Ruprecht-Karls-Universität Heidelberg

Fakultät für Mathematik und Informatik

Institut für Informatik

Bachelorarbeit

Invertieren von LHC Detektoreffekten mit Konditionalen INNs

Inverting LHC Detector Effects with Conditional INNs

Name: Armand Rousselot

Matrikelnummer: 3462443

Betreuer: U. Köthe

Datum der Abgabe: 16.03.2020

Erklärung

Zusammenfassung

Generative Modellierung hat in aktuellen Studien die Fähigkeit bewiesen simulierte Interaktionen, wie z.B. Detektoreffekte, in Large Hadron Collider (LHC) Analysen zu invertieren. Diese Modelle eröffnen vielfältige Methoden, um neue Informationen über Prozesse, die während der Kollision von Partikeln stattfinden, zu gewinnen. Kürzlich haben Konditionale Invertierbare Neuronale Netzwerke (cINNs) wettbewerbsfähige Leistungen in diversen generativen Aufgaben gezeigt. Per Konstruktion leiden diese nicht unter *mode collapse* wie Conditional Generative Adversarial Networks (cGANs). In dieser Arbeit zeigen wir, dass cINNs die Ergebnisse von cGANs in der Aufgabe Detektoreffekte zu invertieren in vielen Aspekten übertreffen. Wir demonstrieren die Fähigkeit unseres Modells der Leistung des cGAN im gesamten und in Teilen des Phasenraums gleichzukommen. Unser Modell erlangt verbesserte Resultate für die invarianten Massen von Teilen des Phasenraums. Zusätzlich zeigen wir, dass das cINN statistisch stimmigere Ergebnisse liefert. Wir stellen die Limitierungen dieser neuen Architektur dar, indem wir zusätzliche Störeffekte in der Form von *initial state radiation* in die Simulation einführen. Zuletzt evaluieren wir die Fähigkeit des cINNs physikalische Strukturen in Testdaten zu erkennen, die in den Trainingsdaten nicht präsent waren.

Abstract

Generative modeling has recently been shown capable of inverting simulated physical interactions such as detector effects in Large Hadron Collider (LHC) analyses. These models provide a powerful tool for gaining new information about the processes during particle collisions. Recently, Conditional Invertible Neural Networks (cINNs) have shown competitive performance in several generative tasks. By construction they don't suffer from mode collapse as Conditional Generative Adversarial Networks (cGANs) do. In this work we show that cINNs surpass the results recently achieved with cGANs in several aspects in the task of inverting detector effects. We demonstrate that our model matches the cGAN's performance on the full and parts of the phase space. We achieve improvements on the invariant mass of cuts of the phase space. Additionally, we show that the cINN creates statistically more coherent results. We outline the limits of the new architecture when introducing new disruptive effects in the form of initial state radiation. Finally we evaluate the cINN's ability to recognize physical structures in test data which were not present in its training data.

Contents

1. Introduction

One of the pillars of modern physics are the experiments and insights gained at particle colliders. These colliders launch particles at each other at velocities close to the speed of light. In each collision the Large Hadron Collider (LHC) accelerates two protons, which belong to the class of hadrons, composites made up of quarks among other particles [1]. When these protons hit each other, they produce a wide range of new particles that give us an insight into the relations and interactions of matter at the scale of quarks. Large amounts of data are produced at particle colliders, which brings the need of automated processing. In 2018 the LHC alone produced upwards of 88 PB of data, even after filtering out over 99% of it [9]. The processes at colliders where substantial amounts of momentum are transferred are called hard processes. We need to be able to inspect measured data with as little disruptive effects as possible to gain information about the processes and find new relations, e.g. by detecting anomalies or unexpected data peaks.

Removing the particles' interactions on the way to the detector and measurement errors introduced in particle colliders (unfolding) and simulating collision data have already been attempted using Generative Adversarial Networks [5][7]. However, these models deal with limitations. Most importantly they are incapable of correctly capturing mass correlations of particles over parts of the distributions. We propose a new model to improve the results on this task, based on Invertible Neural Networks [14] [2] [21]. These networks are related to Normalizing Flows, which have been used in event generation before [15][22]. Invertible Neural Networks have recently been adapted to a conditional architecture for generative tasks [3]. The target distribution, in this case the distribution of the particles before any of the interactions on the way to the detector and without any measurement error, is transformed into a prescribed latent distribution, e.g. a gaussian normal distribution. This mapping is conditioned on the measured detector data, which can be transformed by an arbitrary subnetwork beforehand. Generating events is easily achieved, due to the invertibility of the network. Samples from the known latent distribution can efficiently be transformed back to the target domain, thus providing a prediction for the particles shortly after the collision conditioned on the noisy detector data. In this work, we show the Conditional Invertible Neural Networks' performance in the task of removing disruptive effects from simulated events and compare them to the results of the cGAN built by [5]. We show that our new approach outperforms the adversarial networks in several points and apply it to data with additional sources of uncertainties.

2. Physics Background

For a better description and understanding of the task introduced in the previous chapter it is necessary to take a basic look at the physics inside a particle collider. As mentioned in Chapter 1 we specifically look at the collision of protons. Their constituents interact with each other, creating new particles. We call the physical quantities of the particles directly created by the interaction of two hadrons the parton-level observables. These observables can provide an insight into the interactions between particles and guide the way to gaining new knowledge about physics.

The process that will be discussed in this thesis is the production of a Z and a W Boson, resulting from a collision of two protons at the LHC [1]. In our experiments we only consider events where the Z Boson decays into two leptons, while the W Boson decays into quarks [6]. Each quark undergoes a process called showering, where due to the emission of gluons many hadrons are created. These particles are then clustered into a single object, a so-called jet. The process is given by $pp \to ZW^{\pm} \to (\ell^+\ell^-)(q_1q_2)$ and can be described by a feynman diagram 2.1. We represent this process by a simulation we call the parton shower.

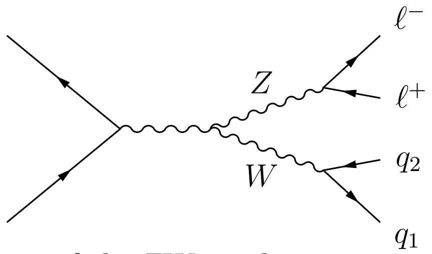

Figure 2.1.: A feynman diagram of the ZW production process. On the left, the initial state particles collide, resulting in a Z and a W Boson, which decay further into two leptons (ℓ^+/ℓ^-) and two jets (q_1/q_2) respectively.

Each of these particles can be represented as a four-vector [38]. The first entry contains the particle's energy, which is always non-negative, while the following three entries signify the x-, y- and z-momentum respectively, meaning we can write any particle x_p as

$$x_p = \begin{pmatrix} E \\ \mathbf{p}_x \\ \mathbf{p}_y \\ \mathbf{p}_z \end{pmatrix} . \tag{2.1}$$

Additionally, via the fact that energy and momentum are conserved in a closed system [38], if a particle decays, the sum of all of its decay products' four-vectors will result in the original particle's four-vector. Finally, special relativity states that the energy of a system can be calculated using its momentum $\vec{\mathbf{p}}$ and total resting mass $\mathbf{m}$, as well as the speed of light c.

$$E^2 = ||\vec{\mathbf{p}}||_2^2 c^2 + \mathbf{m}^2 c^4 \tag{2.2}$$

This means the energy of a particle is computable if its mass and momentum are known. This only holds true as long as the calculations are performed in a closed system, i.e. no energy or momentum can be given off to another particle that is not considered. In the case of the quarks and leptons produced by the process, their masses are negligible, therefore their energies can be estimated by $E^2 \approx ||\vec{\mathbf{p}}||_2^2 c^2$. This means they can be represented using only the lower three entries of a four-vector. Furthermore, to ease the notation, every time a mass or momentum is written, it can instead be expressed in terms of an energy, so for example $\vec{p} = \begin{pmatrix}\mathbf{p}_x c, & \mathbf{p}_y c, & \mathbf{p}_z c\end{pmatrix}^T$, instead of Equation 2.1. In the same way masses can be written as $m = \mathbf{m}c^2$. All of the following equations will be given in these quantities.

