Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Sander Hummerich

born in Heidelberg (Germany)

2021

Hybrid Flows for LHC Events

Increasing Precision of LHC Event Generation with Normalizing Flows Using Hybrid Objectives

This bachelor thesis has been carried out by Sander Hummerich at the
Institute for Theoretical Physics Heidelberg
under the supervision of
Prof. Tilman Plehn

Abstract

At the LHC experiments, proper big data is produced, entailing the need for fast analysis tools. For event generation, the long established Monte Carlo methods cannot keep pace with the rising amount of future collider runs. Modern approaches use machine learning models for more efficient event generation. In this thesis, we will show how to add precision to event generation of LHC processes with normalizing flows using hybrid objectives.

Zusammenfassung

In den LHC Experimenten werden große Datenmengen erzeugt, für deren Verarbeitung schnelle Analyse-Methoden benötigt werden. Etablierte Monte-Carlo-Methoden, zur Simulation von LHC Prozessen, können nur schwer mit den steigenden Datenmengen von zukünftigen Experimenten schritthalten. Modernere Methoden bedienen sich Modellen des maschinellen Lernens, um die die Effizienz der Simulationen zu erhöhen. In dieser Arbeit wird gezeigt, wie man mithilfe hybriden Trainings die Präzision von generativen, invertierbaren neuronalen Netzwerken erhöhen kann.

Contents

1 Introduction

The *Standard Model of Particle Physics* (SM), which describes the fundamentals of our universe with *quarks*, *leptons* and *bosons*, has been established in numerous experiments. A concluding moment was the discovery of the Higgs Boson at the *Large Hardron Collider* (LHC) at CERN. In search of new physics beyond the SM, new experiments at the LHC are already in progress, while data from old events is not even analyzed yet. To analyze the enormous amounts of data produced in past and future collider runs, powerful tools, such as algorithms for event generation, are needed. Until now, the established methods are Monte Carlo Simulation based event generators such as *Madgraph* [1] et cetera. More modern techniques use machine learning models for event simulations. Using density estimation techniques, generating collider events becomes more efficient than Monte Carlo based methods. Still, since the generation process does not rely on a physical background but only on the learned distribution, the density estimators often lack precision on multidimensional physical correlations.

Normalizing flow models or invertible neural networks find a wide range of applications in the scientific world. These range from parameter estimation of pandemic models to speech synthesis . The unique invertible structure of flow-based models gives the opportunity to solve problems in an invertible manner. For LHC physics, many neural network based approaches for various kinds of applications have been proposed in recent years. These include generative adversarial networks variational autoencoders and normalizing flows applied to event generation [2], unfolding [3, 4], event unweighting [5], super resolution for jet images [6] or jet tagging [7, 8].

Extending the previous approaches we will use a normalizing flow model for event generation. Furthermore, we show how an additional *hybrid* objective might help to obtain control over the generation process and to nudge the model towards learning highly correlated data correctly. In this context, we combine a usual normalizing flow model with an adversarial discriminative network as proposed in [9]. Our network setup will be applied to event generation of the well-known Drell-Yan Process as well as WZ-diboson production.

2 Neural Networks

1 Introduction to Neural Networks

In this section, the fundamentals of neural networks will be discussed.

The basic building block of a neural network is a node referred to as a *neuron*. A neuron gets inputs x_i, $i \in \{1, ..., n\}$ and transforms it to an output z. This transformation is usually split into two parts as illustrated in figure 2.1. First, a linear preactivation is applied, and then a in general nonlinear activation function is applied to the preactivated input:

$$\text{Preactivation: } \tilde{z} = \beta \cdot \mathbf{x} + b, \tag{2.1}$$

$$\text{Activation: } z = \Phi(\tilde{z}). \tag{2.2}$$

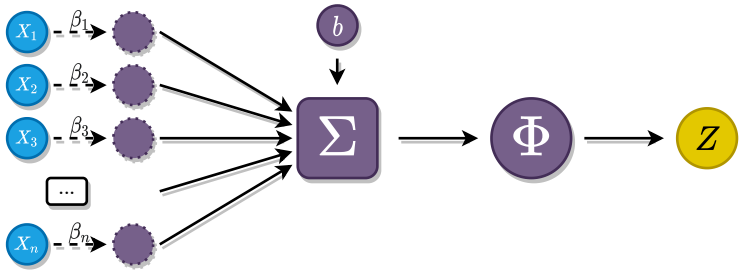
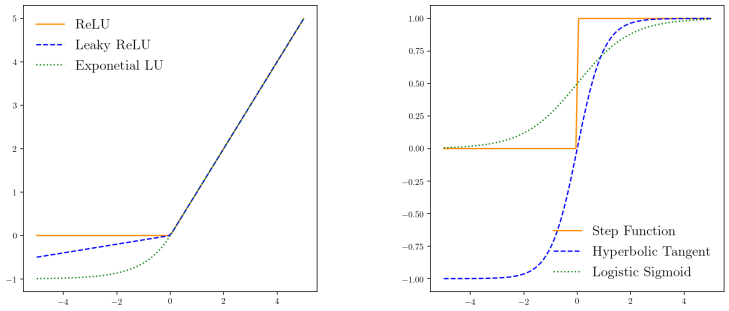


Figure 2.1: Single neuron

β is called the *weight vector* which has the same dimension as the input $\mathbf{x}$. The product of β and $\mathbf{x}$ is a scalar product. The *bias* b is a scalar offset. Common choices for the activation function are the *Rectified Linear Unit* (ReLU) and its variants, such as the *Exponential Linear Unit* (ELU), as well as the *logistic sigmoid function* (σ):

$$\text{ReLU}(\tilde{z}) = \begin{cases} \tilde{z} & \text{if } \tilde{z} > 0 \\ 0 & \text{else} \end{cases}, \quad \text{ELU}(\tilde{z}) = \begin{cases} \tilde{z} & \text{if } \tilde{z} > 0 \\ \rho e^{\tilde{z}-1} & \text{else} \end{cases}, \quad \sigma(\tilde{z}) = \frac{1}{1+e^{\tilde{z}}}. \tag{2.3}$$

These three activation functions as well as other widely used activation functions are shown in figure 2.2.



(a) (b)

Figure 2.2: Common activation functions (a) ReLU and its variations (b) other popular activation functions

Single neurons are arranged in subsequent layers $l \in \{1, ..., L\}$, holding neurons $m \in \{1, ..., M_l\}$. The data input, the output of the network, and all layers in between are referred to as *input layer*, *output layer*, and *hidden layers* respectively. For example in classification tasks, each hidden layer successively transforms the data such that the problem becomes linear in the output layer. It becomes clear that data with certain complexity can only be transformed to a linear problem if the depth L and widths M_l of the network are sufficiently big.

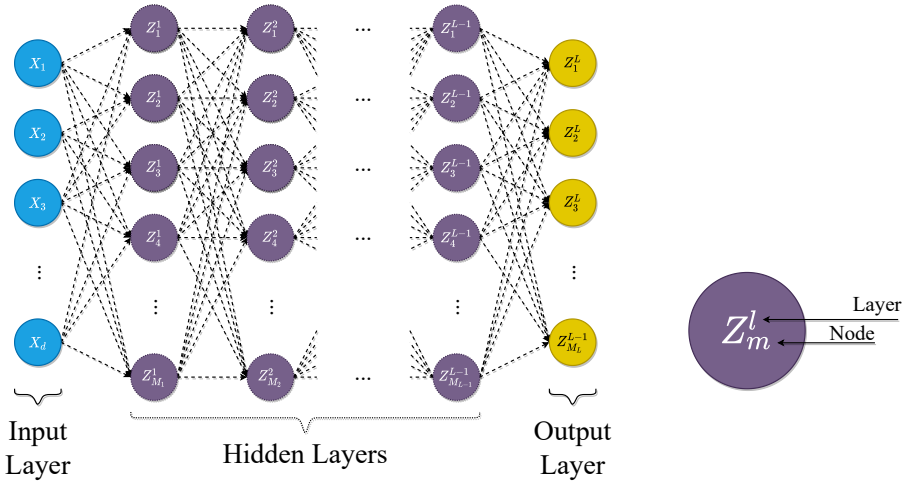


Figure 2.3: Fully connected neural network

In a bigger picture, a neural network can be seen as a transformation $f_\Theta : \mathbf{x} \mapsto \mathbf{z}$ with

parameters Θ, which have to be optimized by minimizing a cost function. Due to the non-linearity of the activation function the functional space on $\mathbb{R}^d$, being covered by a neural network, is huge. The *Universal Approximation Theorem* [10] states that an arbitrary function $f \in L_p = \{f|(\int_{\mathbb{R}} |f(x)|^p dx)^{\frac{1}{p}}\}$ can be approximated arbitrarily well by a sufficiently wide neural network.

2 Training Neural Networks

To optimize the parameters Θ of a neural network, one has numerous optimization algorithms to choose from. Still *Gradient Descent* is the fundamental concept behind all of them and therefore worth explaining.

The parameters Θ of a neural network are chosen such that a loss function $\mathcal{L}$ is minimized. For sufficient complex models these optimal parameters cannot be determined analytically. To optimize the parameters, one therefore computes the gradient of the loss $\nabla_{\Theta_i}\mathcal{L}$ with respect to each parameter and updates the parameters in the opposite direction of steepest descent:

$$\Theta_i^{[t]} \leftarrow \Theta_i^{[t-1]} - \alpha \nabla_{\Theta_i}\mathcal{L}. \tag{2.4}$$

Here we introduced the *learning rate* α, which determines the size of each update step. These updates are performed until the gradient vanishes. The goal of this method is to find the global loss minimum, which is not guaranteed since the gradient vanishes in a *bad local minimum* as well, as illustrated in figure 2.4.

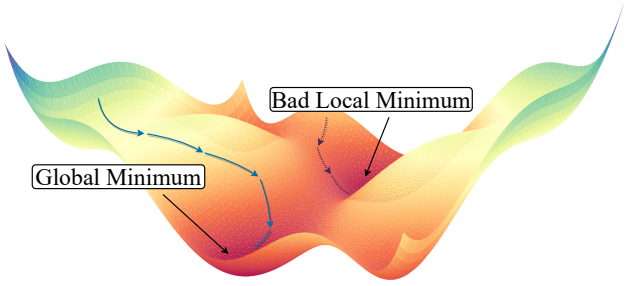


Figure 2.4: Stochastic Gradient Descent in a made up loss-landscape

Plain Gradient Descent often leads to bad convergence which can be improved by updating the parameters Θ more frequently. Therefore, the total training set $\{x_i\}_{i=1}^N$ is subdivided into small batches $\{B_1, ..., B_B\}$, performing gradient optimization steps for each of these batches successively. This method is called *stochastic Gradient Descent* (SGD) or *mini-batch Gradient Descent*.

Optimization algorithms like *Momentum* [11] build on this basic idea. Their purpose is to stabilize training by updating the parameters Θ using a *momentum* term, which is the exponential weighted moving average of the gradients so far:

$$\Theta_i^{[t]} \leftarrow \Theta_i^{[t-1]} - \alpha m^{[t]}, \qquad m^{[t]} = \beta_1 m^{[t-1]} + (1-\beta_1)\nabla_{\Theta_i}\mathcal{L}, \qquad \beta_1 \in [0,1]. \tag{2.5}$$

Through this momentum term, the resistance to statistical outliers and bad local minima is increased making the convergence more stable as illustrated in figure 2.5. The parameter β_1 regulates how much the newly computed gradient affects the optimization step. In static setups with a constant loss-landscape β_1 is chosen rather high, whereas in dynamical systems such as a *Generative Adversarial Network*, which will be discussed later, β_1 must be chosen smaller to allow for some flexibility.

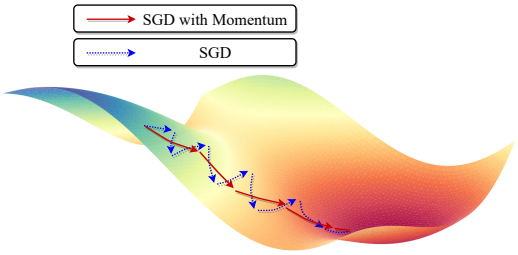


Figure 2.5: SGD in comparison to SGD with Momentum

The even more advanced *Adam* optimizer [12] improves on the fact that the momentum term leads to inattention to atypical samples, by introducing an acceleration term for each dimension separately:

$$v^{[t]} = \beta_2 v^{[t-1]} + (1-\beta_2)(\nabla_{\Theta_i}\mathcal{L})^2, \qquad \beta_2 \in [0,1]. \tag{2.6}$$

This acceleration term, which is the exponential weighted second moment of the gradient, serves as a measurement of the variance of the gradient along each direction. The gradient

steps along directions with high gradient are weighted by the inverse squareroot of the acceleration term:

$$\Theta_i^{[t]} \leftarrow \Theta_i^{[t-1]} - \alpha \frac{m^{[t]}}{\sqrt{v^{[t]}} + \epsilon}, \qquad \epsilon > 0. \tag{2.7}$$

The method to compute the gradient of the loss for every single parameter is *Backpropagation*. Starting with the gradient of the last layer, it computes the gradients of the previous layer using the chain rule and the already computed gradients [13].

In equation 2.4 the learning rate α was introduced. Training a neural network is a balancing act between *exploration* (large α) and *exploitation* (small α). A good learning rate scheduler speeds up convergence by large gradient steps in the beginning, while maintaining precision with small steps at the end. Schedulers used throughout this thesis are a *step* scheduler decreasing α after a fixed number of epochs and a *OneCycle* scheduler introduced by [14].

