr. Jan-Martin Pawlowski

Prof. Dr. Monica Dunford

Abstract

Deep Learning is becoming a standard tool across science and industry to optimally solve a variety of tasks. A challenge of great importance for the research program carried over at the Large Hadron Collider is realising a generative model to sample synthetic data from a desired probability density. While generative models such as Generative Adversarial Networks and Normalizing Flows have been originally designed to solve Machine Learning tasks such as classification and data generation, we illustrate how they can also be employed to statistically invert Monte Carlo simulations of detector effects. In particular, we show how conditional Generative Adversarial Networks and Normalizing Flows are capable of unfolding detector effects, using ZW production at the LHC as a benchmarking process.

Two technical by-products of interest stemming from these studies are the introduction of a Bayesian Normalizing Flow and of the Latent Space Refinement (LaSeR) protocol. The former has been introduced in order to address the crucial question of explainability and uncertainty estimation of deep generative models, which is achieved by reformulating the training and prediction phases of Normalizing Flows as a Bayesian inference task. Finally, LaSeR is a method to refine a model's output using classifier weights. We show how LaSeR can critically improve the performances of a Normalizing Flow whenever the training data contains topological obstructions.

Zusammenfassung

Deep Learning wird gegenwärtig in Wissenschaft und Industrie zu einem Standardwerkzeug, um unterschiedlichste Aufgaben optimal zu lösen. Eine Herausforderung von großer Bedeutung für LHC-Analysen ist die Realisierung eines generativen Modells, um synthetische Daten aus einer gewünschten Wahrscheinlichkeitsdichte zu generieren. Während generative Modelle wie Generative Adversarial Networks und Normalizing Flows ursprünglich entwickelt wurden, um Machine-Learning-Aufgaben wie Klassifikation und Datengenerierung zu lösen, zeigen wir, wie sie auch verwendet werden können, um Monte-Carlo-Simulationen von Detektoreffekten statistisch zu invertieren. Insbesondere zeigen wir, wie conditional-Generative Adversarial Networks und -Normalizing Flows Detektoreffekte umkehren können, indem wir die ZW-Produktion am LHC als Benchmarking-Prozess verwenden.

Zwei interessante technische Nebenprodukte aus diesen Studien sind die Einführung einer Bayesian-Normalizing Flow und des Latent Space Refinement (LaSeR)-Protokolls. Ersteres wurde eingeführt, um die entscheidende Frage der Erklärbarkeit und Unsicherheitsschätzung von tiefen generativen Modellen zu beantworten, die durch die Neuformulierung der Trainings- und Vorhersagephasen von Normalizing Flows als Bayes'sche Inferenzaufgabe erreicht wird. Letzteres ist eine Methode, um die Ausgabe eines Modells mithilfe von Klassifikatorgewichten zu verfeinern. Wir zeigen, wie LaSeR die Leistung einer Normalizing Flow entscheidend verbessern kann, wenn die Trainingsdaten topologische Hindernisse enthalten.

Contents

1 Introduction **1**
- 1.1 Generative models . . . 1
  - 1.1.1 Gaussian Mixture Models . . . 2
- 1.2 Deep Generative Models . . . 3
  - 1.2.1 Normaling Flows . . . 4
  - 1.2.2 Generative Adversarial Networks . . . 7
- 1.3 Unfolding . . . 8
  - 1.3.1 Iterated Bayesian Unfolding . . . 8
  - 1.3.2 OmniFold . . . 10

2 Unfolding with deep generative models **13**
- 2.1 Introduction . . . 13
- 2.2 Unfolding basics . . . 14
  - 2.2.1 Binned toy model and locality . . . 14
  - 2.2.2 Bayes' theorem and model dependence . . . 15
  - 2.2.3 Reference process . . . 16
- 2.3 GAN unfolding . . . 16
  - 2.3.1 Naive GAN . . . 17
  - 2.3.2 Conditional GAN . . . 19
  - 2.3.3 New physics injection . . . 23
- 2.4 INN unfolding . . . 25
  - 2.4.1 Naive INN . . . 26
  - 2.4.2 Noise-extended INN . . . 27
  - 2.4.3 Conditional INN . . . 30
- 2.5 Unfolding with jet radiation . . . 33
  - 2.5.1 Individual n-jet samples . . . 34
  - 2.5.2 Combined n-jet sample . . . 36
- 2.6 Conclusions and outlook . . . 38

3 What is the uncertainty of generative models? **39**
- 3.1 Introduction . . . 39
- 3.2 Generative networks with uncertainties . . . 40
  - 3.2.1 Uncertainties on event samples . . . 40
  - 3.2.2 Bayesian INN . . . 41
- 3.3 Toy events with uncertainties . . . 43
  - 3.3.1 Wedge ramp . . . 44
  - 3.3.2 Quadratic ramp . . . 46

3.3.3 Gaussian ring . . . 47
3.3.4 Errors vs training statistics . . . 48
3.3.5 Marginalizing phase space . . . 50
3.4 LHC events with uncertainties . . . 52
3.5 Conclusions and outlook . . . 53

4 Topology improvement for invertible architectures 55
4.1 Introduction . . . 55
4.2 Background . . . 57
4.2.1 Generative models and coordinate transformations . . . 57
4.2.2 Topological obstructions . . . 58
4.3 Proposed methods . . . 59
4.4 Toy examples . . . 60
4.4.1 Implementation details . . . 60
4.4.2 Probability distance measures . . . 61
4.4.3 Experimental results . . . 62
4.5 LHC applications . . . 64
4.6 Conclusions and outlook . . . 66

5 Summary and Outlook 67

6 Acknowledgements 69

Appendices 71

A Appendix: conditional GAN 73
A.1 Performance . . . 73
A.2 Staggered vs cooling MMD . . . 74

References 77

Preface

The research presented in this thesis was conducted at the Institute for Theoretical Physics at Heidelberg University from February 2019 to February 2022. The contents of the Chapters 2-4 are based on work in collaboration with other authors and have previously been published as

[1] M. Bellagente, A. Butter, G. Kasieeczka, T. Plehn and R. Winterhalder,
"How to GAN away Detector Effects",
SciPost Phys. **8** (2020) no. 4, 070, `arXiv:1912.00477 [hep-ph]`

[2] M. Bellagente, A. Butter, G. Kasieeczka, T. Plehn, R. Winterhalder, L. Ardizzone and U. Köthe,
"Invertible networks or partons to detector and back again",
SciPost Phys. **9** (2020) 074, `arXiv:2006.06685 [hep-ph]`

[3] M. Bellagente, M. Luchmann, M. Haußmann and T. Plehn,
"Understanding Event- Generation Networks via Uncertainties",
`arXiv:2104.04543 [hep-ph]`

[4] R. Winterhalder, M. Bellagente, and B. Nachmann
"Latent Space Refinement for Deep Generative Models",
`arXiv:2106.00792 [stat.ML]`

1 | Introduction

The research program carried out at the Large Hadron Collider (LHC) relies on vast amount of synthetic data to perform hypothesis tests [1–4]. Simulations of parton level interactions and the subsequent processes of parton showering and hadronization, as well as the simulation of detector effects, are traditionally performed with Monte Carlo methods [5–8]. These methods are built using first-principles, in the sense that they are expected to describe the corresponding physical system to the best of our knowledge, being that a detector or a quantum field theory. However, precision and interpretability do not come for free, and a major shortcoming of these methods lies in their computational efficiency. In fact, Monte Carlo simulations take the biggest share of computational resources of collaborations working at the LHC, with detector simulations being the biggest contributors [3]. As a notable example, the time needed to fully simulate a detector response with GEANT4 [9] can be of the order of minutes. If we give up on the idea of simulating a physical system from first principles, we can reformulate the problem entirely as a sampling task: a detector image may be for instance regarded as a sample from an hypothetical distribution $p_{detector}$ defined on some high-dimensional space. The same idea may be applied to other stages of the simulation pipeline, such as the simulation of parton level events, or parton showers. If samples drawn from such a distribution can be used as a surrogate for first principle simulations, massive gains in computational time are theoretically possible, hence the recent interest in the potential of deep generative models for particle physics [10–25].

Deep generative models are a natural tool for supporting (or replacing) traditional Monte Carlo simulations, as they have the necessary expressive power to accurately reproduce complex, high-dimensional and multi-modal distributions. Furthermore, the field of high energy physics does not suffer from a typical bottleneck of deep learning, where clean, (labelled) natural images, texts, or biomedical data, can be expensive to obtain or be protected by privacy laws. Conversely, truly large, particle physics datasets are either publicly available or can be in most cases generated using open source software.

Furthermore, the potential of deep generative models goes beyond the production of synthetic data. Depending on the particular framework, they can solve a variety of tasks, including unfolding [26–28] classification [29, 30], model proposal [31], out-of-distribution detection [32–35], parton distribution functions compression [36] and anomaly detection [37–40]. Given the variety of applications and the number of publications on the topic, we refer to https://github.com/iml-wg/HEPML-LivingReview for a complete overview.

The thesis is organized as follows: in the remaining part of Chap. 1 we give a definition of generative models and introduce the unfolding problem, the two main topics of the thesis; in Chap. 2 we illustrate a method for unfolding detector effects based on deep generative models; in Chap 3 we then address a key aspect for a realistic use in LHC analysis, namely how to define uncertainties over a deep generative model's output; in Chap. 4 we introduce a method to handle topological obstructions induced by parton level cuts. We conclude with a summary of the main results and with possible future directions. The work of this thesis is based on Ref. [41–44].

1.1 Generative models

Given data instances $\mathcal{X} = \{x_1, \dots, x_N\}$, a generative model is a probabilistic model of $p(x)$, i.e. of the underlying probability density from which $\mathcal{X}$ is sampled. In machine learning, generative modelling, or density estimation, is the unsupervised task of approximating a probability density via a parametrized

function G_θ, whose parameters θ are inferred in an automatic fashion. If G_θ is a good approximation of $p(x)$, new and realistic samples can be drawn from it. Similarly, if the data comes with labels $\mathcal{Y}$, we wish to approximate the conditional probability density $p(x|y)$ with a conditional probabilistic model $G_\theta(x|y)$. The underlying assumption of density estimation is that the manifold on which $\mathcal{X}$ lies can be described by a number of parameters which is lower than the dimensionality of the data itself. This way G_θ is forced to discover regularities and patterns of the input data.

1.1.1 Gaussian Mixture Models

As an example, given some multi-modal distribution $p(x)$, we may represent it compactly with a finite mixture

$$p(x) = \sum_{k=1}^{K} \pi_k p_k(x) \tag{1.1}$$

with mixture weights π_k constrained as

$$0 \leq \pi_k \leq 1, \quad \sum_{k=1}^{K} \pi_k = 1. \tag{1.2}$$

The components $p_k(x)$ belong to a family of a basic distributions, e.g. Gaussians $\mathcal{N}_k(x|\mu_k, \Sigma_k)$, in which case we call the model Gaussian Mixture Model (GMM) [45]. The model's parameters $\theta = \{\mu_k, \Sigma_k, \pi_k;\ k = 1, \ldots, K\}$ can be inferred via maximum likelihood. Assuming that the points in the dataset $\mathcal{X} = \{x_1, \ldots, x_N\}$ are identically and independently distributed, the likelihood can be written in the factorized form

$$p(\mathcal{X}|\theta) = \prod_{n=1}^{N} p(x_n|\theta), \quad p(x_n|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}_k(x_n|\mu_k, \Sigma_k), \tag{1.3}$$

leading to the log-likelihood

$$\mathcal{L} \coloneqq \log p(\mathcal{X}|\theta) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}_k(x_n|\mu_k, \Sigma_k). \tag{1.4}$$

The optimal parameters θ^* are those that maximize the log-likelihood, and can therefore be obtained by solving for θ the equation $\mathrm{d}\mathcal{L}/\mathrm{d}\theta = 0$. As a concrete example, taking the derivative of $\mathcal{L}$ with respect to μ_k yelds

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = \sum_{n=1}^{N} \frac{1}{p(x_n|\theta)} \frac{\partial p(x_n|\theta)}{\partial \mu_k} = \sum_{n=1}^{N} (x_n - \mu_k)^T \Sigma_k^{-1} \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_j)} \tag{1.5}$$

$$= \sum_{n=1}^{N} r_{nk} (x_n - \mu_k)^T \Sigma_k^{-1}. \tag{1.6}$$

where we have defined the responsibilities

$$r_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_j)}. \tag{1.7}$$

Setting $\partial \mathcal{L}/\partial \mu_k = 0$ we obtain

$$\mu_k^* = \frac{1}{\sum_{n=1}^{N} r_{nk}} \sum_{n=1}^{N} r_{nk} x_n, \tag{1.8}$$

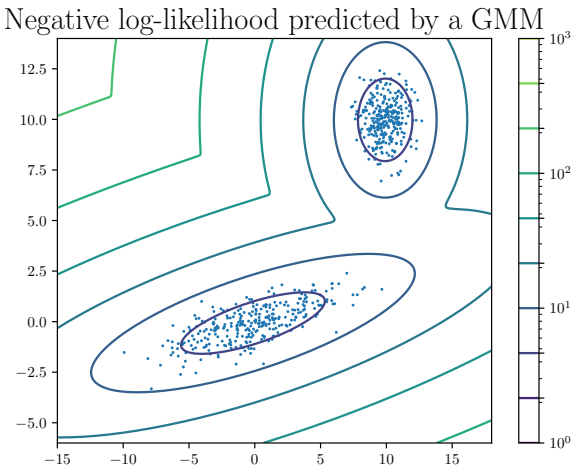

Figure 1.1: Gaussian mixture model with two components. The equal probability surfaces are highlighted in colours.

Similarly, it is possible to show that both π_k^* and Σ_k^* can also be expressed in terms of r_{nk}. As it is often the case in machine learning, there is no closed form solution to the maximization problem for $k > 1$, and we rely instead on an updating scheme which, starting from randomized values of θ, iteratively finds a better approximation of θ^*. In the contest of GMMs, this iterative solution is called Expectation-Minimization algorithm [46], and can be summarized as

- Initialize μ_k, Σ_k, π_k;
- E-step: evaluate the responsibilities r_{nk} using current parameters μ_k, Σ_k, π_k;
- M-step: estimate the new values $\mu_k^*, \Sigma_k^*, \pi_k^*$ using the current responsibilities.

Each E/M-step increases the likelihood [47]. The number of iterations and mixtures are hyperparameters. In Fig. 1.1 we show a simple example of a Gaussian mixture model used to predict the density of a 2-dimensional dataset.

1.2 Deep Generative Models

We may regard the GMM as the simplest form of statistical generative model. Recent years have seen the advent of novel methods for generative modelling which we collectively refer to as Deep Generative Models (DGM), in complete analogy to what deep learning means with respect to machine learning. Before describing the most relevant forms of DGMs, and with no claim of completeness, we recall that some of the key features of deep learning include

- models are parametrized by Neural Networks (NN) whose specific form depends on the particular task;
- the NN parameters are inferred via a stochastic gradient descent approach;
- in order to really exploit NNs, vast amount of high-fidelity data is needed for their training.

The last bullet is particularly attractive for LHC analysis, in that not only scientific collaborations such as ATLAS and CMS have already collected petabytes of data (with a lot more to come with the next upgrade of the collider), but we can also prepare data in a controlled environment for benchmarking models.

Concerning the practical realisation of DGMs, the three most successful frameworks studied and employed in the literature are

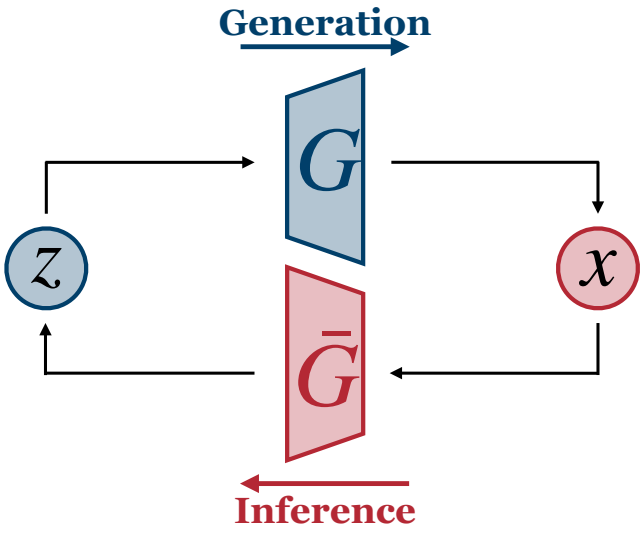

Figure 1.2: Schematic representation of a NF: a random variable z sampled from a fixed, simple distribution is transformed via a map G to a sample x sampled from a complex target distribution. If G is a bijection, and its inverse $\bar{G} = G^{-1}$ can be easily evaluated, the inverse direction can be used to train the model with Maximum Likelihood.

- Normalizing Flows (NF [48–56]);
- Generative Adversarial Networks (GAN) [57–62];
- Variational Autoencoders [63, 64].

The first two have been extensively employed in this thesis, and will be therefore further elaborated in the following sections.

1.2.1 Normaling Flows

Normalizing Flow are defined in terms of a parametrized change of variable $G_\theta : z \to x$ between distributions

$$p_X(x) = p_Z(z) \left| \det \frac{\partial G_\theta(z)}{\partial z} \right|^{-1} = p_Z(\bar{G}_\theta(x)) \left| \det \frac{\partial \bar{G}_\theta(x)}{\partial x} \right|, \tag{1.9}$$

where $\bar{G}_\theta = G_\theta^{-1}$. Regardless of how complicated $p_X(x)$ may be, if there exists a bijection G_θ such that $p_Z(z)$ is a simple distribution, e.g. a Gaussian $\mathcal{N}_{\mu=0,\Sigma=\mathbb{I}}$, we can draw samples from $p_X(x)$ via the generative pipeline

$$z \sim p_Z(z) \longrightarrow x = G_\theta(z) \sim p_X(x), \tag{1.10}$$

see Fig. 1.2 for a schematic representation. Similar to the GMM example in Sec. 1.1, the parameters

of G_θ are be chosen to minimize the negative log-likelihood

$$\begin{aligned}\mathcal{L} &= -\sum_{n=1}^{N} \log\left[p_Z(\bar{G}_\theta(x_n))\left|\det\frac{\partial \bar{G}_\theta(x)}{\partial x_n}\right|\right] \\ &= -\sum_{n=1}^{N} \log p_Z(\bar{G}_\theta(x_n)) + \log\left|\det\frac{\partial \bar{G}_\theta(x_n)}{\partial x_n}\right| \\ &= \sum_{n=1}^{N} \frac{(\bar{G}_\theta(x_n))^2}{2} - \log\left|\det\frac{\partial \bar{G}_\theta(x_n)}{\partial x_n}\right|,\end{aligned} \tag{1.11}$$

where we have used $p_Z(z) = \mathcal{N}_{\mu=0,\Sigma=\mathbb{I}}$ and have discarded irrelevant constant terms. In practice, we only need to compute $\bar{G}_\theta(x_n)$, the corresponding determinant of the Jacobian, and minimize the combination in the final line of Eq. 1.11.

This rather straightforward blueprint hides three complications one needs to carefully address. First of all, not only G_θ must be a bijection, i.e. its inverse must exist, but both G_θ and $\bar{G}_\theta$ must be efficiently computable, as one is required for training and the other for generating. Finally, the map must have a tractable Jacobian. If G_θ is parametrized by a standard NN, such as a Multi-Layer Perceptron (MLP), none of the three properties is satisfied. These constraints restrict the class of allowed transformations, limiting the expressive power at our disposal. However, not all hope is lost, as we can build progressively more complex maps by simply stacking many simple functions, while maintaining the tractability of the Jacobian. Given a bijection G_i with inverse $\bar{G}_i$ and a cheap Jacobian, the composition

$$G = G_n \cdot G_{n-1} \ldots \cdot G_1 \tag{1.12}$$

has inverse

$$\bar{G} = \bar{G}_1 \cdot \ldots \cdot \bar{G}_{n-1} \cdot \bar{G}_n, \tag{1.13}$$

and its determinant is the product of each individual determinant

$$\det G = \prod_{i=1}^{N} \det G_i. \tag{1.14}$$

This means that a composition of bijections is again a bijection whose computational complexity simply grows linearly with the number of compositions. As a result, this research line is primarily dedicated to finding the optimal trade-off between computational complexity and model expressive power. In the remaining of this section we give an overview of past and currently used methods.

Planar and Radial Flows

The original work on NFs [65] introduces the simple planar and radial flows. Despite representing seemingly simple transformations, they are not easily invertible. A planar flow takes the form

$$G(x) = x + uh(w^T x + b), \tag{1.15}$$

where u, w, b are parameters and h is a non-linear activation function. This transformation expands and contracts the distribution of x along certain axis. The planar flow has the simple Jacobian

$$\det\frac{\partial G}{\partial x} = 1 + h'(w^T x + b)u^T w \tag{1.16}$$

which can be computed in $\mathcal{O}(D)$ time but cannot be inverted in closed form and its inverse does not exist in general. Radial flows modify instead the distribution around a point x_0 as

$$G(x) = x + \frac{\beta}{\alpha + ||x - x_0||}(x - x_0). \tag{1.17}$$

The same considerations apply here, the determinant of a radial flow can be efficiently computed, but its inverse is not always defined and cannot be computed in closed form.

Dimensional Partitioning

Introduced in [49,66], this approach builds an invertible transformation by splitting a D-dimensional input x into two subspaces (x_1, x_2), and defining the bijection G, normally called a coupling layer, as

$$y_1 = h(x_1, f(x_2)) \tag{1.18}$$
$$y_2 = x_2. \tag{1.19}$$

with inverse

$$x_1 = h^{-1}(y_1, f(y_2)) \tag{1.20}$$
$$x_2 = y_2. \tag{1.21}$$

The main feature of a coupling layer is that even if f is an arbitrary function, the Jacobian of G is always a triangular matrix which does not depend on the derivative of f. Different strategies can be used to perform the partitioning and ensure that every dimension gets transformed, such as alternating splittings in half and random permutations [66], using masked flow [49] or using (1×1) convolutions [50]. This is the method of choice for all the studies in this thesis, and in Chap. 2 we explain in details our implementation.

Autoregressive Flows

An autoregressive flow is a non-linear generalization of a multiplication by a triangular matrix [67,68]. Motivated by the product rule of probability, stating that a joint density can be computed as a product of one dimensional conditionals

$$p(x) = \prod_i p(x_i|x_{1:i-1}), \tag{1.22}$$

in an autoregressive model the output $y = G(x)$ is computed step-by-step as a function conditioned on the previous entries, i.e.

$$y_t = h(x_t, f_t(x_{1:t-1})), \tag{1.23}$$

where $x_{1:t} = (x_1, \dots, x_t)$, and f_t is an arbitrary map taking a $t-1$ dimensional input. Similarly to a coupling layer, since each output y_t only depends on $x_{1:t}$, the resulting Jacobian is again triangular and its determinant can be efficiently computed. The main difference with respect to coupling layers is that the inverse function, which can be trivially computed for the former, must be computed recursively as

$$x_t = h^{-1}(y_t, f_t(x_{1:t-1})). \tag{1.24}$$

As this operation is inherently sequential, it cannot be parallelized and represent therefore a bottleneck for training this kind of model. Depending on the application, one may want to either have a fast forward direction, in which case one gets a so called masked autoregressive flow, or a fast inverse direction, giving instead an inverse autoregressive flow.

Residual Flows and ODE

The last method we present is based on residual connections, i.e. maps with the form

$$G(x) = x + f(x). \tag{1.25}$$

Residual blocks have been used for a long time [69], and have been adapted into invertible architecture in Ref. [70,71]. Splitting input and output as (x_1, x_2), (y_1, y_2), the map

$$y_1 = x_1 + f(x_2) \tag{1.26}$$
$$y_2 = x_2 + l(y_1) \tag{1.27}$$

is trivially invertible, but has an inefficient Jacobian. Ref. [70, 71] introduces strategies to enforce the invertibility. Most notably, however, is the fact one can see a residual connection as the discretized version of an ordinary differential equation. The starting point is an ordinary differential equation of the form

$$\frac{d}{dt}x(t) = F(x(t), \theta(t)), \tag{1.28}$$

where $F : \mathbb{R}^D \times \Theta \to \mathbb{R}^D$ encodes the dynamics of the system, Θ is a set of parameters and $\theta : \mathbb{R} \to \Theta$ is a parametrization. The discrete version of Eq. 1.28

$$x_{n+1} - x_n = \epsilon F(x_n, \theta_n), \tag{1.29}$$

is equivalent to the equation of a residual connection with residual block ϵF. Starting from this intuition, Ref. [55, 56, 72] propose to directly solve the continuous version Eq. 1.28, which in this differential equation picture would correspond to training an infinitely deep network.

We conclude this subsection with a comment motivating the study of Chap. 4. As already stated, invertible models are bijections, or homeomorphisms [72, 73] (note that this is a strict equivalence, i.e. an invertible model does not approximate a homeomorphism, it is one), and as such they are by construction unable to map manifolds with different topological features, such as the number of holes and disconnected patches. In particular, if we stick to the standard formulation and only consider Gaussian latent spaces, we may only be able to represent data whose underlying manifold is compact and has genus zero. As this may be a limiting factor for the naive use of normalizing flows in particle physics, in Chap. 4 we illustrate a possible method to account for it.

1.2.2 Generative Adversarial Networks

GANs introduce a brand new paradigm for generative modelling which, unlike the previously described methods, does not involve computing the likelihood at all, and is therefore a form of likelihood-free machine learning. Instead, GANs rely on a two-sample test between distributions, i.e. a statistical test to determine whether or not two samples are drawn from the same distribution. This test is formulated as a differentiable objective function so that it can be minimized with stochastic gradient descent methods, in such a way that it is minimal iff the two distributions are statistically identical. The statistical test is performed by a discriminator D_ϕ whose job is to tell apart real and fake samples. The final piece of a GAN is a generator G_θ, a map from noise z to target samples $\tilde{x}$, see Fig. 1.3. In DL, both D_ϕ and G_θ are properly shaped NNs. While many different ways to define the GAN objective have been proposed, we refer to the original formulation [57] as it is close to the one employed in Chap. 2. Formally, we consider the min-max game

$$\min_\theta \max_\phi \mathbb{E}_{x\sim p_X(x)} \left[\log D_\phi(x)\right] + \mathbb{E}_{z\sim p_Z(z)} \left[\log\left(1 - D_\phi(G_\theta(z))\right)\right]. \tag{1.30}$$

For a fixed generator, this is nothing other than a binary classification task, and an optimal discriminator should output 1 for any $x \sim p_X(x)$ and 0 otherwise, which we can summarize as

$$D_{\phi^*}(x) = \frac{p_X(x)}{p_X(x) + p_G(x)}, \tag{1.31}$$

with $p_G(x)$ being the distribution implicitly defined by the generator. Fixing $\phi = \phi^*$, the generator part of the objective is equivalent to minimizing the Jenson-Shannon divergence $JS(p_X, p_G)$, with

$$JS\left(p, q\right) = \frac{1}{2}\left(KL\left(p, \frac{p+q}{2}\right) + KL\left(q, \frac{p+q}{2}\right)\right), \tag{1.32}$$

and $KL(a, b)$ is the Kullback-Leibler divergence between p and q. This objective has a global minimum for $p_G = p_X$. It turns out that if the discriminator is optimal, the objective function has vanishing gradient for the generator, leading to the standard alternating training of GANs: every k updates of ϕ we perform an update of θ, k being a hyperparameter. Training a GAN is a highly non trivial task with many potential issues if extra care is not taken, notably

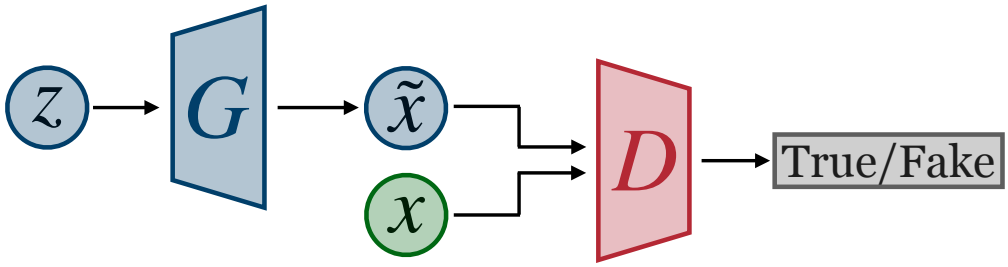

Figure 1.3: Schematic representation of a GAN: a generator G deterministically maps noise z to fake samples $\tilde{x}$. The discriminator D is trained as a binary classifier of true versus fake data, while G tries to fool it.

- the optimization procedure is typically unstable and needs to be regularized;
- for multi-modal targets, generators can collapse to a single mode;
- since the objective function is not convex, there is no clear way to evaluate the model performances or to determine a stopping point.

An important difference between the two presented frameworks is that a GAN represents a target density only implicitly, i.e. G_θ is only a sampler from p_X, but cannot be used to estimate its value (with the exception of low dimensional problems where kernel methods may be used to smooth a binned distribution). On the other hand, the explicit value of the density can be trivially computed with a NF from Eq. 1.9 if we assume a perfect map to the latent space distribution.

1.3 Unfolding

In any quantitative science we wish to be able to estimate the possible value of "true" parameters, directly connected to some underlying theory, given that we are only able of observing a distorted version of them in the best scenario, more realistically we will have to be satisfied with observing a different set of parameters entirely. In particular, we are often interested in lifting the effects induced by the detectors and possible physical and theoretical backgrounds. This procedure is normally called de-convolution, and takes the name of unfolding in particle physics. In this section we first revise a standard method for unfolding based on Bayesian inference, and then illustrate how to set-up a deep-learning version of it.

1.3.1 Iterated Bayesian Unfolding

In this section we will define the fundamental ideas of unfolding following closely Ref. [74,75]. Bayesian inference is a framework to estimate unknown parameters from observed data. A starting point for Bayesian unfolding is the discretization of the problem, in the form of binning of observable distribution of events, with each bin being treated as an independent variables. Borrowing the language of graphical models, the problem can be described by the Bayesian network of Fig. 1.4, C_i being the true number of events in each cause bin, given that we observe E_j events per bin. It's important to notice that as the links between cause and effect is probabilistic, so will be the number of true events per bin. Concretely, given causes C_i and effects E_j, we wish to estimate the conditional distribution

$$P(C_i|E_j) \tag{1.33}$$

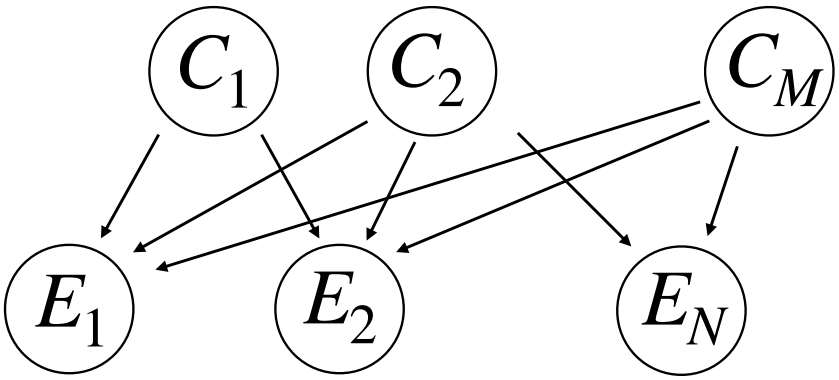

Figure 1.4: Graphical model of causes and effects. Each arrow represents a probabilistic link

Following standard Bayesian inference, Eq. 1.33 may be computed as

$$P(C_i|E_j) = \frac{P(E_j|C_i)P(C_i)}{\sum_{l=1}^{n_E} P(E_j|C_l)P(C_l)}, \tag{1.34}$$

Eq. 1.34 is Bayes' theorem stating that the posterior is proportional to the likelihood, provided that we multiply by the prior

$$P(C_i|E_j) \propto P(E_j|C_i)P(C_i), \tag{1.35}$$

and the denominator is a normalization factor. The unfolding algorithm proceeds as follows:

- estimate the number of expected events in bin C_i given that $n(E_i)$ is the observed counting bin E_j

$$n(C_i)|_{n(E_j)} \sim P(C_i|E_j)n(E_j); \tag{1.36}$$

- include effects from all observations

$$n(C_i) = \sum_j P(C_i|E_j)n(E_j); \tag{1.37}$$

- correct for the effect of different efficiencies ϵ_i

$$\tilde{n}(C_i) = \frac{1}{\epsilon_i} n(C_i) = \frac{1}{\epsilon_i} \sum_{j=1}^{n_E} P(C_i|E_j)n(E_j); \tag{1.38}$$

with the efficiencies computed from the smearing matrix as

$$\epsilon_i = \sum_{j=1}^{n_E} P(E_j|C_i) = \sum_{j=1}^{n_E} \lambda_{ij}, \tag{1.39}$$

where $\lambda_{ij} = P(E_j|C_i)$ are the entries of a smearing matrix Λ estimated from Monte Carlo simulations.