The hard process itself is not directly observable. We need to measure the products that resulted from the collision. Among other methods, particle colliders employ calorimeters to absorb jets and measure their energies, as well as trackers, to follow the paths of charged particles. A calorimeter forms a cylinder around the accelerator at the collision site, which is divided into multiple cells. Each of these cells is able to absorb and measure the energy given off by a particle [26]. Trackers operate via a magnetic field that bends the paths of charged particles moving across. This effect can be measured and used to calculate the momentum of the particles [31]. The combined measurements are then used to reconstruct four-vectors of each jet, creating the detector-level observables. While we can determine the momentum and energy of the particles at that point in time to reasonable accuracy, we do not immediately know how the observables looked on parton-level. As the decay products travel the distance from the collision site to the calorimeter until they are absorbed, they interact on the way. For example, uncharged particles can be created, which are hard to detect and might not be attributed to the jet. Additionally, errors in the detector measurement lead to further "smearing" of the result. Problematically, our main interest lies in the parton-level observables.

It is important to point out that the process of measuring particles is probabilistic by nature, once one simulates the particle and detector interactions realistically. There are many parton-level events that can be the origin of a given detector measurement. As a consequence we do not just need a way to undo the smearing and particle interactions and create a single set of parton-level observables. Instead we want a probabilistic mapping from detector to parton. There already exist Monte-Carlo methods to simulate the process from parton- to detector-level, such as Pythia and Delphes [34][12]. These programs use physical relations to estimate probability distributions of certain interactions. By repeatedly sampling from these distributions it is possible to create realistic data for parton- and detector-level measurements. However, these methods only allow for the simulation of the direction from parton- to detector-level, inverting them is not feasible [5]. The goal of this work is to create a model that can simulate the reverse direction via machine learning, more specifically via an architecture called Conditional Invertible Neural Networks, introduced in Chapter 5. Pythia and Delphes are used to create events for training this model.

2.1. Invariant Mass

Transforming Equation 2.2 yields that the mass of each particle can be calculated as

$$m^2 = E^2 - ||\vec{p}||_2^2. \tag{2.3}$$

On parton-level, the invariant mass of the W Boson can be approximated by a breit-wigner distribution [34]

$$p(x) \propto \frac{\Gamma^2}{(x^2 - M^2)^2 + M^2\Gamma^2}. \tag{2.4}$$

A breit-wigner distribution does not possess a finite standard deviation. Therefore Γ, which controls the width of the distribution, is called the scale parameter and M, which controls the location of the peak, is called the location parameter. The error in jet reconstruction leads to the distribution of the masses changing to be wider spread than before (see Figure 2.2).

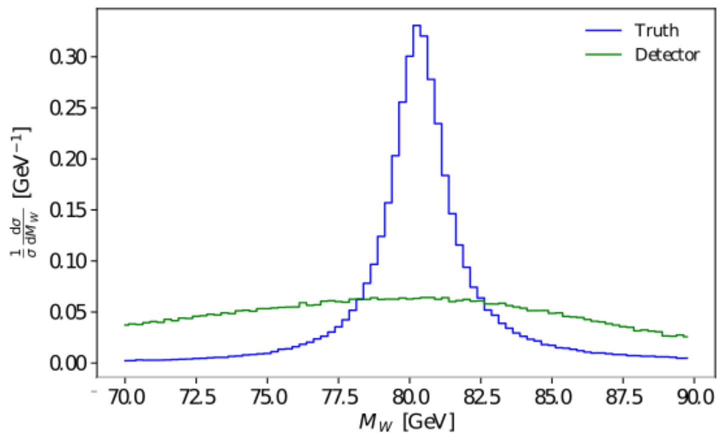

Figure 2.2.: Invariant mass distributions of the W Boson on parton- (blue) and detector-level (green).

As has been found by [5] and reconfirmed during this work, it is quite hard for models to learn this original mass distribution on their own. A solution will be introduced in Section 5.4.

2.2. Initial State Radiation

Before the collision, the partons involved in the hard process have a chance to give off radiation in their initial state (ISR) [34]. While the effect of this radiation on the hard process that occurs afterwards does not change the fundamental kinematics, ISR can introduce disturbances in the measurement in another way. In a calorimeter the measurements of each cell are clustered into jets using algorithms such as the anti-k_t algorithm [8]. If the ISR hits the detector in close proximity to the jets created by the W Boson, some of the particles in either jet might be falsely attributed by the clustering algorithm. Additionally, while the ISR is rarely as hard (i.e. its transverse momentum is as high) as the W jets, they sometimes lie in the same order of magnitude. In this case, it is hard to differentiate which quarks are created by the W and which stem from ISR. Picking the wrong detector-level jet to predict the parton-level jets from obviously ends up preventing a correct reconstruction. For the baseline results, ISR is not included in the data. In Section 6.5 we train our model on a separate dataset with ISR.

3. Machine Learning Background

3.1. Neural Networks

Artificial Neural Networks are a form of learning algorithm that is loosely based on the way the brain processes information. In the brain, inputs are processed by a multitude of neurons that form internal connections. As soon as a neuron receives signals that exceed a certain threshold, the neuron sends a signal to all other neurons connected to it.

3.1.1. Layers

Inspired by the parallels to the brain, the basic building blocks of every neural network are called neurons. A neuron transforms its input data vector by assigning a weight to each entry, then summing over the weighted data. Finally an activation function is applied to the output, to introduce non-linearity to the network [37]. Each neuron can thus be described by the transformation [30]

$$y = \phi(w^T x). \tag{3.1}$$

Where ϕ is the activation function, w the weight vector and x and y the inputs and output respectively. By extending the weight and input vector $\hat{w} = \begin{pmatrix} b \\ w \end{pmatrix}$, $\hat{x} = \begin{pmatrix} 1 \\ x \end{pmatrix}$ we can introduce a bias, to account for non-centered data. The simplest neural network, is built of only a single neuron (Figure 3.1).

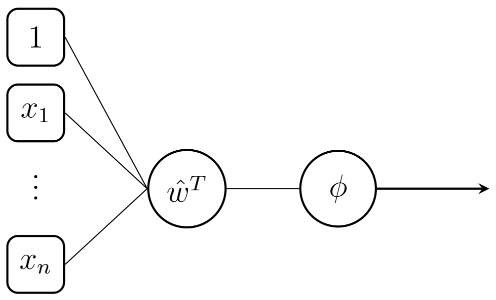

Figure 3.1.: A single neuron. The inputs $x_1, ..., x_n$ are concatenated with a constant input 1 to form the input vector. Its vector product with $\hat{w}^T$ is then used as the input for the activation function ϕ to form the output of the neuron.