A neural network is trained on a set of data instances $\{x_i\}_{i=1}^N$ drawn from the true probability density distribution $P(x)$. The expectation value of the loss function is approximated by a mean over these data instances:

$$\langle ... \rangle_{x \sim P(x)} \approx \frac{1}{N} \sum_{i=1}^{N} ... \,. \tag{2.8}$$

Since neural networks can represent highly complex functions and we train on a finite number of data instances *overfitting* can occur. This happens if the network does not account for statistical fluctuations in the data, perfectly fitting each data point including statistical outliers. In figure 2.6 we show how a cubic spline interpolation overfitts a cosine correlation with some additional noise.

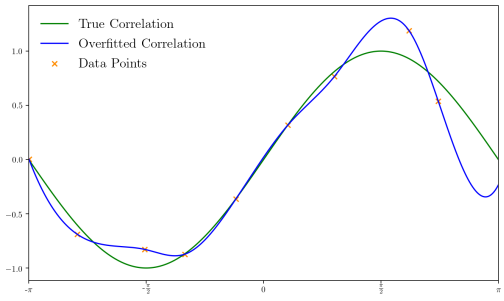


Figure 2.6: Simple example of overfitting

Methods to prevent overfitting and to stabilize training to a certain degree include *L1* and *L2 Regularization*, punishing large weights, *Spectral Normalization* [15], normalizing the input of each layer by the spectral norm, and *Dropout* [16], randomly setting parts of neuron inputs to zero.

Another significant factor in training a neural network successfully is preprocessing the input data. Thereby, it is important to ensure all data dimensions are of the same magnitude such that all dimensions are about equally significant for the optimization. This can be achieved by simply normalizing each dimension by subtracting the mean μ and dividing by the variance σ. Another powerful preprocessing step is called *Principal Component Analysis* (PCA), which transforms the data to a new coordinate system where the coordinate axes are in the direction of biggest variance in the data. Furthermore, applying specific functions to the data instances might help the network in some scenarios.

3 Unsupervised Learning and Generative Models

In machine learning, there are in general three types of problems: *supervised learning*, *unsupervised learning* and *reinforcement learning*. The major difference separating unsupervised learning from the other two is that in unsupervised learning the model does not learn correlations between data instances x_i, and labels y_i (supervised learning) or rewards r_i (reinforcement learning) but rather learns structures and correlations in the data itself.

In unsupervised learning, a key assumption is that the data is drawn from an unknown probability density distribution $P(x)$. The main learning motives are *density approximation*, where the probability density is learned, and *feature extraction*, where crucial properties of the distribution are learned [17].

The goal of generative models is to approximate the true probability density $P_\Theta(x) \approx P(x)$ by a model with parameters Θ. This model then can be used to generate new data instances that fit well into the learned data distribution. The most common objective to train a generative model results from the *Maximum Likelihood Principle* introduced by R. A. Fisher. The idea of this principle is to choose the model parameters Θ such that the likelihood of the true data $\{x_i\}_{i=1}^N$ given the model P_Θ and its parameters is maximized. The objective can be expressed as minimizing the *Negative Log-Likelihood* [18]:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \sum_{i=1}^{N} -\log[P_\Theta(x_i)]. \tag{2.9}$$

A big problem in physics is that the theories evolved so far work well in certain scenarios and limits, but many effects are not yet understood by theorists. Therefore, numerically finding some hidden structures in a data set might provide understanding beyond the known theoretical models. Furthermore, generative models yield the possibility to simulate huge amounts of data way faster than common Monte Carlo based methods, which is essential for future LHC and other collider runs.

4 Normalizing Flows

Normalizing flows or Invertible Neural Networks (INNs) [19] are models that realize a mapping f_Θ between two probability density distributions $P(x)$ and $P(z)$. In this thesis, they are used as a generative model, where $P(x)$ is the probability density the model should learn and $P(z)$ is a known probability density that can be chosen arbitrarily but usually is a multidimensional standard normal distribution $P(z) = \mathcal{N}(0, \mathbb{I})$ in the so-called *latent space*. The catch of normalizing flows is that the mapping f_Θ is invertible such that one can map from data space $\mathcal{X} \subseteq \mathbb{R}^{d_{data}}$ to latent space $\mathcal{Z} \subseteq \mathbb{R}^{d_{latent}}$ and vice versa:

$$f_\Theta : \mathcal{X} \to \mathcal{Z},\ x \mapsto z, \tag{2.10}$$

$$f_\Theta^{-1} : \mathcal{Z} \to \mathcal{X},\ z \mapsto x. \tag{2.11}$$

Therefore, the model can be trained in the forward direction ($\mathcal{X} \to \mathcal{Z}$) and generate in backward direction ($\mathcal{Z} \to \mathcal{X}$), sampling from the known latent space distribution.

To ensure the invertibility of f_Θ, it is constructed as a homeomorphism between the spaces $\mathcal{X}$ and $\mathcal{Z}$. This yields some restrictions for the model. Firstly, the dimension of the latent space has to be the same as the dimension of the data space because otherwise, according to the *Theorem of the Invariance of the Dimension*, $\mathcal{X}$ can not be homeomorphic to $\mathcal{Z}$:

$$d_{data} = d_{latent}. \tag{2.12}$$

Secondly, the network f_Θ has to be a bijection, which only can be attained if all layer transformations Φ_l are bijective:

$$f_\Theta = \Phi_L \circ ... \circ \Phi_1, \tag{2.13}$$

$$f_\Theta^{-1} = \Phi_1^{-1} \circ ... \circ \Phi_L^{-1}. \tag{2.14}$$

Due to the design of f_Θ as a homeomorphism the topological properties of the data in $\mathcal{X}$ are preserved in $\mathcal{Z}$.

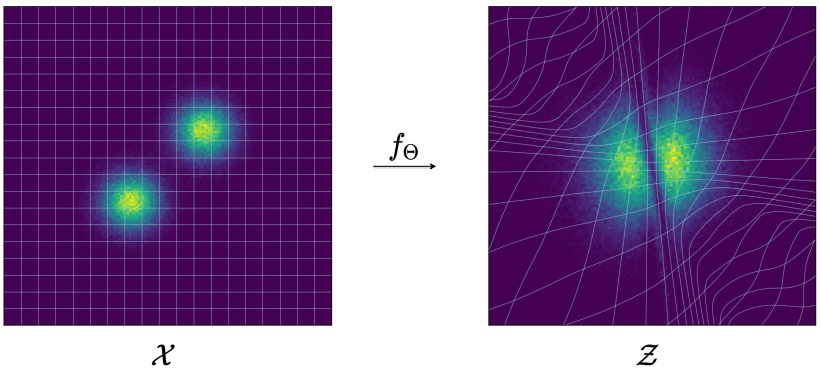


Figure 2.7: Topological transformation from data space $\mathcal{X}$ to latent space $\mathcal{Z}$ under f_Θ for a two dimensional toy model

Sampling $z \sim P(z)$ then approximates sampling from the true probability distribution $P(x)$ by the change of variables formula:

$$P(x) \approx P_\Theta(x) = P(z) \cdot |J_z|, \qquad J_z = \det\Big(\frac{\partial f_\Theta(x)}{\partial x}\Big). \tag{2.15}$$

The Jacobian accounts for the change of probability density under f_Θ. Naively computing J_z is quite expensive ($\mathcal{O}(d^3)$). Therefore, so-called *coupling layers* are used, not only implementing the invertibility but also ensuring that the Jacobian determinant is tractable. The transformation of an input x to an output z is done in the following procedure [19]:

1. Split the input $x = [x_1, ..., x_D] \rightarrow x_u = [x_1, ..., x_{\frac{D}{2}}], x_l = [x_{\frac{D}{2}+1}, ..., x_D]$
2. Compute some parameters $\theta(x_u)$ of a function h_θ depending on x_u
3. Transform $x_l \mapsto h_\theta(x_l)$, $x_u \mapsto x_u$
4. Output $z = [x_u, h_\theta(x_l)] = [z_u, z_l]$

This procedure is illustrated schematically in figure 2.8.

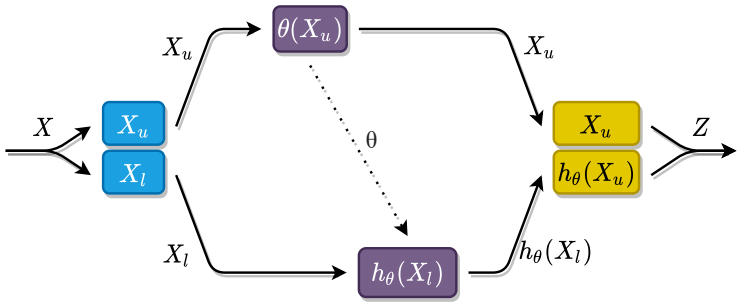


Figure 2.8: Coupling layer

The transformation h_θ is a homeomorphism, which is applied element wise. The parameters θ are usually computed by an arbitrary neural network. Because only transforming the last $\frac{D}{2}$ instances in each layer is not beneficial, permutation layers are built inbetween the coupling layers:

$$f_\Theta = \Phi_1 \circ O_1 \circ ... \circ O_{L-1} \circ \Phi_L. \tag{2.16}$$

In this thesis, we will use *soft*-permutation layers, which are randomly generated rotation matrices from $SO(d)$, as provided by [20].

Due to this layer design, the Jacobian matrix for each coupling layer becomes triangular and the determinant simply the product of the diagonal entries.

$$\frac{\partial \Phi_l(x)}{\partial x} = \begin{pmatrix} \mathbb{1} & \frac{\partial h_\theta(x_l)}{\partial x_u} \\ 0 & \frac{\partial h_\theta(x_l)}{\partial x_l} \end{pmatrix}, \qquad \frac{\partial h_\theta(x_l)}{\partial x_u} < \infty.$$

There are several possibilities to implement h_θ. Herinafter we want to discuss two of them.

Affine transformation: An affine transformation consists of an elementwise multiplication ($\odot$) with $s(x_u)$ and addition ($\oplus$) of $t(x_u)$:

$$x_l \mapsto x_l \odot s(x_u) \oplus t(x_u) = z_l, \tag{2.17}$$
$$z_l \mapsto (y_l \ominus t(x_u)) \oslash s(x_u) = x_l. \tag{2.18}$$

This transformation is invertible regardless of the functions $s(\cdot)$ and $t(\cdot)$, which therefore can be implemented as non-invertible neural networks [19]. For reasons of numerical stability two adjustments to equation 2.18 are implemented:

$$s(x_u) \leftarrow \exp\Big[a \tanh\Big[\frac{1}{b} s(x_u)\Big]\Big], \qquad a, b > 0. \tag{2.19}$$

Firstly, we use a tanh : $(-\infty, \infty) \rightarrow [-1, 1]$ to *soft-clamp* $s(x_u)$ in the interval $[-a, a]$. Secondly, the soft clamped value is exponentiated such that the division by $s(x_u)$ becomes a multiplication with $\exp[-s(x_u)]$.

Cubic Spline Transformation: A cubic spline coupling transformation entails more flexibility due to a mapping $h_\theta : [0, 1] \mapsto [0, 1]$ that consists of joining together multiple monotonic increasing cubic polynomials in a way that the result is differentiable. The free parameters θ of this mapping, which will be computed by a neural network with input x_u, are the positions of the *knots* at which the individual polynomials are linked and the boundary derivatives. [21] The first knot must be at $(0, 0)$ and the last at $(1, 1)$. To realize the mapping from $[0, 1]$ to $[0, 1]$ we scale down a previously defined interval $[\min, \max]$ at the input of each coupling layer.

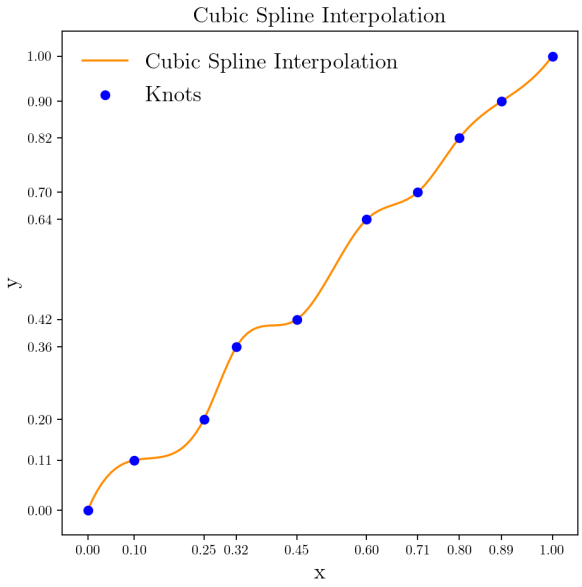


Figure 2.9: Cubic spline interpolation

In figure 2.9 we show a cubic spline interpolation on the interval $[0, 1]$ with nine bins.