The above algorithm is essentially the first form of Bayesian unfolding presented in Ref. [74]. Ref. [75] introduced several improvements over the first version. A first improvements concerns the explicit modelling of the smearing matrix. In the original formulation, one starts by generating a large number of events in each cell and counting where they end up after the simulation. The entries λ_{ij} are estimated as

$$\lambda_{ij} \sim \frac{n(E_j)^{MC}}{n(C_i)^{MC}}. \tag{1.40}$$

We can both improve this estimate, and include its uncertainty, by employing again Bayes theorem and computing a posterior over the λ_{ij}. Given a column λ_i of Λ, we have

$$f(\lambda_i|n(E)^{MC}, n(C_i)^{MC}) \propto P(n(E)^{MC}|n(C_i)^{MC}, \lambda_i) f(\lambda_i). \tag{1.41}$$

If we choose a Dirichlet prior for λ_i it is possible to show

$$\lambda_i \sim \mathrm{Dir}(\alpha_{\mathrm{posterior},i}), \quad \alpha_{\mathrm{posterior},i} = \alpha_{\mathrm{prior},i} + n(E)^{MC}|_{n(C_i)^{MC}}, \tag{1.42}$$

where

$$\mathrm{Dir}(x|\alpha) = x_1^{\alpha_1 - 1} \dots x_n^{\alpha_n - 1}. \tag{1.43}$$

We can now account for the extra uncertainty in $P(C_i|E_j)$ by computing

$$P(C_i|E_j) \longrightarrow P(C_i|E_j, \Lambda); \qquad P(C_i|E_j) = \int P(C_i|E_j) f(\Lambda) d\Lambda, \tag{1.44}$$

i.e. the posterior over causes depends explicitly on the smearing matrix Λ, which is sampled from the posterior distribution $f(\Lambda)$. Most notably, however, is the additional iteration of the algorithm. Bayesian inference only makes sense when we have a prior belief of what the possible values of the unobserved parameters are. Choosing a "correct" prior is a tricky task, as this choice, which is arbitrary, affects the posterior distribution. Ref. [75] suggests that the dependence on the prior can be reduced by iterating the unfolding procedure, by using the result of unfolding step n as the prior of unfolding $n+1$. This method is therefore called Iterated Bayesian Unfolding (IBU).

We conclude with a brief discussions of the limitations of iterated Bayesian unfolding, and equivalent methods. The first obvious limitation is that the method relies on binning measurements into histograms. Binning can be problematic as aside with the exception of few limited case there is no obvious best binning, and each individual observable typically requires an ad hoc manual choice of the binning. Directly related to this is the fact that the total number of bins grows exponentially with the number of dimensions, each being a different observable. This means that differential cross section measurements beyond a few dimensions are impractical.

1.3.2 OmniFold

In parallel to the work of Ref. [41, 42], which constitutes the main content of Chap. 2, another deep-learning framework for unfolding has been published. Because of its relevance as the only other existing method for unfolding relying on deep-learning, and for its connection to IBU, we believe that an explanation of the OmniFold method [26] deserves its own section.

OmniFold has been proposed to overcome the two main shortcomings of IBU, binning and handling multiple dimensions, by generalizing the iterated version of Eq. 1.38 to the continuous, multi-dimensional case. In Fig. 1.5 we illustrate schematically the method. Starting with particle level events t with initial weights $\nu_0(t)$ simulated with Monte Carlo, we produce a paired set of detector level events m with induced weights $\nu_0^{\mathrm{push}}(m) = \nu_0(t)$. We then compute new weights $\omega_0(m)$, and use them to obtain the next set of particle level weights $\nu_1(t)$. The key idea to achieve the correct reweighting is that the likelihood ratio

$$L\left[(w, X), (w', X')\right](x) = \frac{p_{w,X}(x)}{p_{w',X'}(x)}, \tag{1.45}$$

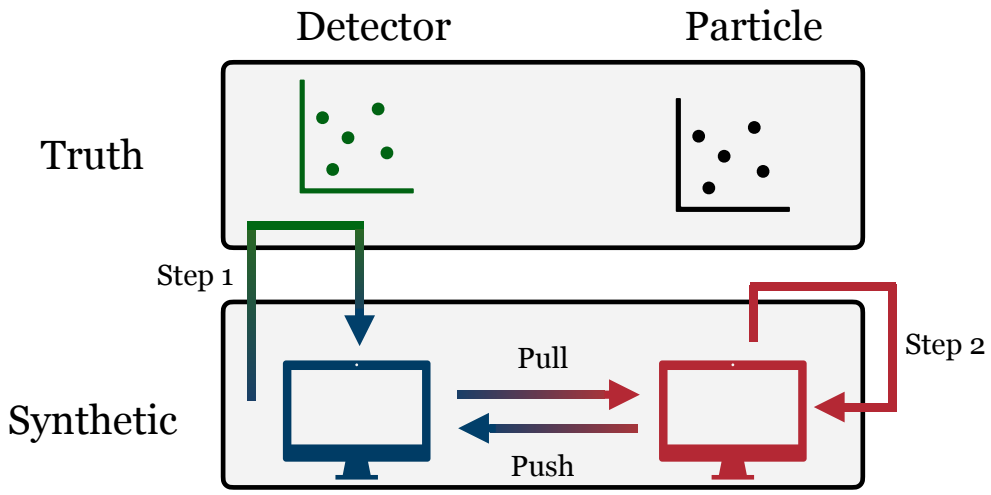

Figure 1.5: Illustration of the OmniFold method: synthetic data is reweighted against the true data at the detector level. This weights are pulled back at the particle level to induce weights on the particle level data. In the second step the original synthetic particle level data is weighted to match its reweighted version. The process is then iterated.

can be approximated using a classifier trained to distinguish (w, X) against $(w^{'}, X^{'})$ [76–80]. With the likelihood ratio it is straightforward to reweight events as

$$\omega_n(m) = \nu_{n-1}^{\text{push}} L\left[(1, \text{truth, det})), (\nu_{n-1}^{\text{push}}, \text{Synt, det})\right], \tag{1.46}$$

$$\nu_n(t) = \nu_{n-1} L\left[(\omega_n, \text{synt, part})), (\nu_{n-1}, \text{synt, part}))\right]. \tag{1.47}$$

It is possible to show that after one iteration the new weights are given by

$$\nu_1 p_{synt,part}(t) = \int dm^{'} p_{\text{ synt, part| Synt, Det}}(t|m^{'}) p_{\text{truth, det}}(m^{'}) \tag{1.48}$$

which is a continuous version of Eq. 1.38. On top of that, OmniFold is not limited by the dimensionality of the observables, and can in principle unfold the full phase space of an event, provided that the classifier employed is powerful enough.

2 | Unfolding with deep generative models

LHC analysis accounts for the distortion induced by detector measurements using an unfolding procedure. Two shortcomings of existing unfolding techniques are their reliance on binned distributions, and, consequentially, their restriction to a limited number of selected observables. We show how conditional generative models can be used to map detector level observables to a pre-defined hard process. Specifically, we consider ZW production at the LHC, with either a fixed or variable number of QCD jets reconstructed by the detector. While additional work remain to be done to assess the prior-independence of generative unfolding, it represents a first step toward multi-dimensional unbinned unfolding.

2.1 Introduction

Our understanding of LHC data from first principles is a unique strength of particle physics. It is based on a simulation pipeline which starts from a hard process described by perturbative QCD, then adds the logarithmically enhanced QCD parton shower, fragmentation and hadronization, and concludes with a simulation of the detector [81]. In this henceforth called forward direction, simulations are based on highly optimized, reliable Monte Carlo techniques, and ideally we would just compare measured and simulated events to draw conclusions about a new physics hypothesis.

Because our simulation chain is well defined only in one direction, the typical LHC analysis starts with a new, theory-inspired hypothesis encoded in a Lagrangian as new particles and couplings. For every point in the new physics parameter space we simulate events, compare them to the measured data using likelihood methods, and discard the new physics hypothesis.

We propose to use generative models to invert Monte Carlo simulations. Technically, we propose to use GANs or INNs to invert part of the LHC simulation chain with a simplified template. This application builds on a long list of one-directional applications of generative or similar networks to LHC simulations, including phase space integration [82,83], amplitudes [84,85], event generation [19,86–89], detector simulations [10, 11, 14, 15, 90–94], parton showers [95–98], or searches for physics beyond the Standard Model [99]. In particle physics, INNs have proven useful for instance in phase space generation [100], linking integration with generation [101, 102], or anomaly detection [103].

This chapter is organized as follows: in Sec 2.2 we begin by revising the inverse problem and present in details the data preparation for the reference process considered in this chapter. Then, in Sec. 2.3, we show step-by-step how to formulate the generative inversion. We start with a naive GAN inversion and see how a mismatch between local structures at parton level and detector level leads to problems, leading to the introduction of a fully conditional GAN (FCGAN) . We will see how the conditional setting gives us all the required properties of an inverted detector simulation. Building on that, in Sec. 2.4 we show how the bijective structure of the INN makes their training especially stable. If we add sufficiently many random numbers to the INN we can start generating probability distributions in the parton-level phase space. The conditional INN [104, 105] adds even more sampling elements to the generation of unfolded configurations. For arbitrary kinematic distributions we can test the calibration of all proposed methods output using truth information and find that the cINN is capable

of generating meaningful posterior distributions in the multi-dimensional parton-level phase space. Finally, we illustrate in Sec. 2.5 how the inversion can link two phase spaces with different dimension. This allows us to unfold based on a model with a variable number of final state particles at the detector level and is crucial to include higher-order perturbative corrections. We show how the cINN can account for jet radiation and unfolds it together with the detector effects.

2.2 Unfolding basics

Unfolding particle physics events is a classic example for an inverse problem [106–108]. In the limit where detector effects can be described by Gaussian noise, it is similar to unblurring images. However, actual detector effects depend on the individual objects, the global energy deposition per event, and the proximity of objects, which means they are much more complicated than Gaussian noise. The situation gets more complicated when we add effects like QCD jet radiation, where the radiation pattern depends for instance on the quantum numbers of the incoming partons and on the energy scale of the hard process.

What we do know is that we can describe the measurement of phase space detector-level distributions $\mathrm{d}\sigma/\mathrm{d}x_d$ as a random process. This means, in turn, that estimating parton-level distributions $\mathrm{d}\sigma/\mathrm{d}x_p$ should also be treated as a problem of statistical inversion.

2.2.1 Binned toy model and locality

As a one-dimensional toy example we can look at a binned (parton-level) distribution $\sigma_j^{(p)}$ which gets transformed into another binned (detector-level) distribution $\sigma_j^{(d)}$ by the kernel or response function g_{ij},

$$\sigma_i^{(d)} = \sum_{j=1}^{N} g_{ij} \sigma_j^{(p)} \; . \tag{2.1}$$

We can postulate the existence of an inversion with the kernel $\bar{g}$ through the relation

$$\sigma_k^{(p)} = \sum_{i=1}^{N} \bar{g}_{ki} \sigma_i^{(d)} = \sum_{j=1}^{N} \left(\sum_{i=1}^{N} \bar{g}_{ki} g_{ij} \right) \sigma_j^{(p)} \quad \text{with} \quad \sum_{i=1}^{N} \bar{g}_{ki} g_{ij} = \delta_{kj} \; . \tag{2.2}$$

If we assume that we know the N^2 entries of the kernel g, this form gives us the N^2 conditions to compute its inverse $\bar{g}$. We illustrate this one-dimensional binned case with a semi-realistic smearing matrix

$$g = \begin{pmatrix} 1-x & x & 0 \\ x & 1-2x & x \\ 0 & x & 1-x \end{pmatrix} \; . \tag{2.3}$$

We illustrate the smearing pattern with two input vectors, keeping in mind that in an unfolding problem we typically only have one kinematic distribution to determine the inverse matrix $\bar{g}$,

$$\begin{aligned} \sigma^{(p)} = n \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad &\Rightarrow \quad \sigma^{(d)} = \sigma^{(p)} \; , \\ \sigma^{(p)} = \begin{pmatrix} 1 \\ n \\ 0 \end{pmatrix} \quad &\Rightarrow \quad \sigma^{(d)} = \sigma^{(p)} + x \begin{pmatrix} n-1 \\ -2n+1 \\ n \end{pmatrix} \; . \end{aligned} \tag{2.4}$$

The first example shows how for a symmetric smearing matrix a flat distribution removes all information about the detector effects. This implies that we might end up with a choice of reference process

and phase space such that we cannot extract the detector effects from the available data. The second example illustrates that for bin migration from a dominant peak the information from the original $\sigma^{(p)}$ gets overwhelmed easily. We can also compute the inverse of the smearing matrix in Eq.(2.3) and find

$$\bar{g} \approx \frac{1}{1-4x} \begin{pmatrix} 1-3x & -x & x^2 \\ -x & 1-2x & -x \\ x^2 & -x & 1-3x \end{pmatrix} , \tag{2.5}$$

where we neglect the sub-leading x^2-terms whenever there is a linear term as well. We see from Eq. 2.5 that even in an idealized toy model, in which the inverse $\bar{g}$ does exist and can be explicitly computed, a local transfer matrix leads to a global unfolding matrix, though the most off-diagonal entries are suppressed.

2.2.2 Bayes' theorem and model dependence

Over the continuous phase space a detector simulation can be written as

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x_d} = \int \mathrm{d}x_p \; g(x_d, x_p) \; \frac{\mathrm{d}\sigma}{\mathrm{d}x_p} , \tag{2.6}$$

where x_d is a kinematic variable at detector level, x_p the same variable at parton level, and g a kernel or transfer function which links these two arguments. We ignore efficiency factors for now, because they can be absorbed into the parton-level rate. To invert the detector simulation we define a second transfer function $\bar{g}$ such that [109–111]

$$\frac{\mathrm{d}\sigma}{dx_p} = \int \mathrm{d}x_d \; \bar{g}(x_p, x_d) \; \frac{\mathrm{d}\sigma}{\mathrm{d}x_d} = \int \mathrm{d}x'_p \; \frac{\mathrm{d}\sigma}{dx'_p} \int \mathrm{d}x_d \; \bar{g}(x_p, x_d) g(x_d, x'_p) . \tag{2.7}$$

This inversion is fulfilled if we construct the inverse $\bar{g}$ of g defined by

$$\int \mathrm{d}x_d \; \bar{g}(x_p, x_d) g(x_d, x'_p) = \delta(x_p - x'_p) , \tag{2.8}$$

all in complete analogy to the binned form above. The symmetric form of Eq.(2.6) and Eq.(2.7) indicates that g and $\bar{g}$ are both defined as distributions. In the g-direction we use Monte Carlo simulation and sample in x_p, while $\bar{g}$ needs to be sampled in $g(x_p)$ or x_d. In both directions this statistical nature implies that we should only attempt to unfold sufficiently large event samples.

The above definitions can be linked to Bayes' theorem if we identify the kernels with probabilities. We now look at $\bar{g}(x_d|x_p)$ in the slightly modified notation as the probability of observing x_d given the model prediction x_p and $g(x_p|x_d)$ gives the probability of the model x_p being true given the observation x_d [112, 113]. In this language Eq.(2.6) and (2.7) describe conditional probabilities, and we can write something analogous to Bayes' theorem,

$$\bar{g}(x_p|x_d) \; \frac{\mathrm{d}\sigma}{\mathrm{d}x_d} \sim g(x_d|x_p) \; \frac{\mathrm{d}\sigma}{\mathrm{d}x_p} . \tag{2.9}$$

In this form $\bar{g}(x_p|x_d)$ is the posterior, $g(x_d|x_p)$ as a function of x_p is the likelihood, $\mathrm{d}\sigma/\mathrm{d}x_p$ is the prior, and the model evidence $\mathrm{d}\sigma/\mathrm{d}x_d$ fixes the normalization of the posterior. From standard Bayesian analyses we know that the posterior will in general depend on the prior, in our case the kinematics of the underlying particle physics process or model.

If the posterior $\bar{g}(x_p|x_d)$ in general depends on the model $\mathrm{d}\sigma/\mathrm{d}x_p$, then Eq.(2.7) does not look useful. On the other hand, Bayesian statistics is based on the assumption that the prior dependence of the posterior defines an iterative process where we start from a very general prior and enter likelihood information step by step to finally converge on the posterior. The same approach can define a kinematic unfolding algorithm [74]. We will not discuss these methods further, but come back to this model dependence throughout the chapter.

2.2.3 Reference process

As a benchmarking process we consider

$$pp \to ZW^{\pm} \to (\ell^-\ell^+)\,(jj)\,, \tag{2.10}$$

One of the contributing Feynman diagrams is shown in Fig. 2.1. Having jets and leptons in the final state, we can test the performances of the model for both small and large detector effects. We generate the ZW events using MADGRAPH [6] without generation cuts and then simulate parton showering with PYTHIA8 [114] and the detector effects with DELPHES [5] using the standard ATLAS card. For jet clustering we use the anti-k_T algorithm [115] with $R = 0.6$ implemented in FASTJET [116]. All jets are required to have

$$p_{T,j} > 25 \text{ GeV} \qquad \text{and} \qquad |\eta_j| < 2.5\,. \tag{2.11}$$

For the hadronically decaying W-boson the limited calorimeter resolution will completely dominate over the parton-level Breit-Wigner distribution. The absolute dimension of the dataset used will be specified at every section, and data will always be split into 90% training and 10% test data.

Initially, we design a minimal set-up which allows us to directly relate the outcome of the hard process to the detector-level observables. This is achieved by having a fixed number of reconstructed elements at the detector level, matching exactly the number of parton level leptons and qcd jets, as well as by switching off initial state radiation. For the data simulation, this means that we need to switch off initial state radiation and select only events with exactly two jets and a pair of same-flavour opposite-sign leptons. In this simplified set-up reconstructed jets and partons are matched by their angular distance, while the detector and parton level leptons are directly assigned by charge. This gives us two samples matched event by event, one at the parton level (x_p) and one including detector effects (x_d). Each of them is given as an unweighted set of four 4-vectors. These 4-vectors can be simplified if we assume all external particles at the parton level to be on-shell. This method can easily accommodate weighted events.

In a second step we relax the constraints allowing both initial state radiation and a variable number of detector-level jets. We still require a pair of same-flavor opposite-sign leptons and at least two jets in agreement with the condition in Eq.(2.11). The four jets with highest p_T are then used as input to the network, ordered by p_T. Events with less than 4 jets are zero-padded. This more challanging set-up is only used for the conditional INN in Sec. 2.4.3.

This does not mean that in a possible application we take a parton shower at face value. All we do is assume that there is a correspondence between a hard parton and its hadronic final state, and that the parton 4-momentum can be reconstructed with the help of a jet algorithm. The question if for instance an anti-k_T algorithm is an appropriate description of sub-jet physics does not arise as long as the jet algorithm reproduces the hard parton momentum.

2.3 GAN unfolding

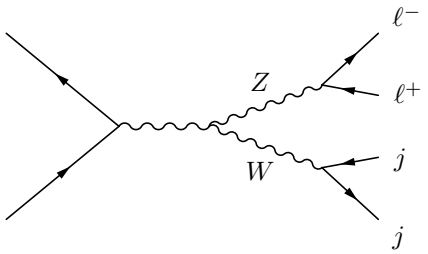

Figure 2.1: Sample Feynman diagram contributing to WZ production, with intermediate on-shell particles labelled.

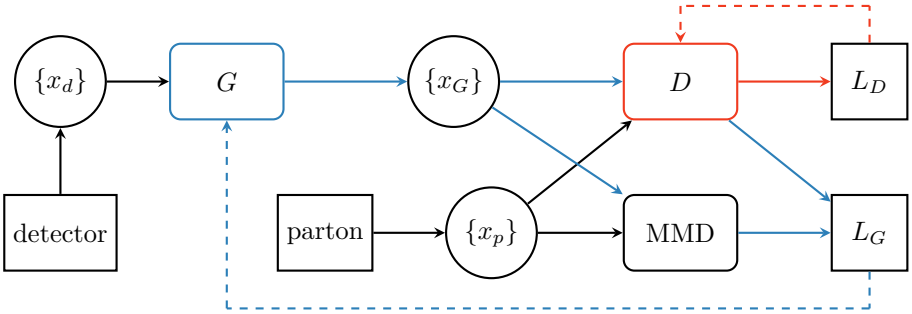

Figure 2.2: Structure of a naive unfolding GAN. The input $\{x_d\}$ describes a batch of events sampled at detector level and $\{x_{G,p}\}$ denotes events sampled from the generator or parton-level data. The blue (red) arrows indicate which connections are used in the training of the generator (discriminator).

2.3.1 Naive GAN

A standard method for fast detector simulation is smearing the outgoing particle momenta with a detector response function. This allows us to generate and sample from a probability distribution of smeared final-state momenta for a single parton-level event. For the inversion we need to rely on event samples, as we can see from a simple example: we start from a sharp Z-peak at the parton level and broaden it with detector effects. Now we look at a detector-level event in the tail and invert the detector simulations, for which we need to know in which direction in the invariant mass the preferred value m_Z lies. This implies that unfolding detector effects requires a model hypothesis, which can be thought of as a condition in a probability of the inversion from the detector level. The problem with this point of view is that the parton-level distribution of the invariant mass requires a dynamic reconstruction of the Breit-Wigner peak, which is not easily combined with a Markov process. In any case, from this argument it is clear that unfolding only makes sense at the level of large enough event samples.

Our GAN follows closely the description given in Sec. 1.2, and consists of a generator network G competing against a discriminator network D in a min-max game, as illustrated in Fig. 2.2. For the implementation we use Keras (v2.2.4) [117] with a TensorFlow (v1.14) backend [118]. As the starting point, G is randomly initialized to produce an output, with the same dimensionality as the target space. As the first toy set-up, we feed the generator samples $\{x_d\}$ of detector events as an input, i.e. $G(\{x_d\}) = \{x_G\}$. The discriminator is instead given batches $\{x_G\}$ and $\{x_p\}$ sampled from P_G and the parton-level target distribution P_p, where P_G is the distribution induced by the generator over the target space. The discriminator is trained as a binary classifier, such that $D(x \in \{x_p\}) = 1$ and $D(x) = 0$ otherwise. Following the conventions of Ref. [88] the discriminator loss function is defined as

$$L_D = \langle -\log D(x) \rangle_{x \sim P_p} + \langle -\log (1 - D(x)) \rangle_{x \sim P_G} \ . \tag{2.12}$$

We add a regularization and obtain the regularized Jensen-Shannon GAN loss function [119]

$$L_D^{(\text{reg})} = L_D + \lambda_D \left\langle (1 - D(x))^2 \, |\nabla \phi|^2 \right\rangle_{x \sim P_p} + \lambda_D \left\langle D(x)^2 \, |\nabla \phi|^2 \right\rangle_{x \sim P_G} , \tag{2.13}$$

with a properly chosen pre-factor λ_D and where we define $\phi(x) = \log \frac{D(x)}{1-D(x)}$. The discriminator training at fixed P_p and P_G alternates with the generator training, which is instead trained to maximize the second term in Eq.(2.12) using the truth encoded in D. This can be written as

$$L_G = \langle -\log D(x) \rangle_{x \sim P_G} \ . \tag{2.14}$$

If the training of the generator and the discriminator with their respective losses Eq.(2.13) and Eq.(2.14) is properly balanced, the distribution P_G converges to the parton-level distribution P_p, while the discriminator unable to distinguish between real and generated samples.

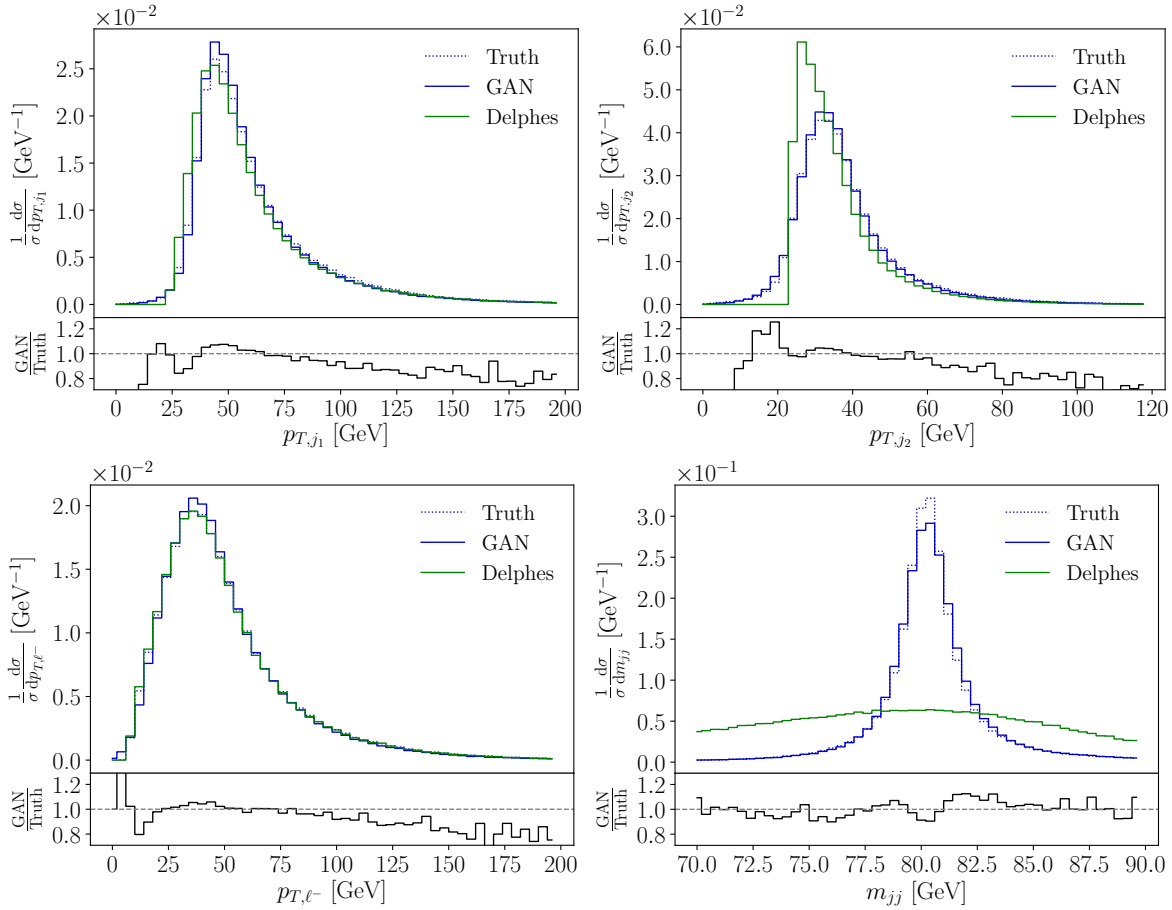

Figure 2.3: Example distributions for parton level truth, after detector simulation, and GANned back to parton level. The lower panels give the ratio of parton level truth and reconstructed parton level.

If we want to describe particularly sharp phase space features, such as invariant masses of unstable resonances, it can be useful to add a maximum mean discrepancy (MMD) [120] contribution to the loss function. It allows us to compare pre-defined distributions, for instance the one-dimensional invariant mass of an intermediate particle. Given batches of true and generated parton-level events we define the additional contribution to the generator loss as

$$\mathrm{MMD} = \left[\langle k(x,x')\rangle_{x,x'\sim P_G} + \langle k(y,y')\rangle_{y,y'\sim P_p} - 2\langle k(x,y)\rangle_{x\sim P_G, y\sim P_p}\right]^{1/2} , \tag{2.15}$$

with another pre-factor λ_G. Note that we use MMD instead of MMD^2 to enhance the sensitivity of the model [121]. In Ref. [88] we have compared common choices, like Gaussian or Breit-Wigner kernels with a given width σ,

$$k_{\mathrm{Gauss}}(x,y) = \exp\frac{-(x-y)^2}{2\sigma^2} \qquad \text{or} \qquad k_{\mathrm{BW}}(x,y) = \frac{\sigma^2}{(x-y)^2+\sigma^2} . \tag{2.16}$$

As a naive approach to GAN inversion, we use detector-level event samples as generator input. The network input is always a set of four 4-vectors, one for each particle in the final state, with their masses fixed. In practice, the GAN is trained to map detector-level events to parton-level events. Both networks consist of 12 layers with 512 units per layer. With $\lambda_G = 1$, $\lambda_D = 10^{-3}$ and a batch size of 512 events, we run for 1200 epochs and 500 iterations per epoch.

For all GAN trainings we use a dataset of 300k events. While the smearing of the lepton momenta is modest, the observed widths of the hadronically decaying W-boson will be much larger than the

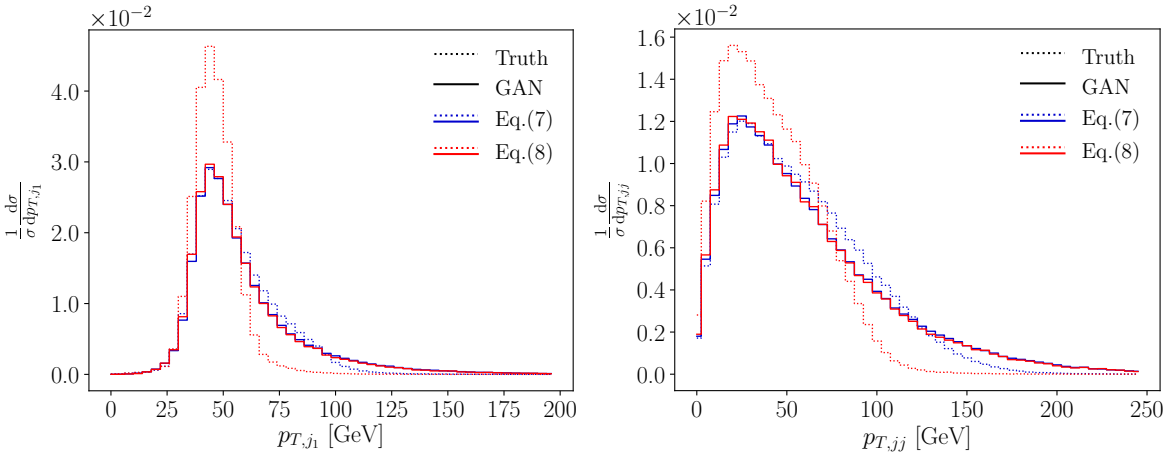

Figure 2.4: Parton level truth and GANned distributions when we train the GAN on the full data set but only unfold parts of phase space defined in Eq.(7) and Eq.(8).

parton-level Breit-Wigner distribution. For this reason, we focus on showing hadronic observables to benchmark the performance of our set-up.

In Fig. 2.3 we compare true parton-level events to the GAN's output. We run the GAN on a set of statistically independent, but simulation-wise identical sets of detector-level events. Both, the relatively flat p_{T,j_1} and the peaked m_{jj} distributions agree well between the true parton-level events and the GAN-inverted sample, indicating that the model is able to reproduce the parton level observables correctly.

This first approach only serves as a check that the training procedure of the GAN leads to the correct outcome, as it suffers from severe limitations. First of all, we wish to invert the detector simulation stochastically, i.e. we want the full posterior distribution of possible parton level events associated to a single detector level event. As this setting does not accommodate any random input, and the generator itself is a deterministic map, this is not possible. Secondly, the detector information is completely ignored by the model, as in the training batches of parton and detector get shuffled independently, and there's therefore no connection between each them. In order to illustrate this second point better, we invert an event sample which is not statistically equivalent to the training data, specifically by testing the GAN on data covering only part of the detector-level phase space. We apply the two sets of jet cuts

$$\text{Cut I}: \quad p_{T,j_1} = 30 \ldots 100 \text{ GeV} \tag{7}$$

$$\text{Cut II}: \quad p_{T,j_1} = 30 \ldots 60 \text{ GeV} \quad \text{and} \quad p_{T,j_2} = 30 \ldots 50 \text{ GeV}\,, \tag{8}$$

which leave us with 88% and 38% of events, respectively. This approach ensures that the training has access to the full information, while the test sample is a significantly reduced sub-set of the full sample.

In Fig. 2.4 we show a set of kinematic distributions with detector cuts propagated to the parton level. As before, we compare the original parton-level shapes of the distributions with the results from inverting the fast detector simulation. We see that while there's a strong correlation in the paired dataset used for the training, the model is completely insensitive to it, and simply reproduces the full parton level information. This is clear indications that the naive GAN approach does not exploit the fact that parton and detector events are paired. We discuss a solution in the next section.

2.3.2 Conditional GAN

The issues highlighted in the previous section can be simultaneously solved by switching to a conditional GAN set-up [41], which we show in Fig. 2.2. First of all, we recover the physical intuition that this entire mapping is statistical in nature by feeding the generator random numbers $\{z\}$ sampled from a

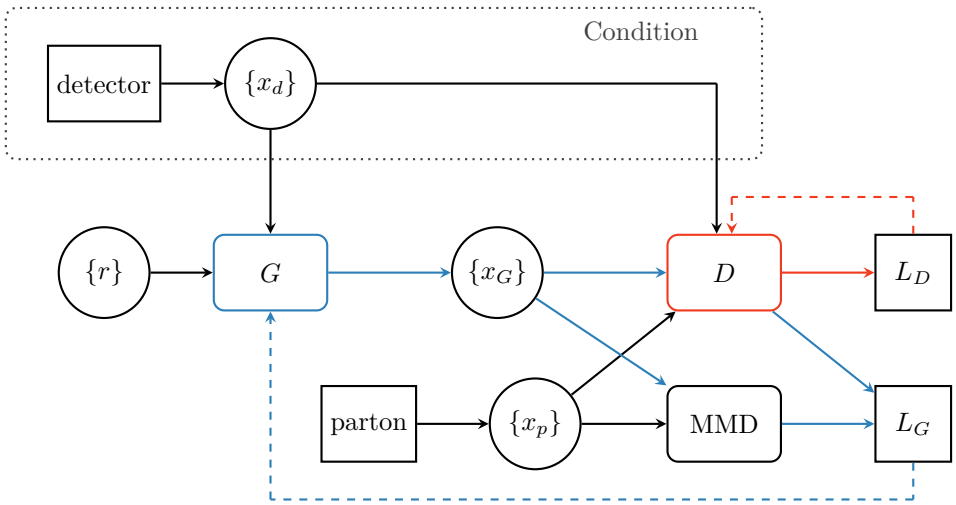

Figure 2.5: Structure of our fully conditional FCGAN. The input $\{r\}$ describes a batch of random numbers and $\{x_{G,d,p}\}$ denotes events sampled from the generator, detector-level data, or parton-level data. The blue (red) arrows indicate which connections are used in the training of the generator (discriminator).

simple, fixed distribution, in our case a multivariate Gaussian with zero mean and the identity as the covariance matrix. The input is taken with the same dimensionality as the target space. Secondly, the detector-level information $\{x_d\}$ is used as an event-by-event conditional input on the link between a set of random numbers and the parton-level output, i.e. $G(\{r\},\{x_d\}) = \{x_G\}$. This way the conditional model can generate parton-level events from random noise but still using the detector-level information as input.