Looking again at Equation 3.1, it becomes apparent that, no matter the dimensionality of x, the output of a single neuron is one-dimensional. In case a higher-dimensional output is required, multiple different neurons have to be stacked on top of each other, which is called a (fully connected) layer. The activation function can also be separated from the weight multiplication to further specify the layers. The former is then called an activation layer, the latter a linear layer. We may also concatenate multiple layers laterally to increase the predictive power of the neural network. The inner layers (or hidden layers) widths (i.e. number of neurons) are usually chosen larger than the input/output layer width. Otherwise we would force the network to compress information while passing data through these layers. A simple neural network might now look something like illustrated in Figure 3.2. Training the neural network means finding the optimal weight vector $\hat{w}_i$ for each of the neurons n_i in the network.

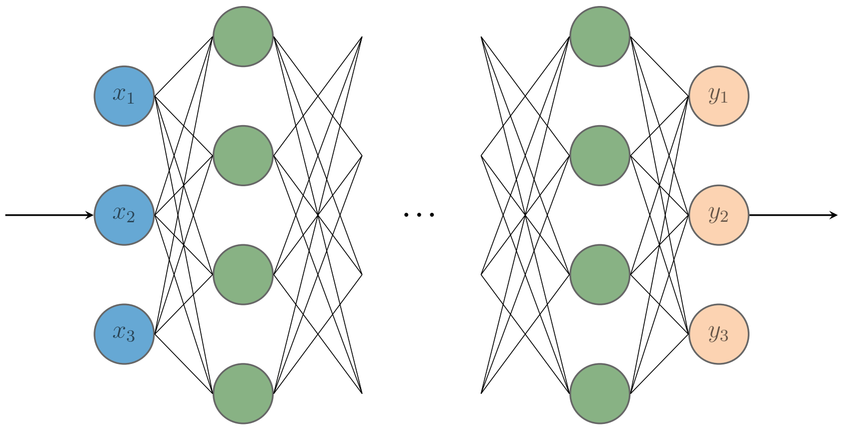

Figure 3.2.: A sketch of a small neural network. Each circle represents a neuron. The input layer on the left is marked in blue, the hidden layers in green and the output layer on the right in orange.

3.1.2. Optimizers

Before training a neural network the goal of the training has to be defined first. This is done by a short function that scores the network outputs based on the inputs it received. Such a function is called a loss function. Traditionally the loss is minimized, i.e. the output of the loss function should decrease the better the network gets. The gradient of the loss function is calculated and propagated backwards through the network to find out how the trainable parameters of the model (e.g. the weight matrices of each neuron) need to be tuned in order to minimize the loss. This can be done recursively, starting from the last layer, the gradient of the loss with respect to each layer can be calculated from the gradient of the following layer and the weights of the current layer. With x_l being the input to layer l and W_l the weight matrix of the same layer we can express this by the equation

$$\begin{aligned}\tilde{\delta}_l &:= \frac{\partial Loss}{\partial(W_l x_l)} = \frac{\partial Loss}{\partial(W_{l+1}x_{l+1})}\frac{\partial(W_{l+1}x_{l+1})}{\partial\phi(W_l x_l)}\frac{\partial\phi(W_l x_l)}{\partial(W_l x_l)} \\ \nabla_{W_l}\mathscr{L} &:= \frac{\partial Loss}{\partial W_l} = \frac{\partial Loss}{\partial(W_l x_l)}\frac{\partial W_l x_l}{\partial W_l} = x_l\tilde{\delta}_l^T. \end{aligned} \tag{3.2}$$

Assuming the gradient of the loss with respect to the output is known, we can now find the gradient to minimize the loss for each layer. As a consequence of Formula 3.2, every step of the prediction process and loss function must be differentiable. Updating each parameter according to its gradient directly often results in unstable training or hinders proper convergence. This can be combated by using optimizers that recalculate the size of each gradient update, preventing overstepping [11]. The loss function can have multiple local minima over the model parameter space. However, the optimization process always aims for a global minimum. The optimizing algorithm's ability to find new, better minima is called exploration. On the other hand, once the global minimum has been found, the goal is to descend into it as accurately as possible. This is called exploitation. There typically exists a trade-off between the two, i.e. the more improvements are made to exploration, the worse the optimizer gets with respect to exploitation and vice-versa. A visualization of the exploration-exploitation trade-off is shown in Figure 3.3

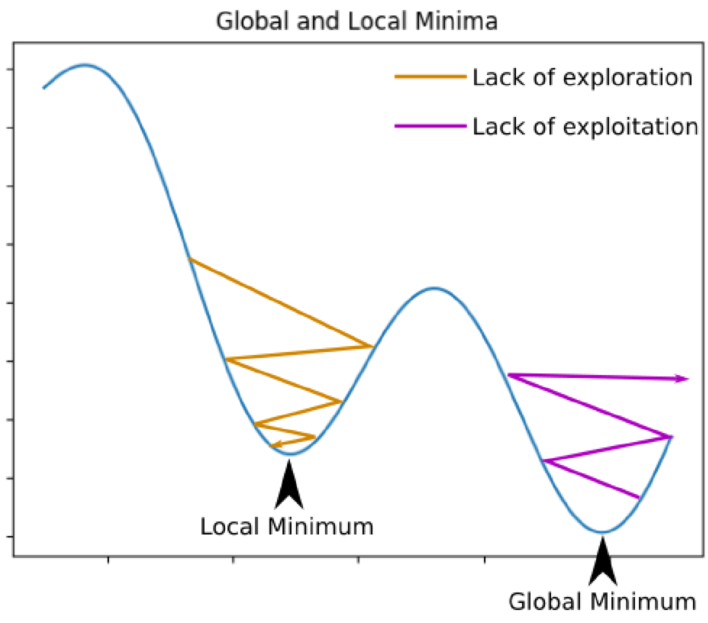

Figure 3.3.: A visualization of the exploration-exploitation trade-off. The blue line shows an exemplified loss function over the parameter space. The orange line shows an optimizer that cannot find the global minimum, as its exploration capability is too bad. The purple line shows an optimizer that has too little exploitation potential and therefore is unable to descend further into the global minimum.

Gradient Descent is the naive approach to optimizing the network's trainable parameters. After the gradient of the loss $\{\mathscr{L}_i\}_{i=1}^N$ for the whole training set of size N is calculated at update step t, we update the model parameters θ according to a learning rate η.

$$\theta_{t+1} = \theta_t - \frac{\eta}{N} \sum_{i=1}^{N} \nabla_{\theta_t} \mathscr{L}_i \tag{3.3}$$

This approach comes with several problems. The gradients for the whole dataset need to be stored in memory simultaneously. If the network, the data dimensions or the dataset size is too large, this approach is not feasible. Additionally the exploration potential of gradient descent is very limited [11].

Stochastic Gradient Descent (SGD) solves this problem by performing the gradient update after every single prediction.

for i **from** 1, ..., N:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \mathscr{L}_i \tag{3.4}$$

This not only removes the need to load the gradients of the whole dataset into memory at once, but increases the ability of the algorithm to find a better global optimum. This is due to the fluctuations in the updates introduced by only considering the outcomes of single data points, which increase the exploration potential. The trade-off is that these fluctuations can also prevent the algorithm to properly converge to a minimum.