The training objective of an INN results from comparing the generated probability density to the desired distribution using the *Kullback-Leibler* (KL) divergence:

$$\begin{aligned} \hat{\Theta} &= \underset{\Theta}{\operatorname{argmin}}\, \mathrm{KL}(P(x)||P_\Theta(x)) &&= \left\langle \log\left[\frac{P(x)}{P_\Theta(x)}\right]\right\rangle_{x\sim P(x)} \\ &= \langle \log[P(x)]\rangle_{x\sim P(x)} - \langle \log[P_\Theta(x)]\rangle_{x\sim P(x)}. \end{aligned} \tag{2.20}$$

The first term is independent of the model and can therefore be dropped. The second term is the negative log likelihood and realizes an exact maximum likelihood objective,

using the change of variables formula in equation 2.15:

$$\mathcal{L}_{Latent} = \langle \log[P(z = f_\Theta(x))] \rangle_{x \sim P(x)} - \log[|J_z|]. \tag{2.21}$$

For latent variables distributed according to a standard normal probability density $\mathcal{N}(0, \mathbb{I})$ the loss then reads:

$$\mathcal{L}_{Latent} = \left\langle \frac{f_\Theta(x)^2}{2} \right\rangle_{x \sim P(x)} - \log[|J_z|]. \tag{2.22}$$

5 Hybrid Learning

As explained in section 4, the loss objective of an ordinary normalizing flow network results from the maximum likelihood principle and is evaluated in latent space $\mathcal{Z}$. Models trained on this objective often lack precision on the generated density distributions compared to other generative models [9]. To improve on accuracy, we can formulate loss objectives in the data space $\mathcal{X}$ and combine them with the maximum likelihood objective in a *hybrid* objective. This setup is illustrated in figure 2.10.

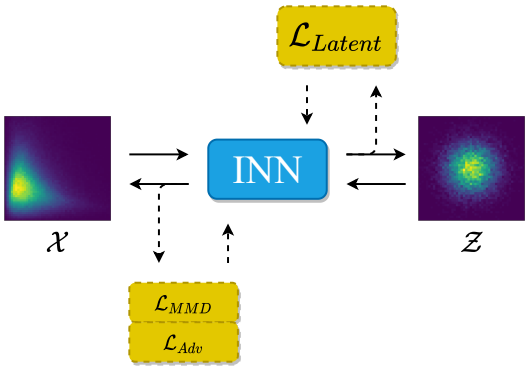


Figure 2.10: Schematic illustration of hybrid learning

5.1 Maximum Mean Discrepancy

If the data space $\mathcal{X}$ is very large, structures and correlations that cover only a small area of the whole space might be overseen by the network. To force the network to learn these tight structures correctly, a *Maximum Mean Discrepancy* (MMD) loss can be helpful. The idea of MMD is to map two probability density distributions of a variable u onto

a so-called *reproducing kernel Hilbert space* $\mathcal{H}$ by a positive-definite *kernel function* and compare them in this space [22]:

$$k : \mathcal{X} \to \mathcal{H}. \tag{2.23}$$

For a variable u distributed according to the true probability density distribution $P(u)$, approximated by the model $P_\Theta(u)$, the MMD loss reads as:

$$\mathcal{L}^2_{MMD} = (\langle k(u)\rangle_{u\sim P(u)} - \langle k(u)\rangle_{u\sim P_\Theta(u)})^2 \tag{2.24}$$

$$= \langle k(u)k(u)\rangle_{u,u\sim P(u)} + \langle k(u)k(u)\rangle_{u,u\sim P_\Theta(u)} - 2\langle k(u)k(u)\rangle_{u\sim P(u),u\sim P_\Theta(u)}. \tag{2.25}$$

To narrow down the loss on the relevant structures, common kernel functions are the Gaussian kernel or the Breit Wigner (BW) kernel as used in [2]:

$$k_{Gaussian} : u \mapsto \exp\Big[\frac{-u^2}{2\sigma}\Big], \qquad k_{BW} : u \mapsto \frac{\sigma^2}{u^2+\sigma}. \tag{2.26}$$

Here, the width σ is a hyperparameter.

Example: The effect of the loss function can be best understood considering a simple example on $\mathbb{R}$: Let the true data be sampled from a normal distribution with variance σ_t^2 and let the generated data be approximated by a normal distribution with variance σ_p^2:

$$P(x) = \exp\Big[\frac{-x^2}{\sigma_t^2}\Big], \qquad P_\Theta(x) = \exp\Big[\frac{-x^2}{\sigma_p^2}\Big].$$

Furthermore, we use the Gaussian kernel function (equation 2.26). The MMD loss then reads as:

$$\begin{aligned}\mathcal{L}^2_{MMD} &= \langle k(x)k(x)\rangle_{x,x\sim P_\Theta(x)} + \langle k(x)k(x)\rangle_{x,x\sim P(x)} - 2\langle k(x)k(x)\rangle_{x\sim P(x),x\sim P_\Theta(y)} \\ &= \int_{\mathbb{R}} dx \int_{\mathbb{R}} dx\; k(x)k(x)P_\Theta(x)P_\Theta(x) + \int_{\mathbb{R}} dx \int_{\mathbb{R}} dx\; k(x)k(x)P(x)P(x) \\ &- 2\int_{\mathbb{R}} dx \int_{\mathbb{R}} dx\; k(x)k(x)P(x)P_\Theta(x) \\ &= \frac{1}{\sqrt{(1+\sigma_p^2)^2-\sigma_p^4}} + \frac{1}{\sqrt{(1+\sigma_t^2)^2-\sigma_t^4}} - 2\frac{1}{\sqrt{(1+\sigma_p^2)(1+\sigma_t^2)-\sigma_p^2\sigma_t^2}}.\end{aligned}$$

This function is depicted for $\sigma_t = 1$ in a two dimensional case in figure 2.11 and has a global minimum at $\sigma_p = \sigma_t$ and thereby $P(x) = P_\Theta(x)$ which is the desired behavior.

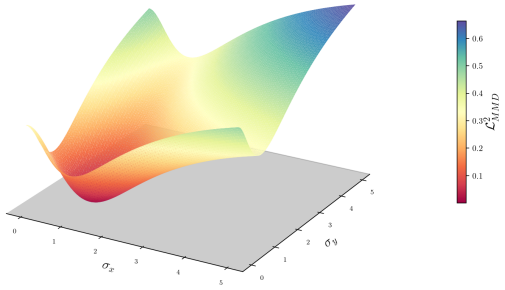


Figure 2.11: MMD loss landscape in two dimensions for the analytically solved example ($\sigma_t = 1$)

5.2 Flow GAN

Another method to tune some structures is training a discriminator network explicitly or implicitly on the problematic structure, using its feedback to optimize the generative network.
In general a discriminator D is a network that learns to distinguish whether an instance u is drawn from the true probability density distribution $P(u)$ or from the models' approximation $P_\Theta(u)$:

$$D : \mathcal{U} \to \mathcal{D} = [0, 1], \qquad u \mapsto D(u). \tag{2.27}$$

Hereby, an output of 1 corresponds to $u \sim P(u)$ and an output of 0 corresponds to $u \sim P_\Theta(u)$. The mapping onto [0,1] is enforced by a sigmoid activation function (equation 2.3) in the output layer of the discriminative network.

The discriminator is trained minimizing the *binary Cross Entropy* (BCE) loss:

$$\mathcal{L}_{Disc} = -\langle \log[D(u)] \rangle_{u \sim P(u)} - \langle \log[1 - D(u)] \rangle_{u \sim P_\Theta(u)}. \tag{2.28}$$

The discriminator output $D^*(u) \in [0, 1]$ for a perfectly trained network can be interpreted as proportional to the probability of the data instance being sampled from the true distribution $P(u \sim P(u)|u)$. Therefore, the discriminator output can be used to rebalance the generated data instances $x_i = f_\Theta^{-1}(z_i)$ by introducing so-called *weights* [23]:

$$w_i = \frac{D(x_i)}{1 - D(x_i)} \propto \frac{P(x_i \sim P(x_i)|x_i)}{P(x_i \sim P_\Theta(x_i)|x_i)}. \tag{2.29}$$

Note that we can train the discriminator on some variables u that not necessarily have to be equal to the generator outputs x but can be a correlated function of them instead

or additionally.

Then, the generative and discriminative model can be trained as a *Generative Adversarial Network* (GAN) [24] by feeding an adversarial loss back to the generative model:

$$\mathcal{L}_{Adv} = \langle \log[D(u)] \rangle_{u \sim P(u)} + \langle \log[1 - D(u)] \rangle_{u \sim P_\Theta(u)}. \tag{2.30}$$

For a perfectly trained discriminator $D^*(u)$, the adversarial loss objective implements a comparison of the generated and true probability density distribution, using the *Jensen-Shannon* (JS) divergence:

$$\mathcal{L}_{Adv} \propto \mathrm{JS}(P_\Theta \| P) + const\ . \tag{2.31}$$

$$\tag{2.32}$$

The JS divergence can be expressed in terms of the KL divergence, which was introduced in equation 2.20:

$$\mathrm{JS}(P_\Theta \| P) = \frac{1}{2}\mathrm{KL}\Big(P_\Theta \Big\| \frac{P_\Theta + P}{2}\Big) + \frac{1}{2}\mathrm{KL}\Big(P \Big\| \frac{P_\Theta + P}{2}\Big). \tag{2.33}$$

A well-performing discriminator outputs $D(u) = 1$ for $u \sim P(u)$ and $D(u) = 0$ for $u \sim P_\Theta(u)$ in most cases. A technical problem with this theoretically optimal behavior is that the adversarial loss becomes close to zero and the gradients of the sigmoid output of the discriminator vanish. A simple solution to this problem is to introduce a non-saturating (ns) adversarial loss [25]:

$$\mathcal{L}^{ns}_{Adv} = -\langle \log[D(u)] \rangle_{u \sim P_\Theta(u)}. \tag{2.34}$$

In our case, the network setup differs from an ordinary GAN in one important point: In an ordinary GAN setup the generative model is trained on the adversarial loss term only. Here, we combine the adversarial loss in data space with a maximum likelihood loss in latent space. Therefore, we call this setup a *Flow-GAN* as proposed in [9].

3 Phenomenology

1 A Glimpse at Relativistic Kinematics

Relativistic kinematics introduces a four dimensional space called *space-time*, consisting of a time dimension additional to the ordinary three dimensional space dimensions. Kinematics in this four dimensional space are best described by *four-vectors* of position $\mathbf{X}$ or momentum $\mathbf{P}$:

$$\mathbf{X} = (t, \mathbf{x})^T, \qquad \mathbf{P} = (E, \mathbf{p})^T, \qquad c = \hbar = 1. \tag{3.1}$$

Here, $\mathbf{x}$ and $\mathbf{p}$ are the three-position and three-momenta of space.
The energy of a relativistic particle consists of its kinetic energy $\mathbf{p}$ and its *invariant mass* M:

$$E^2 = \mathbf{p}^2 + M^2. \tag{3.2}$$

The invariant mass is the mass of a particle measured in a frame where the particle is at rest. To transform or *boost* the four-momenta of a particle to another frame, a *Lorentz-Transformation* is used:

$$\tilde{\mathbf{P}}_{\parallel} = \begin{pmatrix} \tilde{E} \\ \tilde{\mathbf{p}}_{\parallel} \end{pmatrix} = \begin{pmatrix} \gamma & -\gamma\beta \\ -\gamma\beta & \gamma \end{pmatrix} \begin{pmatrix} E \\ \mathbf{p}_{\parallel} \end{pmatrix}, \qquad \beta = \|\mathbf{v}_{frame}\|, \qquad \gamma = \frac{1}{\sqrt{1-\beta^2}}. \tag{3.3}$$

β is a measurement of the velocity of the frame with respect to a globally resting frame. Note that only the momentum $\mathbf{p}_{\parallel}$ parallel to the movement of the frame is affected by the boost, whereas $\tilde{\mathbf{p}}_{\perp} = \mathbf{p}_{\perp}$.
The Minkowski scalar product $\langle\cdot|\cdot\rangle$ and therefore its induced norm $|\cdot|$ is a invariant property under a Lorentz-boost:

$$\langle \mathbf{U}|\mathbf{V}\rangle = U_0V_0 - U_1V_1 - U_2V_2 - U_3V_3 = \langle \tilde{\mathbf{U}}|\tilde{\mathbf{V}}\rangle. \tag{3.4}$$

The norm of a particles' four-momentum $|\mathbf{P}|$ equals its center of mass (CM) energy, which is clear for a single particle where the center of mass energy equals its invariant mass $E_{CM} = M$. This yields the restriction that the CM energy of the decay products of a single particle is limited by the invariant mass of the mother particle due to energy conservation. In fact this restriction is violated in some cases, leading to the introduction of *virtual* or *off-shell* particles that do not live on the mass shell dictated by equation 3.2.

2 Hadron Collider Physics

In hadron colliders like the *Large Hadron Collider* (LHC), hadrons, mostly protons and or antiprotons, are accelerated to a CM energy $\sqrt{s}$ and forced to collide with a Luminosity, indicating the number of collisions per second and square centimeter. At the LHC,

the maximum center of mass energy is 14 TeV [26]. In a collision, partons x_1 and x_2 of the colliding hadrons are involved in the hard interaction. These two partons do not necessarily have the same momentum but follow the so-called *Parton Distribution Function* (PDF) instead. These PDFs and thereby the scattering behavior of the hadrons strongly depend on the energy scale of the process [27]. For high energies, the strong coupling between the partons is only a small perturbation and they can be treated as *asymptotically free*, interacting in *hard* processes. For energies below a certain energy limit, the pertubative description of *Quantum Chromo Dynamics* (QCD) is not sufficient and a non-pertubative description of the then so-called *soft* processes must be established. Due to the individually varying momenta of the interacting partons, the center of mass of the collision is boosted in beam direction.