In Fig. 2.5 we introduce a fully conditional GAN (FCGAN). While the training procedure is identical, we need to modify the objective function as

$$L_D \to L_D^{(\mathrm{FC})} = \langle -\log D(x,y)\rangle_{x\sim P_p, y\sim P_d} + \langle -\log\left(1-D(x,y)\right)\rangle_{x\sim P_G, y\sim P_d} \ , \tag{2.17}$$

and the regularized loss function changes accordingly,

$$\begin{aligned} L_D^{(\mathrm{reg})} \to L_D^{(\mathrm{reg,\,FC})} = L_D^{(\mathrm{FC})} &+ \lambda_D \left\langle \left(1-D(x,y)\right)^2 |\nabla\phi|^2 \right\rangle_{x\sim P_p, y\sim P_d} \\ &+ \lambda_D \left\langle D(x,y)^2 \ |\nabla\phi|^2 \right\rangle_{x\sim P_G, y\sim P_d} , \end{aligned} \tag{2.18}$$

again using the conventions of Ref. [88]. The generator loss function now takes the form

$$L_G \to L_G^{(\mathrm{FC})} = \langle -\log D(x,y)\rangle_{x\sim P_G, y\sim P_d} \ . \tag{2.19}$$

Note, that we do not build a conditional version of the MMD loss. The hyper-parameters of our FCGAN are summarized in Tab. 2.1. Changing from a naive GAN to a conditional GAN we have to

Parameter	Value	Parameter	Value
Layers	12	Batch size	512
Units per layer	512	Epochs	1200
Trainable weights G	3M	Iterations per epoch	500
Trainable weights D	3M	Number of training events	3×10^5
λ_G	1		
λ_D	10^{-3}		

Table 2.1: FCGAN setup.

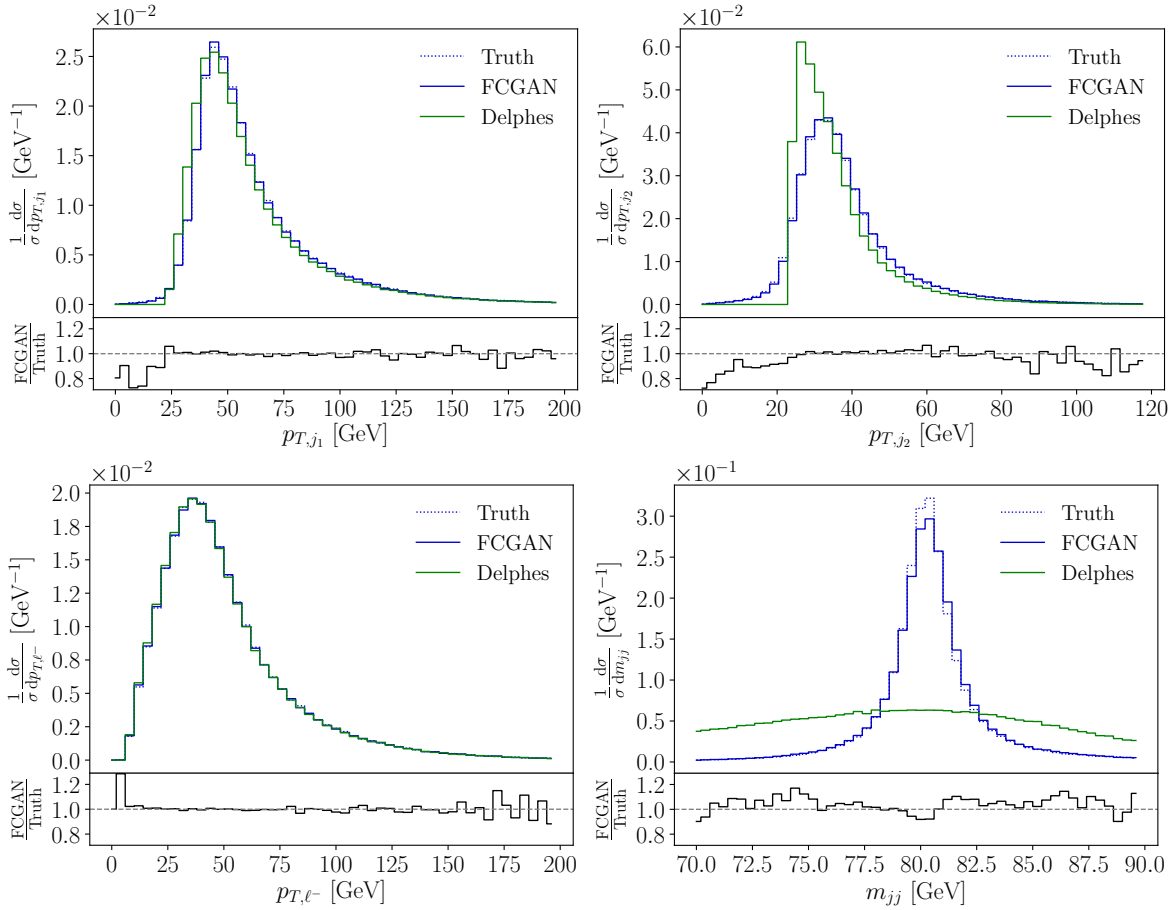

Figure 2.6: Example distributions for parton level truth, after detector simulation, and FCGANned back to parton level. The lower panels give the ratio of parton level truth and reconstructed parton level. The lower panels give the deviation between parton level truth and reconstructed parton level. To be compared with the naive GAN results in Fig. 2.3.

pay a price in the structure of the training sample. While the naive GAN only required event batches to be matched between parton level and detector level, the training of the FCGAN actually requires event-by-event matching.

In Fig. 2.6 we compare the truth and the generated events, trained on and applied to events covering the full phase space. Compared to the naive GAN, inverting the detector effects now works even better. The under-estimate of the GAN rate in tails no longer occurs for the FCGAN. The reconstructed invariant W-mass forces the network to dynamically generate a very narrow physical width from a comparably broad Gaussian peak. Using our usual MMD loss developed in Ref. [88] we reproduce the peak position, width, and peak shape to about 90%. We emphasize that the MMD loss requires us to specify the relevant one-dimensional distribution, in this case m_{jj}, but it then extracts the on-shell mass or width dynamically. The multi-kernel approach we use in this case is explained in the Appendix.

As for our naive ansatz we now test what happens to the network when the training data and the test data do not cover the same phase space region. We train using the entire phase space, but we then only invert to the 88% and 38% of events passing the jet cuts I and II defined in Eq.(7) and Eq.(8). We show the results in Fig. 2.7. As observed before, especially the jet cuts with only 40% survival probability shape our four example distributions. However, we see for example in the $p_{T,jj}$ distribution that the inverted detector-level sample reconstructs the patterns of the true parton-level events perfectly. This comparison indicates that the FCGAN approach deals with differences in the training and test samples very well.

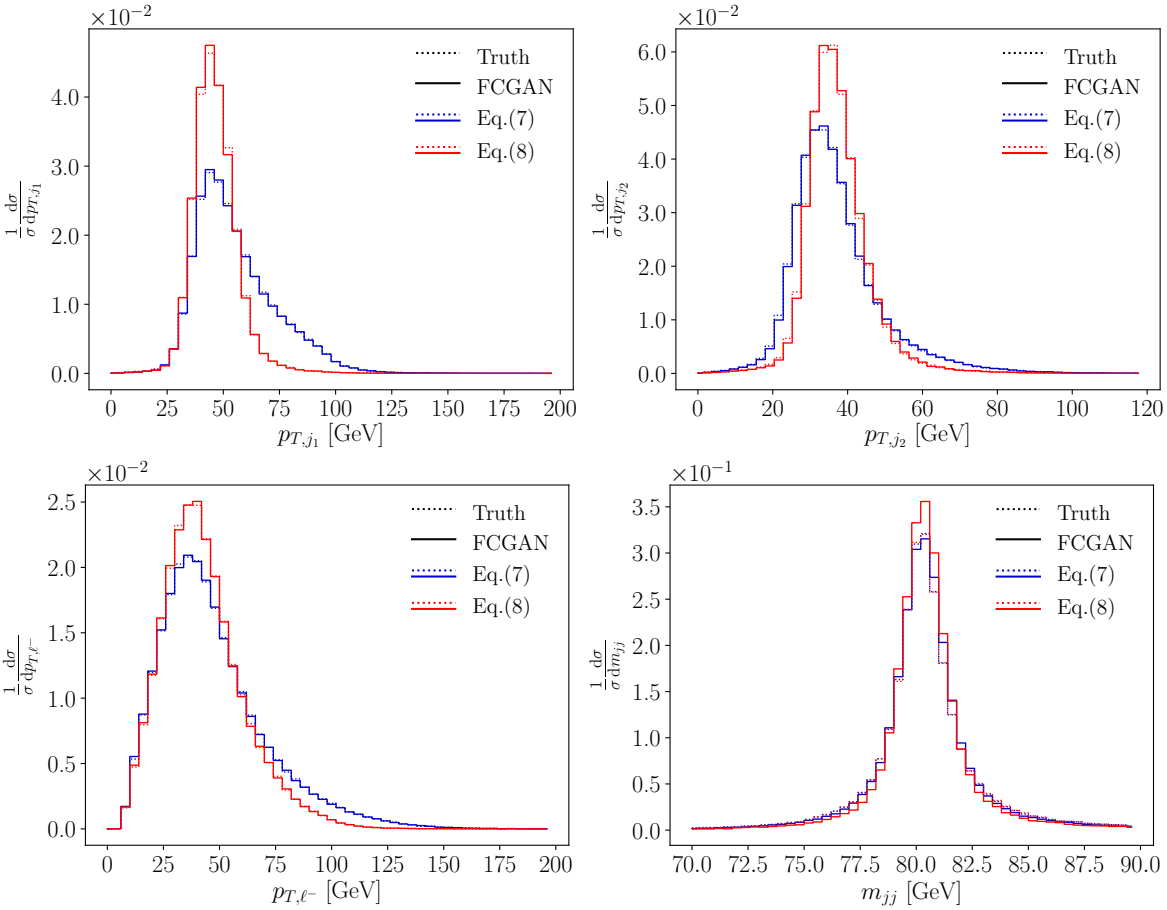

Figure 2.7: Parton level truth and FCGANned distributions when we train the GAN on the full data set but only unfold parts of phase space defined in Eq.(7) and Eq.(8). To be compared with the naive GAN results in Fig.2.4.

In order to test to which degree the inversion holds, we move on to harsher cuts on the inclusive event sample. We start with

$$\text{Cut III}: \quad p_{T,j_1} = 30 \ldots 50 \text{ GeV} \quad p_{T,j_2} = 30 \ldots 40 \text{ GeV} \quad p_{T,\ell^-} = 20 \ldots 50 \text{ GeV} , \tag{12}$$

which 14% of all events pass. In Fig. 2.8 we see that also for this much reduced fraction of test events corresponding to the training sample the FCGAN inversion reproduces the true distributions extremely well, to a level where it appears not really relevant what fraction of the training and test data correspond to each other.

Finally, we apply a cut which not only removes a large fraction of events, but also cuts into the leading peak feature of the p_{T,j_1} distribution and removes one of the side bands needed for an interpolation,

$$\text{Cut IV}: \quad p_{T,j_1} > 60 \text{ GeV} . \tag{13}$$

For this choice 39% of all events pass, but we remove all events at low transverse momentum, as can be seen from Fig. 2.6. This kind of cut could therefore be expected to break the unfolding. Indeed, the red lines in Fig. 2.8 indicate that we have broken the m_{jj} reconstruction through the FCGAN. However, all other (shown) distributions still agree with the parton-level truth extremely well. The problem with the invariant mass distribution is that our implementation of the MMD loss is not actually conditional. At this stage it means that, when pushed towards its limits, the network will first fail to reproduce the correct peak width in the m_{jj} distribution, while all other kinematic variables remain stable.

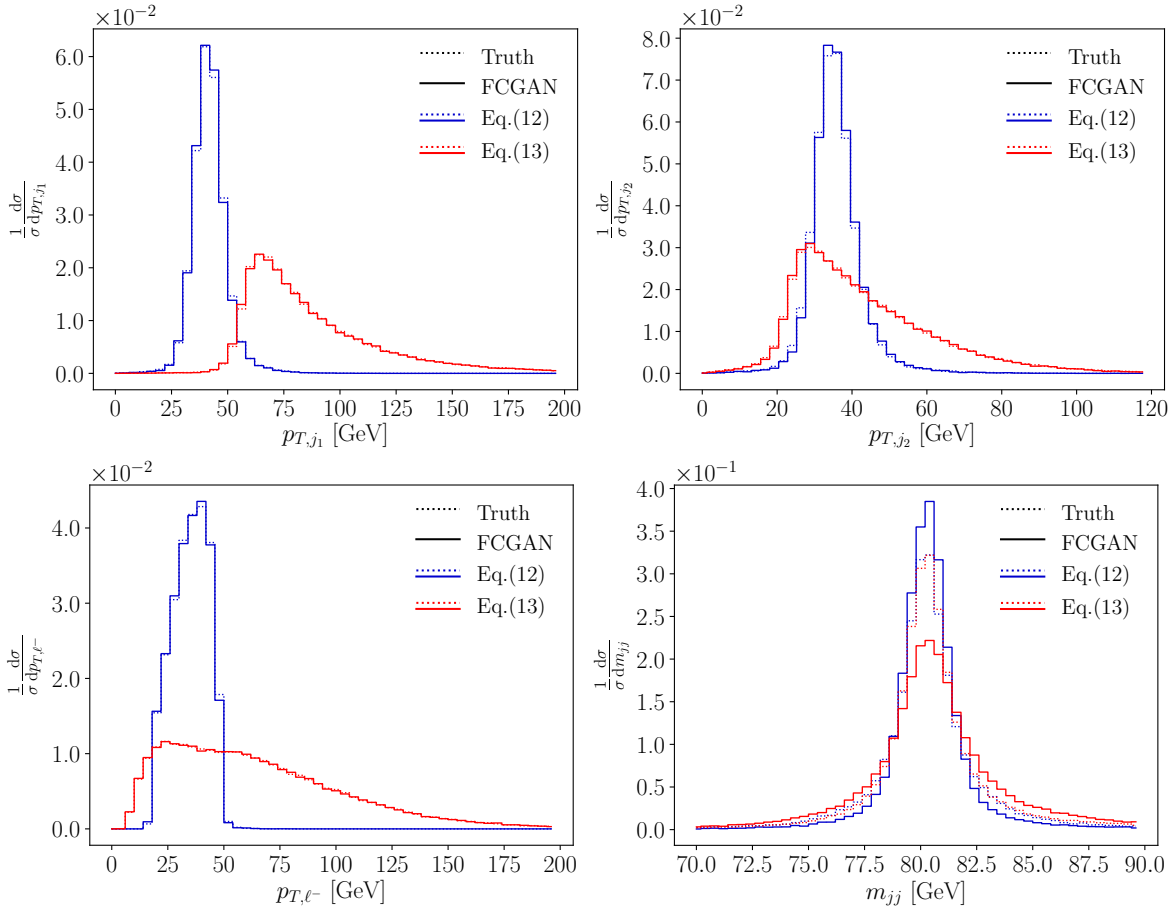

Figure 2.8: Parton level truth and FCGANned distributions when we train the GAN on the full data set but only unfold parts of phase space defined in Eqs.(12) and (13).

Finally, just like in Ref. [88] we show 2-dimensional correlations in Fig. 2.9. We stick to applying the network to the full phase space and show the parton level truth and the FCGAN-inverted events in the two upper panels. Again, we see that the FCGAN reproduces all features of the parton level truth with high precision. The bin-wise relative deviation between the two 2-dimensional distributions only becomes large for small values of E_{j_1}, where the number of training events is extremely small.

2.3.3 New physics injection

As discussed before, unfolding to a hard process is necessarily model-dependent. Until now, we have always assumed the Standard Model to correctly describe the parton-level and detector-level events. An obvious question is what happens if we train our FCGAN on Standard Model data, but apply it to a different hypothesis. This challenge becomes especially interesting if this alternative hypothesis differs from the Standard Model in a local phase space effect. It then allows us to test if the generator networks maps the parton-level and detector-level phase spaces in a structured manner. Such features of neural networks are at the heart of all variational constructions, for instance variational autoencoders which are structurally close to GANs.

To this end we add a fraction of resonant W' events from a triplet extension of the Standard Model [122], representing the hard process

$$pp \to W'^* \to ZW^\pm \to (\ell^-\ell^+)\,(jj) \tag{2.20}$$

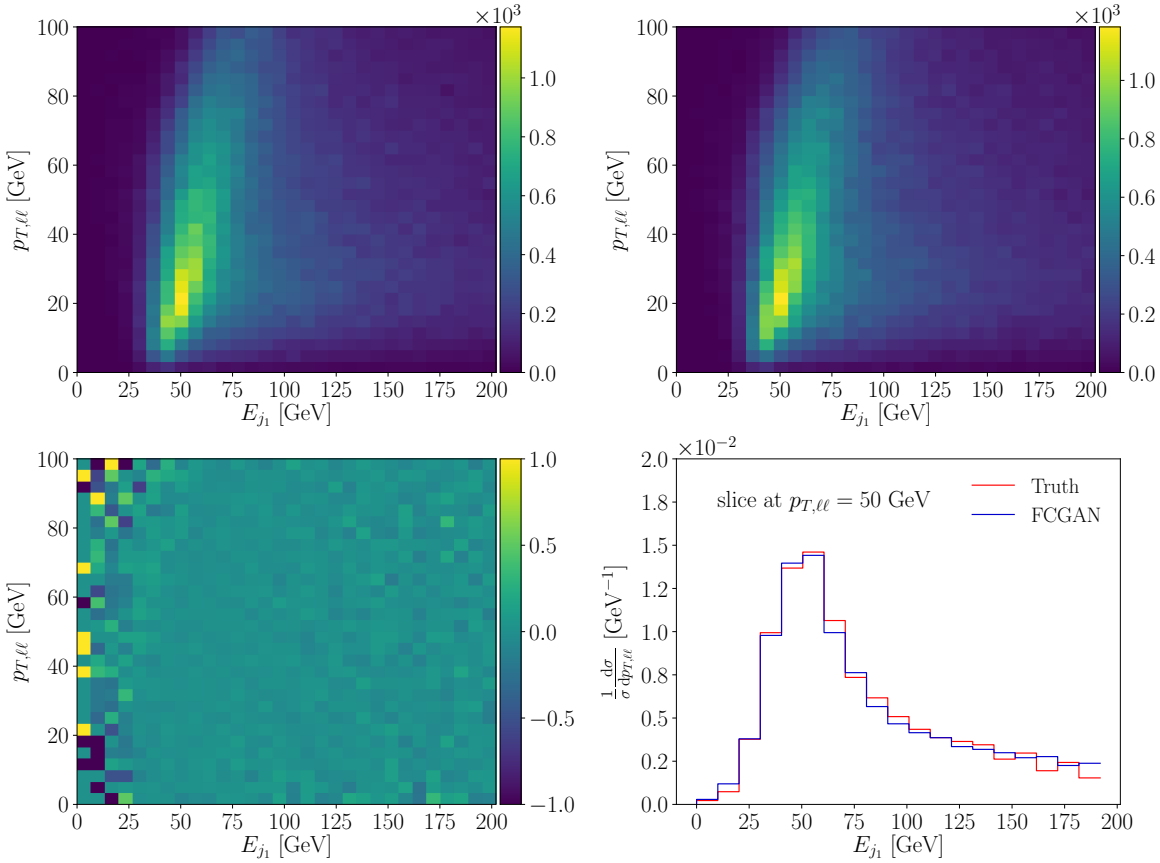

Figure 2.9: Two-dimensional parton level truth (upper left) and FCGANned (upper right) distributions when we train the GAN on the full data set and unfold over the full phase space. The lower panels show the relative deviation between truth and FCGANned and the one-dimensional E_{j_1} distribution along fixed $p_{T,\ell\ell}$.

to the test data. We simulate these events with madgraph using the model implementation of Ref. [123] and denote the new massive charged vector boson with a mass of 1.3 TeV and a width of 15 GeV as W'. For the test sample we combine the usual Standard Model sample with the W'-sample in proportions $90\% - 10\%$. The other new particles do not appear in our process to leading order. Because we want to test how well the GAN maps local phase space structures onto each other, we deliberately choose a small width $\Gamma_{W'}/M_{W'} \sim 1\%$, not exactly typical for such strongly interacting triplet extensions.

The results for this test are shown in Fig. 2.10. First, we look at transverse momentum distribution of final-state particles, which are hardly affected by the new heavy resonance. Both, the leading jet and the lepton distributions are essentially identical for both truth levels and the FCGAN output. The same is true for the invariant mass of the hadronically decaying W-boson, which nevertheless provides a useful test of the stability of our training and testing.

Finally, we show the reconstructed W'-mass in the lower-right pane. Here we see the different (normalized) truth-level distributions for the Standard Model and the W'-injected sample. The FCGAN, trained on the Standard Model, keeps track of local phase space structures and reproduces the W' peak faithfully. It also learn the W'-mass as the central peak position very well. The only issue is the W'-width, which the network over-estimates. However, we know already that dynamically generated width distributions are a challenge to GANs and require for instance an MMD loss. Nevertheless, Fig. 2.10 clearly shows that GAN unfolding shows a high degree of model independence, making use of local structures in the mapping between the two phase spaces. We emphasize that the additional mass peak in the FCGANned events is not a one-dimensional feature, but a localized structure in the

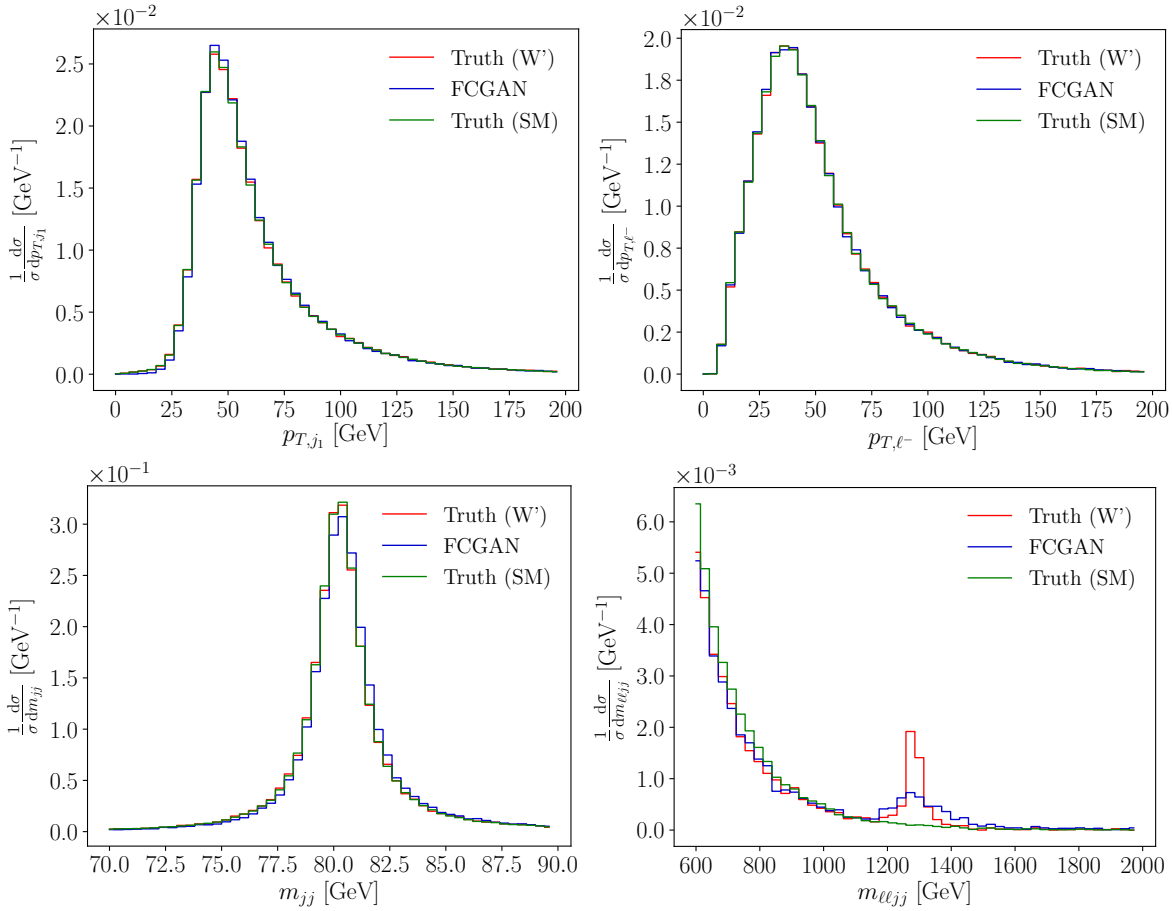

Figure 2.10: Parton level truth and FCGANned distributions when we train the network on the Standard Model only and unfold events with an injection of 10% W' events. The mass of the additional s-channel resonance is 1.3 TeV.

full phase space. This local structure is a feature of neural networks which comes in addition to the known strengths in interpolation.

2.4 INN unfolding

In Sec. 2.3 we have introduced and developed the idea of inverting Monte Carlo simulations using a GAN. However, it should be clear from Sec. 1.2.1 that an invertible model is a more natural tool to use, as the task we wish to solve consists precisely of inverting a well known forward simulation. We will show in this section how the intuition built with the conditional GAN provides us with a useful guideline.

We introduce the conditional INN in two steps, starting with the non-conditional, standard set-up. The construction of the INN we use in our analysis combines two goals [48]:

1. the mapping from input to output is invertible and the Jacobians for both directions are tractable;
2. both directions can be evaluated efficiently. This second property goes beyond some other implementations of normalizing flow [51, 53].

While the final aim is not to actually evaluate our INN in both directions, we will see that these networks can be extremely useful to invert a stochastic process like detector smearing.

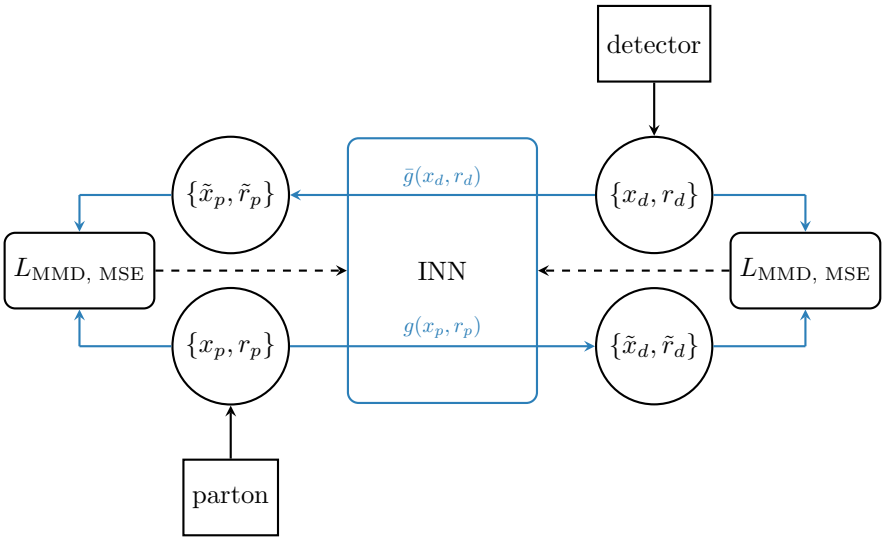

Figure 2.11: Structure of INN. The $\{x_{d,p}\}$ denote detector-level and parton-level events, $\{r_{d,p}\}$ are random numbers to match the phase space dimensionality. A tilde indicates the INN generation.

In Sec. 2.4.3 we will show how the conditional INN retains a proper statistical notion of the inversion to parton level phase space. This avoids a major weakness of standard unfolding methods, namely that they only work on large enough event samples condensed to one-dimensional or two-dimensional kinematic distributions. This could be a missing transverse energy distribution in mono-jet searches or the rapidities and transverse momenta in top pair production. To avoid systematics or biases in the full phase space coverage required by the matrix element method, the unfolding needs to construct probability distributions in parton-level phase space, including small numbers of events in tails of kinematic distributions.

2.4.1 Naive INN

While it is clear from our discussion in Sec. 2.3 that a standard INN will not serve our purpose, we still describe it in some detail before we extend it to a conditional network. Following the conventions of our GAN analysis and in analogy to Eqs.(2.6) to (2.8) we define the network input as a vector of hard process information $x_p \in R^{D_p}$ and the output at detector level via the vector $x_d \in \mathbb{R}^{D_d}$. as the INN is a change of variable, the input and output dimensionalities must be the same. If the dimensionality of the spaces are such that $D_p < D_d$ we can add a noise vector r with dimension $D_d - D_p$ to define the bijective, invertible transformation,

$$\begin{pmatrix} x_p \\ r \end{pmatrix} \xleftarrow[\leftarrow \text{ unfolding:}\bar{g}]{\text{PYTHIA,DELPHES:}g\rightarrow} x_d \; . \tag{2.21}$$

A correctly trained network g with the parameters θ then reproduces x_d from the combination x_p and r. Its inverse $\bar{g}$ instead reproduces the combination of x_p and r from x_d.

We have introduced in Sec. 1.2.1 the general features of invertible models, i.e. models that learn both directions of the bijective mapping in parallel and encodes them into one network as illustrated in Fig. 2.11 . In particular, we have explained how we can think of an INN as the composition of multiple copies of a single base function with the required properties of invertibility and cheap Jacobian. For our INN we have chosen invertible coupling layers [49, 66] based on dimensional partitioning. For notational purposes we ignore the random numbers in Eq.(2.21) and assume that this layer links an input vector x_p to an output vector x_d after splitting both of them in halves, $x_{p,i}$ and $x_{d,i}$ for $i = 1, 2$. The relation between input and output is given by a sub-network, which encodes arbitrary functions $s_{1,2}$ and $t_{1,2}$. Using an element-wise multiplication $\odot$ and sum one could for instance define an output

$x_{d,1}(x_p) = x_{p,1} \odot s_2(x_{p,2}) + t_2(x_{p,2})$. In order to avoid numerical instabilities caused by the division with $s(x)$ in the inverse direction, we include an exponential to obtain

$$\begin{pmatrix} x_{d,1} \\ x_{d,2} \end{pmatrix} = \begin{pmatrix} x_{p,1} \odot e^{s_2(x_{p,2})} + t_2(x_{p,2}) \\ x_{p,2} \odot e^{s_1(x_{d,1})} + t_1(x_{d,1}) \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} x_{p,1} \\ x_{p,2} \end{pmatrix} = \begin{pmatrix} (x_{d,1} - t_2(x_{p,2})) \odot e^{-s_2(x_{p,2})} \\ (x_{d,2} - t_1(x_{d,1})) \odot e^{-s_1(x_{d,1})} \end{pmatrix} . \tag{2.22}$$

By construction, this inversion works independent of the form of s and t. If we write the coupling block function as $g(x_p) \sim x_d$, again omitting the random numbers r, the Jacobian of the network function has a triangular form

$$\frac{\partial g(x_p)}{\partial x_p} = \begin{pmatrix} \text{diag } e^{s_2(x_{p,2})} & \text{finite} \\ 0 & \text{diag } e^{s_1(x_{d,1})} \end{pmatrix} , \tag{2.23}$$

so its determinant is easy to compute. Such coupling layer transformations define the normalizing flow, when we view it as transforming an initial probability density into a very general form of probability density through a series of invertible steps. We can relate the two probability densities as long as the Jacobians of the individual layers can be efficiently calculated.

Since the first use of the invertible coupling layer, much effort has gone into improving its efficiency. The All-in-One (AIO) coupling layer includes two features, introduced by Ref. [49] and Ref. [50]. The first modification replaces the transformation of $x_{p,2}$ by a permutation of the output of each layer. Due to the permutation each component still gets modified after passing through several layers. The second modification includes a global affine transformation to include a global bias and linear scaling that maps $x \to sx + b$. Finally, we apply a bijective soft clamping after the exponential function in Eq.(2.22) to prevent instabilities from diverging outputs.

The INN in our simplified example combines three contributions to the loss function. First, it tests if in the DELPHES direction of Eq.(2.21) we indeed find $g(x_p) = x_d$ via the mean squared error (MSE) function. While this is theoretically sufficient to obtain the inverse function, also testing the inverse direction $\bar{g}(x_d) = x_p$ greatly improves the efficiency and stability of the training. Third, to resolve special sharp features like the invariant mass of intermediate particles we use the MMD loss introduced in Sec. 2.3. For the INN architecture the Breit-Wigner kernel is the best choice to analyze the distribution of the random numbers as part of the loss function [48].