Mini-Batch Gradient Descent forms a compromise of SGD and Gradient Descent by averaging the gradient not over the whole dataset but over a small subset, or batch, with a fixed size b.

for i **from** 1, ..., $\frac{N}{b}$:

$$\theta_{t+1} = \theta_t - \frac{\eta}{b} \sum_{j=i}^{i+b} \nabla_{\theta_t} \mathscr{L}_j \tag{3.5}$$

Momentum aims to reduce the erratic nature of SGD and smooth out the updates by introducing a fraction γ of the previous update to the next one (moving average).

$$\begin{aligned} m_{t+1} &= \gamma m_t + \eta \nabla_{\theta_t} \mathscr{L} \\ \theta_{t+1} &= \theta_t - m_{t+1} \end{aligned} \tag{3.6}$$

This especially helps reduce the oscillation of updates in regions of the parameter space where large gradients in some dimensions are paired with low gradients in other dimensions (see Figure 3.4). The disadvantage of this method can also be extrapolated from the illustration. Once we reach the minimum, the momentum we have accumulated might be so big that we overshoot it entirely.

The Nesterov accelerated gradient is built to prevent this overshooting of the minima by performing a prospective gradient update [29]. Instead of calculating the gradient from the point in the parameter space the model parameters are currently located at, we first apply the momentum term. Only then is the gradient calculated, which takes

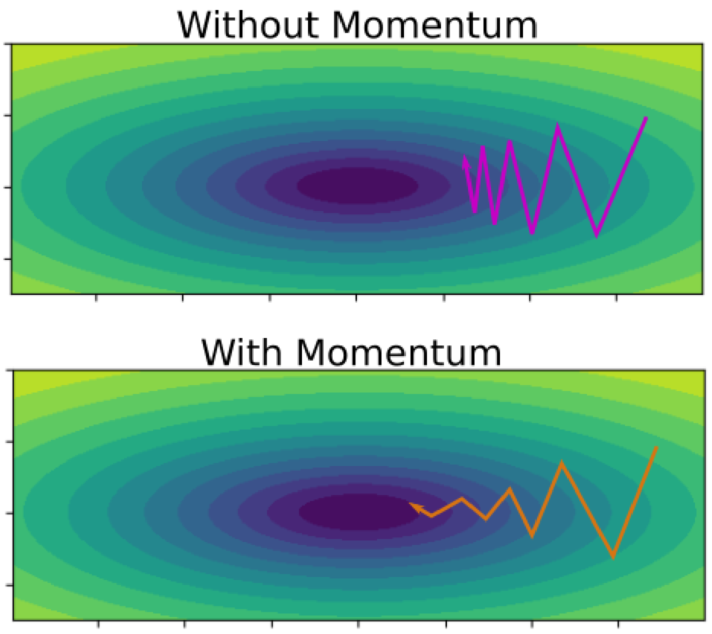

Figure 3.4.: An illustration of the advantage that momentum provides. In the top panel, the optimizer oscillates a lot in the vertical direction, whereas on the bottom the momentum gradually amplifies the horizontal direction of the updates

the anticipated step by momentum into account. If the momentum term now overshoots the minimum, the gradient will compensate for this.

$$
\begin{aligned}
m_{t+1} &= \gamma m_t + \eta \nabla_{\theta_t + \gamma m_t} \mathscr{L} \\
\theta_{t+1} &= \theta_t - m_{t+1}
\end{aligned}
\tag{3.7}
$$

RMSProp is founded on the idea that it is not just enough to have a global learning rate, but instead a separate learning rate for each parameter is required. This is motivated by the observation that magnitudes of gradients can vary from layer to layer within a neural network. Bigger updates in the early layers, where the gradients can become very small, are advantageous. Another factor constitute the input numbers (fan-in) of each neuron. When the weight of each of the inputs of a neuron is updated simultaneously the risk of overcompensating exists if the fan-in is large. As the fan-in can vary greatly throughout the network, we need to be able to adapt the gradient updates accordingly. RMSProp enables this by keeping track of the moving average of the squared gradient updates. If a certain parameter is constantly updated by a low gradient, the root of the mean gradient squares (RMS) will be small, and vice-versa. Consequently dividing by the RMS will adjust the gradient updates for each parameter.

$$
\begin{aligned}
v_{t+1} &= \gamma v_t + (1-\gamma)(\nabla_{\theta_t} \mathscr{L})^2 \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon} \nabla_{\theta_t} \mathscr{L}
\end{aligned}
\tag{3.8}
$$

Here, ϵ is a regularization parameter, to avoid instabilities from small gradients.

ADAM combines momentum and RMSProp to adaptively estimate the moment of each gradient in an update [20]. Additionally, it addresses another problem with both RMSProp and momentum. Their moving averages are biased towards the gradients of the first batches at the start of the training. ADAM divides the moving averages by a bias correction factor that decreases with training time to account for this.

$$\begin{aligned}
m_{t+1} &= \beta_1 m_t + (1-\beta_1)\nabla_{\theta_t}\mathscr{L} \\
v_{t+1} &= \beta_2 v_t + (1-\beta_2)(\nabla_{\theta_t}\mathscr{L})^2 \\
\hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^t} \\
\hat{v}_{t+1} &= \frac{v_{t+1}}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}}+\epsilon}\hat{m}_{t+1} \qquad (3.9)
\end{aligned}$$

Apart from choosing a smart optimization function, it is advisable to additionally decrease the overall learning rate during training. There are many different methods to adjust the learning rate, e.g. exponential decay, decaying in steps at set epochs or reducing the learning rate once the validation loss reaches a plateau. On one hand, not reducing the learning rate can obviously lead to overshooting the loss minimum. Reducing the learning rate too early on the other hand might leave us in a local minimum, unable to reach the global one. This is another example of the exploration-exploitation trade-off. For the training performed in this work, we used the ADAM optimizer, combined with a learning rate decay by a factor of 0.4 on loss plateaus.

3.1.3. Activation Functions

When creating the activation layers there are several popular choices regarding the function $\phi(x)$ [30]. In the following section, some of the most important ones are showcased.

The Sigmoid function is given by the formula

$$\phi(x) = \frac{1}{1+e^{-x}}. \qquad (3.10)$$

This function provides a lot of advantageous properties. Its derivative is positive everywhere and it is bounded, which prevents the explosion of values in the forward pass. However, it has proven to also introduce some problems, especially when training deep networks. Since its derivative approaches zero at both edges, the gradient gets dampened and when back-propagating through many layers we quickly encounter the "vanishing gradient" problem. The gradient for the layers in front becomes so small that they converge very slowly to the optimum.

The ReLU function is given by the formula

$$\phi(x) = max(0, x). \tag{3.11}$$

The ReLU function rectifies negative inputs to 0, but leaves positives inputs untouched. It thereby solves the vanishing gradient problem. Another big advantage of the ReLU function is that it is very easy to compute, compared to the exponentiation and division in the sigmoid function. Another property that can sometimes be used is that it introduces sparsity in its outputs, as all negative numbers are set to 0. However, this very effect can also become a disadvantage. As training progresses, some neurons effectively "die" meaning their output is always set to 0 and the neuron is completely ignored.

The leaky ReLU function is given by the formula

$$\phi(x) = \begin{cases} x & x \geq 0 \\ \alpha x & \text{else} \end{cases} \tag{3.12}$$

The leaky ReLU function tackles the problem of dead neurons, as the contributions of negative outputs are now non-zero. Typically α is chosen very small, as to not significantly change the results of the training procedure apart from the dead neuron problem. On the downside, we obviously loose the sparsity introduced by ReLU.