The kinematics of the collision are best described by the *Pseudorapidity* η, the *transverse momentum* p_T, and the *azimuthal angle* ϕ, where η is defined by the *polar angle* θ [28]:

$$\eta = -\log[\tan\left(\frac{\theta}{2}\right)]. \tag{3.5}$$

For each particle involved or produced in the collision the Cartesian four-momentum can be expressed by (m, η, p_T, ϕ):

$$\mathbf{p} = \begin{pmatrix} \sin(\phi)p_T \\ \cos(\phi)p_T \\ \sinh(\eta)p_T \end{pmatrix}, \qquad E = \sqrt{m^2 - (p_x^2 + p_y^2 + p_z^2)}, \qquad \mathbf{P} = (E, \mathbf{p})^T. \tag{3.6}$$

3 Jets

At the LHC, heavy resonances of the *Standard Model* (SM), such as the $W^{\pm}$-, Z- or H-boson as well as the t-quark, are frequently produced in hard processes. These resonances from the hard scattering decay in further hard processes into quarks and leptons, which then produce *showers* of quark and lepton radiation. At lower energies around Λ_{soft}, the produced quarks begin to *hadronize* forming bound states of quark anti-quark pairs, *mesons*, or *baryons* consisting of three quarks. [28]

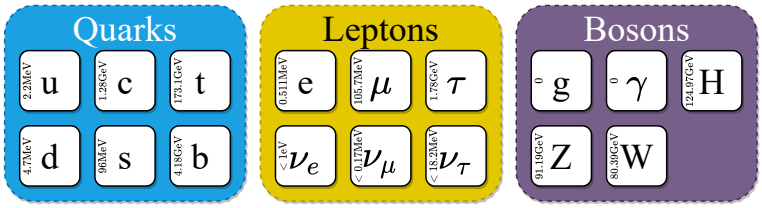


Figure 3.1: Particles of the Standard Model

These *Jets* of particles produced in the soft processes are then measured by the detector. Recombination algorithms like the k_T [29], *Cambridge-Aachen* [30] or *anti-k_T* [31] algorithm then can be used to reconstruct information about the particles from the hard process. For massive particles, these jets are often called *fat-jets* due to their large spread in the η-ϕ plane. For fat jets, reconstruction is best done with the four-momenta of the particles, because we want to take the invariant mass of the particles into account. After reconstruction, the massive particles can be found in the reconstruction chain by searching for invariant mass changes that correspond to the specific masses. [32].

4 Drell-Yan Process

The Drell-Yan process is probably the best known process occuring at the LHC and other particle colliders. The underlying process is rather simple:

$$q\overline{q} \to Z/\gamma* \to l^+l^-. \tag{3.7}$$

Two quarks annihilate to form a on-shell Z-boson, with a resting energy of about 91.2 GeV, or a virtual photon that then decays leptonically. The corresponding feynman graph is shown in figure 3.2.

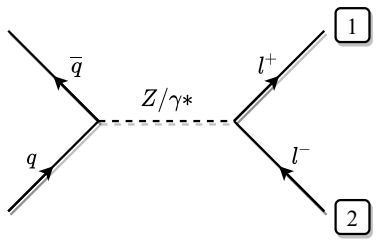


Figure 3.2: Feynman graph: Drell-Yan process

Assuming we get the four-momenta $\{\mathbf{P}_i\}_{i=1}^2$ for the two final state particles in figure 3.2 from a reconstruction algorithm or simply from a simulation of the hard process, we can reconstruct the invariant masses of the Z-boson:

$$M_Z^2 = |\mathbf{P}_1 + \mathbf{P}_2|^2. \tag{3.8}$$

Considering the fact that the original Z-boson is only boosted in the beam direction, we know that the momenta of the two jets transverse to the beam direction must sum up to zero in each direction, respectively:

$$\sum_{i=1}^{2} p_{x,i} = 0, \qquad \sum_{i=1}^{2} p_{y,i} = 0 \qquad \text{or} \qquad p_{T,1} = p_{T,2}, \qquad \Delta\phi_{1,2} = \pi. \tag{3.9}$$

5 WZ Production and Decay

Another process observed in the LHC experiments is the production of a WZ-diboson state, where the W-boson decays hadronically and the Z-boson leptonically. At the LHC, the dibosonic state is dominantly produced by a quark, anti-quark initial state with an intermediate single off-shell W-boson [33]:

$$q\bar{q} \to W \to WZ \to q\bar{q}l^+l^-. \tag{3.10}$$

Here, both bosons are on-shell with masses $M_W = 80.4\,\text{GeV}$ and $M_Z = 91.2\,\text{GeV}$. The corresponding feynman graph is shown in figure 3.3.

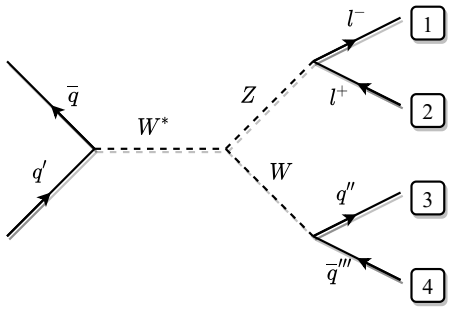


Figure 3.3: Feynman graph: WZ diboson production (s-channel) and decay

We can reconstruct the invariant masses for the W- and Z-bosons from the four-momenta of the two quarks and two leptons in the final state:

$$M_Z^2 = |\mathbf{P}_1 + \mathbf{P}_2|^2, \qquad M_W^2 = |\mathbf{P}_3 + \mathbf{P}_4|^2. \tag{3.11}$$

Furthermore, we know from the momentum conservation:

$$\sum_{i=1}^{4} p_{x,i} = 0, \qquad \sum_{i=1}^{4} p_{y,i} = 0 \qquad \text{or} \qquad p_{T,W} = p_{T,Z}, \qquad \Delta\phi_{W,Z} = \pi. \tag{3.12}$$

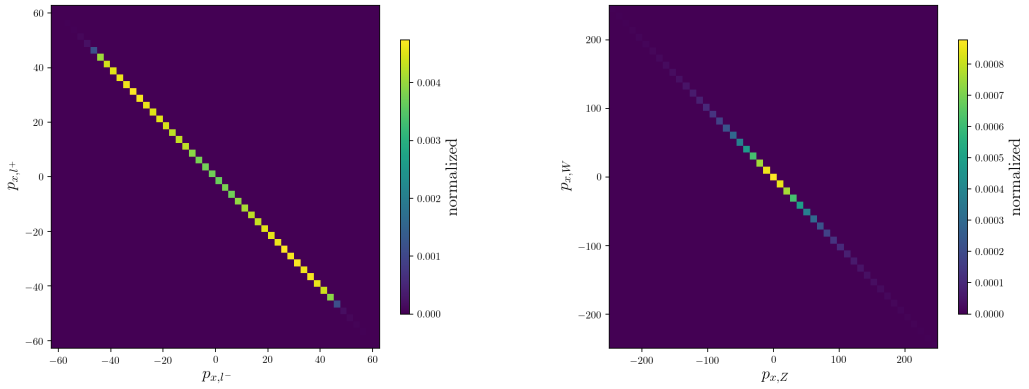
4 Methodology

1 Training Data

The training data for the INN is generated using *Madgraph*. The data is simulated on parton level, which means that no showers, hadronization or detector effects have been simulated. Hence, the data space $\mathcal{X}$ will in future be referred to as *parton space*. From these events 50% are used for training (Train) and the other 50% for validation (True). We have two data sets we want to apply our network on. The simplest *Drell-Yan* data set consists of two final state leptons resulting from the Drell-Yan process as described in section 4. The more complicated data set, the WZ data set, consists of two final state leptons and two final state jets from the process explained in section 5. In parton space $\mathcal{X}$ the INN is trained on the observables

$$\{p_{T,i}, \phi_i, \eta_i\} \qquad \text{or} \qquad \{p_{x,i}, p_{y,i}, p_{z,i}\}$$

for $i = 1, 2$ or $i = 1, ..., 4$, depending on the data set. This is not exactly true, since we know from equations 3.9 and 3.12 that some of these observables can directly be derived from others, which reduces the dimension of the manifold of the data in parton space.



(a) (b)

Figure 4.1: Momentum conservation in x direction (a) Drell-Yan Data Set (b) WZ Data Set. The momentum density in x direction is plotted for the leptons in the Drell-Yan data set and for the Z- and W-boson of the WZ data set.

The Drell-Yan data set satisfies the momentum conservation exactly, while there is a deviation of magnitude $\sim 10^{-4}$ in the WZ data set. Therefore, we enforce the momentum conservation when training on the Drell-Yan data set, since not considering the reduced

dimension in parton space might inhibit the homeomorphic behavior of normalizing flows. As explained in section 4, the latent variables z in the equal dimensional latent space $\mathcal{Z}$ are distributed according to a standard normal distribution. In these latent variables the network should encode the relevant information about the distribution of the input data and correlations in the data.

2 Generative Model

The implemented INN architecture makes use of the *Framework for Easily Invertible Architectures* (FrEIA) [20]. The main coupling blocks used are the *AllInOne*-Block from FrEIA and a *CubicSpline*-Block adapted from [21]. The subnetworks of these coupling blocks consist of dense layers with ELU activation. Optional for the subnetworks is dropout and spectral norm. The optimization algorithm of choice is the *Adam* optimizer [12]. Optional a L2 regularization is used with this optimizer. Additionally to the acceleration term in the optimization step, we use a *OneCycle* or *Step* scheduler, as introduced in section 2. For each instance in data space $\mathcal{X}$ the observables described in section 1 are available. We use several preprocessing steps $PreP : \mathcal{X} \to \tilde{\mathcal{X}}$ to transform the data into a new space $\tilde{\mathcal{X}}$ in which the data is easier to learn for the network. The preprocessing steps include:

1. $p_T \mapsto \log[p_T - p_{T,min}]$ (if trained on p_T)
2. $\phi \mapsto \tanh^{-1}[\frac{\phi}{\pi}]$ (if trained on ϕ)
3. Normalization
4. Principal Component Analysis

The first and second preprocessing steps transform the p_T and ϕ distribution to a distribution more similar to a standard normal distribution, as shown for the ϕ observable in figure 4.3.

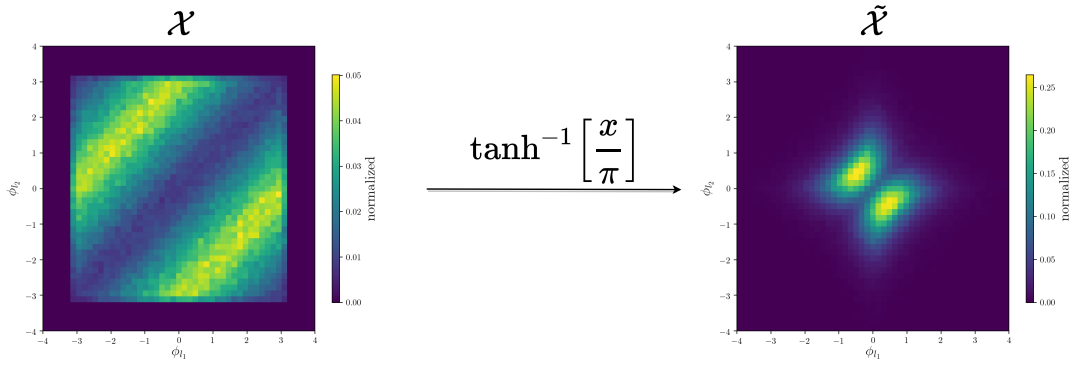


Figure 4.2: Preprocessing of the ϕ distribution. Left: unpreprocessed data distribution for ϕ_1 and ϕ_2 as a 2D histogram, right: 2D histogram after applying $\tanh^{-1}\left[\frac{x}{\pi}\right]$ to each data instance.

3 Hybrid Training with MMD

The MMD objective will be used exclusively to improve the generated probability density distributions of the mass correlation. Since we do not train on the mass distribution directly, the mass correlation must be computed from the output of the generative model using equation 3.6 and 3.2. The mass distributions only cover a thin hypersurface in the total phase space. Hence, we use the kernels introduced in equations 2.26 to narrow down the effect of the MMD objective on the relevant structure. As suggested in [2], the width σ of the kernel should be adjusted to the width of the distributions predicted by the model, in order to avoid that relevant instances to the MMD Loss are suppressed by the kernel. Therefore, the width must be adaptive to the broadness of the generated distribution during training. This can be realized in several ways.

1. **Cooling-Kernel:** A Cooling-Kernel simply uses an epoch-wise decaying kernel width, for example an exponential decay:

   $$\lambda^t = \lambda^0 e^{-t\tau}.$$

   Here t is the epoch and τ a time constant. The main obstacle using this method is to adjust τ to the rapidity of changes in distribution width.

2. **Estimate-Kernel:** A Estimate-Kernel estimates the current width of the distribution every now and then, to use this estimate as a new kernel width. A problem arising with this method are an instable training when the kernel width and therefore the MMD loss is changed abruptly.

3. **Multi-Kernel:** A Multi-Kernel in general is the sum of multiple MMD losses using multiple kernel widths. The used kernel widths are chosen such that for

every distribution width a more or less matching kernel is contributing to the loss. This method is quite easy to apply and does not suffer from numerical instabilities. That is why a multi-kernel was the method of choice for this thesis.

4 Hybrid Training as a Flow-GAN

The Flow-GAN objective will be used for all distributions we want to improve in precision. Because we want to train the discriminative network not only on the output of the generative network in $\tilde{\mathcal{X}}$, we need to transform the output data to a new space $\mathcal{U}$ by a function $DPreP : \tilde{\mathcal{X}} \to \mathcal{U}$, where we define new observables we want to train the discriminator on. Unless stated otherwise, these observables are normalized.