We now use the INN network to map parton-level events to detector-level events or vice-versa. In a statistical analysis we then use standard kinematic distributions and compare the respective truth and INN-inverted shapes for both directions. The left panels of Fig. 2.12 shows the transverse momentum distributions of the two jets and their invariant mass for both directions of the INN. The truth events at parton level and at detector level are marked as dashed lines. Starting from each of the truth events we can apply the INN describing the detector effects as $x_d = g(x_p)$ or unfolding the detector effects as $x_p = \bar{g}(x_d)$ in Eq.(2.21). The corresponding solid lines have to be compared to the dotted truth lines, where we need to keep in mind that at the parton level the relevant objects are quarks while at the detector level they are jets.

For the leading jet the truth and INN-generated detector-level agree very well, while for the second jet the naive INN fails to capture the hard cut imposed by the jet definition. For the invariant mass we find that the smearing due to the detector effects is reproduced well with some small deviations in the tails. In the unfolding direction both p_T distributions follow the parton level truth. The only difference is a systematic lack of events in the tail for the second quark. This is especially visible in the ratio of the INN-unfolded events and the parton-level truth, indicating that also at small p_T the network does not fill the phase space sufficiently. Combining both directions we see that in forward direction the INN produces a too broad p_T-distribution, the unfolding direction of the INN produces a too narrow distribution. The conceptual advantage of the INN actually implies a disadvantage for the inversion of particular difficult features. Finally, the invariant mass of the W is reproduced perfectly without any systematic deviation.

2.4.2 Noise-extended INN

While our simplified example in the previous section shows the potential of INNs, it fails to incorporate key aspects of the physical process. First of all, the number of degrees of freedom is not actually the

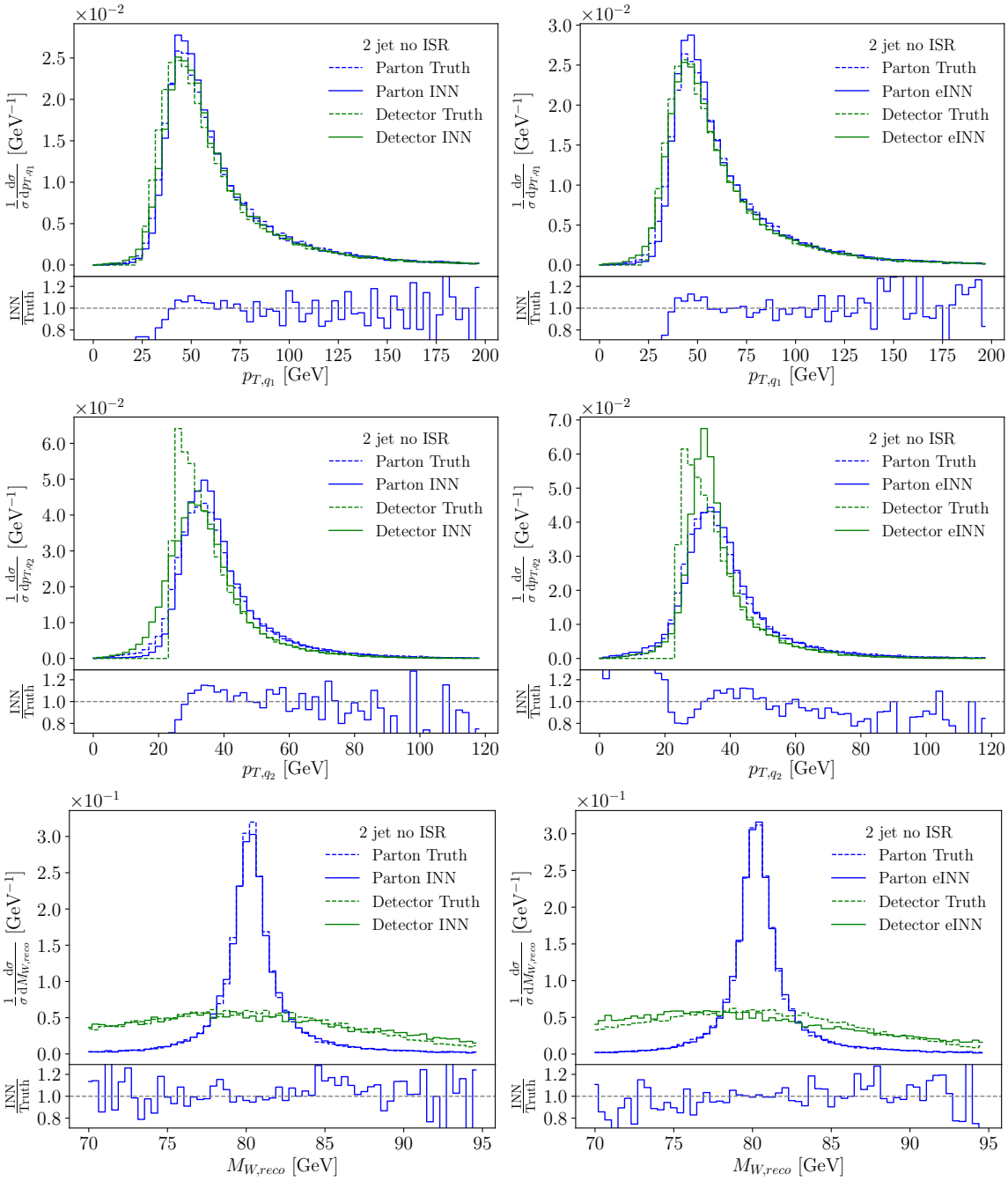

Figure 2.12: INN-generated $p_{T,q}$ and $M_{W,\text{reco}}$ distributions from a naive INN (left) and the noise-extended eINN (right). In green we compare the detector-level truth to INNed events transformed from parton level. In blue we compare the parton-level truth to INNed events transformed from detector level. The secondary panels show the ratio of INNed events over parton-level truth. More distributions can be found in the pdf files submitted to the arXiv.

same at parton level and at detector level. External partons are on their mass shell, while jets come with a range of jet masses. This mismatch becomes crucial when we include missing transverse momentum in the signature. We generally need fewer parameters to describe the partonic scattering than the detector-level process. For a fixed set of parton-level momenta we usually smear each momentum component to simulate the detector measurement. These additional degrees of freedom are of stochastic

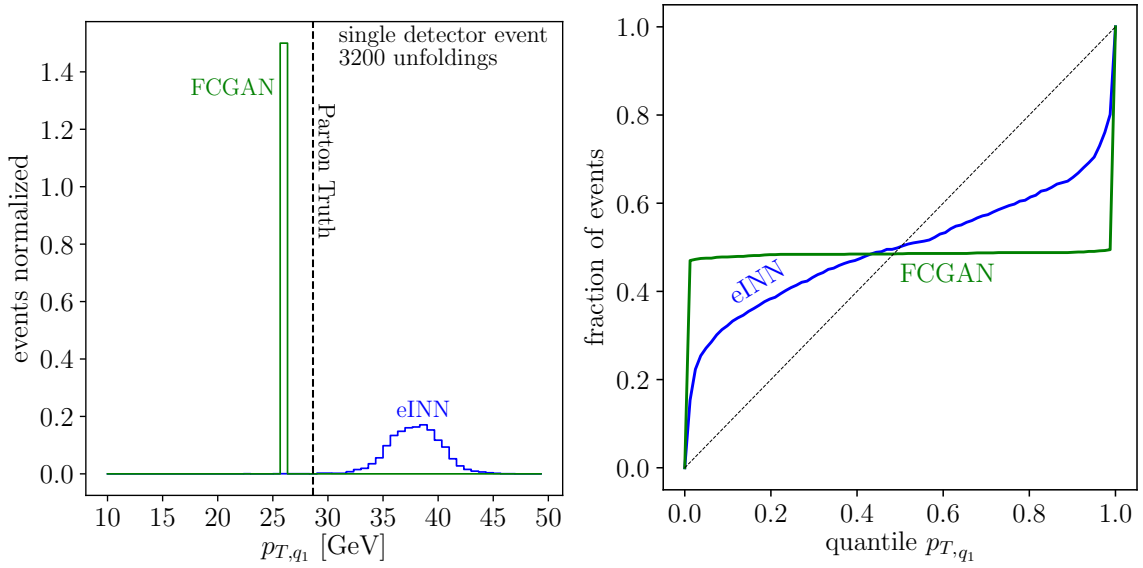

Figure 2.13: Left: illustration of the statistical interpretation of unfolded events for one event. Right: calibration curves for p_{T,q_1} extracted from the FCGAN and the noise-extended eINN.

nature, so adding Gaussian random variable on the parton side of the INN could be a first step to address this problem.

To also account for potentially unobservable degrees of freedom at the parton level we extend each side of the INN by a random number vector. The mapping in Eq.(2.21) now includes two random number vectors with dimensions $D_{r_d} = D_p$ and $D_{r_p} = D_d$,

$$\begin{pmatrix} x_p \\ r_p \end{pmatrix} \xleftarrow[\leftarrow \text{ unfolding:} \bar{g}]{\text{PYTHIA,DELPHES:} g \rightarrow} \begin{pmatrix} x_d \\ r_d \end{pmatrix} . \tag{2.24}$$

In addition, a pure MSE loss can not capture the fact that the additional noise generates a distribution of detector-level events given fixed parton momenta. It would just predict a mean value of this distribution and minimize the effect of the noise. A better solution is an MMD loss for each degree of freedom in the event and the masses of intermediate particles, as well as the Gaussian random variables. On the side of the random numbers this MMD loss ensures that they really only encode noise. Again it is beneficial for the training to use the inverse direction and apply additional MMD losses to the parton level events as well as the corresponding Gaussian inputs. Finally we add a weak MSE loss on the four vectors of each side to stabilize the training.

In the right panels of Fig. 2.12 we show results for this noise-extended INN (eINN). The generated distributions are similar to the naive INN case and match the truth at the parton level. A notable difference appears in the second jet, the weak spot of the naive INN. The additional random numbers and MMDs provide more freedom to generate the peak in the forward direction and also improve the unfolding in the low-p_T and high-p_T regimes.

The noise extension allows for a statistic interpretation of the generated distributions and a test of the integrity of the INN-inverted distributions. In the left panel of Fig. 2.13 we illustrate the goal of the statistical treatment: we start from a single event at the detector level and generate a set of unfolded events. For each of them we evaluate for instance p_{T,q_1}. Already in this illustration we see that the GAN output is lacking a statistical behaviour at the level of individual events, while the noise-extended eINN returns a reasonable distribution of unfolded events.

To see if the width of this INN output is correct we take 1500 parton-level and detector-level event pairs and unfold each event 60 times, sampling over the random variables. This gives us 1500 combinations like the one shown in the left panel of Fig. 2.13: a single parton-level truth configuration and a

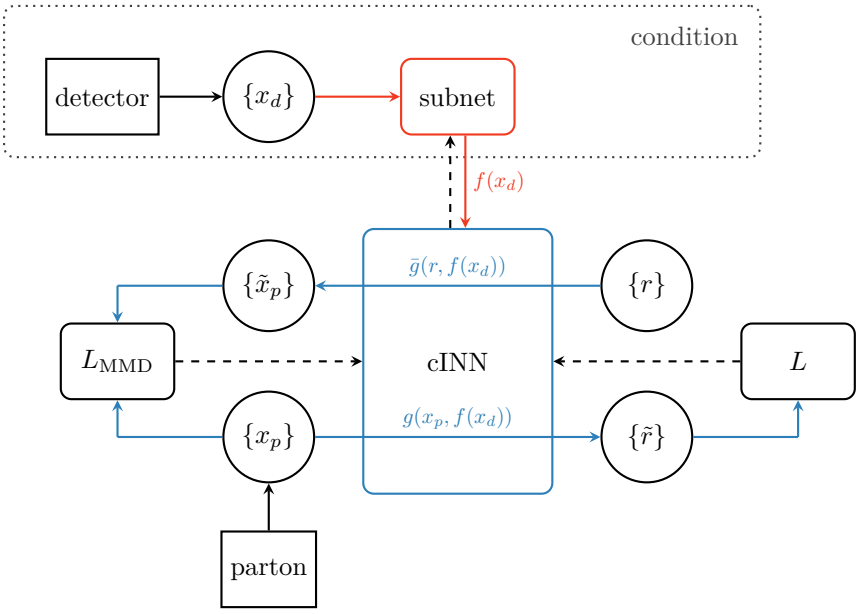

Figure 2.14: Structure of the conditional INN. The input are random numbers $\{r\}$ while $\{x_{d,p}\}$ denote detector-level and parton-level data. The latent dimension loss L follows Eq.(2.25), a tilde indicates the INN generation.

distribution of the INNed configuration. To see if the central value and the width of the INNed distribution can be interpreted statistically as a posterior probability distribution in parton phase space we analyse where the truth lies within the INN distribution for each of the 1500 events. For a correctly calibrated curve we start for instance from the left of the kinematic distribution and expect 10% of the 1500 events in the 10% quantile of the respective probability distribution, 20% of events in the 20% quantile, etc. The corresponding calibration curves for the noise-extended eINN are shown in the right panel of Fig. 2.13. While they indicate that we can attempt a statistical interpretation of the INN unfolding, the calibration is not (yet) perfect. A steep rise for the lower quantile indicates that too many events end up in the first 10% quantile. In other words, the distributions we obtain by sampling over the Gaussian noise for each event are too narrow.

While our noise-extended eINN takes several steps in the right direction, it still faces major challenges: the combination of many different loss functions is sensitive to their relative weights; the balance between MSE and MMD on event constituents has to be calibrated carefully to generate reasonable quantile distributions; when we want to extend the INN to include more detector-level information we have to include an equally large number of random variable on the parton level which makes the training very inefficient. This leads us again [28] to adopt a conditional set-up.

2.4.3 Conditional INN

If a distribution of parton-level events can be described by n degrees of freedom, we should be able to use normalizing flows or an INN to map a n-dimensional random number vector onto parton-level 4-momenta. To capture the information from the detector-level events we need to condition the INN on these events [28, 41, 57], so we link the parton-level data x_p to random noise r under the condition of x_d. Trained on a given process the network should now be able to generate probability distributions for parton-level configurations given a detector-level event and an unfolding model. We note that the cINN is still invertible in the sense that it includes a bi-directional training from Gaussian random numbers to parton-level events and back. While this bi-directional training does not represent the inversion of a detector simulation anymore, it does stabilize the training by requiring the noise to be Gaussian.

A graphic representation of this conditional INN or cINN is given in Fig. 2.14. We first process the detector-level data by a small subnet, i.e. $x_d \to f(x_d)$, to optimize its usability for the cINN [104]. The subnet is trained alongside the cINN and does not need to be reversed or adapted. We choose a shallow and wide architecture of two layers with a width of 1024 internally, because four layers degrade already the conditional information and allow the cINN to ignore it. When a deeper subnet is required we advertise to use an encoder, which is initialized by pre-training it as part of an autoencoder. We apply this technique when using the larger ISR input, where it leads to a more efficient training. After this preprocessing, the detector information is passed to the functions s_i and t_i in Eq.(2.22), which now depend on the input, the output, and on the fixed condition. Since the invertibility of the network is independent of the values of s_i and t_i, the network remains invertible between the parton-level events $\{x_p\}$ and the random variables $\{r\}$. This feature stabilizes the training. The cINN loss function is motivated by the simple argument that for the correct set of network parameters θ describing s_i and t_i we maximize the (posterior) probability $p(\theta|x_p, x_d)$ or minimize

$$\begin{aligned} L &= -\left\langle \log p(\theta|x_p, x_d) \right\rangle_{x_p \sim P_p, x_d \sim P_d} \\ &= -\left\langle \log p(x_d|x_p, \theta) + \log p(\theta|x_p) - \log p(x_d|x_p) \right\rangle_{x_p \sim P_p, x_d \sim P_d} \\ &= -\left\langle \log p(x_d|x_p, \theta) \right\rangle_{x_p \sim P_p, x_d \sim P_d} - \log p(\theta) + \text{const.} \\ &= -\left\langle \log p(g(x_p, x_d)) + \log \left| \frac{\partial g(x_p, x_d)}{\partial x_p} \right| \right\rangle_{x_p \sim P_p, x_d \sim P_d} - \log p(\theta) + \text{const.} \,, \end{aligned} \tag{2.25}$$

where we first use Bayes' theorem, then ignore all terms irrelevant for the minimization, and finally apply a simple coordinate transformation for the bijective mapping. The last term is a simple weight regularization, while the first two terms are called the maximum likelihood loss. Since we impose the latent distribution of the random variable $p(g(x_p, x_d))$ to produce a normal distribution centered around zero and with width one, the first term becomes

$$\log p(g(x_p, x_d)) = -\frac{||g(x_p, x_d))||_2^2}{2} \,. \tag{2.26}$$

The final network set-up after tuning of the hyper-parameters are listed In Tab. 2.3. We verified that the network performance is stable under small changes of these parameters.

In the left panels of Fig. 2.15 we show the unfolding performance of the cINN, trained and tested on the same exclusive 2-jet events as the simpler INNs in Fig. 2.12. Unlike the naive and the noise-extended INNs we cannot evaluate the cINN in both direction, detector simulation and unfolding, so we focus on the detector unfolding. The agreement between parton-level truth and the INN-unfolded distribution

Parameter	INN	eINN
Blocks	24	24
Layers per block	2	2
Units per layer	256	256
Trainable weights	$\sim$ 150k	$\sim$ 270k
Epochs	1000	1000
Learning rate	$8 \cdot 10^{-4}$	$8 \cdot 10^{-4}$
Batch size	512	512
Training/testing events	290k / 30k	290k / 30k
Kernel widths	$\sim 2, 8, 25, 67$	$\sim 2, 8, 25, 67$
$D_p + D_{r_p}$	$12 + 4$	$12 + 16$
$D_d + D_{r_d}$	$16 + 0$	$16 + 12$
λ_{MMD}	0.1 (masses only)	0.2
λ_{MMD} increase	-	-

Table 2.2: INN and noise-extended eINN setup and hyper-parameters, as implemented in PyTorch (v1.2.0) [124].

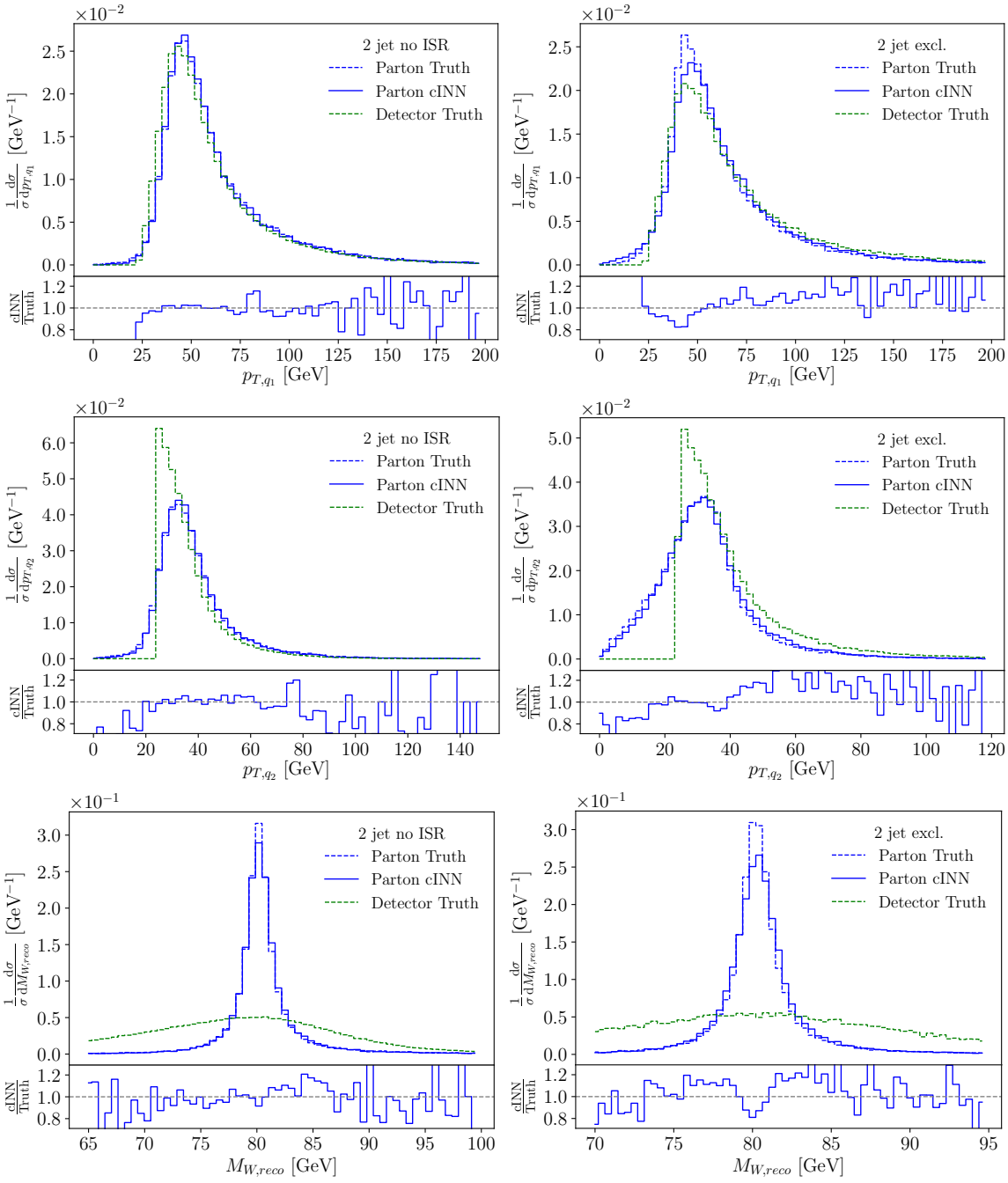

Figure 2.15: cINNed $p_{T,q}$ and $m_{W,\text{reco}}$ distributions. Training and testing events include exactly two jets. In the left panels we use a data set without ISR, while in the right panels we use the two-jet events in the full data set with ISR. The lower panels give the ratio of cINNed to parton-level truth.

is around 10% for the bulk of the p_T distributions, with the usual larger relative deviations in the tails. An interesting feature is still the cut $p_{T,j} > 20$ GeV at the detector level, because it leads to a slight shift in the peak of the p_{T,j_2} distribution. Finally, the reconstructed invariant W-mass and the physical W-width agree extremely well with the Monte Carlo truth owing to the MMD loss.

As in Fig. 2.13 we can interpret the unfolding output for a given detector-level event statistically. First, in the left panel of Fig. 2.16 we show a single event and how the FCGAN, INN, and cINN output is

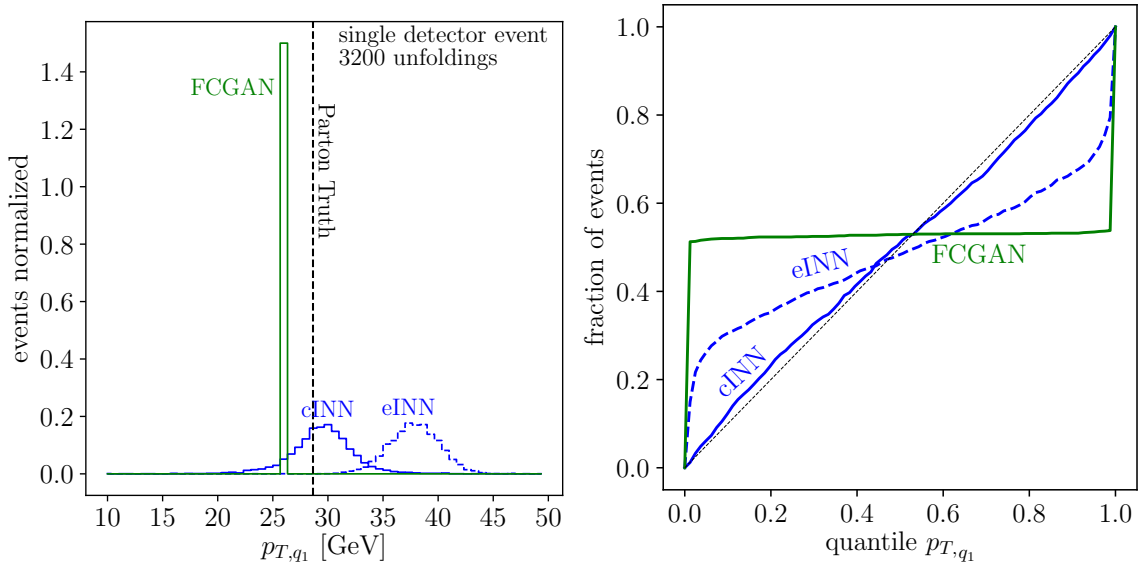

Figure 2.16: Left: illustration of the statistical interpretation of unfolded events for one event. Right: calibration curves for p_{T,q_1} extracted from the FCGAN and the noise-extended eINN, as shown in Fig. 2.13, and the cINN.

distributed in parton level phase space. The separation between truth and sampled distributions does not have any significance, but we see that the cINN inherits the beneficial features of the noise-extended eINN. In the right panel of Fig. 2.16 we again reconstruct the individual probability distribution from the unfolding numerically. We then determine the position of the parton-level truth in its respective probability distribution for the INN and the cINN. We expect a given percentage of the 1500 events to fall into the correct quantile of its respective probability distribution. The corresponding calibration curve for the cINN is added to the right panel of Fig. 2.16, indicating that without additional calibration the output of the cINN unfolding can be interpreted as a probability distribution in parton-level phase space for a single detector-level event, as always assuming an unfolding model. Instead of the transverse momentum of the harder parton-level quark we could use any other kinematic distribution at parton level. This marks the final step for a statistically interpretable unfolding.

2.5 Unfolding with jet radiation

In the previous chapter we used a simplified data set to explore different possibilities to unfold detector level information with invertible networks. We limit the data to events with exactly two jets, by switching off initial state radiation (ISR). This guarantees that the two jets come from the W-decay. In a realistic QCD environment we do not have that information, because additional QCD jets will be radiated off the initial and final state partons. In this section we demonstrate how we can unfold a sample of events including ISR and hence with a variable number of jets. We know that with very few exceptions [125] the radiation of QCD jets does not help us understand the nature of the hard process. In such cases, we would like to interpret a measurement with an appropriately defined hard process, leading to the question if an unfolding network can invert detector effects and QCD jet radiation. Technically, this means inverting jet radiation and kinematic modifications to the hard process as, in our case, done by Pythia.

We emphasize that this approach requires us to define a specific hard process with any number of external jets and other features. We can illustrate this choice for two examples. First, a di-tau resonance search typically probes the hard process $pp \to \mu^+\mu^- + X$, where X denotes any number of additional, analysis-irrelevant jets. We invert the corresponding measurements to the partonic process

$pp \to \mu^+\mu^-$. A similar mono-jet analysis instead probes the process $pp \to Z'j(j) + X$, where Z' is a dark matter mediator decaying to two invisible dark matter candidate. Depending on the analysis, the relevant process to invert is $pp \to Z'j$ or $pp \to Z'jj$, where a reported missing transverse momentum recoils against one or two hard jets. Because our inversion network in trained on Monte Carlo data, we automatically define the appropriate hard process when generating the training data. This covers any combination of signal and background matrix elements contributing to such a hard process, even non-SM processes to quantify a remaining model dependence. A final caveat — in the hard process we do not include subjet aspects at this stage. As long as subjet information is used for tagging purposes it factorizes from the hard process information and can easily be included in terms of efficiencies. A problem would arise in unfolding or inverting analyses relying on different hard processes, like a fat mono-jet analysis, where the above choice of recoil jets is left to a sub-jet algorithm.

2.5.1 Individual n-jet samples

In Sec. 2.4.3 we have shown that our cINN can unfold detector effects for ZW-production at the LHC. The crucial new feature of the cINN is that it provides probability distribution in parton-level phase space for a given detector-level event. The actual unfolding results are illustrated in Fig. 2.15, focusing on the two critical distribution known from the corresponding FCGAN analysis [28]. The event sample used throughout Sec. 2.4 includes exactly two partons from a W-decay with minimal phase space cuts on the corresponding jets. Strictly speaking, these phase space cuts are not necessary in this simulation. The correct definition of a process described by perturbative QCD includes a free number of additional jets,

$$pp \to ZW^\pm + \text{jets} \to (\ell^-\ell^+)\,(jj) + \text{jets}\,, \tag{2.27}$$

For the additional jets we need to include for instance a p_T cut to regularize the soft and collinear divergences at fixed-order perturbation theory. The proper way of generating events is therefore to allow for any number of additional jets and then cut on the number of hard jets. Since ISR can lead to jets with larger p_T than the W-decay jets, an assignment of the hardest jets to hard partons does not work. We simply sort jets and partons by their respective p_T and let the network work out their relations. We limit the number of jets to four because larger jet number appear very rarely and would not give us enough training events.

Combining all jet multiplicities we use 780k events, out of which 530k include exactly two jets, 190k events include three jets and 60k have four or more jets. We split the data into 80% training data

Parameter	cINN no ISR	cINN ISR incl.
Blocks	24	24
Layers per block	2	3
Units per layer	256	256
Condition/encoder layers	2	8
Units per condition/encoder layer	1024	1024
Condition/encoder output dimension	256	256
Trainable weights	~ 2 M	~ 10 M
Encoder pre training epochs	-	300
Epochs	1000	900
Learning rate	$8 \cdot 10^{-4}$	$8 \cdot 10^{-4}$
Batch size	512	512
Training/testing events	290k / 30k	620k / 160k
Kernel widths	$\sim 2, 8, 25, 67$	$\sim 2, 8, 25, 67$
D_p	12	12
D_d	16	25
λ_{MMD}	0.5	0.04
λ_{MMD} increase	-	1.6 / 100 epochs

Table 2.3: cINN setup and hyper-parameters, as implemented in PyTorch (v1.2.0) [124].

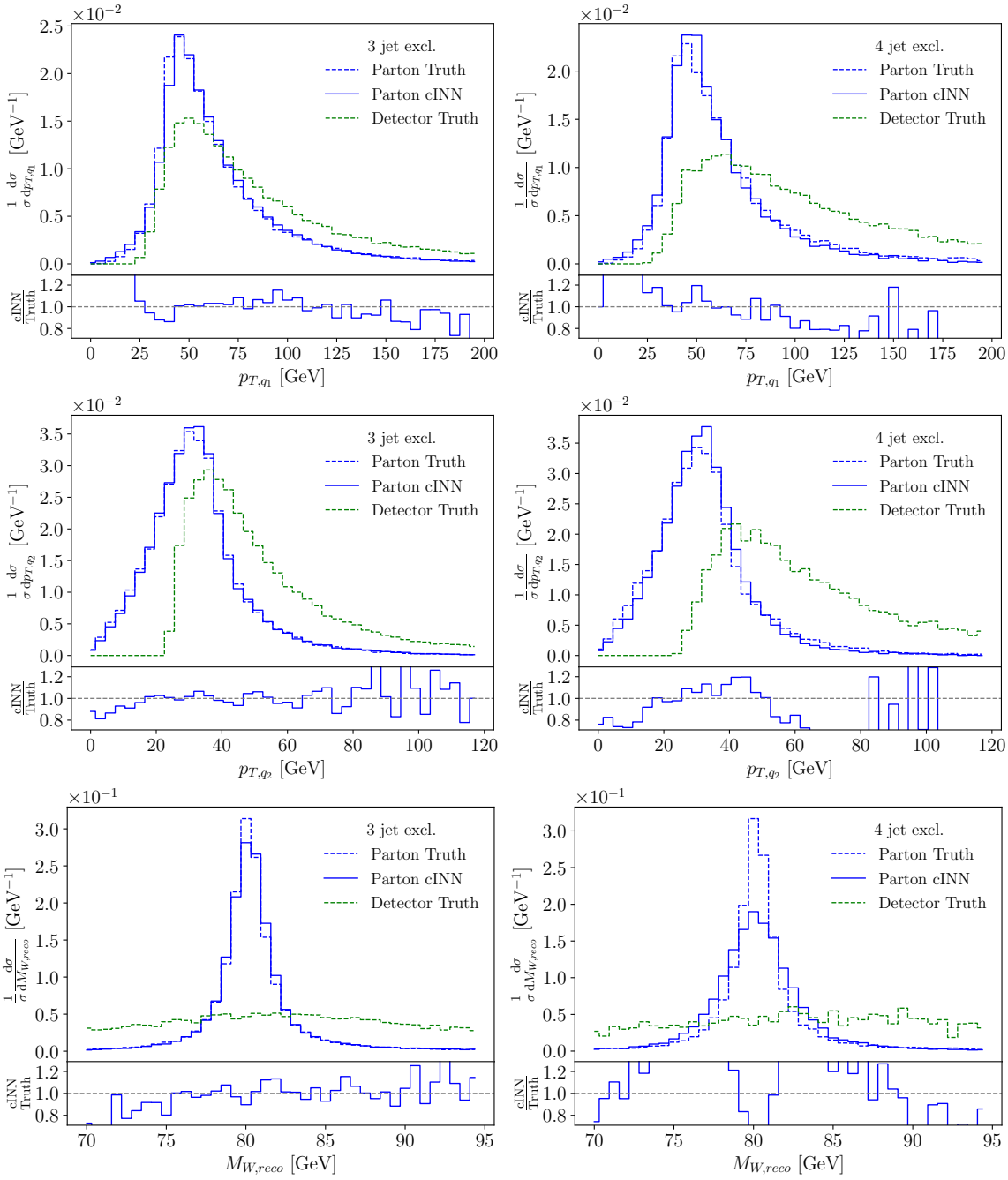

Figure 2.17: cINNed $p_{T,q}$ and $m_{W,\text{reco}}$ distributions. Training and testing events include exactly three (left) and four (right) jets from the data set including ISR.

and 20% test data to produce the shown plots. For the network input we zero-pad the event-vector for events with less than four jets and add the number of jets as additional information. The training samples are then split by exclusive jet multiplicity, such that the cINN reconstructs the 2-quark parton-level kinematics from two, three, and four jets at the detector level.