The parametric ReLU (PReLU) function is given by the formula

$$\phi_i(x) = \begin{cases} x & x \geq 0 \\ \alpha_i x & \text{else} \end{cases} \tag{3.13}$$

Improvements over the standard leaky ReLU have been found by allowing the network to learn a different factor α_i for each of the neurons n_i during training.

The Softmax function is given by the formula

$$\phi(x_i) = \frac{e^{x_i}}{\sum\limits_{j=1}^{l} e^{x_j}}. \tag{3.14}$$

where the current layer contains neurons $n_1, ..., n_l$. This activation function produces an output that ranges between 0 and 1, based on the amplitude of x_i in relation to the rest of the layer's outputs. This definition yields a probability distribution over the outputs $\phi(x_1), ...\phi(x_l)$, i.e. $\sum\limits_{j=1}^{l} \phi(x_j) = 1$. As such, it is used for creating the class probability output in the final layer of a model trained on classifying tasks.

The SoftPlus function is given by the formula

$$\phi(x) = \log(1 + e^x). \tag{3.15}$$

This function constitutes a smooth version of the ReLU function.

A visualization of the Sigmoid, (Leaky /P)ReLU and SoftPlus functions can be found in Figure 3.5. For this work, we utilized the ReLU function as the network activation functions. In Section 6.5 we use a classifying network to differentiate the origin of jets. We apply the Softmax function in the final layer of the classifier to create the label probabilities.

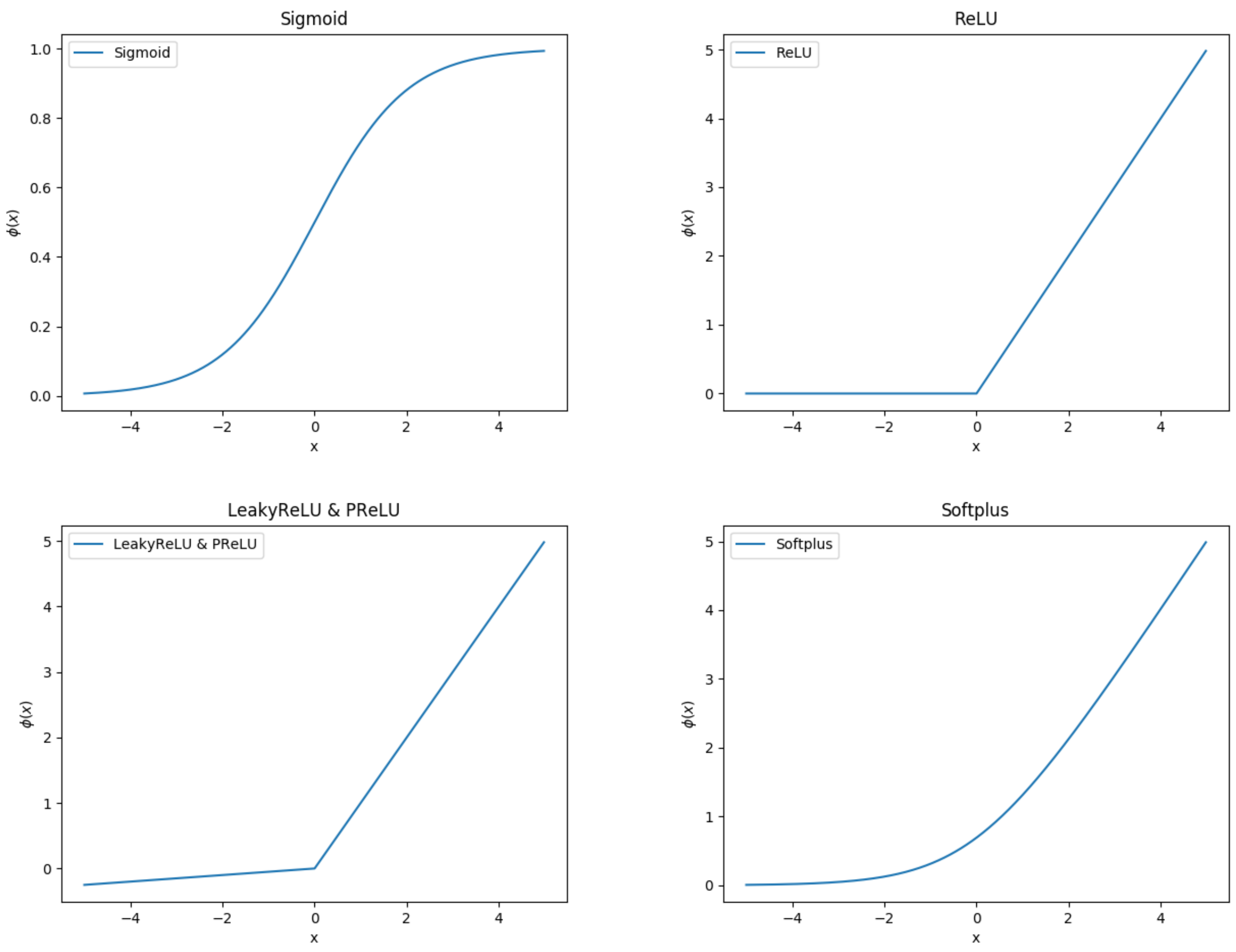

Figure 3.5.: A visualization of the different activation functions described in this section. The difference between Leaky ReLU and PRelU lies only in the slope of the negative x side of the function being learnable in the latter. The Softmax function is not depicted as its concrete form depends on the output of all of the layer's neurons.

3.1.4. Overfitting

A common problem that arises during the training of neural networks is overfitting. In an optimal setting, we would like to generate new data for every batch to train the model with. Realistically this is of course rarely possible. Thus, after a certain point, it becomes necessary to reiterate over the entire dataset from the start. As the model tailors its predictions on the training data that it sees repeatedly it can ”overspecialize”, or in other words it overfits. Overfitting is easily detectable by using a separate test set to evaluate the model performance. If the loss for the training set is significantly lower than for the test set, the model has learned to base its predictions on details about the training data that are insignificant for the process that the network is trained on. However, by remembering these natural variations in the data, the model can differentiate between individual data points and provide the predictions that are associated with them. The bigger and more powerful our model is, the larger its capability to overfit becomes, as it can remember more single data points.

The obvious first solution to this problem is to either reduce the model size or use more training data. This is oftentimes not possible. When a model is trained to perform a complex task, we can not resort to shrinking it beyond a certain point, as its predictive power will become too small. Increasing the dataset size in real world applications can be very expensive, both in terms of time and money.

Dropout is a more sophisticated method to reduce overfitting of neural networks [18]. As mentioned earlier, overfitting can occur when the predictions of the model are based on a (intermediate) feature of the data that is insignificant for the underlying correlation. By randomly setting single features (neuron activations or input features) within the network to 0 for every batch, we take away the network's ability to rely on just a single aspect of the data. Unfortunately, this does not come without a disadvantage. Choosing the chance of dropout too low might not lead to the desired impact, choosing it too high can severely decrease the training success.

Regularization or weight decay stems from the observation that overfitting networks oftentimes have disproportionally big weights [24]. With the right combination of features, which might be present in the training data, the effect of these weights can cancel out. Once the network is applied on the test data, where the feature combination is not always present, the difference is amplified by the big weights and the effect of overfitting becomes worse. Using weight regularization an additional constraint is added to our training objective in the form of the mean squared value of the network weights. Minimizing the loss now also inevitably forces the model to retain small weights, otherwise the regularization term would grow. This method suffers from the same disadvantage as dropout.