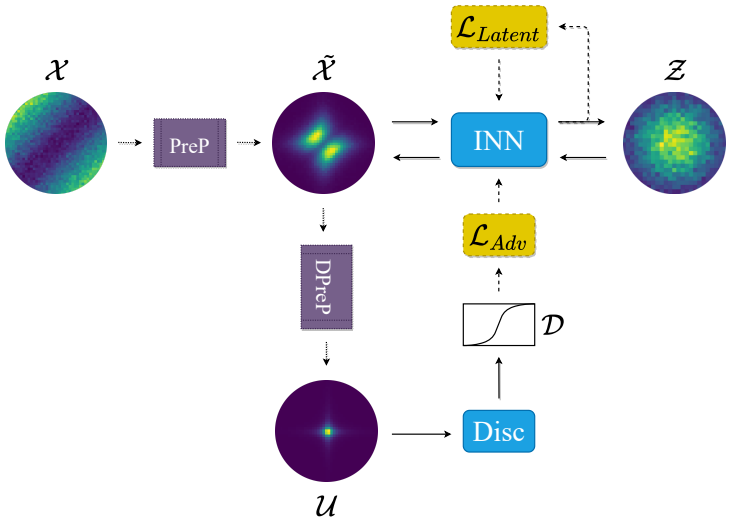


Figure 4.3: Schematic illustration of the Flow-GAN setup

As a discriminative network a fully connected network with Leaky ReLU activation will be used. Just as with the generative network, the *Adam* optimizer will be used for the gradient steps. Additional, a L2 regularization, gradient penalty, or spectral norm will be applied if needed. When training the generative and discriminative networks as adversaries, we change the parameters (β_1, β_2) from their default values of $(0.9, 0.99)$ to $(0.5, 0.9)$ as discussed in section 2.

5 Results

1 Drell-Yan Data Set

The simplest problem was the Drell-Yan data set. Here we had to design the generative network intentionally small to have some space for improvement. A baseline model with an architecture noted in table 5.1 was trained for 300 epochs.

input observables	$\{p_{x,l1}, p_{y_l}1, p_{z,l1}, p_{z,l2}\}$
block type	AllInOne
number blocks	20
layers per block	3
nodes per layer	256
soft clamping	5
parameters to optimize	1.4M
spectralnorm	✓
L2	0.0
learning rate	$2 \cdot 10^{-4}$ (OneCycle)
batchsize	2k

Table 5.1: Setup of the generative model for the Drell-Yan data set

The resulting probability density distribution for the mass correlation is shown in figure 5.1.

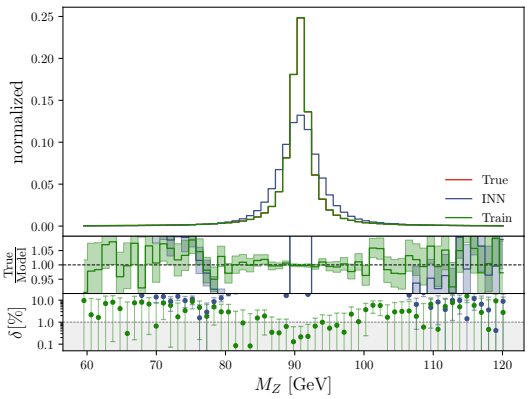
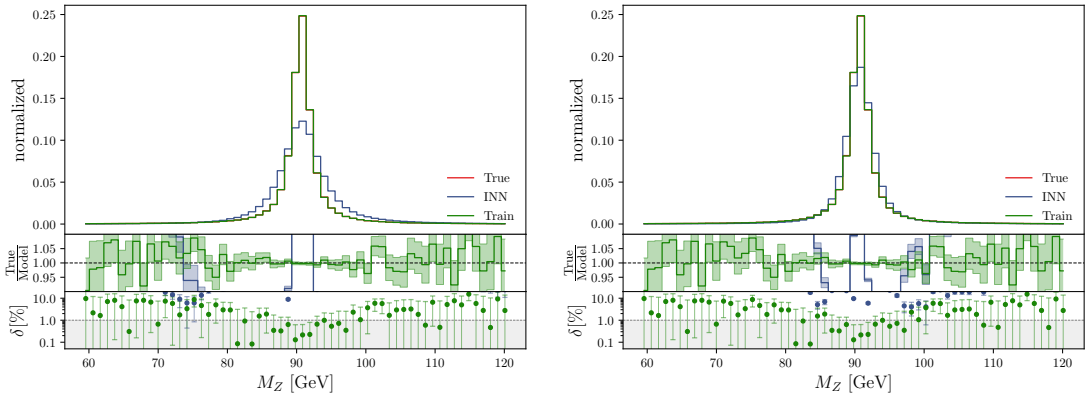


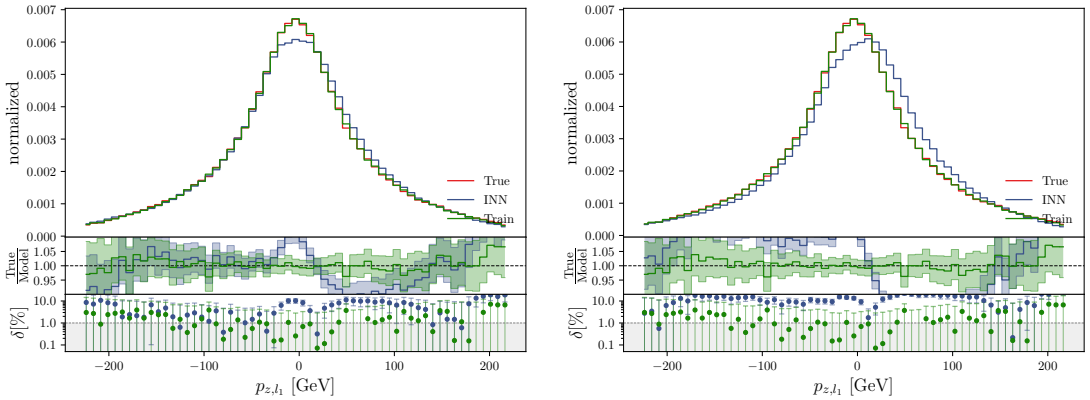
Figure 5.1: Baseline mass distribution for the Drell-Yan data set. Panel 1: probability density distribution of events, panel 2: quotient of the true and generated density (bin-wise), panel 3: relative deviation from the true density (bin-wise).

First, we apply a MMD loss objective to the mass distribution. The Loss term $\mathcal{L}_{MMD}$ (equation 2.25) is added to the Loss of the generative network weighted with a factor λ_{MMD}. Considering the Baseline Mass distribution in figure 5.1 using multiple Gaussian kernels with widths $\sigma \in \{5, 2, 1\}$ seemed appropriate. For $\lambda_{MMD} \in \{0.2, 2\}$ the effects of the MMD Loss are shown in figure 5.2.



(a) (b)

Figure 5.2: Mass distributions with hybrid MMD objective for the Drell-Yan data set (a) $\lambda_{MMD} = 0.2$ (b) $\lambda_{MMD} = 2$



(a) (b)

Figure 5.3: Momentum distributions in z direction with hybrid MMD objective for the Drell-Yan data set (a) $\lambda_{MMD} = 0.2$ (b) $\lambda_{MMD} = 2$

We can see how the additional loss term affects the performance on the mass distribu-

tion with increasing weight. Nevertheless, considering other distributions, we see that the MMD loss objective does not force the generative network to converge towards the global minimum, since loosing precision in other phase space regions, as shown for the $p_{z,l1}$ distribution in figure 5.3. Here we see how the generative INN optimizes for the MMD loss correctly. Still, we do not have any control over how the two loss terms interact in the hybrid objective. The main hyperparameter to tune is the balance of both loss terms λ_{MMD}, which does not influence how the MMD objective acts in other phase space regions.

Next, we train the network as a Flow GAN, explained in section 5.2. To train in adversarial mode we introduce a discriminative network with properties listed in table 5.2.

input observables	$\{M_Z\}$
number layers	4
nodes per layer	64
parameters to optimize	9k
spectralnorm	✗
gradient penalty	✗
L2	$1 \cdot 10^{-3}$
learning rate	$1 \cdot 10^{-4}$ (OneCycle)
batchsize	2k

Table 5.2: Setup of the discriminative model for the mass distributions of the Drell-Yan data set

As a first step, we train the discriminative network for 20 epochs by itself, minimizing the BCE loss objective $\mathcal{L}_{Disc}$ (equation 2.28). Using the discriminator output on the generated events, we can compute weights according to equation 2.29 and reweight each event. The resulting probability density of the reweighted mass distribution is shown in figure 5.4.

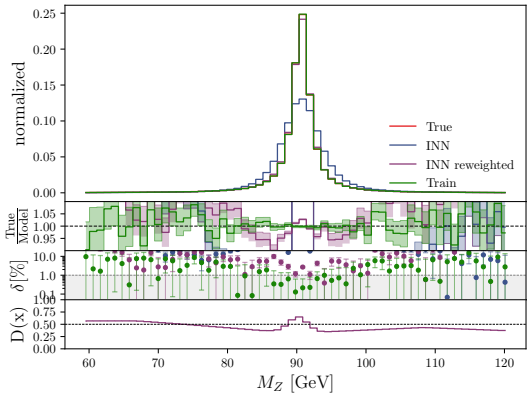
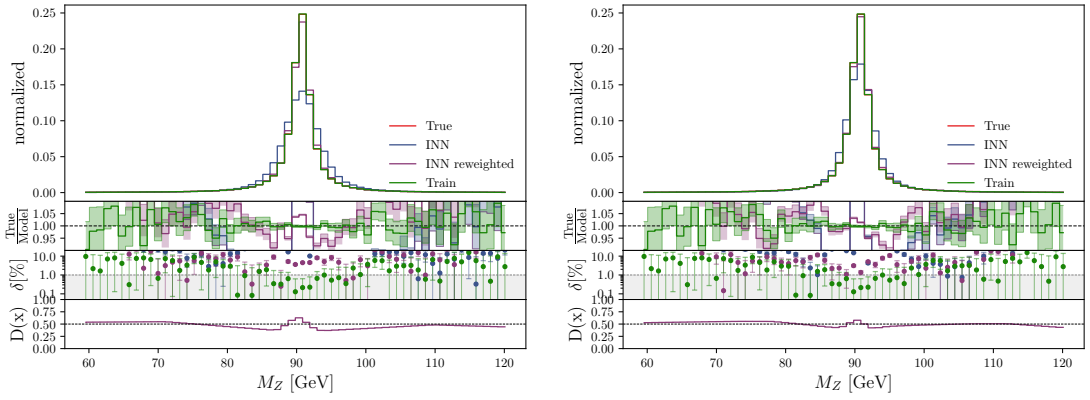


Figure 5.4: Reweighted mass distribution for the Drell-Yan data set. Panel 1-3: see figure 5.1, panel 4: discriminator output (bin-wise).

The discriminative network converges almost perfectly to the theoretical minimum D^*, where the probability interpretation of the output matches the ratio of the generated and true probability density distributions.

Next, we train the generative and discriminative network as adversaries for 20 more epochs by adding the adversarial loss term $\mathcal{L}_{Adv}$ (equation 2.30) weighted with a factor λ_{Adv} to the loss objective of the generative network. For $\lambda_{Adv} \in \{1, 10\}$, the effects of the hybrid objective on the density distribution of the Z-boson mass is shown in figure 5.5.

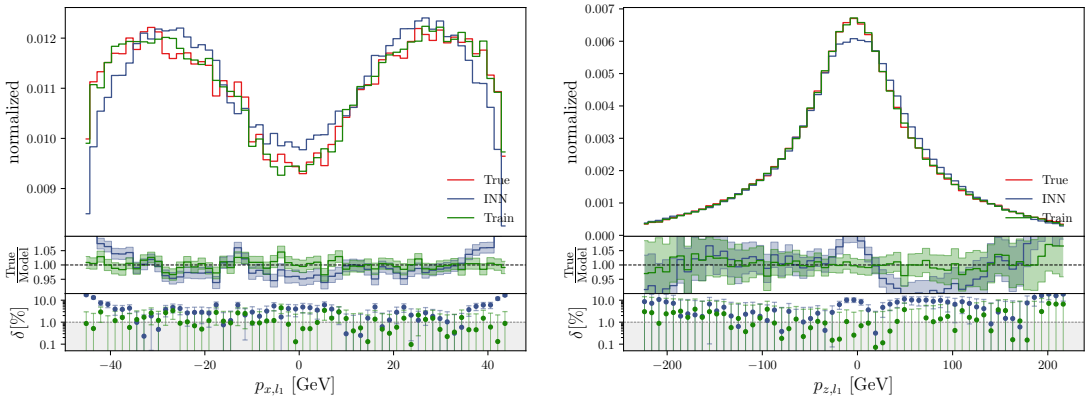


(a) (b)

Figure 5.5: Mass distributions with Flow-GAN objective for the Drell-Yan data set (a) $\lambda_{Adv} = 1$ (b) $\lambda_{Adv} = 10$

Using the hybrid objective, the generative network is nudged towards learning the mass density distribution correctly. Still, when considering the other distributions it becomes clear that the network is not converging to a joint minimum of both loss terms in the hybrid objective, because the precision of other distributions decreases when the precision of the mass distribution increases. Density histograms for other phase space regions can be found in the appendix.

The next step to retain precision in other phase space regions, is to train the generative network on other distributions as well. The momentum distributions generated by the generative network described in table 5.1 are shown in figure 5.6. Since we enforced momentum conservation, the momentum distributions for both leptons are equal. Furthermore, the distributions in x and y direction are symmetrical. Therefore, it is sufficient to consider the $p_{x,l1}$ and $p_{z,l1}$ distributions.