As before, we can start with the sample including exactly two jets. The difference to the sample used before is that now one of the W-decay jets might not pass the jet p_T condition in Eq.(2.11), so it will be replaced by an ISR jet in the 2-jet sample. Going back to Fig. 2.15 we see in the right panel how these events are slightly different from the sample with only decay jet. The main difference is in

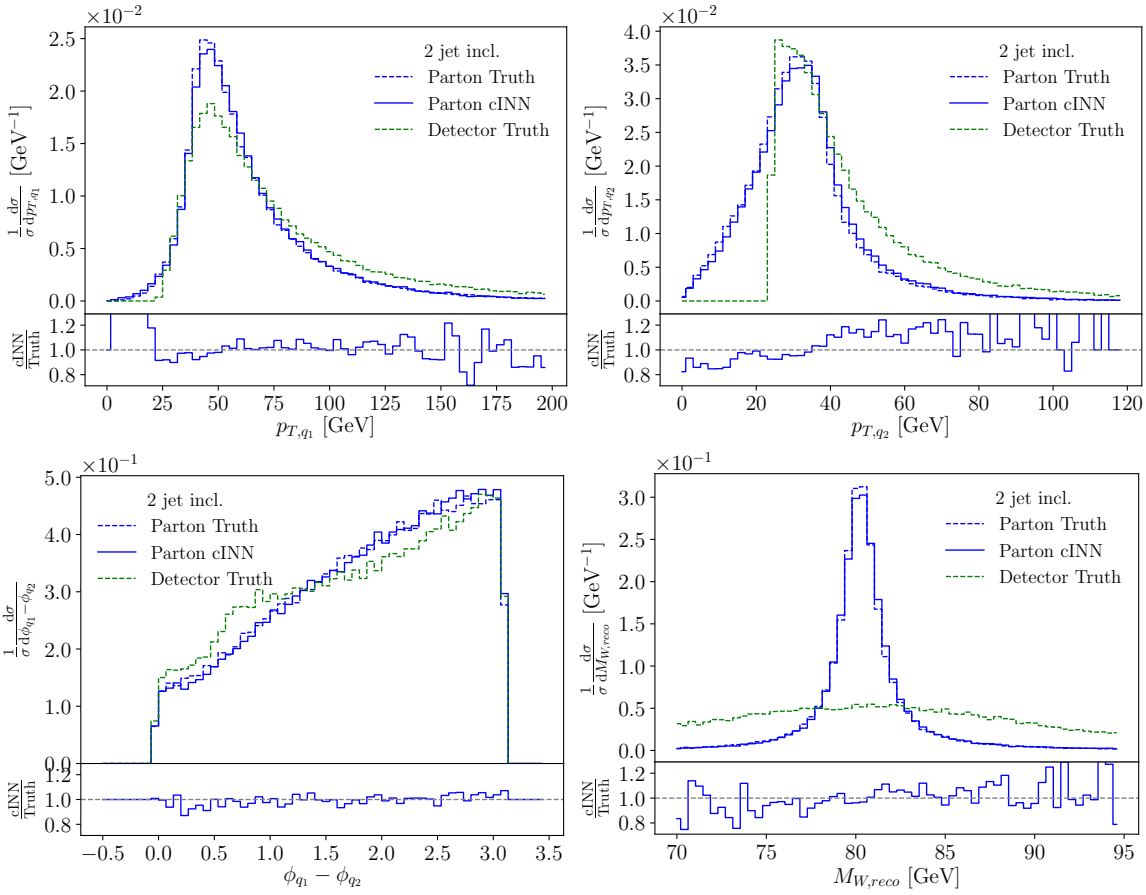

Figure 2.18: cINNed example distributions. Training and testing events include two to four jets, combining the samples from Fig. 2.15 and Fig. 2.17 in one network. At the parton level there exist only two W-decay quarks.

p_{T,q_2}, where the QCD radiation produces significantly more soft jets. Still, the network learns these features, and the unfolding for the sample without ISR and the 2-jet exclusive sample has a similar quality. In Fig. 2.17 we see the same distributions for the exclusive 3-jet and 4-jet samples. In this case we omit the secondary panels because they are dominated by the statistical uncertainties of the training sample. For these samples the network has to extract the parton-level kinematics with two jets only from up to four jets in the final state. In many cases this corresponds to just ignoring the two softest jets and mapping the two hardest jets on the two W-decay quarks, but from the p_{T,q_2} distributions in Fig. 2.15 we know that this is not always the correct solution. Especially in the critical m_{jj} peak reconstruction we see that the network feels the challenge, even though the other unfolded distributions look fine.

2.5.2 Combined n-jet sample

The obvious final question is if our INN can also reconstruct the hard scattering process with its two W-decay quarks from a sample with a variable number of jets. Instead of separate samples as in Sec. 2.5.1 we now interpret the process in Eq.(2.27) as jet-inclusive. This means that the hard process includes only the two W-decay jets, and all additional jets are understood as jet radiation, described either by resummed ISR or by fixed-order QCD corrections. The training sample consists of the combination of the right panels in Fig. 2.15 and the two panels in Fig. 2.17. This means that the network has to deal with the different number of jets in the final state and how they can be related

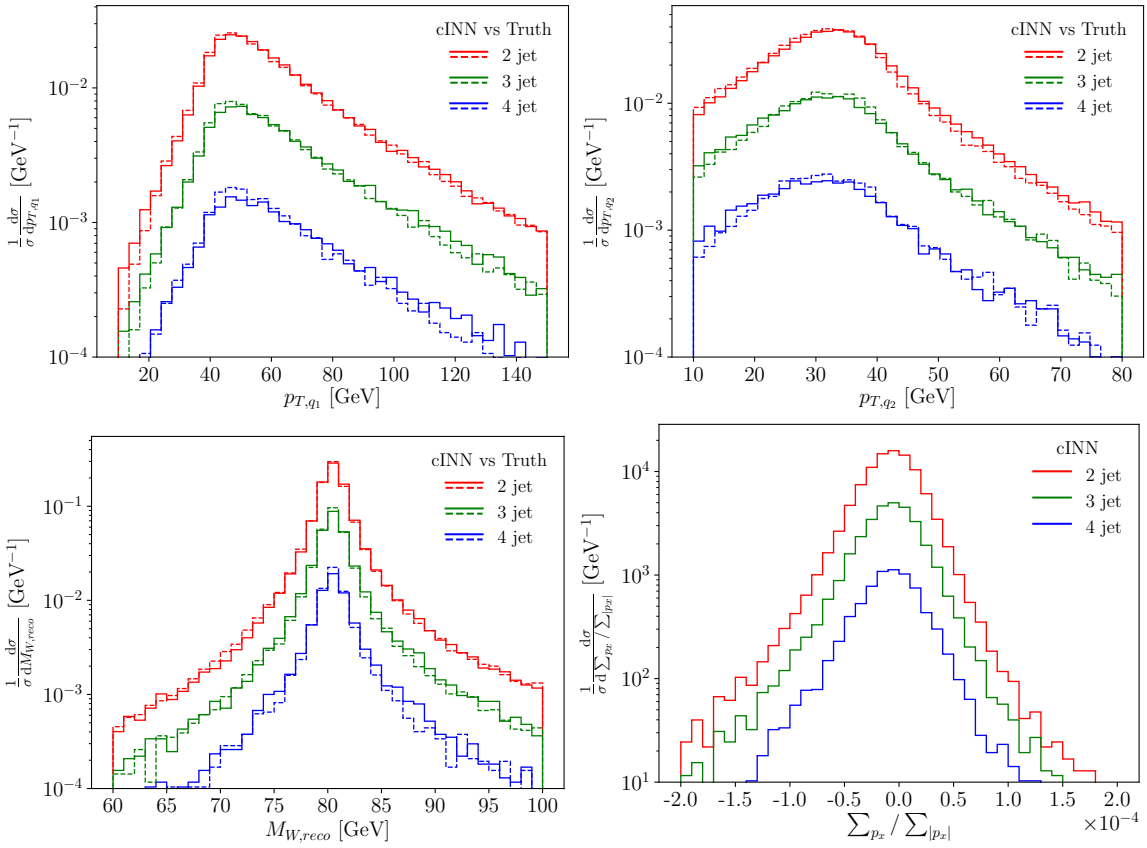

Figure 2.19: cINNed example distributions. Training and testing events include two to four events, combining the samples from Fig. 2.15 and Fig. 2.17 in one network. The parton-level events are stacked by number of jets at detector level.

to the two hard jets of the partonic $ZW \to \ell\ell jj$ process. The number of jets in the final state is not given by individual hard partons, but by the jet algorithm and its R-separation.

In Fig. 2.18 we show a set of unfolded distributions. First, we see that the $p_{T,j}$ thresholds at the detector level are corrected to allow for $p_{T,q}$ values to zero. Next, we see that the comparably flat azimuthal angle difference at the parton level is reproduced to better than 10% over the entire range. Finally, the m_{jj} distribution with its MMD loss re-generates the W-mass peak at the parton level almost perfectly. The precision of this unfolding is not any worse than it is for the case where the number of hard partons and jets have to match and we only unfold the detector effects.

In Fig. 2.19 we split the unfolded distributions in Fig. 2.18 by the number of 2, 3, and 4 jets in the detector-level events. In the first two panels we see that the transverse momentum spectra of the hard partons are essentially independent of the QCD jet radiation. In the language of higher-order calculations this means that we can describe extra jet radiation with a constant K-factor, if necessary with the appropriate phase space mapping. Also the reconstruction of the W-mass is not affected by the extra jets, confirming that the neural network correctly identifies the W-decay jets and separates them from the ISR jets. Finally, we test the transverse momentum conservation at the unfolded parton level. Independent of the number of jets in the final state the energy and momentum for the pre-defined hard process is conserved at the 10^{-4} level. The kinematic modifications from the ISR simulation are unfolded correctly, so we can compute the matrix element for the hard process and use it for instance for inference.

2.6 Conclusions and outlook

In this chapter we have demonstrated how deep generative models have the required flexibility and expressive power to invert Monte Carlo simulations like a a fast detector simulation. For our example process $WZ \to (jj)(\ell\ell)$ at the LHC we have worked out several possible configurations of (conditional-) GANs and INNs, giving emphasis to the strengths and weaknesses of each version.

Starting with the GAN, we have shown how a naive approach works extremely well when the training sample and the test sample are very similar. In that case the GAN benefits from the fact that we do not actually need an event-by-event matching of the parton-level and detector-level samples.

If the training and test samples become significantly different we may use instead a conditional model. It maps random noise parton-level events with conditional, event-by-event detector-level input and learns to generate parton-level events from detector-level events. First, we noticed that the conditional-GAN with its structured mapping provides much more stable predictions in tails of distributions, where the training sample is statistics limited. Then, we have shown that a network trained on the full phase space can be applied to much smaller parts of phase space, even including cuts in the main kinematic features. The conditional-GAN successfully maintains a notion of events close to each other at detector level and at parton level and maps them onto each other. This approach only breaks eventually because the MMD loss needed to map narrow Breit-Wigner propagators is not (yet) conditional in our specific setup.

In the second half of the chapter we have instead focused on invertible architectures. We have shown how an INN and in particular a conditional INN can be used to unfold detector effects for the same reference process. The cINN is not only able to unfold the process over the entire phase space, but it also gives correctly calibrated posterior probability distributions over parton-level phase space for given detector-level events.

Finally, we have extended the inversion to a variable number of jets in the final state. This situation will automatically appear whenever we include higher-order corrections in perturbative QCD for a given hard process. The hard process at parton level is defined at the training level. We find that the cINN also unfolds QCD jet radiation in the sense that it identifies the ISR jets and corrects the kinematics of the hard process to ensure energy-momentum conservation in the hard scattering.

In combination, these features should enable analysis techniques like the matrix element method and efficient ways to communicate analysis results including multi-dimensional kinematic distributions. While the ZW production process used in this analysis, we expect these results to carry over to more complex processes with intermediate particles and the impact of a SM-training hypothesis should be under control, the next step will be to test this new framework in a realistic LHC example with proper analysis of the uncertainties.

3 | What is the uncertainty of generative models?

Following the growing interest in refining LHC simulations with deep generative models, a crucial question to realistically use them in an analysis, is how to define and estimate the uncertainty associated to models output. We illustrate how approximate Bayesian inference can be used to define an uncertainty over a generative model's output. Our findings show a highly non-trivial pattern in the model uncertainty, which may be understood in terms of a functional fit of the learnt density.

3.1 Introduction

In Chap. 2 we have shown that deep generative models certainly possess the expressive power that we need to produce realistic synthetic data. However, expressive power is not the only feature needed for their realistic use in particle physics analysis. When data is simulated via standard Monte Carlo methods, there's a clear understanding of what is the associated statistical error.

Assuming a simple set-up in which we wish to estimate the observable

$$\mathcal{O} = \int_x dx f(x), \tag{3.1}$$

we can estimate $\mathcal{O}$ via Monte Carlo

$$\mathbb{E}\left[\mathcal{O}\right] = \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{3.2}$$

with uncertainty

$$\mathrm{Var}\left[\mathbb{E}\left[\mathcal{O}\right]\right] = \frac{\mathrm{Var}\left[f(x)\right]}{N}. \tag{3.3}$$

It is not within the scope of this chapter to discuss standard methods such as optimized importance sampling to improve the standard integration described above, instead, we want emphasize that as $\mathcal{O}$ is estimated using the "true" function $f(x)$, the only relevant figure is the number of samples N. This is clearly not the case for approaches like the one described in Ref. [88], where a generative model completely replaces the integrator. As generative models only approximates a desired distribution, in this case the integrand $f(x)$, they should be used with caution if and when precision is required. In other worlds, if an additional rejection sampling based on the truth is not added on top of the generative model, our prediction will also include uncertainties from the fact that we generate samples from some function $\tilde{f} \sim f$, and this extra uncertainty collects everything associated to model's training. This does not mean that such an approach should be discarded entirely, but rather that we should think of a way to estimate this approximation uncertainty.

The fact that experiments, field theory calculations, and simulations control their uncertainties implies that we can work with a complete uncertainty budget, including statistical, systematic, and theory uncertainties. To sustain this approach at the upcoming HL-LHC, with a data set more than 25 times

the current Run 2 data set, the challenge is to provide faster simulations while keeping full control of the uncertainties.

In recent years it has been shown that modern machine learning can improve LHC event simulations in various ways. Promising techniques include deep generative models such as generative adversarial networks (GAN) [57,58], variational autoencoders [63,64], and invertible models [48–53,53,54]. Further developments are fully NN-based event generation [19, 86–89], detector simulation [10, 11, 14, 15, 90–94, 126], or parton showering [95–98, 127]. Generative models may also improve searches for physics beyond the Standard Model [99], anomaly detection [103, 128], detector resolution [129, 130], and inference [131, 132]. Finally, conditional GANs and INNs allow us to invert the simulation chain to unfold detector effects [27,28] and extract the hard scattering process at parton level [42]. The problem with these applications is that little is known about

1. how these generative networks work, and
2. what the uncertainty on the generative network output is.

As we will see in this chapter, these two questions may be related.

In general, we can track statistical uncertainties in neural network outputs with Bayesian networks [133–136]. Such networks have been used in particle physics for a long time [137–139]. For the LHC we have proposed to use them to extract uncertainties in jet classification [140] and jet calibration [141]. They can cover uncertainties related to statistical and structural limitations of the training sample [142], while being obviously unable to track systematic uncertainties. Similar ideas can be used as part of ensemble techniques [143]. We propose to use a Bayesian INN (BINN) to extract uncertainties induced by the network training on a generated event sample.

Because invertible models allows to evaluate the likelihood exactly, they are a natural tool for evaluating the uncertainty over the learnt distribution, compared to implicit models like generative adversarial networks. While Bayesian classification [140] and regression networks [141] highlight the statistical and nature of uncertainties, our Bayesian generative network exhibits a very different structure. We will discuss the learning pattern of the Bayesian INN in details for a set of simple toy processes in Sec. 3.3, before we apply the network to a semi-realistic LHC example in Sec. 3.4.

3.2 Generative networks with uncertainties

We start by reminding that in the literature it is often assumed that a generative model has learned a phase-space density perfectly, so that the only remaining source of uncertainty is the statistics of the generated sample binned in phase space. However, such an assumption is not realistic [140, 141], and we need to estimate the effect of statistical limitations of the training data, as well as in the representation power of the model and the training procedure. A problem with existing methods to estimate models uncertainties is that they rely on training many networks, with the assumption that randomly initializing their weights is enough to get independent samples and finally comparing their outcome. For some applications this can be an infeasible option, so we will show how an alternative solution could look.

3.2.1 Uncertainties on event samples

Uncertainties on a simulated kinematic or phase space distribution are crucial for any LHC analysis. We denote the complete phase space weight for a given phase space point as $p(x)$, such that we can express a total cross section as

$$\sigma_{\text{tot}} = \int_0^1 dx\, p(x) \quad \text{with} \quad p(x) > 0 \,. \tag{3.4}$$

In this simplified notation x stands for a generally multi-dimensional phase space. For each phase space position, we can likewise define an uncertainty $\sigma(x)$.

One contribution to the error budget are systematic and theory uncertainties, $\sigma_{\text{th/sys}}(x)$. The former reflect our ignorance of aspects of the training data, which do not decrease when we increase the amount of training data. The latter captures the degree to which we trust our prediction, for instance based on self-consistency arguments. For example, we can account for possible large, momentum-dependent logarithms as a simple function of phase space. If we use a numerical variation of the factorization and renormalization scale to estimate a theory uncertainty, we typically re-weight events with the scales. Another uncertainty arises from the statistical limitations of the training data, $\sigma_{\text{stat}}(x)$. In the Gaussian limit, a statistical uncertainty can be defined by binning the phase space and in that limit we expect a scaling like $\sigma_{\text{stat}}(x) \sim \sqrt{p(x)}$, and we will test that hypothesis in detail in Sec. 3.3.

Once we know the uncertainties as a function of the phase space position, we can account for them as additional entries in unweighted or weighted events. For instance, relative uncertainties can be easily added to unweighted events,

$$\text{ev}_i = \begin{pmatrix} \sigma_{\text{stat}}/p \\ \sigma_{\text{syst}}/p \\ \sigma_{\text{th}}/p \\ \{x_{\mu,j}\} \\ \{p_{\mu,j}\} \end{pmatrix} , \qquad \text{with } \mu = 0 \ldots 3 \text{ for each particle } j. \tag{3.5}$$

The entries σ or σ/p are smooth functions of phase space. The challenge in working with this definition is how to extract σ_{stat} without binning. Specific theory counterparts can be either computed directly or extracted by appropriately modifying the training data [140, 141].

3.2.2 Bayesian INN

To model complex densities such as LHC phase space distributions, we can employ invertible models of the kind described in Chap. 2.

An INN composes multiple transformation maps into coupling layers with dimensional partitioning. The input vector z into a layer is split in half, $z = (z_1, z_2)$, allowing us to compute the output $x = (x_1, x_2)$ of the layer as

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z_1 \odot e^{s_2(z_2)} + t_2(z_2) \\ z_2 \odot e^{s_1(x_1)} + t_1(x_1) \end{pmatrix} , \tag{3.6}$$

where s_i, t_i $(i = 1, 2)$ are arbitrary functions, and $\odot$ is the element-wise product. In practice, each is a small multi-layer perceptron. This transformation has the benefit of being trivially invertible, given a vector $x = (x_1, x_2)$ the inverse is given by

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} (x_1 - t_2(z_2)) \odot e^{-s_2(z_2)} \\ (x_2 - t_1(x_1)) \odot e^{-s_1(x_1)} \end{pmatrix} . \tag{3.7}$$

Additionally, its Jacobian is an upper triangular matrix

$$\frac{\partial G(z)}{\partial z} = \begin{pmatrix} \text{diag}\left(e^{s_2(z_2)}\right) & \text{finite} \\ 0 & \text{diag}\left(e^{s_1(x_1)}\right) \end{pmatrix} , \tag{3.8}$$

whose determinant is just the product of the diagonal entries, irrespective of the off-diagonal entries. As such, it is computationally cheap and easily composable.

We refer to the overall map composing a sequence of such coupling layers as $G(z; \theta)$, where we collected the parameters of the individual nets s, t of each layer into a joint θ. Note that each coupling layer has a separate set of nets, whose indices we suppress (e.g. s^l, t^l for the l-th layer). We can then train the model via a maximum likelihood approach. It relies on the assumption that we have access to a data set of N samples $\mathcal{D} = \{x_1, \ldots, x_N\}$ of the intractable target phase space distribution $p^*_X(x)$ and

want to fit our model distribution $p_X(x;\theta)$ via the INN G. The maximum likelihood loss is

$$\begin{aligned}\mathcal{L}_{\text{ML}} &= -\sum_{n=1}^{N} \log p_X(x_n;\theta) \\ &= -\sum_{n=1}^{N} \log p_Z\big(\overline{G}(x_n;\theta)\big) + \log\left|\det \frac{\partial \overline{G}(x_n;\theta)}{\partial x_n}\right| .\end{aligned} \tag{3.9}$$

Given the structure of $\overline{G}(x;\theta)$ and the base distribution p_Z each of the terms is tractable and can be computed efficiently. We can approximate the sum over the complete training data via a mini-batch and optimize the overall objective with a stochastic gradient descent approach. Note that one can see this maximum likelihood approach as minimizing the Kullback-Leibler (KL) divergence between the true but unknown phase space distribution $p_X^*(x)$ and our approximating distribution $p_X(x;\theta)$.

The invertible model provides us with a powerful generative model of the underlying data distribution. However, it lacks a mechanism to account for the uncertainty in the parameters θthemselves. In order to model it, we switch from deterministic to probabilistic transformations, replacing the deterministic sub-networks $s_{1,2}$ and $t_{1,2}$ in each of the coupling layers with Bayesian neural nets. In this section, we first review the structure of a classical Bayesian neural network (BNN) [144,145] as used in a supervised learning task, and then explain how we can use BNNs for our problem of modelling the phase space density, extending the INN into a Bayesian invertible neural network (BINN).

Bayesian Neural Networks Assuming a data set $\mathcal{D}$ consisting of N pairs of observations $(\mathbf{x}_i, y_i)$, $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$, in the supervised learning problem we want to model the relation $y = f_\theta(\mathbf{x})$ through a neural network parameterised by weights θ. Placing a prior over the weights and allowing for some observation noise, the generative model is given as

$$\begin{aligned} \theta &\sim p(\theta) , \\ y_i|\theta, \mathbf{x}_i &\sim p(y_i|\theta, \mathbf{x}_i), \qquad\qquad i = 1, \dots, N . \end{aligned} \tag{3.10}$$

In case of a regression with $y_i \in \mathbb{R}$ we often use a Gaussian likelihood, $p(y_i|\theta, \mathbf{x}_i) = \mathcal{N}\left(y_i|f_\theta(\mathbf{x}_i), \alpha^{-1}\right)$, and a Gaussian prior over the weights $p(\theta) = \mathcal{N}\left(\theta|\mathbf{0}, \beta^{-1}\mathbf{1}\right)$, with precisions α, β and $\mathbf{1}$ the identity matrix of suitable dimensionality [141]. We are not bound to these distributions and could for example choose a prior with a strongly sparsifying character for further regularization [146, 147]. Given the highly nonlinear structure of f_θ the posterior $p(\theta|\mathcal{D})$ is, for practically relevant applications, analytically intractable. While MCMC-based approaches can work for specific use cases and small networks [148], they quickly become too expensive for large architectures, so we instead rely on variational inference (VI) [149]. A VI-based model approximates the posterior $p(\theta|\mathcal{D})$ with a tractable simplified family of distributions, $q_\phi(\theta)$, parameterized by ϕ. We will rely on mean-field Gaussians throughout this work, learning a separate mean and variance parameter for each network weight. These parameters are learned by minimizing the KL-divergence

$$\min_\phi \text{KL}\big(q_\phi(\theta), p(\theta|\mathcal{D})\big) . \tag{3.11}$$

However, this objective is intractable, as it relies on the unknown posterior. Using Bayes' theorem we reformulate it as

$$\begin{aligned}\text{KL}\big(q_\phi(\theta), p(\theta|\mathcal{D})\big) &= -\int d\theta\; q_\phi(\theta) \log \frac{p(\mathcal{D}|\theta)p(\theta)/p(\mathcal{D})}{q_\phi(\theta)} \\ &= -\int d\theta\; q_\phi(\theta) \log p(\mathcal{D}|\theta) - \int d\theta\; q_\phi(\theta) \log \frac{p(\theta)}{q_\phi(\theta)} + \log p(\mathcal{D}) .\end{aligned} \tag{3.12}$$

Now, the log evidence $\log p(\mathcal{D})$ is bounded from below as

$$\begin{aligned}\log p(\mathcal{D}) &= \text{KL}\big(q_\phi(\theta), p(\theta|\mathcal{D})\big) + \int d\theta\; q_\phi(\theta) \log p(\mathcal{D}|\theta) - \text{KL}\big(q_\phi(\theta), p(\theta)\big) \\ &\geq \int d\theta\; q_\phi(\theta) \log p(\mathcal{D}|\theta) - \text{KL}\big(q_\phi(\theta), p(\theta)\big) .\end{aligned} \tag{3.13}$$

Maximizing this evidence lower bound (ELBO) then is equivalent to minimizing Eq.(3.11), giving us as the objective without the intractable posterior

$$\mathcal{L}_{\text{ELBO}} = \sum_{i=1}^{N} \Big\langle \log p(y_i|\theta, \mathbf{x}_i) \Big\rangle_{\theta\sim q_\phi(\theta)} - \text{KL}\big(q_\phi(\theta), p(\theta)\big) \; . \tag{3.14}$$

This turns the inference problem into an optimization problem, which allows us to take advantage of gradient descent methods such as Adam [150]. As the choice of prior $p(\theta)$ is under our control, the KL-term between the variational posterior and the prior is tractable. The intractable expectation in the first term we can approximate by taking S samples from the variational posterior and instead of computing the gradient over the whole data set in each iteration switch to a stochastic gradient setup, approximating the sum with a mini-batch of size M, giving us

$$\mathcal{L}_{\text{ELBO}} \approx \frac{N}{M} \sum_{i=1}^{M} \frac{1}{S} \sum_{s=1}^{S} \log p(y_i|\theta^{(s)}, \mathbf{x}_i) - \text{KL}\big(q_\phi(\theta), p(\theta)\big) \qquad \text{with} \quad \theta^{(s)} \sim q_\phi(\theta) \; . \tag{3.15}$$

In practice, it is often sufficient to approximate the expectation via a single sample ($S = 1$) per forward pass to keep the computational cost low and further rely on local re-parametrization [151] to reduce the variance of the gradients.

Bayesian INN As discussed in Sec. 3.2.2, our generative model consists of a map $G : z \to x$ from a base distribution $p_Z(z)$ to the phase-space $p_X(x)$ parameterized via an INN. Replacing the deterministic sub-networks $s_{1,2}$ and $t_{1,2}$ in Eq.(3.6) with BNNs we get as the generative pipeline for our BINN

$$\begin{aligned} \theta &\sim p(\theta), \\ x|\theta &\sim p_X(x|\theta) = p_Z(\overline{G}(x;\theta)) \Big| \det \frac{\partial \overline{G}(x;\theta)}{\partial x} \Big| \; . \end{aligned} \tag{3.16}$$

Given a set of N observations $\mathcal{D} = \{x_1, \ldots, x_N\}$ we can approximate the intractable posterior $p(\theta|\mathcal{D})$ as before with a mean-field Gaussian as the variational posterior $q_\phi(\theta)$. Learning the map and the posterior then is achieved by maximizing the equivalent of the ELBO loss in Eq.(3.15) for event samples,

$$\begin{aligned} \mathcal{L} &= \sum_{n=1}^{N} \langle \log p_X(x_n|\theta) \rangle_{\theta\sim q_\phi(\theta)} - \text{KL}\big(q_\phi(\theta), p(\theta)\big) \\ &= \sum_{n=1}^{N} \Big\langle \log p_Z\big(\overline{G}(x_n;\theta)\big) + \log \Big| \det \frac{\partial \overline{G}(x_n;\theta)}{\partial x_n} \Big| \Big\rangle_{\theta\sim q_\phi(\theta)} - \text{KL}\big(q_\phi(\theta), p(\theta)\big) \\ &\approx \frac{N}{M} \sum_{m=1}^{M} \frac{1}{S} \sum_{s=1}^{S} \log p_Z\big(\overline{G}(x_m;\theta^{(s)})\big) + \log \Big| \det \frac{\partial \overline{G}(x_m;\theta^{(s)})}{\partial x_m} \Big| - \text{KL}\big(q_\phi(\theta), p(\theta)\big) \; , \end{aligned} \tag{3.17}$$

with a mini-batch of size M and S samples $\theta^{(s)}$ from the variational posterior $q_\phi(\theta)$. By design all three terms, the log likelihood, the log determinant of the Jacobian as well as the Kullback-Leibler divergence can be computed easily. Automatic differentiation [124] allows us to get the gradients of $\mathcal{L}$ with respect to ϕ in order to fit our generative pipeline via a stochastic gradient descent update scheme.

3.3 Toy events with uncertainties

Before we tackle a semi-realistic LHC setup, we first study the behaviour of BINNs for a set of toy examples, namely distributions over the minimally allowed two-dimensional parameter space where in one dimension the density is flat. Aside from the fact that these toy examples illustrate that the BINN

actually constructs a meaningful uncertainty distribution, we will use the combination of density and uncertainty maps to analyse how an INN actually learns a density distributions. We will see that the INN representation of the target density may be interpreted as a few-parameter fit, rather than numerically encoding patches over the parameter space independently.

The default architecture for our toy models is a network with 32 units per layer, three layers per coupling block, and a total of 20 coupling blocks. It's implemented in PyTorch [124]. More details are given in Tab. 3.1. Generally, moderate changes of the hyperparameters do not have a visible impact on the performances. For each of the trainings we use a sample of 300k events. The widths of the Gaussian priors is set to one. We check that variations of this over several orders of magnitude did not have a significant impact on the performance.

3.3.1 Wedge ramp

Our first toy example is a two-dimensional ramp distribution, linear in one direction and flat in the other,

$$p(x, y) = \text{Linear}(x \in [0, 1]) \times \text{Const}(y \in [0, 1]) = x \times 2 \, . \tag{3.18}$$

The second term ensures that the distribution $p(x, y)$ is normalized to one, and the network output is shown in Fig. 3.1. The network output is a set of points in the two-dimensional parameters space, (x, y). We show one-dimensional distributions after marginalizing over the flat direction and find that the network reproduces Eq.(3.18) well.

In Fig. 3.2 we include the predicted uncertainty given by the BINN. For this purpose we train a network on the two-dimensional parameter space and evaluate it for a set of points with $x \in [0, 1]$ and a constant y-value. In the left panel we indicate the predicted uncertainty as an error bar around the density estimate. Throughout the chapter we always remove the phase space boundaries, because we

Parameter	Flow
Hidden layers (per block)	3
Units per hidden layer	32
Batch size	512
Epochs	300
Trainable weights	75k
Optimizer	Adam
$(\alpha,\ \beta_1,\ \beta_2)$	$(1 \times 10^{-3},\ 0.9,\ 0.999)$
Coupling layers	20
Training size	300k
Prior width	1

Table 3.1: Hyper-parameters for all toy models, implemented in pytorch(v1.4.0) [124].

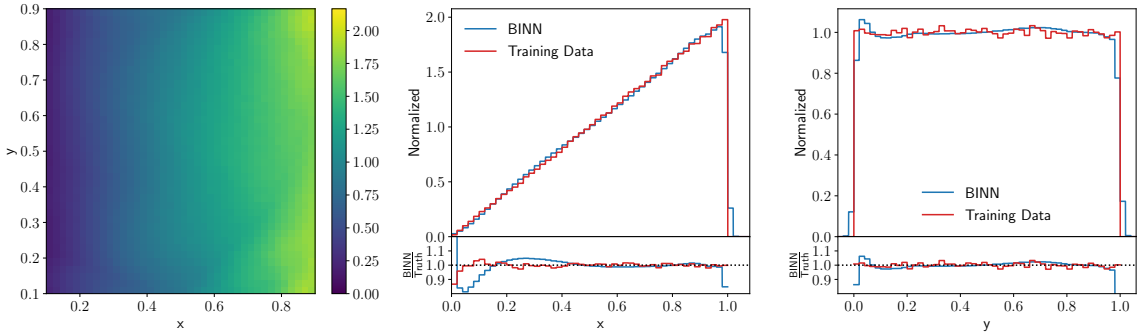

Figure 3.1: Two-dimensional and marginal densities for the linear wedge ramp.