Early Stopping can also combat overfitting [35]. Even if the optimizer manages to find the global optimum the model can still overfit if the training continues for too long. After learning the true correlation as accurately as possible, the only way for the model to improve its predictions is to incorporate additional features into its prediction process, which are unique to the training data. When using a test set we can track the difference of test and training loss and quit once the test loss starts to rise.

In this work, we use a small weight decay of 1e-5. We found that early stopping and dropout were not necessary, as overfitting did not have much of an effect once we used adequately sized datasets.

4. Related Work

4.1. Generative Adversarial Networks

A promising way to learn detector unfolding are Generative Adversarial Networks (GANs). The authors of [5] set up a GAN for the task of unfolding detector effects. This form of network consists of two opposing parts, a generator and a discriminator, which compete in a min-max game. In their version, the former part receives a detector-level event x_d as its input and outputs generated parton-level events x_{gen}. In other words, the generator transforms the detector distribution into the generated parton distribution $G : x_d \sim p_d(x_d) \to x_p \sim p_{gen}(x_p)$. The discriminator on the other hand takes events from parton-level, either real or generated, as an input. It outputs the probability that the given event stemmed from $p_{real}(x_p)$, i.e. $D : x_p \sim p_{real/gen}(x_p) \to [0, 1]$. The objective of the discriminator is to correctly tell apart samples from either distribution, while the generator's objective is to fool the discriminator. As a consequence, over the course of the training, the generator will create more realistic samples, as the discriminator learns more defining features about the true distribution $p_{real}(x_p)$. We can formulate this goal in terms of the loss function [16]

$$\mathscr{L}_D = \mathbb{E}_{x_p \sim p_{real}(x_p)} \left[\log(D(x_p))\right] + \mathbb{E}_{x_d \sim p_d(x_d)} \left[\log\left(1 - D\left(G\left(x_d\right)\right)\right)\right]. \tag{4.1}$$

The two parts of the network work against each other, meaning that the generator is trained to minimize this loss function, while the discriminator tries to maximize it.

Looking at each term separately we find the goals defined beforehand. The first term $\mathbb{E}_{x_p \sim p_{real}(x_p)} \left[\log(D(x_p))\right]$ represents the objective of the discriminator to correctly identify the true distribution. The higher $D(x_p)$, meaning the more confident the discriminator is about real events being real, the bigger the expected value will be. As a consequence, training to maximize this term will contribute to our first goal. Likewise, in the second term $\mathbb{E}_{x_d \sim p_d(x_d)} \left[\log\left(1 - D\left(G\left(x_d\right)\right)\right)\right]$ the discriminator is expected to output values close to 0, which means that the event was not sampled from the true distribution. The closer to 0, the bigger the term will be. This represents the second half of our first goal. The generator on the other hand learns to minimize the second term, meaning it tries to create samples that the discriminator cannot tell apart from the real distribution $p_{real}(x_p)$, and as a consequence will not classify correctly. The better the generator can fool the discriminator, meaning the closer its output will be to 1, the smaller the second term will become. The generator has to be trained on an additional loss function $\mathscr{L}_{MMD}$ in order to correctly capture the invariant mass distribution of the particles. The invariant mass distributions of the generated and real events are compared using MMD, which will be explained in Section 5.4. Weight decay is used as well to stabilize the training.

The two big advantages with this method are showcased in this example. The first one comes in the form of a problem independent loss function. The discriminator, over

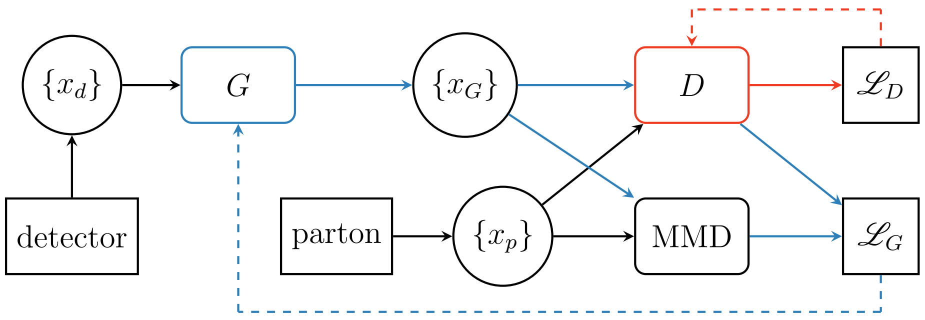

Figure 4.1.: Structure of a naive unfolding GAN. The simulation with Pythia (parton) and Delphes (detector) creates the input events x_p and x_d on parton- and detector-level respectively. The generator produces a generated event x_G, with the goal of resembling the the distribution of x_p. The discriminator classifies the two events and is trained to maximize $\mathscr{L}_D$. The generator is trained to minimize $\mathscr{L}_D$ and additionally maximize $\mathscr{L}_{MMD}$, which compares the invariant mass distribution of the generated and real events. Taken from [5]

the course of the training, learns adequate features to differentiate the two distributions $p_{gen}(x_p)$ and $p_{real}(x_p)$ and penalizes the generator accordingly, no matter the form that the true distribution takes. This means Equation 4.1 does in principle not have to be modified to learn any distribution. The second advantage is the free choice we have for the latent space of the generator. The generator describes an arbitrary function that is not dependent on the latent space distribution by construction. Only during the training does the generator adapt to the sampling we use. This lets us utilize advantageous distributions for said latent space, like the detector-level observables in this case.

A big issue that oftentimes arises is that training a GAN like this is very dependent on a lot of hyperparameters and getting the ratio of generator and discriminator training just right. If this is not the case, one of the networks can overpower the other which won't be able to train correctly, as its gradients become too small. Another known problem of GANs is that they often suffer from mode collapse. This will manifest by the generator only being able to predict a very narrow part of the full distribution that the GAN was trained on. Wasserstein GANs [4] can help to reduce these problems, however for this task they were not used. Apart from this, GANs can pose even further challenges during training. In an optimal training scenario, we get $p_{real}(x_p) \overset{!}{=} p_{gen}(x_p)$. The discriminator should have an exactly 50% chance for a correct prediction for samples from either distribution [16]. Using $D(x) = D(G(z)) = 0.5$ in Equation 4.1 we get an optimal loss value of $\log(2)$ for each half of the loss function. The problem arises from the fact that, to achieve this loss value, we do not in fact have to successfully have trained the GAN to the optimal configuration. We can only tell that something is going wrong when the loss values diverge from their optimal state, but cannot confidently say that the training succeeded when the loss value reached $\log(2)$ on both losses.