(a) (b)

Figure 5.6: Baseline momentum distributions for the Drell-Yan data set (a) $p_{x,l1}$ (b) $p_{z,l1}$

In contrast to the good performance on the mass distribution, simply training a discriminative network on the one-dimensional momentum distributions yields rather poor reweighting results. This can be best understood considering that the discriminative network only 'sees' a one-dimensional distribution and therefore can not learn any correlations that help distinguishing between generated and true events. Furthermore, the discriminator is only trained on a small batch (here 2k samples) of instances that underlie statistical batch-fluctuations as shown in figure 5.7.

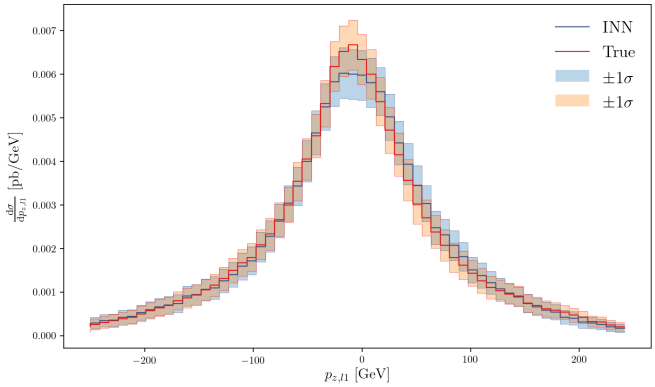


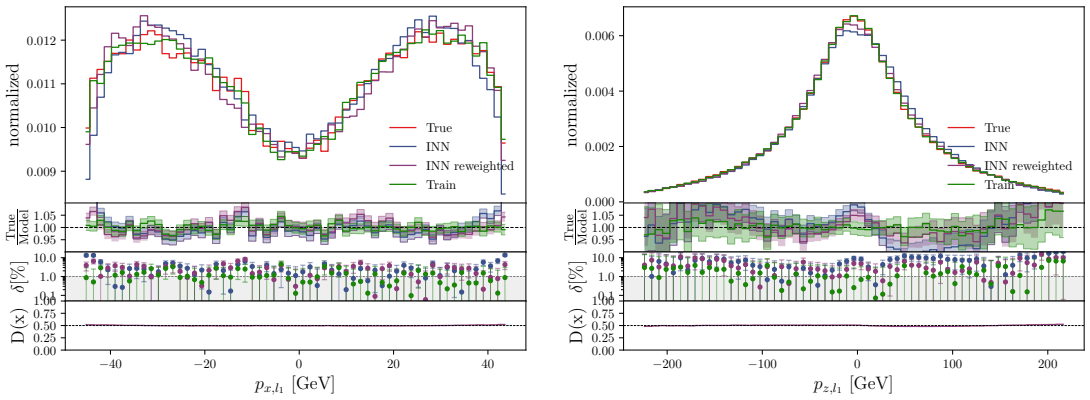
Figure 5.7: Batch fluctuations on the $p_{z,l1}$ distributions. Computed for a batchsize of 2k instances over 10k batches for the Drell-Yan data set.

Still, when training on multidimensional discriminator input including the mass observable, the reweighting results improve, which is expected due to the multidimensional correlations the discriminator now can use for its classification.
Therefore, we trained a pretrained discriminator with a multidimensional input summarized in table 5.3 together with the pretrained generative model as adversaries for 100 epochs.

input observables	$\{M_Z, p_{x,l1}, p_{y_l}1, p_{z,l1}, p_{z,l2}\}$
number layers	6
nodes per layer	128
parameters to optimize	67k
spectralnorm	✗
gradient penalty	✗
L2	$1 \cdot 10^{-3}$
learning rate	$5 \cdot 10^{-5}$ (Step)
batchsize	2k
λ_{Adv}	2

Table 5.3: Setup of the discriminative model for the momentum distributions of the Drell-Yan data set

The effects on the momentum distributions as well as on the mass distribution are shown in figure 5.8.



(a) (b)

Figure 5.8: Distributions after adversarial training for the Drell-Yan data set (a) $p_{x,l1}$ (b) $p_{z,l1}$

Especially in the area around 0GeV of the $p_{x,l1}$ distribution, there is some serious improvement of the generated density with relative errors δ around the 1% mark. Other distributions improve slightly, as shown for the $p_{z,l1}$- distribution in figure 5.8. Stll, considering the course of the loss functions, we see that the minimizing the adversarial loss leads to a increase of the latent loss as show in figure 5.9, until both loss terms saturate in an equilibrium.

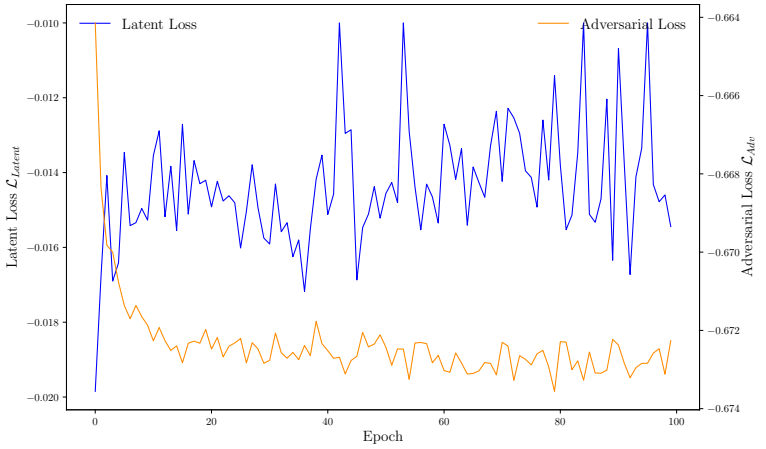


Figure 5.9: Course of the adversarial and latent loss in adversarial training for the Drell Yan data set with pretrained models (Coupling $\lambda_{Adv} = 1$, learning rate $5{\cdot}10^{-5}$ (Step)).

This shows that the current Flow-GAN setup might improve the distributions, but does not lead to finding the joint global minimum of both loss terms.

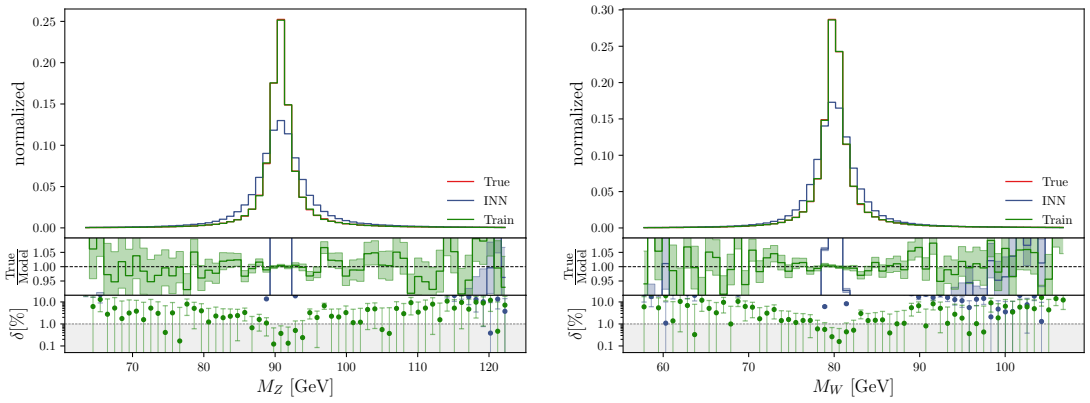
2 WZ Data Set

The WZ data set consists of four final state particles. Since the momentum conservation is not exact in the training data set, we do not enforce it by reducing the dimension of the input. Therefore, we train the generative network on a 16 dimensional input in $\tilde{\mathcal{X}}$, requiring a more powerful generative model summarized in table 5.4.

input observables	$\{p_{T,i}, \eta_i, \phi_i\}_{i=1}^4$
block type	Cubic Splines
number blocks	24
layers per block	3
nodes per layer	128
parameters to optimize	2.7M
spectralnorm	✗
L2	0.0
learning rate	$2 \cdot 10^{-4}$ (OneCycle)
batchsize	4k

Table 5.4: Setup of the generative model for the WZ data set

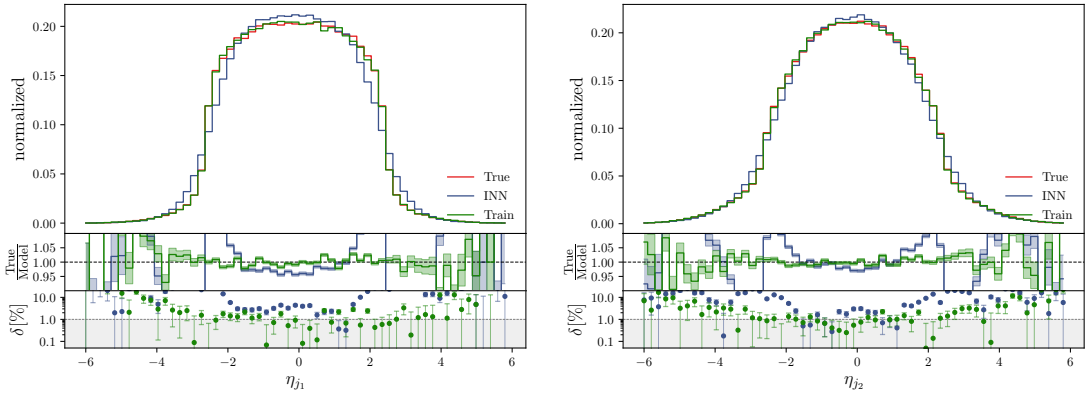
After 500 epochs of training, there is still some room for improvement in the mass density distributions shown in figure 5.10.



(a) (b)

Figure 5.10: Baseline mass distributions for the WZ data set (a) M_Z (b) M_W

In the remaining distributions, especially the distributions of the pseudorapidity η lacks in precision in some areas as one can see for the two jets in figure 5.11.



(a) (b)

Figure 5.11: Baseline pseudorapidity distributions for the WZ data set (a) η_{j1} (b) η_{j2}

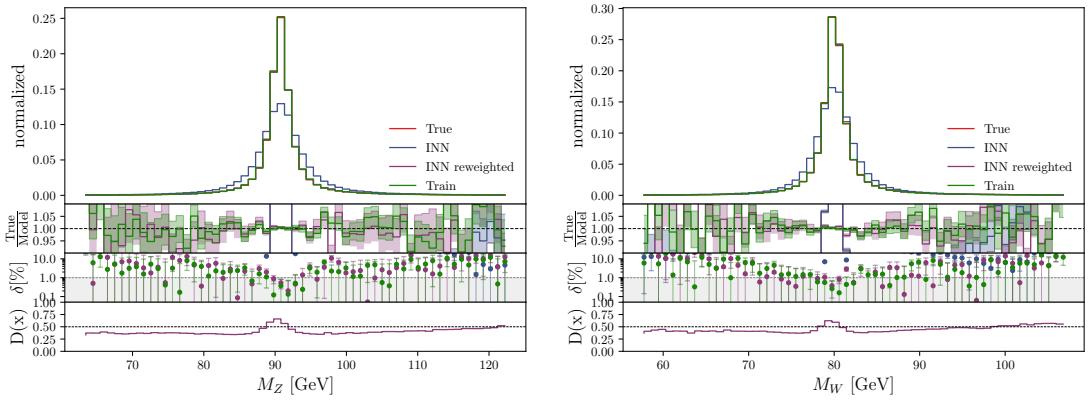
Since the complexity grew on this data set, we designed the input observables of the discriminator in $\mathcal{U}$ more carefully. The preprocessing of the discriminative network $DPreP$ consists of reverting the preprocessing of the generator $PreP$, defining a set of observables on $\mathcal{X}$, normalization and PCA. The discriminative network we used is summarized in table 5.5.

input observables	$\{M_Z, M_W\} \cup \{p_{x,i}, p_{y,i}, p_{z,i}\}_{i=1}^4$
number layers	6
nodes per layer	128
parameters to optimize	67k
spectralnorm	✗
gradient penalty	✗
L2	$1 \cdot 10^{-4}$
learning rate	$2 \cdot 10^{-4}$ (OneCycle)
batchsize	4k

Table 5.5: Setup ot the discriminative model for the WZ data Set

Note that we trained on the Cartesian momenta $\{p_{x,i}, p_{y,i}, p_{z,i}\}$ instead of $\{p_{T,i}, \eta_i, \phi_i\}$ we used as input for the generative model. With these input observables, the discriminative model performed better. This makes sense considering equation 3.6, showing how the Cartesian momenta are a multidimensional correlation of the generator observables.

Using the setup shown in table 5.5, we trained the discriminator for 400 epochs on the previously trained generative model. Since the mass distributions in figure 5.10 deviated the most compared to all other distributions, the discriminative network primarily focuses on the mass correlation.



(a) (b)

Figure 5.12: Reweighted mass distributions for the WZ data set (a) M_Z (b) M_W

The reweighted mass distributions perfectly match the validation curves with relative errors below 1% in the area of the peak.

Considering the weight distributions in the 2D η phase space region, for the discriminator setup in table 5.5, we can see visible weight structures, in the areas the generative model lacks in precision. Those structures in other distributions only become visible in the last few percent of reweighting in the mass peak. This shows how the discriminative model focuses on the most relevant structures first. Furthermore, we see how a large part of reweighting the events with incorrect mass correlation does not affect other phase space regions noticeably. In figure 5.13 we show the 2D weight distributions in the (η_{l1}, η_{l2}) phase space for two discriminative models trained for 200 and 400 epochs, respectively. The model trained for 400 epochs reweights the mass density distributions about 1-2 % better, than the model trained for 200 epochs. In figure 5.12 we show the mass distributions for the longer trained model, while the mass distributions for the shorter trained model can be found in the appendix.