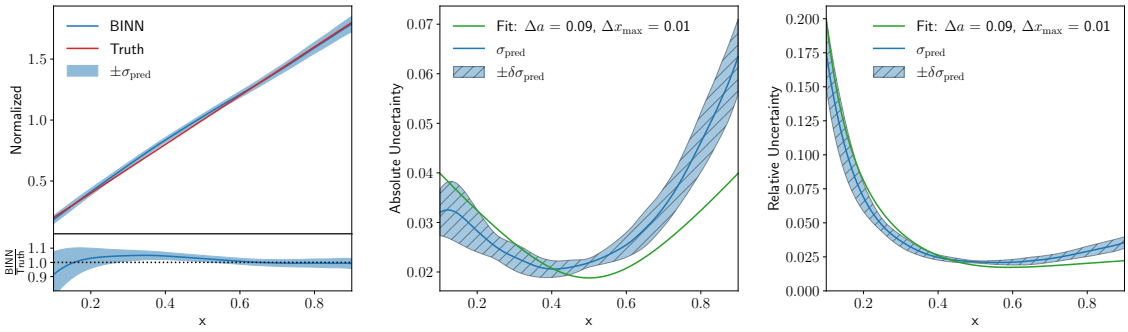

Figure 3.2: Density and predicted uncertainty distribution for the wedge ramp. In the left panel the density and uncertainty are averaged over several lines with constant y. In the central and right panels, the uncertainty band on σ_{pred} is given by their variation. The green curve represents a two-parameter fit to Eq.(3.24).

observe that the model predicts there large uncertainties which would overall dominate the figures. The relative uncertainty grows for small values of x and hence small values of $p(x, y)$, and it covers the deviation of the extracted density from the true density well. These features are common to all our network trainings. In the central and right panel of Fig. 3.2 we show the relative and absolute predicted uncertainties. The error bar indicates how much σ_{pred} varies for different choices of y. We compute it as the standard deviation of different values of σ_{pred}, after confirming that the central values agrees within this range. As expected, the relative uncertainty decreases towards larger x. However, the absolute uncertainty shows a distinctive minimum in σ_{pred} around $x \approx 0.45$. This minimum is a common feature in all our trainings, so we need to explain it.

To understand this non-trivial uncertainty distribution $\sigma_{\text{pred}}(x)$ we focus on the non-trivial x-coordinate and its linear behavior

$$p(x) = ax + b \qquad \text{with} \qquad x \in [0, 1] \, . \tag{3.19}$$

Because the network learns a normalized density, we can remove b by fixing the normalization,

$$p(x) = a\left(x - \frac{1}{2}\right) + 1 \, . \tag{3.20}$$

If we now assume that a network acts like a fit of a, , we can relate the uncertainty Δa to an uncertainty in the density

$$\sigma_{\text{pred}} \equiv \Delta p \approx \left|x - \frac{1}{2}\right| \, \Delta a \, . \tag{3.21}$$

The absolute value appears because the uncertainties are defined to be positive, as encoded in the usual quadratic error propagation. The uncertainty distribution has a minimum at $x = 1/2$, close to the observed value in Fig. 3.2.

The differences between the simple prediction in Eq.(3.21) and our numerical findings in Fig. 3.2 is that the predicted uncertainty is not symmetric and does not reach zero. To account for these sub-leading effects we can expand our very simple ansatz to

$$p(x) = ax + b \qquad \text{with} \qquad x \in [x_{\text{min}}, x_{\text{max}}] \, . \tag{3.22}$$

Using the normalization condition we again remove b and find

$$p(x) = ax + \frac{1 - \frac{a}{2}(x_{\text{max}}^2 - x_{\text{min}}^2)}{x_{\text{max}} - x_{\text{min}}} \, . \tag{3.23}$$

Again assuming a fit-like behaviour of the flow network we expect for the predicted uncertainty

$$\sigma^2_{\text{pred}} \equiv (\Delta p)^2 = \left(x - \frac{1}{2}\right)^2 (\Delta a)^2 + \left(1 + \frac{a}{2}\right)^2 (\Delta x_{\max})^2 + \left(1 - \frac{a}{2}\right)^2 (\Delta x_{\min})^2 \,. \tag{3.24}$$

Adding $x_{\max}$ adds an x-independent offset. Also accounting for $x_{\min}$ does not change the x-dependence of predicted uncertainty. The slight shift of the minimum and the asymmetry between the lower and upper boundaries in x are not explained by this argument. We ascribe them to boundary effects, specifically the challenge for the network to describe the correct approach towards $p(x) \to 0$.

The green line in the lower panels of Fig. 3.2 gives a two-parameter fit of Δa and $\Delta x_{\max}$ to the σ_{pred} distribution from the BINN. It indicates that there is a hierarchy in the way the network extracts the x-independent term with high precision, whereas the uncertainty on the slope a is around 4%.

3.3.2 Quadratic ramp

We can test our findings from the linear wedge ramp using the slightly more complex quadratic ramp,

$$p(x, y) = \text{Quadr}(x \in [0, 1]) \times \text{Const}(y \in [0, 1]) = x^2 \times 3 \,. \tag{3.25}$$

We show the results from the network training for the density in Fig. 3.3 and find that the network describes the density well, limited largely by the flat, low-statistics approach towards the lower boundary with $p(x) \to 0$.

In complete analogy to Fig. 3.2 we show the complete BINN output with the density $p(x, y)$ and the uncertainty $\sigma_{\text{pred}}(x, y)$ in Fig. 3.4. As for the linear case, the BINN reproduces the density well, deviations from the truth being within the uncertainty in all points of phase space. The indicated error bar on $\sigma_{\text{pred}}(x, y)$ is given by the variation of the predictions for different y-values, after ensuring that their central values agree. The relative uncertainty at the lower boundary $x = 0$ is large, reflecting the statistical limitation of this phase-space region. An interesting feature appears again in the absolute uncertainty, namely a maximum-minimum combination as a function of x.

Again in analogy to Eq.(3.22) for the wedge ramp, we start with the parametrization of the density

$$p(x) = a\,(x - x_0)^2 \qquad \text{with} \qquad x \in [x_0, x_{\max}] \,, \tag{3.26}$$

where we assume that the lower boundary coincides with the minimum and there is no constant offset. We choose to describe this density through the minimum position x_0, coinciding the the lower end of the x-range, and $x_{\max}$ as the second parameter. The parameter a can be eliminated through the normalization condition and we find

$$p(x) = 3\frac{(x - x_0)^2}{(x_{\max} - x_0)^3} \,. \tag{3.27}$$

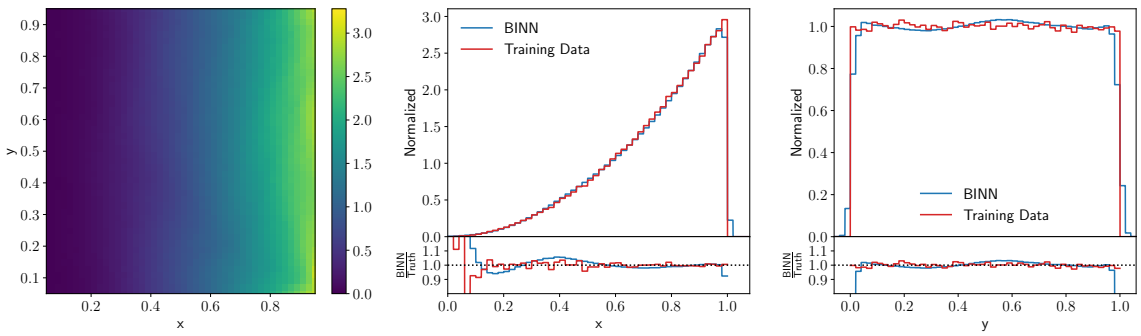

Figure 3.3: Two-dimensional and marginal densities for the quadratic ramp.

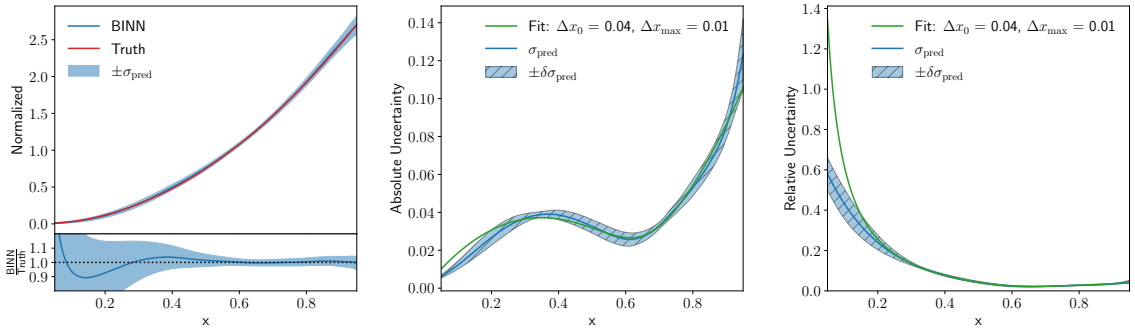

Figure 3.4: Density and predicted uncertainty distribution for the quadratic ramp. In the left panel the density and uncertainty are averaged over several lines with constant y. In the central and right panels, the uncertainty band on σ_{pred} is given by their variation. The green curve represents a two-parameter fit to Eq.(3.28).

If we vary x_0 and x_{max} we can trace two contributions to the uncertainty in the density,

$$\begin{aligned} \sigma_{\text{pred}} \equiv \Delta p \supset \frac{9}{(x_{\text{max}} - x_0)^4} \left| (x - x_0) \left(x - \frac{x_0}{3} - \frac{2x_{\text{max}}}{3} \right) \right| \Delta x_0 \\ \text{and} \qquad \sigma_{\text{pred}} \equiv \Delta p \supset \frac{9}{(x_{\text{max}} - x_0)^4} (x - x_0)^2 \, \Delta x_{\text{max}} \, , \end{aligned} \tag{3.28}$$

one from the variation of x_0 and one from the variation of x_{max}. In analogy to Eq.(3.24) they need to be added in quadrature. If the uncertainty on Δx_0 dominates, the uncertainty has a trivial minimum at $x = 0$ and a non-trivial minimum at $x = 2/3$. From Δx_{max} we get another contribution which scales like $\Delta p \propto p(x)$. In Fig. 3.4 we clearly observe both contributions, and the green line in the lower panels is given by the corresponding 2-parameter fig to the σ_{pred} distribution from the BINN.

3.3.3 Gaussian ring

Our third example is a two dimensional Gaussian ring, which in terms of polar coordinates reads

$$p(r, \phi) = \text{Gauss}(r > 0; \mu = 4, w = 1) \times \text{Const}(\phi \in [0, \pi]) \, , \tag{3.29}$$

We define the Gaussian density as the usual

$$\text{Gauss}(r) = \frac{1}{\sqrt{2\pi}\, w} \exp \left[-\frac{1}{2w^2} (r - \mu)^2 \right] \tag{3.30}$$

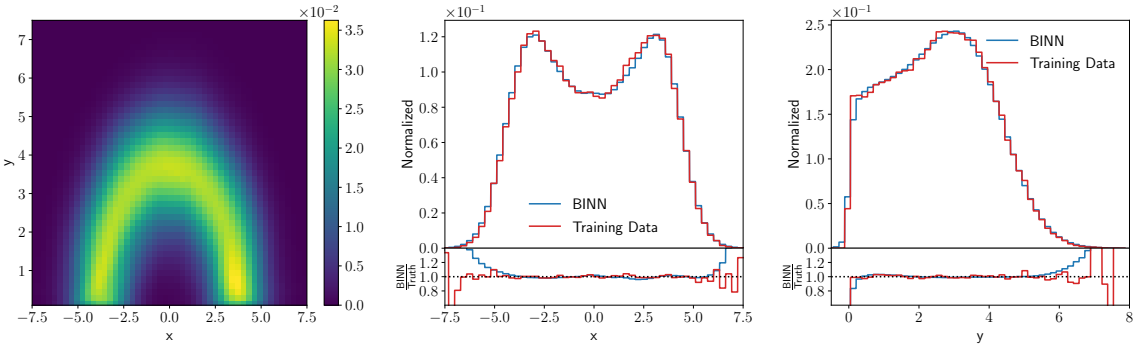

Figure 3.5: Two-dimensional and marginal densities for the Gaussian (half-)ring.

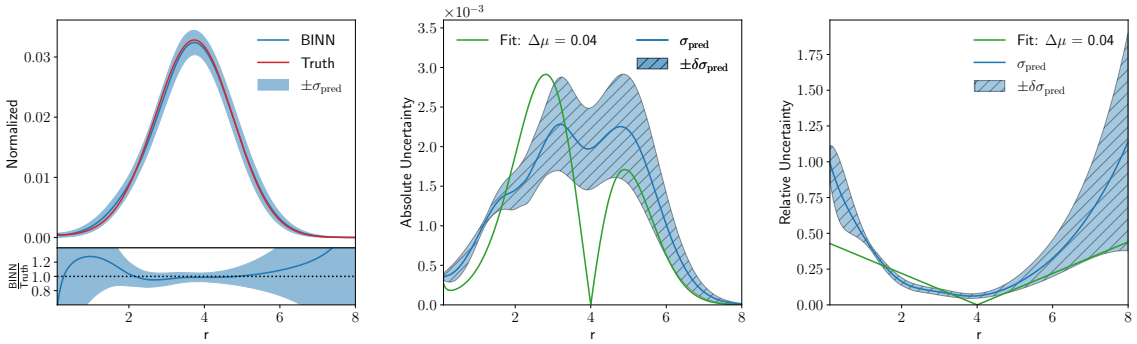

Figure 3.6: Cartesian density and predicted uncertainty distribution for the Gaussian ring. In the left panel the density and uncertainty are averaged over several lines with constant ϕ. In the central and right panels, the uncertainty band on σ_{pred} is given by their variation. The green curve represents a two-parameter fit to Eq.(3.32).

The density defined in Eq.(3.29) can be translated into Cartesian coordinates as

$$p(x,y) = \text{Gauss}(r(x,y); \mu = 4, w = 1) \times \text{Const}(\phi(x,y) \in [0,\pi]) \times \frac{1}{r(x,y)} \tag{3.31}$$

where the additional factor $1/r$ comes from the Jacobian. We train the BINN on Cartesian coordinates, just like in the two examples before, and limit ourselves to $y > 0$ to avoid problems induced by learning a non-trivial topology in mapping the latent and phase spaces. In Fig. 3.5 we once again see that our network describes the true two-dimensional density well.

In Fig. 3.6 we show the Cartesian density but evaluated on a line of constant angle. This form includes the Jacobian and has the expected, slightly shifted peak position at $r_{\text{max}} = 2 + \sqrt{3} = 3.73$. The BINN returns an uncertainty which grows towards both boundaries. The error band easily covers the deviation of the density learned by the BINN and the true density. While the relative predicted uncertainty appears to have a simple minimum around the peak of the density, we again see that the absolute uncertainty has a distinct structure with a local minimum right at the peak. The question is what we can learn about the INN from this pattern in the BINN.

As before, we describe our distribution in the relevant direction in terms of convenient fit parameters. For the Gaussian radial density these are the mean μ and the width w used in Eq.(3.29). The contributions driven by the extraction of the mean in Cartesian coordinates reads

$$\begin{aligned} \sigma_{\text{pred}} \equiv \Delta p &\supset \left| \frac{G(r)}{r} \frac{\mu - r}{w^2} \right| \Delta\mu \\ \text{and} \qquad \sigma_{\text{pred}} \equiv \Delta p &\supset \left| \frac{(r-\mu)^2}{w^3} - \frac{1}{w} \right| \Delta w \; . \end{aligned} \tag{3.32}$$

In analogy to Eq.(3.24) the two contributions need to be added in quadrature for the full, fit-like uncertainty. The contribution from the the mean has a minimum at $r = \mu = 4$ and is otherwise dominated by the exponential behavior of the Gaussian, just as we observe in the BINN result. In the opposite limit of $\Delta\mu \ll \Delta w$ the uncertainty develops the maxima at $r = 3$ and $r = 5$, which we observe in Fig. 3.6. In the central and right panels we show a one-parameter fit of the BINN output and find that the network determined the mean of the Gaussian as $\mu = 4 \pm 0.037$. We observe that including Δw does not improve the goodness of the fit.

3.3.4 Errors vs training statistics

Even though it is clear from the above discussion that we cannot expect the predicted uncertainties to have a simple scaling pattern, like for the regression [141] and classification [140] networks, there still remains the question of how the BINN uncertainties change with the size of the training sample.

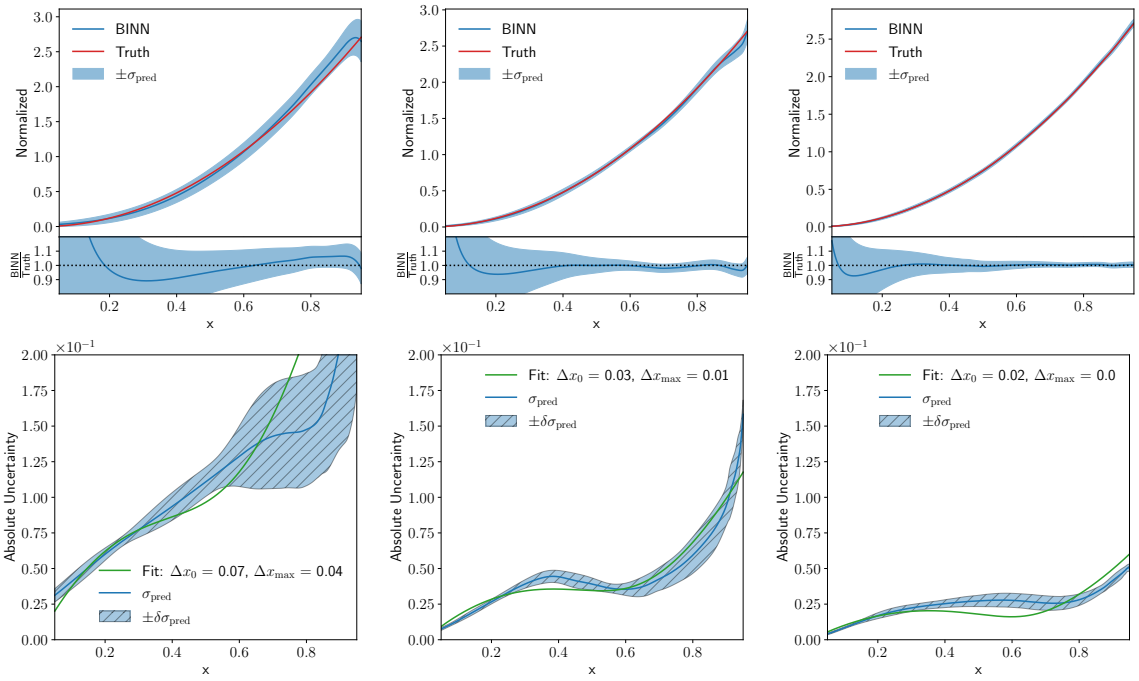

Figure 3.7: Dependence of the density (upper) and absolute uncertainty (lower) on the training statistics for the quadratic ramp. We illustrate BINNs trained on 10k, 100k, and 1M events (left to right), to be compared to 300k events used for Fig. 3.4. Our training routine ensures that all models receive the same number of weights updates, regardless of the training set size.

In Fig. 3.7 we show how the BINN predictions for the density and uncertainty change if we vary the training sample size from 10k events to 1M training events. Note that for all toy models, including the quadratic ramp in Sec. 3.3.2, we use 300k training events. For the small 10k training sample, we see that the instability of the BINN density becomes visible even for our reduced x-range. The peak-dip pattern of the absolute uncertainty, characteristic for the quadratic ramp, is also hardly visible, indicating that the network has not learned the density well enough to determine its shape. Finally, the variation of the predicted density explodes for $x > 0.4$, confirming the picture of a poorly trained model. As a rough estimate, the absolute uncertainty at $x = 0.5$ with a density value $p(x, y) = 0.75$ ranges around $\sigma_{\text{pred}} = 0.11 \ldots 0.15$.

For 100k training events we see that the patterns discussed in Sec. 3.3.2 begin to form. The density and uncertainty encoded in the network are stable, and the peak-dip with a minimum around $x = 2/3$ becomes visible. As a rough estimate we can read off $\sigma_{\text{pred}}(0.5) \approx 0.06 \pm 0.03$. For 1M training events the picture improves even more and the network extracts a stable uncertainty of $\sigma_{\text{pred}}(0.5) \approx 0.03 \pm 0.01$. Crucially, the dip around $x \approx 2/3$ remains, and even compared to Fig. 3.4 with its 300k training events the density and uncertainty at the upper phase space boundary are much better controlled.

Finally, we briefly comment on a frequentist interpretation of the BINN output. We know from simpler Bayesian networks [140,141] that it is possible to reproduce the predicted uncertainty using an ensemble of deterministic networks with the same architecture. However, from those studies we also know that our class of Bayesian networks has a very efficient built-in regularization, so this kind of comparison is not trivial. For the BINN results shown in this chapter we find that the detailed patterns in the absolute uncertainties are extracted by the Bayesian network much more effectively than they would be for ensembles of deterministic INNs. For naive implementations with a similar network size and no fine-tuned regularization these patterns are somewhat harder to extract. On the other hand, in stable regions without distinctive patterns the spread of ensembles of deterministic networks reproduces the predicted uncertainty reported by the BINN.

3.3.5 Marginalizing phase space

Before we move to a more LHC-related problem, we need to study how the BINN provides uncertainties for marginalized kinematic distributions. In all three toy examples the two-dimensional phase space consists of one physical and one trivial direction. For instance, the quadratic ramp in Sec. 3.3.2 has a quadratic physical direction, and in a typical phase space problem we would integrate out the trivial, constant direction and show a one-dimensional kinematic distribution. From our effectively one-dimensional uncertainty extraction we know that the absolute uncertainty has a characteristic maximum-minimum combination, as seen in the lower-right panel of Fig. 3.4.

To compute the uncertainty for a properly marginalized phase space direction, we remind ourselves how the BINN computes the density and the predicted uncertainty by sampling over the weights,

$$\begin{aligned} p(x,y) &= \int d\theta\, q(\theta)\, p(x,y|\theta) \\ \sigma^2_{\text{pred}}(x,y) &= \int d\theta\, q(\theta)\, \big[p(x,y|\theta) - p(x,y)\big]^2 \; . \end{aligned} \tag{3.33}$$

If we integrate over the y-direction, the marginalized density is defined as

$$\begin{aligned} p(x) = \int dy\, p(x,y) &= \int dy d\theta\, q(\theta)\, p(x,y|\theta) \\ &= \int d\theta\, q(\theta) \int dy\, p(x,y|\theta) \equiv \int d\theta\, q(\theta)\, p(x|\theta) \; , \end{aligned} \tag{3.34}$$

which implicitly defines $p(x|\theta)$ in the last step, notably without providing us with a way to extract it in a closed form. The key step in this definition is that we exchange the order of the y and θ integrations. Nevertheless, with this definition at hand, we can *define* the uncertainty on the marginalized distribution as

$$\sigma^2_{\text{pred}}(x) = \int d\theta\, q(\theta)\, \big[p(x|\theta) - p(x)\big]^2 \; . \tag{3.35}$$

We illustrate this construction with a trivial $p(x,y) = p(x,y_0)$, where we can replace the trivial y-dependence by a fixed choice $y = y_0$ just like for the wedge and quadratic ramps. Here we find, modulo a normalization constant in the y-integration

$$\begin{aligned} \sigma^2_{\text{pred}}(x) &= \int d\theta\, q(\theta)\, \big[p(x|\theta) - p(x)\big]^2 \\ &= \int d\theta\, q(\theta) \int dy\; \big[p(x,y_0|\theta) - p(x,y_0)\big]^2 \\ &= \int dy d\theta\, q(\theta)\; \big[p(x,y_0|\theta) - p(x,y_0)\big]^2 = \int dy\, \sigma^2_{\text{pred}}(x,y_0) = \sigma^2_{\text{pred}}(x,y_0) \; . \end{aligned} \tag{3.36}$$

Adding a trivial y-direction does not affect the predicted uncertainty in the physical x-direction.

As mentioned above, unlike for the joint density, $p(x,y|\theta)$ we do not know the closed form of the marginal distributions $p(x)$ or $p(x|\theta)$. Instead, we can approximate the marginalized uncertainties through a combined sampling in y and θ. We start with one set of weights θ_i from the weight distributions, based on one random number per INN weight. We now sample N points in the latent space, z_j, and compute N phase space point x_j using the BINN configuration θ_i. We then bin the wanted phase space direction x and approximate $p(x|\theta_i)$ by a histogram. We repeat this procedure $i = 1 \ldots M$ times to extract M histograms with identical binning. This allows us to compute a mean and a standard deviation from M histograms to approximates $p(x)$ and $\sigma_{\text{pred}}(x)$. The approximation of σ_{pred} should be an over-estimate, because it includes the statistical uncertainty related to a finite number of samples per bin. For $N \gg 1$ this contribution should become negligible. With this procedure we effectively sample $N \times M$ points in phase space.

Following Eq.(3.34), we can also fix the phase space points, so instead of sampling for each weight sample another set of phase space points, we use the same phase space points for each weight sampling.

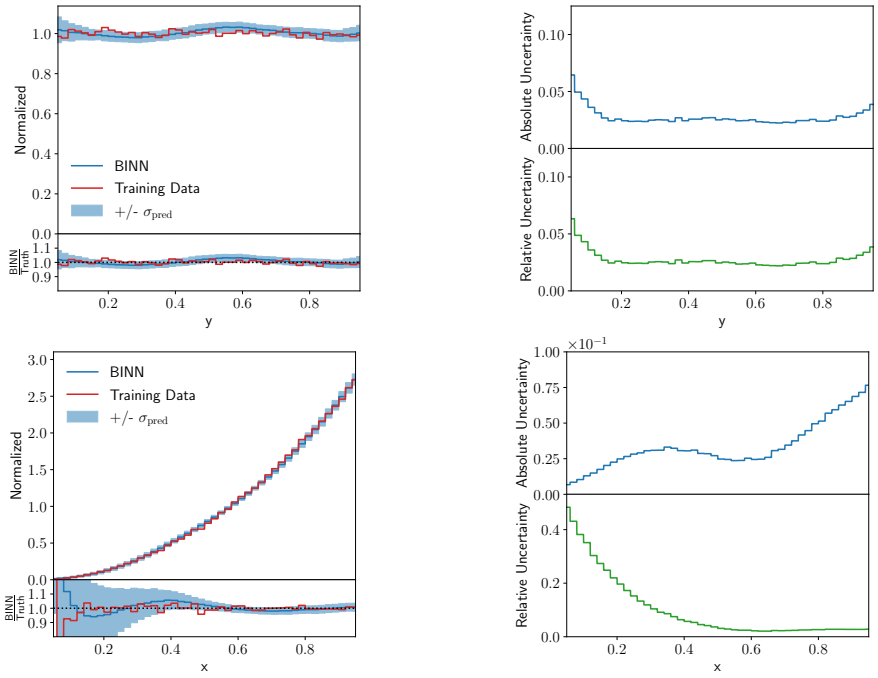

Figure 3.8: Marginalized densities and predicted uncertainties for the quadratic ramp. Instead of the true distribution we now show the training data as a reference, to illustrate possible limitations. We use 10M phase space point to guarantee a stable prediction.

This should stabilize the statistical fluctuations, but with the drawback of relying only on an effective number of N phase space points. Both approaches lead to the same σ_{pred} for sufficiently large N, which we typically set to 10^5 ... 10^6. For the Bayesian weights we find stable results for $M = 30$ ... 50.

In Fig. 3.8 we show the marginalized densities and predicted uncertainties for the quadratic ramp. In y-direction the density and the predicted uncertainty show the expected flat behaviour. The only exception are the phase space boundaries, where the density starts to deviate slightly from the training data and the uncertainty correctly reflects that instability. In x-direction, the marginalized density and uncertainty can be compared to their one-dimensional counterparts in Fig.3.4. While we expect the same peak-dip structure, the key question is if the numerical values for $\sigma_{\text{pred}}(x)$ change. If the network learns the y-direction as uncorrelated additional data, the marginalized uncertainty should decrease through a larger effective training sample. This is what we typically see for Monte Carlo simulations, where a combination of bins in an unobserved directions leads to the usual reduced statistical uncertainty. On the other hand, if the network learns that the y-directions is flat, then adding events in this direction will have no effect on the uncertainty of the marginalized distribution. This would correspond to a set of fully correlated bins, where a combination will not lead to any improvement in the uncertainty. In Fig. 3.8 we see that the $\sigma_{\text{pred}}(x)$ values on the peak, in the dip, and to the upper end of the phase space boundary hardly change from the one-dimensional results in Fig.3.4. This strengthens our general observation, that the (B)INN learns a functional form of the density in both directions, in close analogy to a fit. It also means that the uncertainty from the generative network training is not described by the simple statistical scaling we observed for simpler models [140, 141].

3.4 LHC events with uncertainties

As a physics example we consider the Drell-Yan process

$$pp \to Z \to e^+ e^- \ , \tag{3.37}$$

with its simple $2 \to 2$ phase space combined with the parton density. The training set consists of an unweighted set of 4-vectors simulated with madgraph [6] at 13 TeV collider energy with the NNPDF2.3 parton densities [152]. We fix the masses of the final-state leptons and enforce momentum conservation in the transverse direction, which leaves us with a four-dimensional phase space. In our discussion we limit ourselves to a sufficiently large set of one-dimensional distributions. For these marginalized uncertainties we follow the procedure laid out in Sec. 3.3.5 with 50 samples in the BINN-weight space. In Tab. 3.2 we give the relevant hyper-parameters for this section.

To begin with, we show a set of generated kinematic distributions in Fig. 3.9. The positron energy features the expected strong peak from the Z-resonance. Its sizeable tail to larger energies is well described by the training data to $E_e \approx 280$ GeV. The central value learned by the BINN becomes unstable at slightly lower values of 250 GeV, as expected. The momentum component p_x is not observable given the azimuthal symmetry of the detector, but it's broad distribution is nevertheless reproduced correctly. The predicted uncertainty covers the slight deviations over the entire range. What is observable at the LHC is the transverse momentum of the outgoing leptons, with a similar distribution as the energy, just with the Z-mass peak at the upper end of the distribution. Again, the predicted uncertainty determined by the BINN covers the slight deviations from the truth on the pole and in both tails. In the second row we show the p_z component as an example for a strongly peaked distribution, similar to the Gaussian toy model in Sec. 3.3.3.

While the energy of the lepton pair has a similar basic form as the individual energies, we also show the invariant mass of the electron-positron pair, which is described by the usual Breit-Wigner peak. It is well known that this intermediate resonance is especially hard to learn for a network, because it forms a narrow, highly correlated phase space structure. Going beyond the precision shown here would for instance require an additional MMD loss, as described in Ref. [28]. This resonance peak is the only distribution, where the predicted uncertainty does not cover the deviation of the BINN density from the truth. This apparent failure corresponds to the fact that generative networks always overestimate the width and hence underestimate the height of this mass peak [88]. This is an example of the network being limited by the expressive power in phase space resolution, generating an uncertainty which the Bayesian version cannot account for.

In Fig. 3.10 we show a set of absolute and relative uncertainties from the BINN. The strong peak combined with a narrow tail in the E_e distribution shows two interesting features. Just above the peak the absolute uncertainty drops more rapidly than expected, a feature shared by the wedge and quadratic ramps at their respective upper phase space boundaries. The shoulder around $E_e \approx 280$ GeV indicates that for a while the predicted uncertainty follows the increasingly poor modelling of the phase

Parameter	Flow
Hidden layers (per block)	2
Units per hidden layer	64
Batch size	512
Epochs	500
Trainable weights	$\sim$ 182k
Number of training events	$\sim$ 1M
Optimizer	Adam
(α, β_1, β_2)	(1×10^{-3}, 0.9, 0.999)
Coupling layers	20
Prior width	1

Table 3.2: Hyper-parameters for the Drell-Yan data set, implemented in pytorch(v1.4.0) [124].

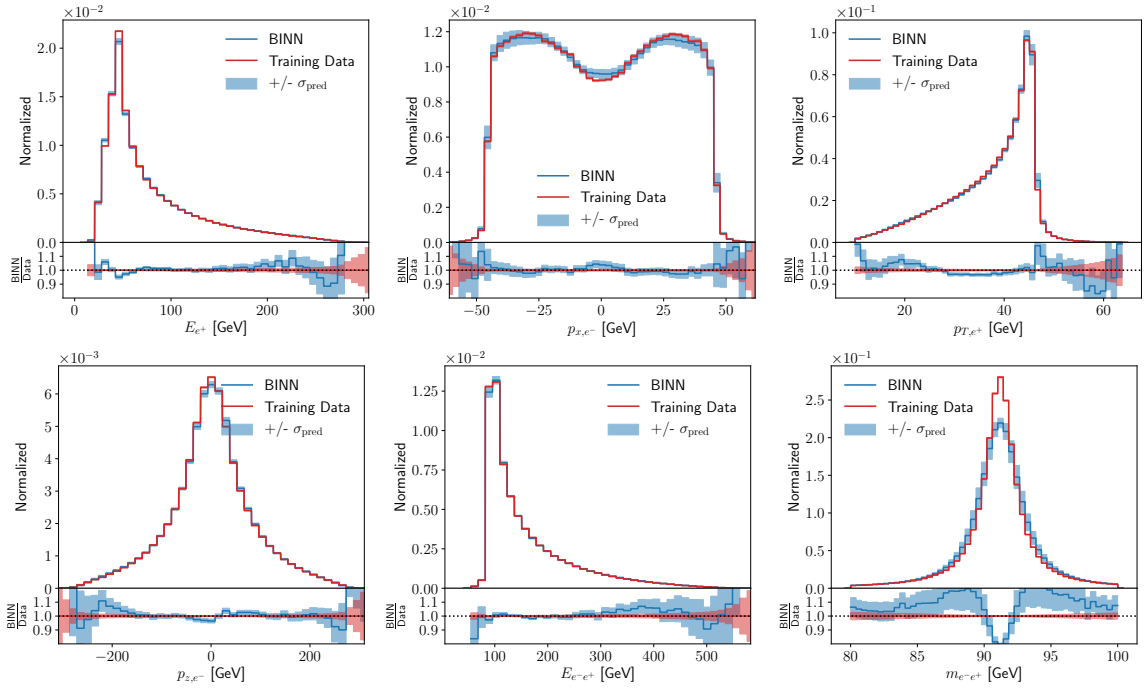

Figure 3.9: One-dimensional (marginalized) kinematic distributions for the Drell-Yan process. We show the central prediction from the BINN and include the predicted uncertainty from the BINN as the blue band. The red band indicates the statistical uncertainty of the training data per bin in the Gaussian limit.

space density by the BINN, to a point where the network stops following the truth curve altogether and the predicted uncertainty is limited by the expressive power of the network. Unlike the absolute uncertainty, the relative uncertainty keeps growing for increasing values of E_e. This behavior illustrates that in phase space regions where the BINN starts failing altogether, we cannot trust the predicted uncertainty either, but we see a pattern in the intermediate phase space regime where the network starts failing.