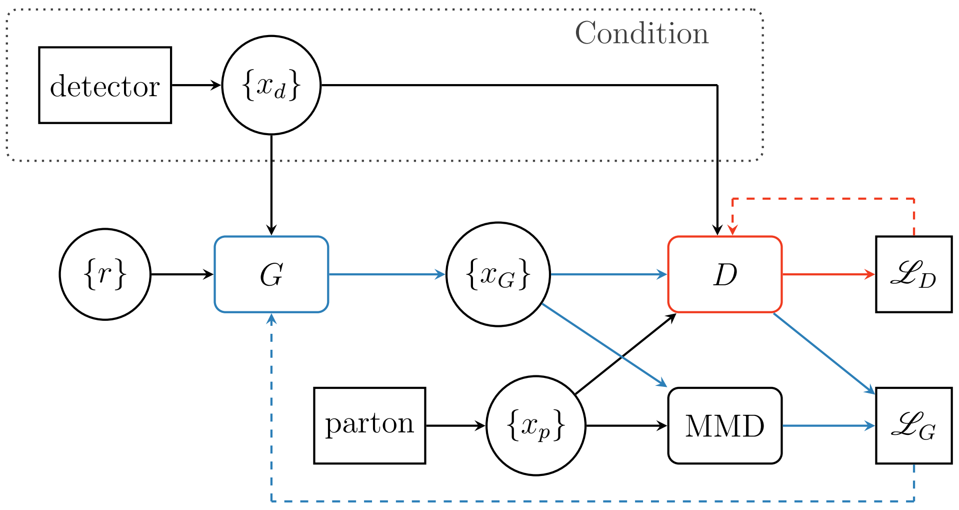

Figure 4.2.: Structure of the cGAN. Compared to the naive unfolding GAN, the detector event x_d is now distributed to both the generator and the discriminator. The generator also receives additional noise r as input. Taken from [5]

4.2. Conditional GANs

The work of [5] has shown that a standard GAN works for generating parton-level events for the full phase space, but it cannot correctly predict the distribution when we only look at part of it. The reason for this is that the discriminator cannot exploit the fact that the events are paired, i.e. that there exists a link between x_p and x_d. A possible solution consists of two steps. Firstly we need to provide the generator with additional gaussian noise, to enable probabilistic predictions by sampling over the noise. This also corresponds with the picture we get from the physical nature of the problem, as mentioned in Chapter 2. The second step is to introduce a fully conditional architecture, now providing also the discriminator with the detector event. This means the network functions now depend on additional parameters $G : z \sim p_z(z), x_d \rightarrow p_{gen}(x_p|x_d)$ and $D : x_p \sim p_{real/gen}(x_p|x_d), x_d \rightarrow [0, 1]$. There exist different approaches on how to provide the GAN with the condition x_d [10]. The simplest one is to just concatenate it to the input of both the generator and the discriminator [27]. [13] have found more success in conditioning the hidden layers directly, by using conditional batch normalization. This also allows for processing the condition by a feature network, which will present the data to the GAN in a more usable form. The conditional GAN in [5] uses the simpler of the two approaches. This new architecture solves the problem the GAN encounters with slicing to a large extent. However the generated invariant mass is still off in some cuts of the phase space, as shown in Section 6.2. The probabilistic interpretation of unfolding is also violated by the cGANs, as they experienced mode collapse. Sampling over the latent space of the generator will not result in a meaningful probability distribution, which is demonstrated in Section 6.3.

5. Method

5.1. Invertible Neural Networks

Invertible Neural Networks (INNs) learn a bijective mapping from input to output and back simultaneously. As we can run the model in any direction, we gain predictive capabilities in the inverse direction that we trained in. Using INNs is advantageous, even in cases when we are not particularly interested in both sides of the mapping simultaneously. If we can define a loss on both sides, network updates that include gradients from both directions are possible. This has been shown to increase training stability and convergence speed [2]. INNs are constructed to fulfill three goals:

- The mapping from input to output is invertible
- The Jacobians for both directions are tractable
- Both directions can be evaluated efficiently

While INNs are closely related to normalizing flows [22], the third restriction is unique to INNs. The nature of our problem is probabilistic, meaning we do not want to map the same detector event to the same value on the parton side every time. As will become clear shortly, this cannot be achieved via a normal INN, as it is deterministic by itself. However, the INN will still be introduced with a deterministic version of the problem for illustrative purposes and later be adapted to a probabilistic architecture. An overview of the INN architecture can be seen in Figure 5.1

5.1.1. Coupling Blocks

The network architecture has to be built specifically to be easily invertible to ensure the properties of an INN mentioned above. This can be achieved via so-called coupling blocks. Each of these coupling blocks is invertible and a model can be constructed by stacking multiple of these blocks laterally. The work of [14] introduces real-valued non-volume preserving transformations (Real NVP), a set of stably invertible, learnable transformations in the form of coupling blocks. The basic idea starts with splitting the input vector u in two parts, u_1 and u_2. The output v will also be computed in two steps

$$v_1 = u_1 \times s_1(u_2) + t_1(u_2) \qquad\qquad v_2 = u_2 \times s_2(v_1) + t_2(v_1). \tag{5.1}$$

Here, s_1, s_2, t_1 and t_2 describe arbitrary transformations on the input that are usually implemented by a neural network. These are the functions we are trying to optimize during training. Note that they can be arbitrarily complex and do not need to be

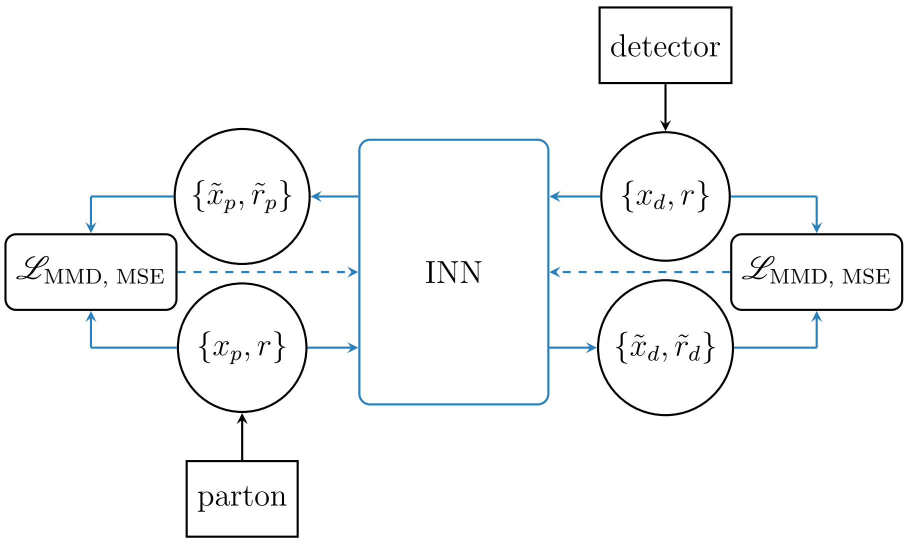
detector
$\{\tilde{x}_p, \tilde{r}_p\}$
$\{x_d, r\}$
$\mathscr{L}_{\text{MMD, MSE}}$
INN
$\mathscr{L}_{\text{MMD, MSE}}$
$\{x_p, r\}$
$\{\tilde{x}_d, \tilde{r}_d\}$
parton

Figure 5.1.: Structure of our regular INN. On the parton side we train the model by comparing the true parton events x_p to the ones generated by the INN $\tilde{x}_p$ from the detector data x_d. This is done via Mean Squared Error (MSE) and MMD simultaneously. The gaussian distribution of the generated noise $\tilde{r}_p$ is enforced with MMD only. On the detector side, the same process is mirrored.

invertible, since we can express the inverse direction $v \to u$ only using their forward directions

$$u_2 = \frac{v_2 - t_2(v_1)}{s_2(v_1)} \qquad u_1 = \frac{v_1 - t_1(u_2)}{s_1(u_2)}. \tag{5.2}$$

The only restriction that arises from this (as from any bijective) mapping is that u and v need to have the same dimensionality. Otherwise, we can not split them into parts of equivalent sizes, which is required to apply the same functions s_i, t_i on both the forward and backward pass. For numerical reasons, we want to avoid the direct division by $s_2(v_1)$ and $s_1(u_2)$. Thus this basic coupling block is modified to include an exponential function

$$v_1 = u_1 \times e^{s_1(u_2)} + t_1(u_2) \qquad v_2 = u_2 \times e^{s_2(v_1)} + t_2(v_1) \tag{5.3}$$

$$u_2 = (v_2 - t_2(v_1)) \times e^{-s_2(v_1)} \qquad u_1 = (v_1 - t_1(u_2)) \times e^{-s_1(u_2)}. \tag{5.4}$$

An illustration of the basic transformation applied inside a coupling block can be found in Figure 5.2.