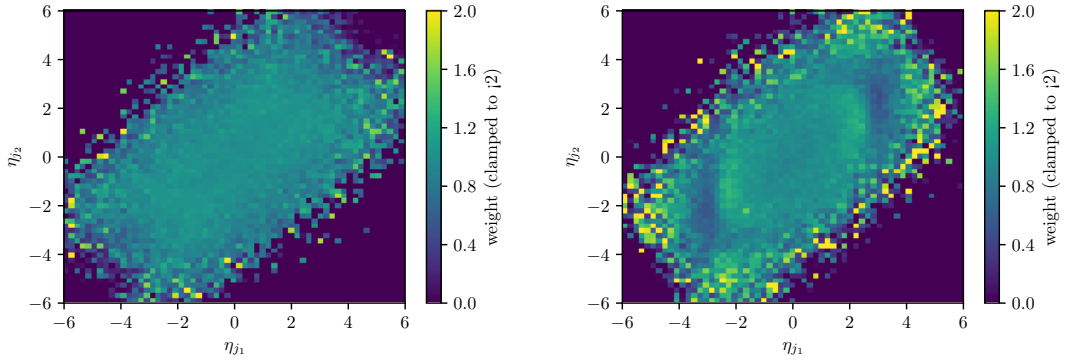


Figure 5.13: Bin-wise weight distribution in the (η_{l1}, η_{l2}) phase space for the discriminative model from table 5.3 (a) trained for 200 epochs (b) trained for 400 epochs

Training on other input, excluding the mass distributions led to great reweighting results for example in the η distributions as shown in figure 5.14.

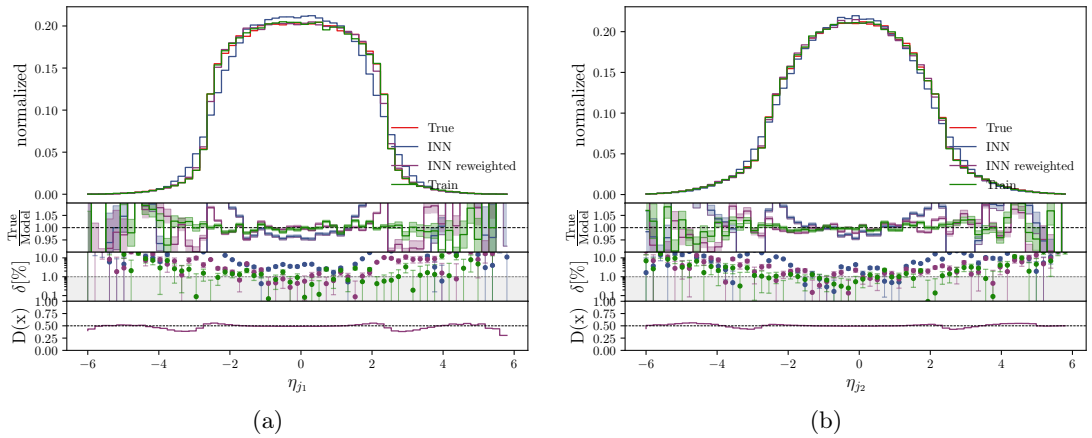


Figure 5.14: Reweighted pseudorapidity distributions for the WZ data set (a) η_{j1} (b) η_{j2}. The discriminative network was trained on the generator output without further preprocessing for 200 epochs

The reweighted pseudorapidity distributions have relative errors around the 1% mark in the regions of high statistics ($\eta \in [-2, 2]$).

In theory, we know for shure that the coupled latent and adversarial loss terms have a well-defined global minimum:

$$\mathcal{L}_{Latent} + \lambda_{Adv}\mathcal{L}_{Adv} \text{ is minimal} \Leftrightarrow P(x) = P_\theta(x).$$

Anyhow, we found, setting up a Flow-GAN converging towards this global minimum, is difficult on the Drell-Yan data set and even more difficult on the more complex WZ data set. For all setups so far, training as adversaries did not lead to a convergence to a global minimum but to bad local minima instead. When training the pretrained models summarized in tables 5.4 and 5.5 as adversaries, both loss objectives increased during training as exemplary shown in figure 5.15.

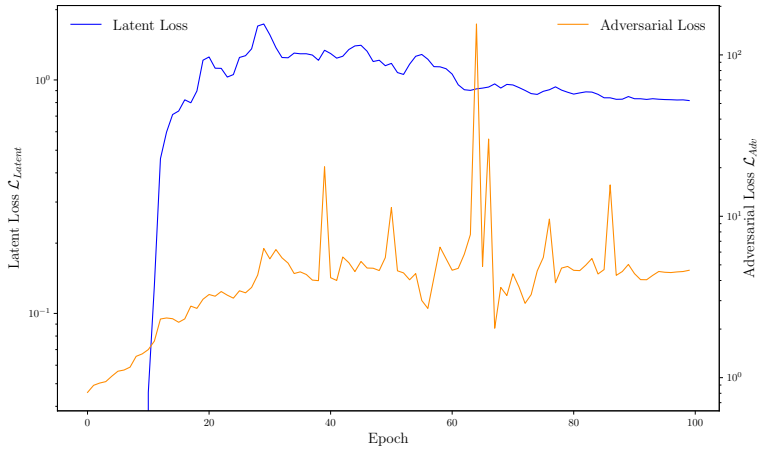


Figure 5.15: Course of the adversarial and latent loss in adversarial training with pretrained models (Coupling $\lambda_{Adv} = 1$, learning rate $1 \cdot 10^{-4}$ (OneCycle)).

Training the generative and discriminative models as adversaries without pretraining, led to a generative network converging to a local minimum in which either the distributions were not affected at all by the adversarial loss, or in a local minimum with improved precision on some distributions but drastically decreased precision on other distributions.

For this problem, using a hybrid objective is possible in theory, but becomes hard to implement for rising complexity of the problem. We think the main caveat making the optimization task for the generative INN a really complex problem is the fact that the losses are computed on both parton space $\mathcal{X}$ and latent space $\mathcal{Z}$. This makes it hard to control in which phase space regions and how the generative network optimizes for each one of the loss objectives. Therefore, we propose to use the discriminative network not on parton space but on the latent space instead.

3 Latent Space Refinement - an Outlook

Since transferring information from the discriminative model to the generative model using an additional adversarial loss term did not improve the generative model, we consider using the discriminative model on latent space. In this section, we want to take a look at the latent space $\mathcal{Z}$ to analyse the mapping behavior of $f_\Theta : \mathcal{X} \to \mathcal{Z}$ and give and outlook on information transfer from the discriminative model to the generative INN not in the parton space but in the latent space itself.

For reasons of better presentation, we consider the three-dimensional Drell-Yan problem in the following two exemplary plots. For the generator described in table 5.1, we transform the test data set under f_Θ onto the latent space. The distributions in latent space $\mathcal{Z}$ are shown in figure 5.16.

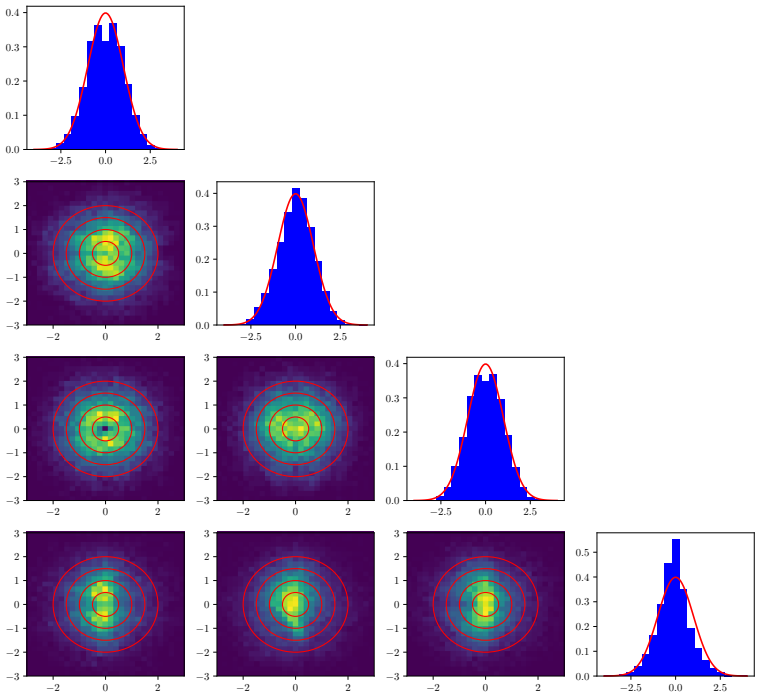


Figure 5.16: Distribution of the test data set transformed to the latent space for the Drell-Yan data set. 2D-Histograms: bottom row: x-axis: latent dimension 1, y-axis latent dimensions 2-4 from left to right, row above: x-axis: latent dimension 2, y-axis latent dimensions 3-4 from left to right, et cetera. 1D-Histograms: distribution in the latent dimension shown on the x-axis of the 2D-histograms in each row.

The optimal latent distribution is a standard normal distribution. In figure 5.16 we can

see how the mapping of real data onto the latent space differs from this optimal mapping behavior. Of course, these deviations can be the result of a lack of expressiveness of f_Θ but also topological constraints might play a role as discussed in section 4.

Additionally, to the already discussed reweighting effect in parton space, we can consider the reweighting effects of the discriminative model described in table 5.3 on the latent space. Therefore, we sample a set of instances $\{z_i\}_{i=1}^N$ from the real latent space distribution $P(z) = \mathcal{N}(0, \mathbb{I})$ and send them through the network in backward direction by applying $f_\Theta^{-1} : z_i \mapsto x_i$, which is the standard data generation procedure discussed in section 4. Then, the discriminative network is used to generate weights w_i to reweight each data pair (z_i, x_i). In figure 5.17 the weight dsitributions in latent space clamped to a maximum value of 2 are shown.

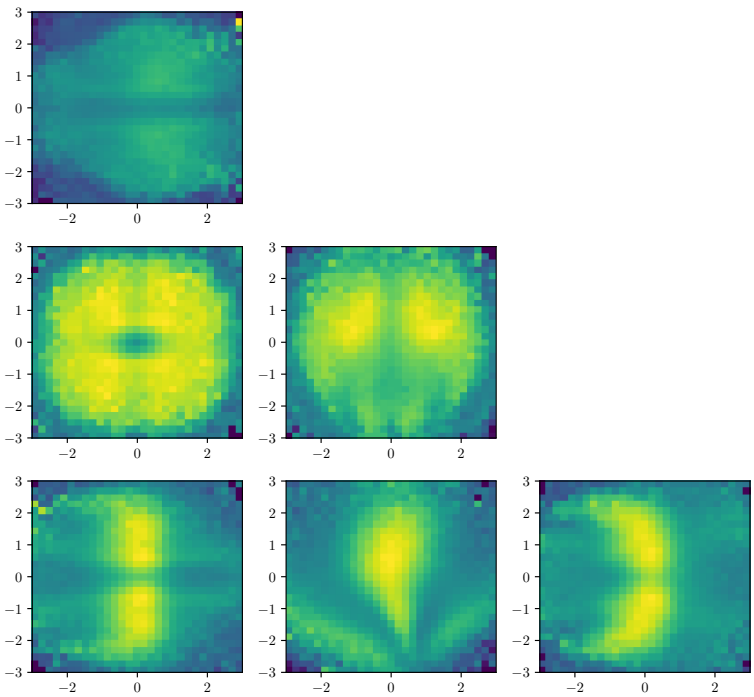


Figure 5.17: Weight distribution in the latent space for the Drell-Yan data set. 2D-Histograms: see figure 5.16

Comparing the 2D-histograms in figure 5.16 and figure 5.17 it becomes clear that reweighting in parton space indirectly enforces sampling from a latent space distribution similar or equal to the transformed data distribution under f_Θ.
The information of the discriminator can be used to create a reweighted latent space to sample from for example using the *Laser-Protocol* proposed in [34].

Furthermore, we can train a discriminative network on the latent space to distinguish between the latent Gaussian distribution and the models' mapping f_Θ. Besides the apparent differences between the one-dimensional distributions, the discriminative network can check whether the generative network maps onto a multidimensional correlated latent distribution $P_\Theta(z)$, since we do not have any correlations in $P(z)$. In this setup the adversarial loss is computed in the latent space, which might make the optimization task easier for the generative network. Furthermore, this approach results in unweighted events in parton space $\mathcal{X}$ since reweighting is directly possible in latent space. Nevertheless, we can not train on the specific observables we want to improve.

Another promising approach is to use the information that the discriminative model provides and modify the latent loss function of the ordinary INN, instead of introducing an additional loss term. Using the discriminator in parton space $\mathcal{X}$, we can rebalance the data $\{(x_i, z_i)\}_{i=1}^N$ by the inverse of the computed weight and compute the latent loss over these rebalanced data instances. That way the attention of the latent loss is moved towards events, with small weights. The modified latent loss objective for a standard normal latent distribution then reads:

$$\mathcal{L}^*_{Latent} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{w_i}\Big(\frac{f_\Theta(x_i)^2}{2} - \log[|J_z|_i]\Big). \tag{5.1}$$

Assuming a perfectly trained discriminative model, this loss has a minimum with respect to the weights if all weights are equal to one, which requires $P(x) = P_\Theta(x)$. First runs with this modified objective yield promising results that can be found in the appendix.

6 Conclusion

In this thesis we showed how an invertible network, used for LHC event generation, behaves with a hybrid objective in parton space.