The second kinematic quantity we select is the x-component of the momentum, which forms a relative flat central plateau with sharp cliffs at each side. Any network will have trouble learning the exact shape of such sharp phase space patterns. Here the BINN keeps track of this, the absolute and the relative uncertainties indeed explode. The only difference between the two is that the density at the foot of the plateau drops even faster than the learned absolute uncertainty, so their ratio keeps growing.

Finally, we show the result for the Breit-Wigner mass peak, the physical counterpart of the Gaussian ring model of Sec. 3.3.3. Indeed, we see exactly the same pattern, namely a distinctive minimum in the predicted uncertainty right on the mass peak. This pattern can be explained by the network learning the general form of a mass peak and then adjusting the mean and the width of this peak. Learning the peak position leads to a minimum of the uncertainty right at the peak, and learning the width brings up two maxima on the shoulders of the mass peak. In combination Fig. 3.9 and 3.10 show that the BINN traces uncertainties in generated LHC events just as for the toy models.

3.5 Conclusions and outlook

Controlling the output of generative networks and quantifying their uncertainties is the main task for any application in LHC physics, be it in forward generation, inversion, or inference. We have proposed to use a Bayesian invertible network (BINN) to quantify the uncertainties from the network training for each generated event. For a series of two-dimensional toy models and an LHC-inspired application

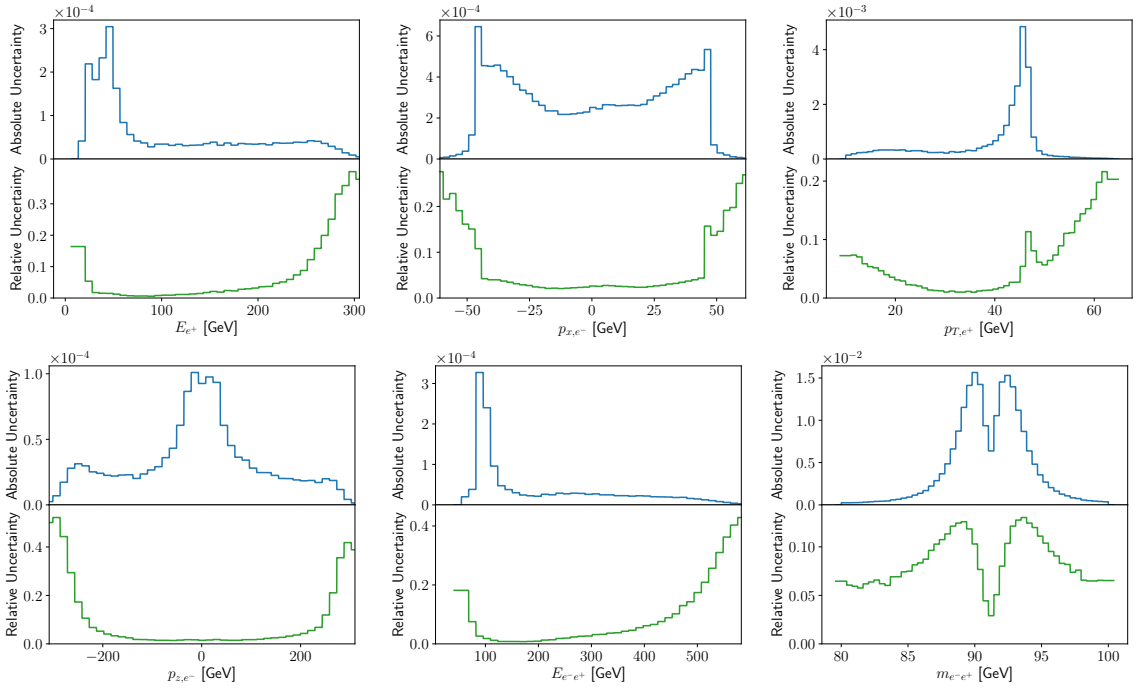

Figure 3.10: Absolute and relative uncertainties as a function of some of the kinematic Drell-Yan observables shown in Fig. 3.9.

we have shown how the Bayesian setup indeed generates an uncertainty distribution, over the full phase space and over marginalized phase spaces. As expected, the learned uncertainty shrinks with an improved training statistics. Our method can be trivially extended from unweighted to weighted events by adapting the simple MLE loss. An intriguing result from our study is that the combined learning of the density and uncertainty distributions allows us to draw conclusions on how a normalizing-flow network like the BINN learns a distribution. We find that the uncertainty distributions are naturally explained by a fit-like behavior of the network, rather than a patch-wise learning of the density. For the LHC, this can be seen for instance in the non-trivial uncertainty for an intermediate Breit-Wigner resonance.

4 | Topology improvement for invertible architectures

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 modelling 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.

4.1 Introduction

In the previous chapters generative models have been introduced as 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) [57, 58], Variational Autoencoders (VAE) [63,64], and Normalizing Flows (NF) [53,153] 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 HEPML-LivingReview

A key difference in generative modelling 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 modelling 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. [154] 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. [155] 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 extend these approaches by introducing the **La**tent **S**pac**e** **R**efinement (LaSeR) protocol, illustrated in Fig. 4.1. Like DctrGAN [76,155], 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 that the results are weighted, which reduces the statistical power of a generated sample. The energy-based framework of Discriminator Driven Latent Sampling (DDLS) [154] 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. [154]. We begin by directly pulling back the weights from our post-hoc classifier to the latent space (we also notice that in case our generator is not surjective, we still may do this via the OmniFold method [26]). This procedure allows us to define the optimal output as the one generated by the same nominal generative model, but the input is sampled from the optimal distribution encoded in the classifier weights. The challenge is then to efficiently sample from the optimal latent space. We 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 $g(\Phi(y))$.

While this procedure is completely general and therefore independent on the particular baseline model, we focus on the case where g is a normalizing flow and Φ is a GAN. This combination is particularly interesting given their different topological properties. 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. [121,156,157], where a standard autoencoder becomes probabilistic via a second model such as a NF trained on the bottleneck layer. Similarly, the authors of Ref. [158] proposed an iterative VAE setup to bring the latent space closer to a target multidimensional Gaussian random variable.

This chapter is organized as follows. We review related work and describe the statistical properties and challenges associated with existing generative models in Section 4.2. We introduce various ways of implementing the LaSeR protocol in Sec. 4.3. Illustrative numerical examples are provided in Sec. 4.4 and the chapter ends with conclusions and outlook in Sec. 4.6.

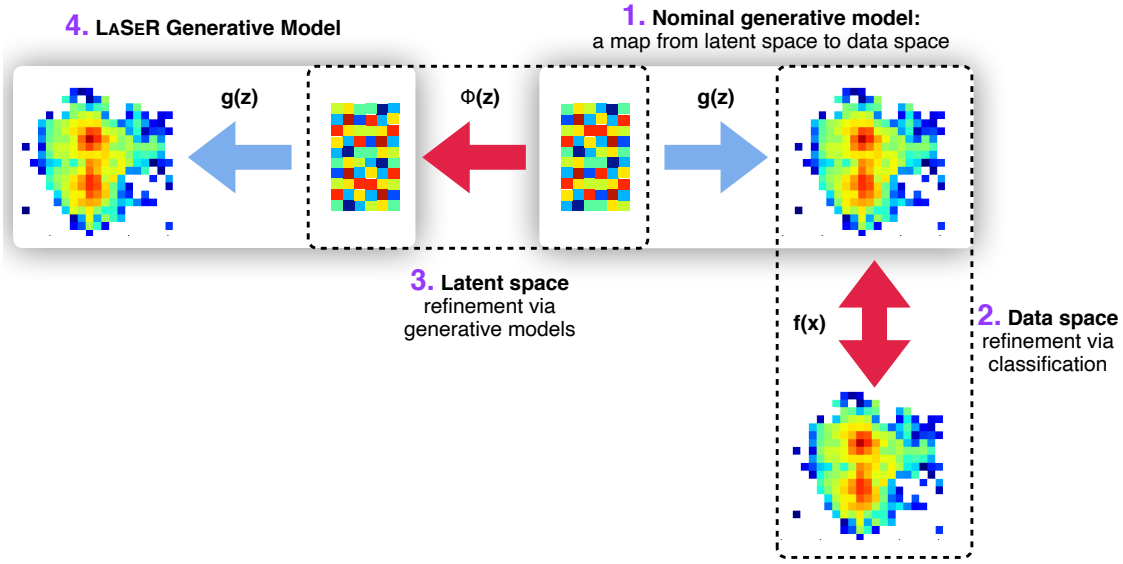

Figure 4.1: A schematic diagram illustrating the LaSeR protocol. 1. The upper right part of the diagram represents a given generative model $g : \mathbb{R}^N \to \mathbb{R}^M$ that maps features from a latent space to a data space. 2. A classifier network $f : \mathbb{R}^M \to \mathbb{R}$ is trained 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 $g(\Phi(y))$.

4.2 Background

As explained in Chap. 1, 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) [57, 58]
- Variational autoencoders (VAEs) [63, 64]
- Normalizing flows (NFs) [51, 53]

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 4.1 and are further discussed in the following.

Table 4.1: 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	✓	✓	✗

4.2.1 Generative models and coordinate transformations

The three generative models introduced in the previous section can all be trained directly on unlabeled data, and 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:

$$p_g(x) \equiv p_g(g(z)) = \left|\frac{\partial g(z)}{\partial z}\right|^{-1} p_\mathcal{Z}(z). \tag{4.1}$$

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:

$$\log p_g(x) = \log p_\mathcal{Z}(g^{-1}(x)) + \log\left|\frac{\partial g^{-1}(x)}{\partial x}\right|. \tag{4.2}$$

4.2.2 Topological obstructions

While the bijective nature of NFs allows for an explicit representation of the target probability density estimate, they inevitably suffer from the significant drawback described at the end of Sec. 1.2.1. In order to find a mapping g which satisfies Eq. (4.1) 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 n^{th} Betti number b_n of a topological space X is defined as the rank of the n^{th} homology group H_n of X [159, 160]

Informally, the n^{th} 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. 4.2 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 $b_2 = 1$.

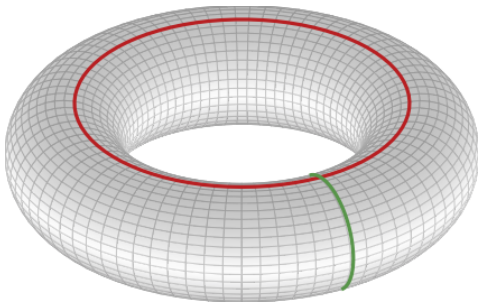

Figure 4.2: For a torus, the first Betti number is $b_1 = 2$, which can be intuitively understood as the number of circular holes.

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. [72, 161]. 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. [162] 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 [162] or by augmenting additional dimensions [72, 73]. Further, the issues arising by non-trivial topologies have also been investigated in the context of autoencoders in Ref. [163].

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 in 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.

4.3 Proposed methods

The LaSeR protocol is illustrated in Fig. 4.1. 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 [164, 165] (and forms the basis of the Dctr protocol [166]) 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 [167, 168]. Throughout this chapter, 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 [26] 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
$$q_{\mathcal{Z}}(z) = p_{\mathcal{Z}}(z)\, w(z)/Z_0, \tag{4.3}$$
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. [154] uses a simple Langevin dynamics algorithm, we suggest to use the more advanced Hamiltonian Monte Carlo (HMC) [169, 170] 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. [171, 172]. 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 4.2 illustrates some of the features of various combinations of generators g and latent space refiner networks Φ.

Table 4.2: 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

4.4 Toy 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.

4.4.1 Implementation details

All models and the training routines are implemented in PyTorch 1.8 [173]. 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 [174] implementation of a NF as the baseline model. Our implementation relies partially on the FrEIA library.* 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 [175] 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 3 hidden layers, 40 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. [171] to accommodate the weights

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

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 [170] 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.

4.4.2 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

$$\mathrm{JSD}(p,q) = \frac{1}{2}\int dx\, p(x) \log\left(\frac{2p(x)}{p(x)+q(x)}\right) + q(x)\log\left(\frac{2p(x)}{p(x)+q(x)}\right). \tag{4.4}$$

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

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

$$\begin{gathered}\mathrm{EMD}(p,q) = \min_{\{f_{ij}>0\}} \sum_{i=1}^{M}\sum_{i=1}^{M'} f_{ij}||x_i - y_j||_2, \\ \sum_{j=1}^{M'} f_{ij} \le p(x_i), \quad \sum_{i=1}^{M} f_{ij} \le q(y_j), \quad \sum_{i=1}^{M}\sum_{i=1}^{M'} f_{ij} = \min\left(\sum_{i=1}^{M} p(x_i), \sum_{j=1}^{M'} q(y_j)\right),\end{gathered} \tag{4.5}$$

where the optimal flow f_{ij} is found by solving this optimization problem. For an efficient implementation we used the POT library [176].

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 4.3.

4.4.3 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. 4.3, 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.

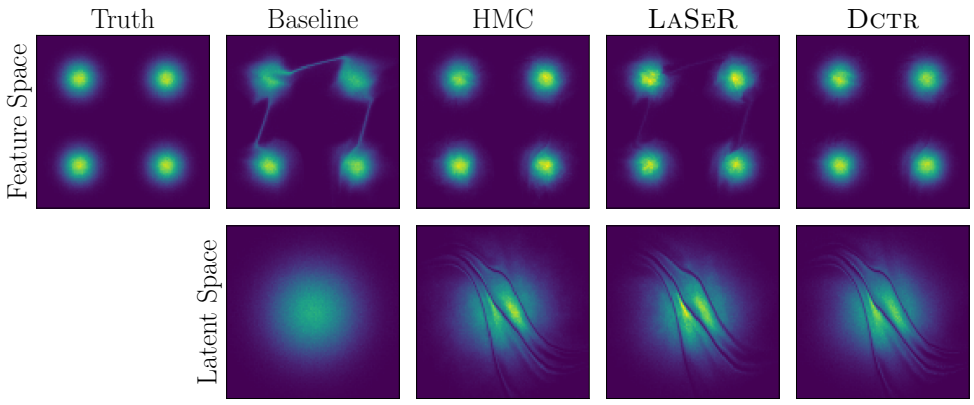

Figure 4.3: Comparison of the baseline model and the refined outputs by either using the HMC, LaSeR or Dctr method on a 2-dimensional multi-modal gaussian distribution with Betti numbers $b_0 = 4$, $b_1 = 0$.

In Fig. 4.4, 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. 4.5. 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 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

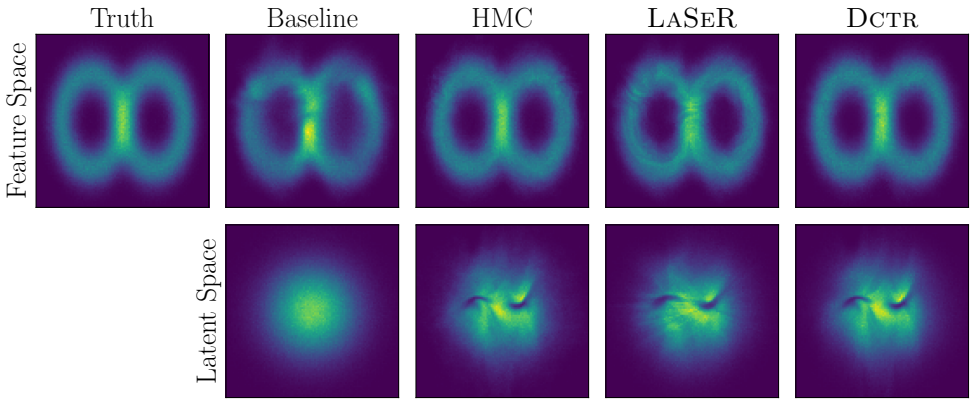

Figure 4.4: Comparison of the baseline generator and the refined outputs by either using the HMC, LaSeR or Dctr method on a 2-dimensional double donut distribution with Betti numbers $b_0 = 1$, $b_1 = 2$.

network hyperparameter optimization.

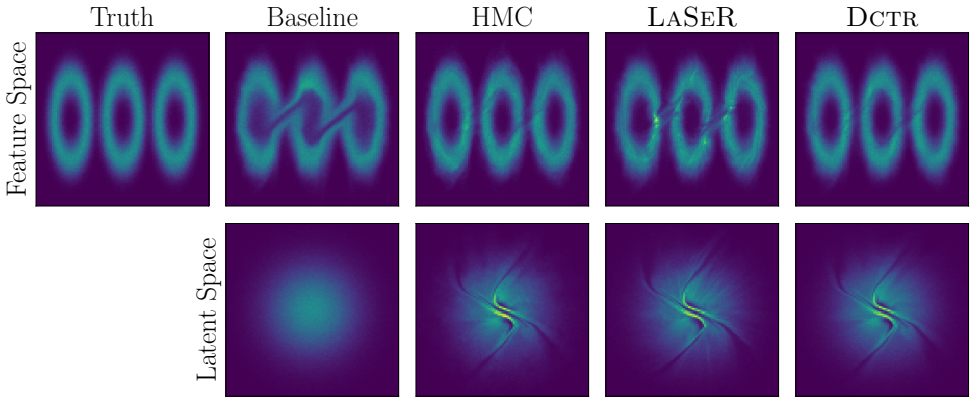

Figure 4.5: Comparison of the baseline generator and the refined outputs by either using the HMC, LaSeR or Dctr method on a 2-dimensional triple ring distribution with Betti numbers $b_0 = 3$, $b_1 = 3$.

Table 4.3: 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 4.3 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.

4.5 LHC applications

The content of this section is part of currently active, and therefore unpublished, research projects. In Sec. 4.4 we have shown how LaSeR) easily handle topology limitations in toy examples. Clearly the key question is whether or not such obstructions are relevant for high energy physics applications. We will immediately show that this is indeed the case. As a prime example, we consider the

$$pp \to W^+ + Nj \tag{4.6}$$

scattering process, with $N = 2, 3(, 4)$ jets. In order to stay in the perturbative regime of QCD we apply a typical cut on the transverse momenta of the jets as

$$p_{T,j} > 25\,\text{GeV}. \tag{4.7}$$

This process has a low phase-space dimensionality but still encodes typical topological features which need to be correctly addressed by the generative model, as the cuts on the transverse momenta introduce a number of holes in the process manifold equal to the number of jets (or any massless parton in general). In Fig. 4.6 we illustrate how the typical parton level cut on the transverse momentum of parton naturally leads to a topological obstruction.

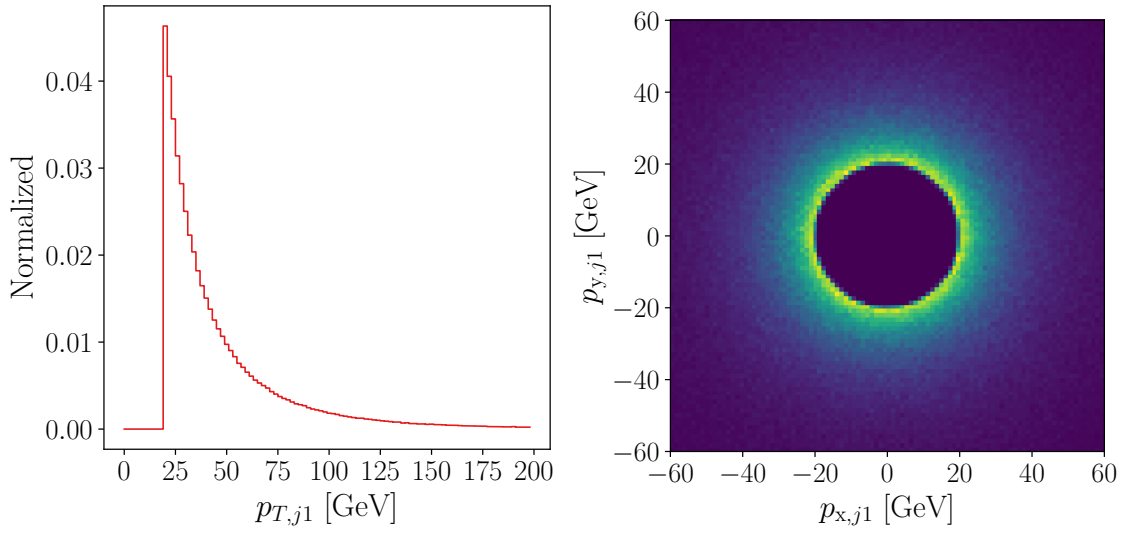

Figure 4.6: Left: the typical shape of the p_T distribution of a jet. Right: the effect of the transverse momentum cut visualized in the p_X-p_Y plane.

A very important comment is in order: some extremely clever cook may think that using a different set of variables rather then 4-momenta provide better results. This is logically flawed, as different

representations, like $\{p_T, \eta, \phi\}$ are again related to the 4-momenta by a change of variable. We conclude by showing in Fig. 4.7 and 4.8 preliminary results on the application of the LaSeR protocol to $pp \to W^+ + 3j$.

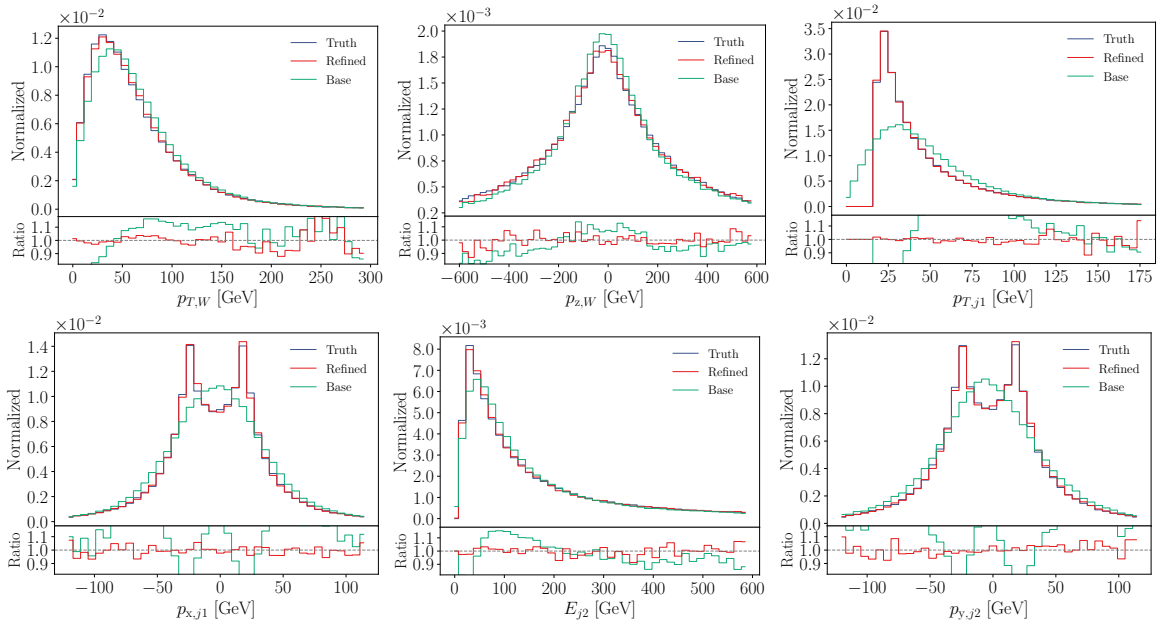

Figure 4.7: Set of kinematic distributions for the W + 3jets process, with and without LaSeR refinement.

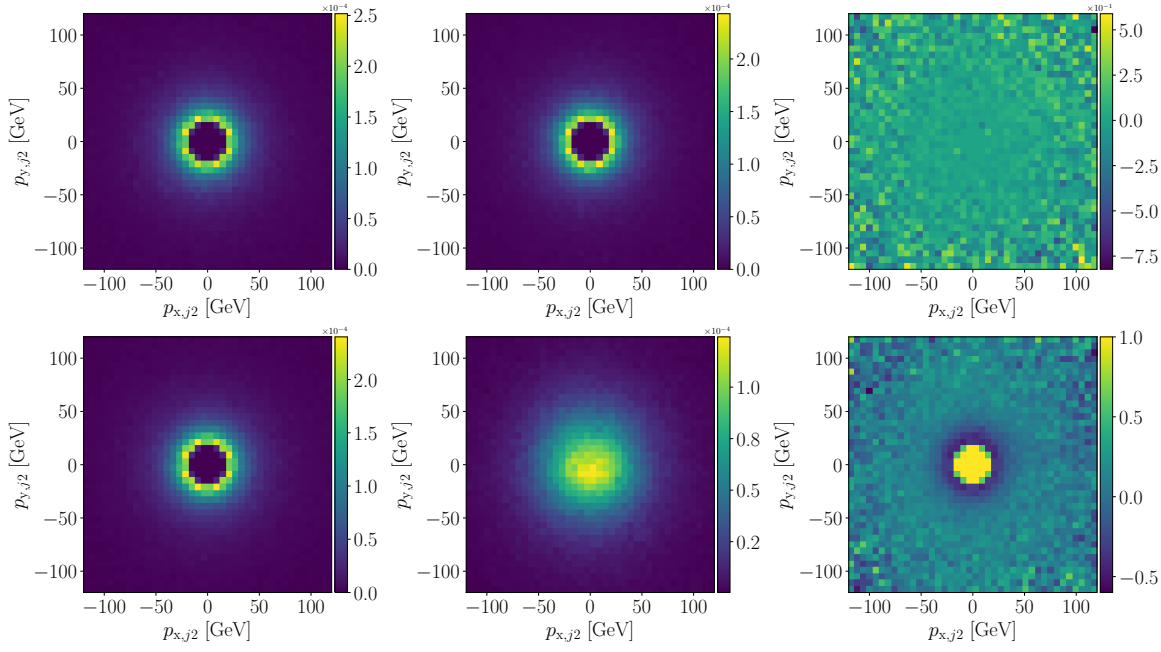

Figure 4.8: Two-dimensional distributions of one of the jets for the W + 3jets process, with and without LaSeR refinement.

4.6 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. We leave further investigations of the statistical power of refined samples and the interplay with hyperparameter choices for future research.

5 | Summary and Outlook

In this thesis we have illustrated how LHC analysis may benefit from the introduction of deep learning-based tools. In particular, we have focused on the possible application of a variety of deep generative models to the problem of unfolding. In parallel we have defined a method to estimate the uncertainty of generative models and an algorithmic procedure to remove topological obstructions from data manifolds.

In Chap. 2 we have employed GANs and INNs as tools for inverting Monte Carlo simulations without relying on binned histograms. We started with a naive GAN and shown that without a dedicated training correlating parton and detector level information, the inversion is meaningless. Accordingly, we changed our model to a conditional setting, showing how phase space slices at the detector level are correctly propagated by the model down to the parton level. This insight is then applied to INNs. We again started with a naive formulation, showing that our invertible architecture is powerful enough to represent the parton level distributions. We then employed a conditional set-up superior to that of the conditional GAN, showing that a conditional INN can generate posterior distributions over single events. We have concluded the chapter by illustrating how the conditional INN is powerful enough to handle a semi-realistic scenario with initial state radiation and a variable number of reconstructed jets.

In Chap. 3 we have addressed to problem of defining estimating the uncertainty associated to distributions generated by deep generative models. Providing with a well-justified, reliable estimate of model's uncertainties is a fundamental aspect for the adoption of deep learning methods in LHC analysis. While we believe that a lot more needs to be done both theoretically, and from the perspective of more realistic examples, we have made a first step in the direction of "trustable" generative models, by formulating the training of an INN as a Bayesian inference procedure.

Finally, in Chap. 4 we introduced the notion of topological obstructions between data manifolds, and explained if and how they may represents a limitation for existing methods using invertible architectures. We have therefore proposed a method which solves the problem by defining a refined latent space via a classifier weights. We have provided toy examples as well as first results on a realistic parton-level process.

The work presented mostly consists of proofs of principle in a new field with vast, unexplored directions. As such, it opens up the path to several possible directions and follow-ups. Concerning the idea of generative unfolding, we have presented in Fig. 2.10 how our method can be robust against the injection of signal not included in the training set. This, however, is far from a complete proof that such a method is independent of the prior distribution it is trained on. Furthermore, the method needs to be tested against other deep learning frameworks and traditional approaches such as OmniFold and IBU.

An obvious direction for the application of topology refinement is in hybrid Monte Carlo/machine learning approaches like i-flow [23, 24]. i-flow uses a normalizing flow to replace the proposal function of a standard Monte Carlo integrator like Sherpa. Phase-space cuts on the transverse momentum of massless particles may affect the final efficiency of the proposed model, and latent space refinement could be a simple way to boost their performances.

In conclusion, the field of deep learning for high energy physics is gaining more and more momentum, with novel applications, architectures and training procedures being proposed on a daily basis. We strongly believe that with a few more steps toward explainable machine learning, more mathematically

rigorous approaches and in general better communication between theorists and experimentalists, this technology will become a standard tool in the life of future researchers.

6 | Acknowledgements

The first thank goes to the amazing supervisors of my phd, Jan Martin Pawlowski, Monica Dunford and Hans-Christian Schultz-Coulon, for their support, the supervision and the scientific discussions.

I'm obviously grateful to the Heidelberg University and the Max Planck Institute for paying my rent and giving me the opportunity of fulfilling my dream of getting a phd and do research.

I'm grateful to the referees of this thesis and the members of my defence committee Jan Martin Pawlowski, Monica Dunford, Fred Hamprecht and Lorenzo Masia for wasting a morning of their life to listen to me.

Secondly, a huge thanks goes to my phd-siblings, the postdocs, students, visitors and all the amazing people I met along the way, Theo, Tanmoy, Ruth, Ramon, all the Sebastians, Sascha, Patrick, Peter, Nina, Natalie, Nastia, Manuel, Jennifer, Luca, Lynton, the Lennarts, Ilaria, Gonzalo, Emma, Diego, Bob, Barry, Armand, Anke, Michel (list is in strict order of importance). Please have mercy of me, if I forgot you it was probably not on purpose (probably). Also, if you expected personalised greetings, well suck on that, I have too much stuff to do these days.

I'm extremely grateful to Ullrich Köthe, Gregor Kasieczka, Simone Marzani, Benjamin Nachman and Michael Spannowsky, the incredible scientists I had the opportunity to (at least partially) work with, and most of all learn from. A special thank of course goes to all the people I directly worked with, Ramon, Lynton, Armand, Manuel and Michel.

A special thanks goes to the proof-readers of the thesis Stefano, Peter, Ramon and Michel.

Whenever I felt hopeless, I could always count on amazing friends, Ante, Elena, Elisa, Franzo, Giulia V., Jens, Manto, the Minuters, Minuti, Nicola, Nigel, Peter, Pischi, Porots, Princess, Ramon, Verra and Michel (again in strict order of importance).

I would never have made it without the support of my family, my parents Annalaura and Benedetto, Ginger, my brother Simone, Krystine and my wonderful niece Aurora. A kind and warm thank goes to Giulia's family, to Alvaro, Elena, Franca, Fuffino, Ester and Federica and to Mirko.

Finally, the biggest, warmest thank goes to my better half, the umami of my life, the kimchi in my pancakes, the guanciale in my carbonara: my dear Giulia, without whom I would only be a fish out of water.

Appendices

A | Appendix: conditional GAN

A.1 Performance

While it is clear from the main text that the FCGAN inversion of the fast detector simulation works extremely well, we can still show some additional standard measures to illustrate this. For instance, in Fig. A.1 we show the event-wise normalized deviation between the parton-level truth kinematics and

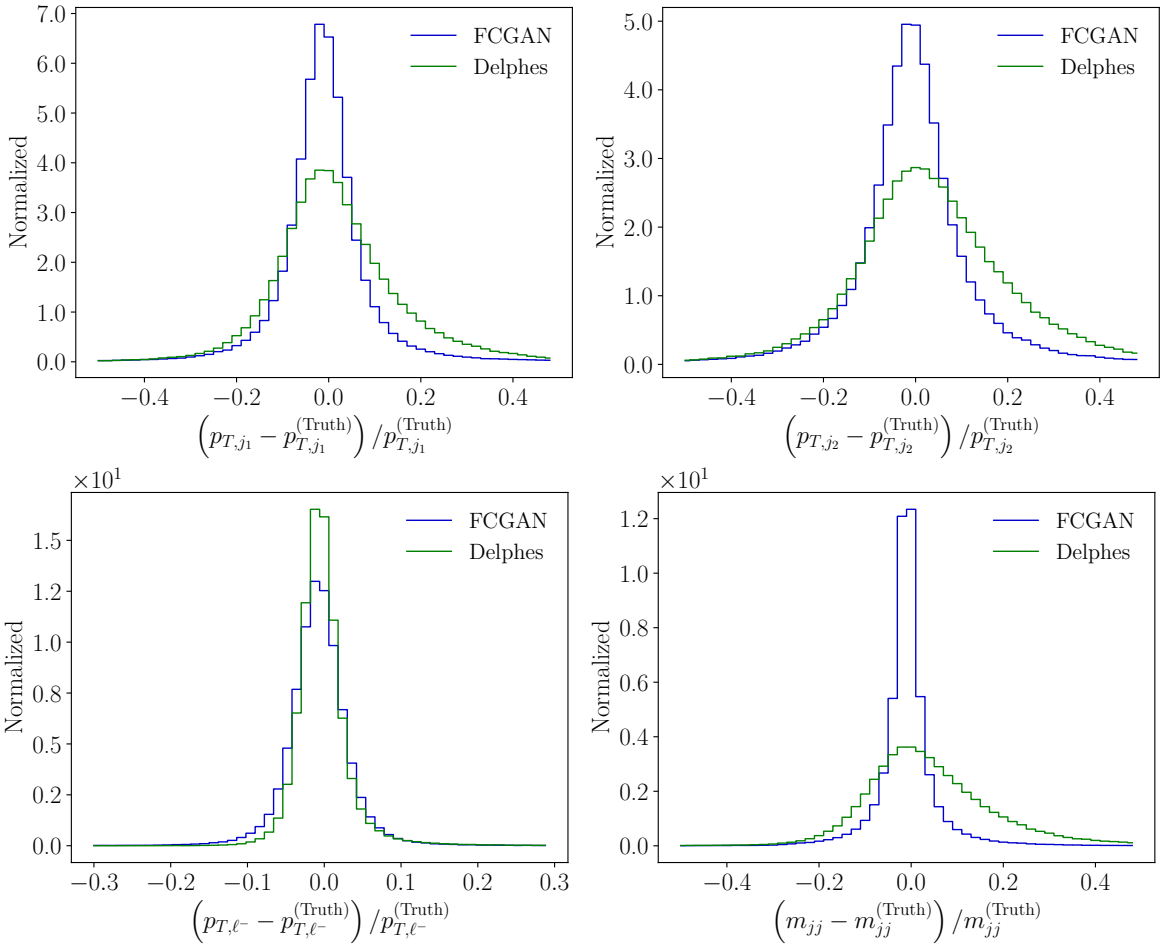

Figure A.1: Normalized deviation between the FCGANned sample and truth (residual) for some of the kinematic variables.

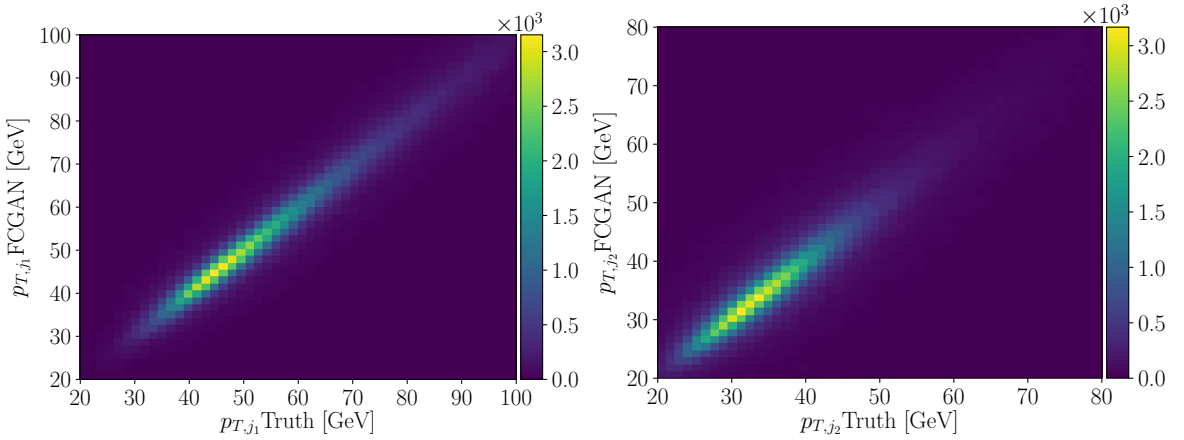

Figure A.2: Correlations between the FCGAN-inverted and parton-level truth kinematics, or migration matrix.

the DELPHES and FCGAN-inverted kinematics, for instance

$$\frac{p_{T,j}^{(\text{FCGAN})} - p_{T,j}^{(\text{Truth})}}{p_{T,j}^{(\text{Truth})}} \quad \text{and} \quad \frac{p_{T,j}^{(\text{DELPHES})} - p_{T,j}^{(\text{Truth})}}{p_{T,j}^{(\text{Truth})}} \,. \tag{A.1}$$

The events shown in these histograms correspond to the full phase space inversion shown in Fig. 2.6, but from the discussion in the main text it is clear that the picture does not change when we invert only part of phase space. As expected, we see narrow peaks around zero, with a width in the $\pm 10\%$ range for the jet momenta and much more narrow for the leptons, which are less affected by detector smearing. For all distributions, but especially the reconstructed W-mass, we see that the FCGAN reconstruction is significantly closer to the parton-level truth than the DELPHES events are.

Finally, we show the migration matrix or correlation between true parton-level and reconstructed parton-level events in terms of some of the kinematic variables in Fig. A.2. Not surprisingly, we observe narrow diagonal lines.

A.2 Staggered vs cooling MMD

The MMD loss is a two-sample test looking at the distance between samples x, x', drawn independently and identically distributed, in terms of a kernel function $k(x, x')$. Implementations of such a kernel, as given in Eq. 2.16, include a fixed width or resolution σ. We employ the MMD loss to reproduce the invariant mass distribution of intermediate on-shell particles M_p. A natural choice of σ is the corresponding particle width. However, this is inefficient at the beginning of the training, when any generated invariant mass M_G is essentially a random uniform distribution. In that case $(x - x')^2 \gg \sigma^2$ for any $x, x' \sim M_G$, and Eq. 2.15 reduces to

$$\text{MMD}\,(k; M_G, M_P) \simeq \sqrt{\langle k\,(y, y')\rangle_{y,y' \sim M_P}} \simeq \text{const}\,, \tag{A.2}$$

and provides little to no gradient.

This can be avoided by computing the MMD loss using multiple kernels with decreasing widths, so that the early training can be driven by wide kernels. A drawback of this approach is that only the small subset of kernels with a resolution close to the evolving width of M_G gives a non-negligible gradient.

Alternatively, we can employ a cooling kernel, which we initialize to some large value and then shrink to the correct particle width. This is an efficient solution at all stages of the training. A subtlety is that the rate of the cooling has to follow the pace of the generator in producing narrower invariant

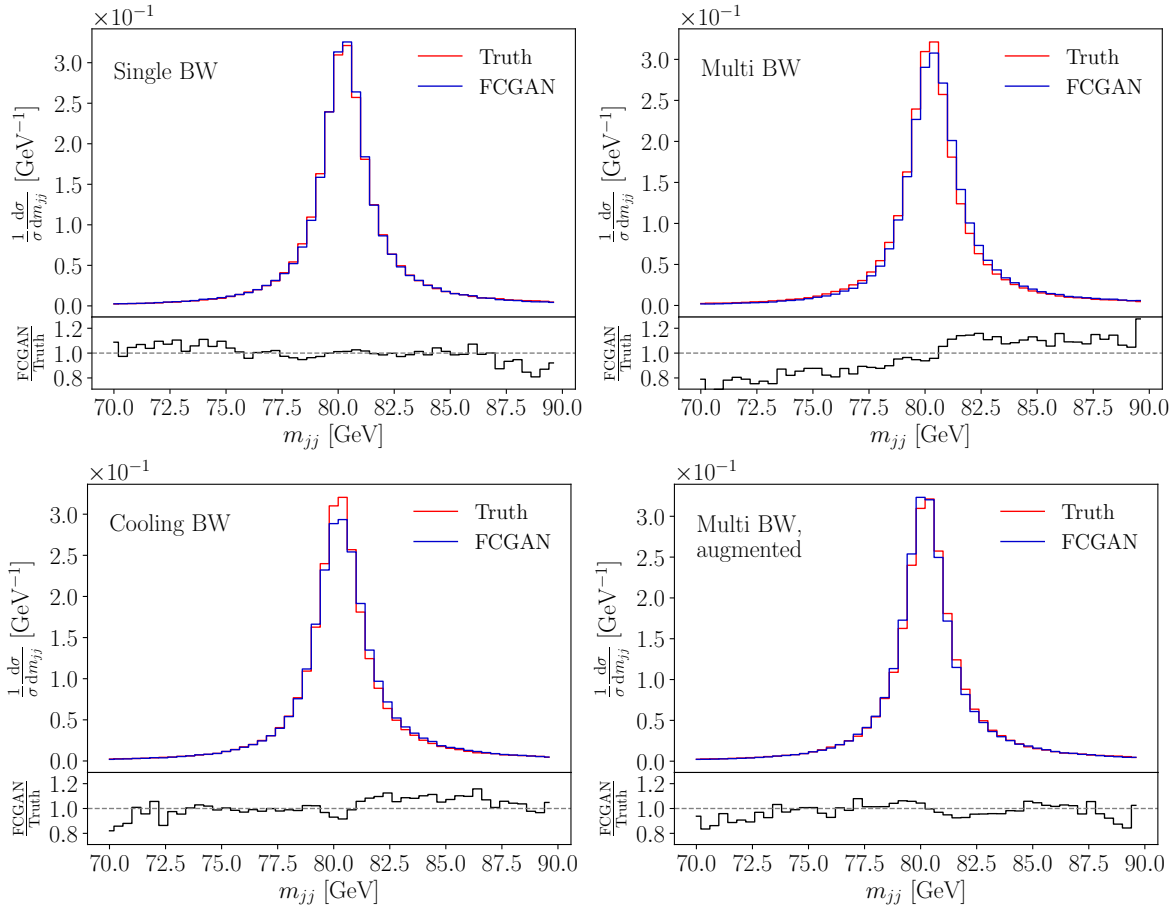

Figure A.3: Invariant jet-jet mass distribution for different MMD loss implementations: single kernel (upper left), multiple kernels (upper right), cooling kernel (lower left) and augmented multiple kernels (lower right).

mass distributions. Ultimately, we want to avoid hand-crafting the cooling process, because it adds hyper-parameters we need to tune. We use a dynamic kernel width as a fixed fraction of the standard deviation of the M_G distribution. This standard deviation as an estimate of the width of M_G can be replaced by any measure of the shape of M_G, such as the full width at half maximum, and our tests show that the performance is largely insensitive to the choice of the fraction.

Yet another approach is based on the observation that the MMD kernel test is not restricted to one-dimensional distributions [121, 177, 178]. This allows us to improve the invariant mass reconstruction by including additional physical information x_i to the test, so that the discrepancy is not computed just between the samples M_P and M_G of real and generated invariant masses, but rather between (M_P, x_P) and (M_G, x_G). In the FCGAN spirit we therefore augment the batches of true and generated invariant masses with one of conditional invariant masses. From the same detector information used to condition the generator and the discriminator, we can extract the detector level invariant masses M_D and accordingly compute $\text{MMD}(k; (M_G, M_D), (M_P, M_D))$. Even tough this does not represent a conditional MMD, training with multiple kernels benefits from using the augmented batches. In Fig. A.3 we compare the same invariant mass distribution using these different MMD implementations.

References

[1] HEP Software Foundation collaboration. HEP Software Foundation Community White Paper Working Group - Detector Simulation. Technical report, Mar 2018.

[2] HEP Software Foundation collaboration. HL-LHC Computing Review: Common Tools and Community Software. Technical report, Aug 2020. 40 pages contribution to Snowmass 2021.

[3] P Calafiura, J Catmore, D Costanzo, and A Di Girolamo. ATLAS HL-LHC Computing Conceptual Design Report. Technical report, CERN, Geneva, Sep 2020.

[4] The new Fast Calorimeter Simulation in ATLAS. Technical report, CERN, Geneva, Jul 2018.

[5] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi. DELPHES 3, A modular framework for fast simulation of a generic collider experiment. *JHEP*, 02:057, 2014.

[6] J. Alwall et al. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *JHEP*, 07:079, 2014.

[7] Torbjorn Sjostrand, Stephen Mrenna, and Peter Z. Skands. A Brief Introduction to PYTHIA 8.1. *Comput. Phys. Commun.*, 178:852–867, 2008.

[8] Enrico Bothmann et al. Event Generation with Sherpa 2.2. *SciPost Phys.*, 7(3):034, 2019.

[9] GEANT4 collaboration. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003.

[10] Martin Erdmann, Lukas Geiger, Jonas Glombitza, and David Schmidt. Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks. *Comput. Softw. Big Sci.*, 2(1):4, 2018.

[11] Martin Erdmann, Jonas Glombitza, and Thorben Quast. Precise simulation of electromagnetic calorimeter showers using a Wasserstein Generative Adversarial Network. *Comput. Softw. Big Sci.*, 3:4, 2019.

[12] Martin Erdmann, Lukas Geiger, Jonas Glombitza, and David Schmidt. Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks. *Comput. Softw. Big Sci.*, 2(1):4, 2018.

[13] Suyong Choi and Jae Hoon Lim. A data-driven event generator for hadron colliders using wasserstein generative adversarial network. *Journal of the Korean Physical Society*, 78(6):482–489, Feb 2021.

[14] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters. *Phys. Rev. Lett.*, 120(4):042003, 2018.

[15] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. *Phys. Rev.*, D97(1):014021, 2018.

[16] Jesus Arjona Martínez, Thong Q Nguyen, Maurizio Pierini, Maria Spiropulu, and Jean-Roch Vlimant. Particle generative adversarial networks for full-event simulation at the lhc and their application to pileup description. *Journal of Physics: Conference Series*, 1525:012081, Apr 2020.

[17] Stefano Carrazza and Frédéric A. Dreyer. Lund jet images from generative and cycle-consistent adversarial networks. *The European Physical Journal C*, 79(11), Nov 2019.

[18] Riccardo Di Sipio, Michele Faucci Giannelli, Sana Ketabchi Haghighat, and Serena Palazzo. DijetGAN: A Generative-Adversarial Network Approach for the Simulation of QCD Dijet Events at the LHC. 2019.

[19] Riccardo Di Sipio, Michele Faucci Giannelli, Sana Ketabchi Haghighat, and Serena Palazzo. DijetGAN: A Generative-Adversarial Network Approach for the Simulation of QCD Dijet Events at the LHC. *JHEP*, 08:110, 2020.

[20] Sascha Diefenbacher, Engin Eren, Gregor Kasieczka, Anatolii Korol, Benjamin Nachman, and David Shih. DCTRGAN: Improving the Precision of Generative Models with Reweighting. 9 2020.

[21] Erik Buhmann, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Anatolii Korol, and Katja Krüger. Getting high: High fidelity simulation of high granularity calorimeters with high speed. *Computing and Software for Big Science*, 5(1), May 2021.

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

[23] Christina Gao, Joshua Isaacson, and Claudius Krause. i- flow: High-dimensional integration and sampling with normalizing flows. *Machine Learning: Science and Technology*, 1(4):045023, Nov 2020.

[24] Christina Gao, Stefan Höche, Joshua Isaacson, Claudius Krause, and Holger Schulz. Event generation with normalizing flows. *Physical Review D*, 101(7), Apr 2020.

[25] Kosei Dohi. Variational autoencoders for jet simulation, 2020.

[26] Anders Andreassen, Patrick T. Komiske, Eric M. Metodiev, Benjamin Nachman, and Jesse Thaler. OmniFold: A Method to Simultaneously Unfold All Observables. *Phys. Rev. Lett.*, 124(18):182001, 2020.

[27] Kaustuv Datta, Deepak Kar, and Debarati Roy. Unfolding with Generative Adversarial Networks. 2018.

[28] Marco Bellagente, Anja Butter, Gregor Kasieczka, Tilman Plehn, and Ramon Winterhalder. How to GAN away Detector Effects. *SciPost Phys.*, 8(4):070, 2020.

[29] Lynton Ardizzone, Radek Mackowiak, Carsten Rother, and Ullrich Köthe. Training normalizing flows with the information bottleneck for competitive generative classification. 2021.

[30] Radek Mackowiak, Lynton Ardizzone, Ullrich Köthe, and Carsten Rother. Generative classifiers as a basis for trustworthy image classification. 2020.

[31] Harold Erbin and Sven Krippendorf. Gans for generating eft models. *Physics Letters B*, 810:135798, Nov 2020.

[32] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? 2019.

[33] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Why normalizing flows fail to detect out-of-distribution data. 2020.

[34] Joan Serrà, David Álvarez, Vicenç Gómez, Olga Slizovskaia, José F. Núñez, and Jordi Luque. Input complexity and out-of-distribution detection with likelihood-based generative models. 2020.

[35] Jiaming Song, Yang Song, and Stefano Ermon. Unsupervised out-of-distribution detection with batch normalization. 2019.

[36] Stefano Carrazza, Juan Cruz-Martinez, and Tanjona R. Rabemananjara. Compressing pdf sets using generative adversarial networks. *The European Physical Journal C*, 81(6), Jun 2021.

[37] Oliver Knapp, Guenther Dissertori, Olmo Cerri, Thong Q. Nguyen, Jean-Roch Vlimant, and Maurizio Pierini. Adversarially learned anomaly detection on cms open data: re-discovering the top quark. 2020.

[38] Adrian Alan Pol, Victor Berger, Gianluca Cerminara, Cecile Germain, and Maurizio Pierini. Anomaly detection with conditional variational autoencoders. 2020.

[39] Taoli Cheng, Jean-François Arguin, Julien Leissner-Martin, Jacinthe Pilette, and Tobias Golling. Variational autoencoders for anomalous jet tagging, 2021.

[40] M. Crispim Romão, N. F. Castro, and R. Pedro. Finding new physics without learning about it: anomaly detection as a tool for searches at colliders. *The European Physical Journal C*, 81(1), Jan 2021.

[41] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. 2014.

[42] Marco Bellagente, Anja Butter, Gregor Kasieczka, Tilman Plehn, Armand Rousselot, Ramon Winterhalder, Lynton Ardizzone, and Ullrich Köthe. Invertible Networks or Partons to Detector and Back Again. *SciPost Phys.*, 9:074, 2020.

[43] Marco Bellagente, Manuel Haußmann, Michel Luchmann, and Tilman Plehn. Understanding Event-Generation Networks via Uncertainties. 4 2021.

[44] Ramon Winterhalder, Marco Bellagente, and Benjamin Nachman. Latent Space Refinement for Deep Generative Models.

[45] Douglas Reynolds. *Gaussian Mixture Models*, pages 659–663. Springer US, Boston, MA, 2009.

[46] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[47] Radford M. Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, pages 355–368, 1999.

[48] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W. Pellegrini, Ralf S. Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. 2018.

[49] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. 2016.

[50] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. 2018.

[51] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France, 07–09 Jul 2015. PMLR.

[52] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. 2019.

[53] Ivan Kobyzev, Simon Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. 2019.

[54] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. 2018.

[55] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. 2018.

[56] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. 2019.

[57] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. Jun 2014.

[58] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, Jan 2018.

[59] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. 2017.

[60] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. 2017.

[61] Jae Hyun Lim and Jong Chul Ye. Geometric gan, 2017.

[62] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2019.

[63] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2014.

[64] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[65] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. *Proceedings of the 32nd International Conference on Machine Learning*, 37:1530–1538, 07–09 Jul 2015.

[66] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. 2014.

[67] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow. 2017.

[68] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. 2018.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2015.

[70] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations, 2017.

[71] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks, 2018.

[72] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. 2019.

[73] Chin-Wei Huang, Laurent Dinh, and Aaron Courville. Augmented normalizing flows: Bridging the gap between generative flows and latent variable models, 2020.

[74] G. D'Agostini. A Multidimensional unfolding method based on Bayes' theorem. *Nucl. Instrum. Meth.*, A362:487–498, 1995.

[75] G. D'Agostini. Improved iterative bayesian unfolding. 2010.

[76] Anders Andreassen and Benjamin Nachman. Neural Networks for Full Phase-space Reweighting and Parameter Tuning. *Phys. Rev. D*, 101(9):091901, 2020.

[77] Johann Brehmer, Kyle Cranmer, Gilles Louppe, and Juan Pavez. Constraining effective field theories with machine learning. *Physical Review Letters*, 121(11), Sep 2018.

[78] Johann Brehmer, Kyle Cranmer, Gilles Louppe, and Juan Pavez. A guide to constraining effective field theories with machine learning. *Physical Review D*, 98(5), Sep 2018.

[79] Johann Brehmer, Gilles Louppe, Juan Pavez, and Kyle Cranmer. Mining gold from implicit models to improve likelihood-free inference. *Proceedings of the National Academy of Sciences*, 117(10):5242–5249, Feb 2020.

[80] Kyle Cranmer, Juan Pavez, and Gilles Louppe. Approximating likelihood ratios with calibrated discriminative classifiers. 2016.

[81] John Campbell, Joey Huston, and Frank Krauss. *The Black Book of Quantum Chromodynamics*. Oxford University Press, 2017.

[82] Matthew D. Klimek and Maxim Perelstein. Neural Network-Based Approach to Phase Space Integration. 2018.

[83] Joshua Bendavid. Efficient Monte Carlo Integration Using Boosted Decision Trees and Generative Deep Neural Networks. 2017.

[84] Fady Bishara and Marc Montull. (Machine) Learning Amplitudes for Faster Event Generation. 12 2019.

[85] Simon Badger and Joseph Bullock. Using neural networks for efficient evaluation of high multiplicity scattering amplitudes. *JHEP*, 06:114, 2020.

[86] Sydney Otten et al. Event Generation and Statistical Sampling with Deep Generative Models and a Density Information Buffer. 2019.

[87] Bobak Hashemi, Nick Amin, Kaustuv Datta, Dominick Olivito, and Maurizio Pierini. LHC analysis-specific datasets with Generative Adversarial Networks. 2019.

[88] Anja Butter, Tilman Plehn, and Ramon Winterhalder. How to GAN LHC Events. *SciPost Phys.*, 7:075, 2019.

[89] Yasir Alanazi, N. Sato, Tianbo Liu, W. Melnitchouk, Michelle P. Kuchera, Evan Pritchard, Michael Robertson, Ryan Strauss, Luisa Velasco, and Yaohang Li. Simulation of electron-proton scattering events by a Feature-Augmented and Transformed Generative Adversarial Network (FAT-GAN). 1 2020.

[90] Pasquale Musella and Francesco Pandolfi. Fast and Accurate Simulation of Particle Detectors Using Generative Adversarial Networks. *Comput. Softw. Big Sci.*, 2(1):8, 2018.

[91] ATLAS Collaboration. Deep generative models for fast shower simulation in ATLAS. ATL-SOFT-PUB-2018-001, 2018. http://cds.cern.ch/record/2630433.

[92] ATLAS Collaboration. Energy resolution with a GAN for Fast Shower Simulation in ATLAS. ATLAS-SIM-2019-004, 2019. https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/SIM-2019-004/.

[93] Dawit Belayneh et al. Calorimetry with Deep Learning: Particle Simulation and Reconstruction for Collider Physics. 12 2019.

[94] Erik Buhmann, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Anatolii Korol, and Katja Krüger. Getting High: High Fidelity Simulation of High Granularity Calorimeters with High Speed. 5 2020.

[95] Enrico Bothmann and Luigi Debbio. Reweighting a parton shower using a neural network: the final-state case. *JHEP*, 01:033, 2019.

[96] Luke de Oliveira, Michela Paganini, and Benjamin Nachman. Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis. *Comput. Softw. Big Sci.*, 1(1):4, 2017.

[97] J. W. Monk. Deep Learning as a Parton Shower. *JHEP*, 12:021, 2018.

[98] Anders Andreassen, Ilya Feige, Christopher Frye, and Matthew D. Schwartz. JUNIPR: a Framework for Unsupervised Machine Learning in Particle Physics. *Eur. Phys. J.*, C79(2):102, 2019.

[99] Joshua Lin, Wahid Bhimji, and Benjamin Nachman. Machine Learning Templates for QCD Factorization in the Search for Physics Beyond the Standard Model. *JHEP*, 05:181, 2019.

[100] Enrico Bothmann, Timo Janßen, Max Knobbe, Tobias Schmale, and Steffen Schumann. Exploring phase space with Neural Importance Sampling. 1 2020.

[101] Christina Gao, Joshua Isaacson, and Claudius Krause. i-flow: High-Dimensional Integration and Sampling with Normalizing Flows. 1 2020.

[102] Christina Gao, Stefan Höche, Joshua Isaacson, Claudius Krause, and Holger Schulz. Event Generation with Normalizing Flows. *Phys. Rev. D*, 101(7):076002, 2020.

[103] Benjamin Nachman and David Shih. Anomaly Detection with Density Estimation. *Phys. Rev. D*, 101:075042, 2020.

[104] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks. 2019.

[105] Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. 2019.

[106] Nikolai D. Gagunashvili. Machine learning approach to inverse problem and unfolding procedure. 4 2010.

[107] Francesco Spano. Unfolding in particle physics: a window on solving inverse problems. *EPJ Web Conf.*, 55:03002, 2013.

[108] Alexander Glazov. Machine learning as an instrument for data unfolding. 12 2017.

[109] G. Cowan. A survey of unfolding methods for particle physics. *Conf. Proc. C*, 0203181:248–257, 2002.

[110] Volker Blobel. *Unfolding Methods in Particle Physics*. Jan 2011.

[111] Rahul Balasubramanian, Lydia Brenner, Carsten Burgard, Glen Cowan, Vincent Croft, Wouter Verkerke, and Pim Verschuuren. Statistical method and comparison of different unfolding techniques using RooFit. 2019.

[112] L. B. Lucy. An iterative technique for the rectification of observed distributions. *Astronomical Journal*, 79(6):745, 1974.

[113] G. Zech. Iterative unfolding with the richardson–lucy algorithm. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 716:1–9, Jul 2013.

[114] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O. Rasmussen, and Peter Z. Skands. An Introduction to PYTHIA 8.2. *Comput. Phys. Commun.*, 191:159–177, 2015.

[115] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. The anti-k_t jet clustering algorithm. *JHEP*, 04:063, 2008.

[116] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. FastJet User Manual. *Eur. Phys. J. C*, 72:1896, 2012.

[117] François Chollet. `https://github.com/fchollet/keras`, 2015.

[118] Martín Abadi et al. Tensorflow: A system for large-scale machine learning. *CoRR*, 2016.

[119] Kevin Roth, Aurélien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. *CoRR*, 2017.

[120] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel method for the two-sample problem. *CoRR*, 2008.

[121] Yujia Li, Kevin Swersky, and Richard Zemel. Generative Moment Matching Networks. page arXiv:1502.02761, 2015.

[122] Anke Biekötter, Alexander Knochel, Michael Krämer, Da Liu, and Francesco Riva. Vices and virtues of Higgs effective field theories at large energy. *Phys. Rev.*, D91:055029, 2015.

[123] Johann Brehmer, Ayres Freitas, David Lopez-Val, and Tilman Plehn. Pushing Higgs Effective Theory to its Limits. *Phys. Rev.*, D93(7):075014, 2016.

[124] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[125] Matthew R. Buckley, Tilman Plehn, and Michael J. Ramsey-Musolf. Top squark with mass close to the top quark. *Phys. Rev. D*, 90(1):014046, 2014.

[126] Erik Buhmann, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Anatolii Korol, and Katja Krüger. Decoding Photons: Physics in the Latent Space of a BIB-AE Generative Network. 2 2021.

[127] Kosei Dohi. Variational Autoencoders for Jet Simulation. 9 2020.

[128] Oliver Knapp, Guenther Dissertori, Olmo Cerri, Thong Q. Nguyen, Jean-Roch Vlimant, and Maurizio Pierini. Adversarially Learned Anomaly Detection on CMS Open Data: re-discovering the top quark. 5 2020.

[129] Francesco Armando Di Bello, Sanmay Ganguly, Eilam Gross, Marumi Kado, Michael Pitt, Lorenzo Santi, and Jonathan Shlomi. Towards a Computer Vision Particle Flow. *Eur. Phys. J. C*, 81(2):107, 2021.

[130] Pierre Baldi, Lukas Blecher, Anja Butter, Julian Collado, Jessica N. Howard, Fabian Keilbach, Tilman Plehn, Gregor Kasieczka, and Daniel Whiteson. How to GAN Higher Jet Resolution. 12 2020.

[131] Johann Brehmer and Kyle Cranmer. Flows for simultaneous manifold learning and density estimation. 3 2020.

[132] Stefan T. Radev, Ulf K. Mertens, Andreass Voss, Lynton Ardizzone, and Ullrich Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. 2020.

[133] David MacKay. Probable Networks and Plausible Predictions – A Review of Practical Bayesian Methods for Supervised Neural Networks. *Comp. in Neural Systems*, 6:4679, 1995.

[134] Radford M. Neal. *Bayesian learning for neural networks*. PhD thesis, Toronto, 1995.

[135] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, Cambridge, 2016.

[136] Alex Kendall and Yarin Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? *Proc. NIPS*, 2017.

[137] Pushpalatha C. Bhat and Harrison B. Prosper. Bayesian neural networks. *Conf. Proc. C*, 050912:151, 2005.

[138] Skyler Richard Saucedo. Bayesian Neural Networks for Classification. Master's thesis, Florida State University, 2007.

[139] Ye Xu, Weiwei Xu, Yixiong Meng, Kaien Zhu, and Wei Xu. Applying Bayesian Neural Networks to Event Reconstruction in Reactor Neutrino Experiments. *Nucl. Instrum. Meth. A*, 592:451, 2008.

[140] Sven Bollweg, Manuel Haußmann, Gregor Kasieczka, Michel Luchmann, Tilman Plehn, and Jennifer Thompson. Deep-Learning Jets with Uncertainties and More. *SciPost Phys.*, 8(1):006, 2020.

[141] Gregor Kasieczka, Michel Luchmann, Florian Otterpohl, and Tilman Plehn. Per-Object Systematics using Deep-Learned Calibration. 3 2020.

[142] Benjamin Nachman. A guide for deploying Deep Learning in LHC searches: How to achieve optimality and account for uncertainty. 9 2019.

[143] Jack Y. Araz and Michael Spannowsky. Combine and Conquer: Event Reconstruction with Bayesian Ensemble Neural Networks. 2 2021.

[144] David JC MacKay. Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469–505, 1995.

[145] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[146] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[147] Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors. In *International Conference on Machine Learning*, pages 1744–1753. PMLR, 2018.

[148] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[149] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[150] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[151] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015.

[152] Richard D. Ball, Valerio Bertone, Stefano Carrazza, Luigi Del Debbio, Stefano Forte, Alberto Guffanti, Nathan P. Hartland, and Juan Rojo. Parton distributions with QED corrections. *Nucl. Phys.*, B877:290–320, 2013.

[153] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *International Conference on Machine Learning*, 37:1530, 2015.

[154] Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your gan is secretly an energy-based model and you should use discriminator driven latent sampling. 2020.

[155] 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:P11004, 2020.

[156] Vanessa Böhm and U Seljak. Probabilistic auto-encoder, 2020.

[157] Zhisheng Xiao, Qing Yan, and Yali Amit. Generative latent flow, 2019.

[158] Bin Dai and David Wipf. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations*, 2019.

[159] Omer Bobrowski and Sayan Mukherjee. The topology of probability distributions on manifolds. *Probability Theory and Related Fields*, 161, 07 2013.

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

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

[162] Rob Cornish, Anthony L. Caterini, George Deligiannidis, and Arnaud Doucet. Relaxing bijectivity constraints with continuously indexed normalising flows, 2021.

[163] Joshua Batson, C. Grace Haaf, Yonatan Kahn, and Daniel A. Roberts. Topological Obstructions to Autoencoding. 2 2021.

[164] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[165] Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori. *Density Ratio Estimation in Machine Learning*. Cambridge University Press, 2012.

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

[167] Raffaele Tito D'Agnolo and Andrea Wulzer. Learning New Physics from a Machine. *Phys. Rev.*, D99(1):015014, 2019.

[168] Benjamin Nachman and Jesse Thaler. E Pluribus Unum Ex Machina: Learning from Many Collider Events at Once. 1 2021.

[169] Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216–222, 1987.

[170] Radford M. Neal. MCMC using Hamiltonian dynamics. 2012.

[171] Mathias Backes, Anja Butter, Tilman Plehn, and Ramon Winterhalder. How to GAN Event Unweighting. 12 2020.

[172] Rob Verheyen and Bob Stienen. Phase Space Sampling and Inference from Weighted Events with Autoregressive Flows. 11 2020.

[173] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[174] Laurent Dinh, Jascha Sohl-Dickstein, and Samy 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.

[175] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[176] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[177] Suman Ravuri, Shakir Mohamed, Mihaela Rosca, and Oriol Vinyals. Learning Implicit Generative Models with the Method of Learned Moments. page arXiv:1806.11006, 2018.

[178] Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. Training generative neural networks via Maximum Mean Discrepancy optimization. page arXiv:1505.03906, 2015.