As a first application, we applied the model to the Drell Yan data set. On this setup, we first used a MMD loss function computed on parton space additional to the maximum likelihood objective in latent space. For the mass distribution the MMD loss is applied to, we found an increase in precision when training the generative model on this hybrid objective. Anyway, we had no control over the effects of the additional loss objective in other phase space regions. This becomes particularly relevant, since the mass observable of each event is a multidimensional correlation of other observables. Using the Flow-GAN setup, we were able to control how the discriminative model influences other phase space regions, using the output of the discriminator to reweight all distributions in phase space event-wise. Furthermore, we were able to influence the the adversarial feedback by choosing the input observables for the discriminative model. Nevertheless, we found that it is hard to find a Flow-GAN setup in which both loss terms are minimzed simultaneously. Instead, the generative model converged to a equilibrium state, in which either the adversarial or the latent loss objective were minimized depending on the balance of both loss terms.

The second application was the WZ data set. Here the complexity of the problem is doubled as this data set consists of four final state particles instead of two. Using a discriminative on the generated events, we were able to reweight the generated events to a precision of around 1%, which is a massive improvement. Still, transferring the feedback of the discriminative model to the generative model using an adversarial loss term in parton space, to obtain unweighted events, failed.

For our current setup, using a hybrid objective in latent space and parton space did not lead to a generative model converging towards the global minimum. In theory and in application this is possible as shown in [9]. Nonetheless, for our problem it seems that implementing a hybrid objective in parton space is not very productive yet. Therefore, we propose to use the discriminative network not via a hybrid objective, but to reweight the latent space using the Laser protocol [34], to apply a discriminative model in the latent space, or use the discriminative model to rebalance the latent loss function. The latter recently showed promising results, which hopefully will be presented in an upcoming paper.

The summarized results of this thesis are:

1. We can use a MMD objective to improve precision on specific distributions, but have no control over the optimization effects in other phase space regions.

2. Using a discriminative network gives more control over how and which distributions are affected by the hybrid objective.
3. We can reweight the generated events to obtain distributions with relative errors around the 1% mark with respect to the validation data.
4. We can not yet use an adversarial loss term to nudge the generative model towards the theoretical global minimum.
5. We could do a reweighting procedure in latent space, not transferring the information of the discriminative model to the generative model.
6. We found first promising results modifying the latent loss function, using the discriminative network.

Anyhow, using a discriminative model on the generated density distributions helps us, since machine learning methods for event generation of LHC processes often lack precision and control over the generation process. By using a discriminative network, we get ahead in both of these caveats. The reweighting procedure gives precisely weighted events in good agreement with the events simulated using Monte-Carlo-Methods. Furthermore, we can use the discriminative feedback as a measurement on how precise our simulations are in certain phase space regions. Generating unweighted events needs further exploration of various approaches. Especially the latent space refinement leaves open a large area of further research. We hope to find a more elegant way, to give the generative model external feedback.

Bibliography

[1] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.-S. Shao, T. Stelzer, P. Torrielli, and M. Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. https://arxiv.org/abs/1405.0301, 2014.

[2] Anja Butter, Tilman Plehn, and Ramon Winterhalder. How to gan lhc events. https://arxiv.org/abs/1907.03764, 2019.

[3] Marco Bellagente, Anja Butter, Gregor Kasieczka, Tilman Plehn, and Ramon Winterhalder. How to GAN away Detector Effects. https://arxiv.org/abs/1912.00477, 2020.

[4] 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. https://arxiv.org/abs/2006.06685, 2020.

[5] Mathias Backes, Anja Butter, Tilman Plehn, and Ramon Winterhalder. How to GAN Event Unweighting. https://arxiv.org/abs/2012.07873, 2021.

[6] 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.

[7] Anja Butter et al. The Machine Learning landscape of top taggers. https://arxiv.org/abs/1902.09914, 2019.

[8] Michela Paganini. Machine Learning Algorithms for b-Jet Tagging at the ATLAS Experiment. https://arxiv.org/abs/1711.08811, 2018.

[9] Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Bridging implicit and prescribed learning in generative models. https://arxiv.org/abs/1705.08868, 2017.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. https://www.sciencedirect.com/science/article/abs/pii/0893608089900208?via%3Dihub, 1989.

[11] Ning Qian. On the momentum term in gradient descent learning algorithms. https://doi.org/10.1016/s0893-6080(98)00116-6, 1999.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2017.

[13] Yoann Boget. Adversarial regression. generative adversarial networks for non-linear regression: Theory and assessment. https://arxiv.org/abs/1910.09106, 2019.

[14] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. http://arxiv.org/abs/1708.07120, 2017.

[15] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. https://arxiv.org/abs/1802.05957, 2018.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. https://www.jmlr.org/papers/volume15/srivastava14a/, 2014.

[17] Peter Dayan, Maneesh Sahani, and Grégoire Deback. Unsupervised learning. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.2531, 1999.

[18] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. https://arxiv.org/abs/1803.08823, May 2019.

[19] 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. https://arxiv.org/abs/1808.04730, 2019.

[20] Framework for easily invertible architectures. https://github.com/VLL-HD/FrEIA.

[21] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. https://arxiv.org/abs/1906.02145, 2019.

[22] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. http://www.gatsby.ucl.ac.uk/~gretton/papers/SmoGreSonSch07.pdf, 2007.

[23] Anders Andreassen and Benjamin Nachman. Neural networks for full phase-space reweighting and parameter tuning. https://arxiv.org/abs/1907.08209, 2020.

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. https://arxiv.org/abs/1406.2661, 2014.

[25] Maciej Wiatrak, Stefano V. Albrecht, and Andrew Nystrom. Stabilizing generative adversarial networks: A survey. https://arxiv.org/abs/1910.00927, 2020.

[26] Eva Halkiadakis. Introduction to the LHC Experiments. https://inspirehep.net/literature/853601, 2010.

[27] Richard D. Ball, Valerio Bertone, Stefano Carrazza, Luigi Del Debbio, Stefano Forte, Patrick Groth-Merrild, Alberto Guffanti, Nathan P. Hartland, Zahari Kassabov, José I. Latorre, and et al. Parton distributions from high-precision collider data. https://arxiv.org/abs/1706.00428, 2017.

[28] Simone Marzani, Gregory Soyez, and Michael Spannowsky. Looking inside jets. https://arxiv.org/abs/1901.10342, 2019.

[29] Stephen D. Ellis and Davison E. Soper. Successive combination jet algorithm for hadron collisions. https://arxiv.org/abs/hep-ph/9305266, 1993.

[30] Yu.L Dokshitzer, G.D Leder, S Moretti, and B.R Webber. Better jet clustering algorithms. https://arxiv.org/abs/hep-ph/9707323, 1997.

[31] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. The anti-ktjet clustering algorithm. https://arxiv.org/abs/0802.1189, 2008.

[32] Tilman Plehn. Lectures on lhc physics. https://arxiv.org/abs/0910.4182, 2012.

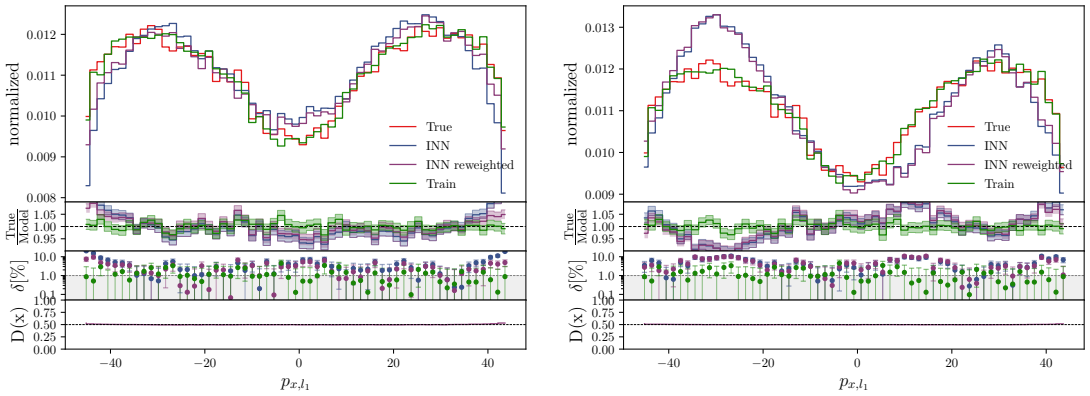
[33] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, A. A. Abdelalim, O. Abdinov, R. Aben, B. Abi, and et al. Measurement of w ± z production in proton-proton collisions at $\sqrt{s} = 7$ TeV with the atlas detector. https://arxiv.org/abs/1208.1390, 2012.

[34] Ramon Winterhalder, Marco Bellagente, and Benjamin Nachman. Latent space refinement for deep generative models. https://arxiv.org/abs/2106.00792, 2021.

List of Figures

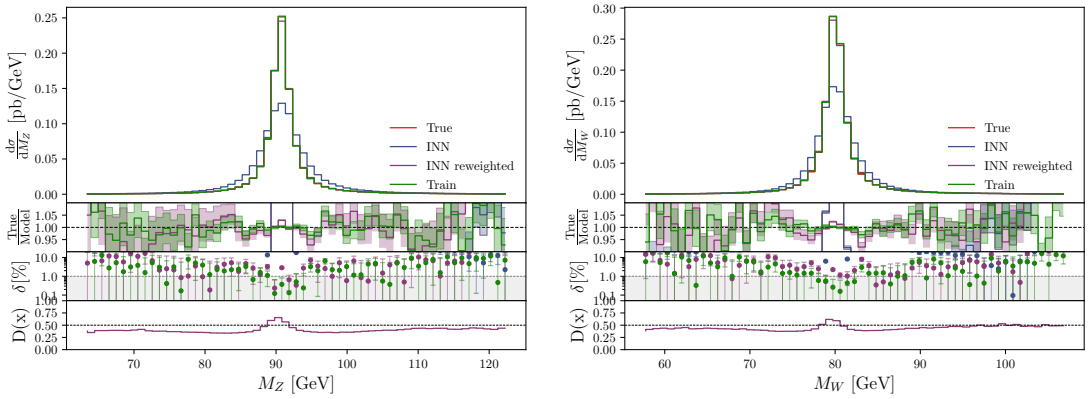
List of Tables

Appendix



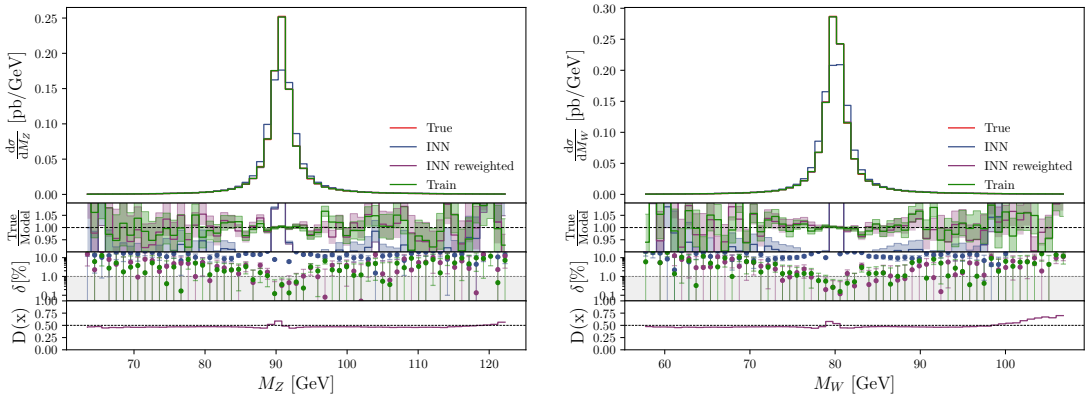
(a) (b)

$p_{x,l1}$ distributions with Flow-GAN objective for the Drell-Yan data set. The discriminative model is trained on $\{M_Z\}$ only. (a) $\lambda_{Adv} = 1$ (b) $\lambda_{Adv} = 10$

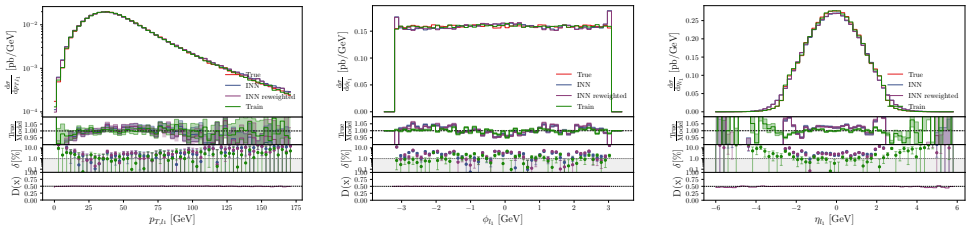


(c) (d)

Reweighted mass distributions for the discriminative model from table 5.5 trained for 200 epochs



(e) (f)



(g) (h) (i)

Generated event distributions using the reweighted latent loss objective. (a),(b): mass distributions of the Z- and W-boson, (c),(d),(e): p_T, η, ϕ distributions for the first lepton. The setup of the generative network is equal to the setup in table 5.4. The discriminative dense network has 256 layers à 256 nodes. Each of the networks was trained for 400 epochs in total.

Acknowledgements

First of all, I want to thank Tilman and Anja to give me the opportunity to work on this highly interesting topic in a great atmosphere. Moreover, I am highly grateful to Theo, who not only implemented the code base of our project but helped me out all along these months of research by answering questions, putting us back on track, and finally taking a look at my thesis. In regard to answering questions and providing code, I would also like to thank Armand and Ramon for their great support. Working together with Tobias was always a lot of fun and very helpful when discussing new ideas. Last but not least I want to thank the whole group for making this research project a really good time. It was a pleasure getting to know all of you!

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 7.8.2021, Sander Hummerich