The coupling blocks also need to ensure that the Jacobian is tractable throughout the network. The Jacobian of each coupling block takes the following form after the first half of the forward pass [14]

$$J_1 = \frac{\partial \begin{pmatrix} v_1 \\ u_2 \end{pmatrix}}{\partial u} = \begin{bmatrix} \text{diag}(e^{s_1(u_2)}) & \frac{\partial v_1}{\partial u_2} \\ 0 & \mathbb{I} \end{bmatrix}. \tag{5.5}$$

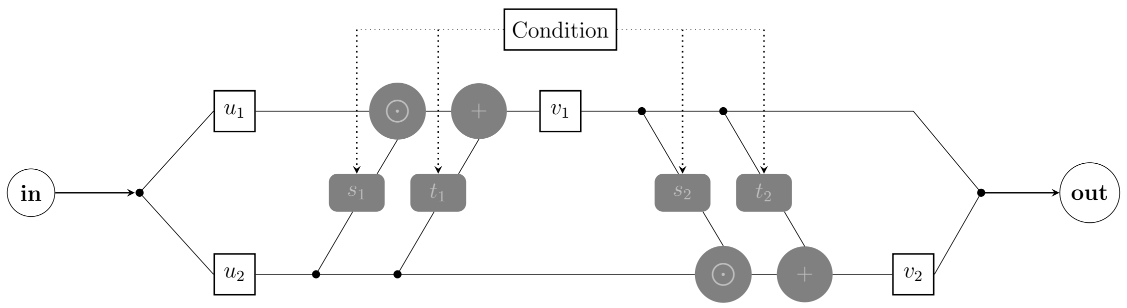

Figure 5.2.: Structure of a coupling block. On the left, the input u is split in two parts u_1 and u_2. The former is transformed via Equation 5.1 into v_1, with the coefficients predicted by the subnetworks s_1 and t_1. This process is then mirrored to transform u_2 into v_2. In the end v_1 and v_2 are concatenated to form the output v

We can look at each quadrant of the matrix individually to understand this fact. The lower half of the matrix consists of $\frac{\partial u_2}{\partial u}$. After the first half of the pass the lower half of the Jacobian is just the identity matrix, since the corresponding data vector has not changed. In the upper half, we get $\frac{\partial v_1}{\partial u_1}$ in the left quadrant and $\frac{\partial v_1}{\partial u_2}$ in the right quadrant. Substituting Equation 5.3 for v_1 in the upper left quadrant, we get $\frac{\partial v_1}{\partial u_1} = \operatorname{diag}(e^{s_1(u_2)})$. Therefore we end up with an upper right triangular Jacobian after the first half of the forward pass. The same argument applied to the second half of the forward pass results in

$$J_2 = \frac{\partial v}{\partial \begin{pmatrix} v_1 \\ u_2 \end{pmatrix}} = \begin{bmatrix} \mathbb{I} & 0 \\ \frac{\partial v_2}{\partial v_1} & \operatorname{diag}(e^{s_2(v_1)}) \end{bmatrix}. \tag{5.6}$$

For the complete coupling block, the Jacobian is given by $\frac{\partial v}{\partial u} = J_2 * J_1$. While this is no longer a triangular matrix, as will be explained in Section 5.2, we are only interested in the Jacobian (logarithmic) determinant. Despite the Jacobian taking a non-triangular form, by using $\det(A*B) = \det(A)\det(B)$ we still can efficiently calculate its logarithmic determinant

$$\begin{aligned} &\log(\det(J)) = \log(\det(J_1)) + \log(\det(J_2)) = \\ &\log\left(\prod_{i=1}^{\dim u_2} e^{s_1(u_2)_i}\right) + \log\left(\prod_{i=1}^{\dim v_1} e^{s_2(v_1)_i}\right) = \\ &\operatorname{sum}\{s_1(u_2)\} + \operatorname{sum}\{s_2(v_1)\}. \end{aligned} \tag{5.7}$$

We can use the same factorization rule to calculate the Jacobian determinants throughout multiple successive coupling blocks. In principle, the backwards pass works the same as the forward pass, therefore the same argument for the triangular shape of the Jacobians holds here as well. From this, we can see that a network built from these coupling blocks has tractable Jacobians.

5.1.2. Loss Functions

The standard INN combines three different contributions to the loss function. Each of them can be exemplified by picking the naive approach to detector unfolding as a toy example. In this approach, we try to learn a deterministic mapping from the detector-level to parton-level. Explicitly, this means we take a detector event $x_d \in R^{D_d}$ of dimensionality D_d and want to transform it into its corresponding parton event $x_p \in R^{D_p}$ of dimensionality D_p. Since we assume that the conservation of energy-momentum holds for particles on the parton-level, the energy can be calculated by their momenta and the masses with the help of Equation 2.2. For this reason, we remove the energy from the four-vectors of the parton level events. However, the same can not be done for the detector level, due to particles that are not measured or wrongly attributed and detector smearing. We are no longer in a closed system, preventing us from using the energy-momentum conservation to remove the energies. This leaves us with $D_p < D_d$. However, the inputs of our INN need to have the same dimensionality by construction. We pad the smaller input vector x_p with random numbers z from a gaussian distribution of dimensionality $D_r = D_d - D_p$ to fix this issue. These additional degrees of freedom, in some sense, describe the additional information gain that is created by the stochastic nature of the detector simulation. In a nutshell, our INN describes the mapping:

$$\begin{pmatrix} x_p \\ r \end{pmatrix} \leftarrow INN \rightarrow x_d \,. \tag{5.8}$$

For training this network represented by the functions $g_y : R^{D_d} \rightarrow R^{D_p}$ and $\bar{g}_y : R^{D_p+D_r} \rightarrow R^{D_d}$ and $g_z : R^{D_d} \rightarrow R^{D_r}$, we define three different loss functions:

$$\begin{aligned} \mathscr{L}_y &= ||x_p - g_y(x_d)|| \\ \mathscr{L}_x &= ||x_d - \bar{g}_y(x_p, z)|| \\ \mathscr{L}_z &= < g_z(x_d), \mathcal{N}^{D_r}(0,1) > \end{aligned} \tag{5.9}$$

$\mathscr{L}_y$ describes our goal to correctly map each detector-level event to the corresponding parton-level event, while $\mathscr{L}_x$ describes the inverse direction from parton- to detector-level. As our network is bijective, once $\mathscr{L}_x$ reaches zero, so will $\mathscr{L}_y$. This means $\mathscr{L}_y$ is not vital for training, still it helps with convergence and training stability [2]. $\mathscr{L}_z$ describes our restraint that the random numbers follow a gaussian normal distribution. We only look at the marginalized distribution $p(z) = \int_{R^{D_p}} p(z|y)p(y)dy$. This means the random numbers should (in the limit of infinite training time) become independent of the parton output x_p and the network will not encode redundant information in them. $\mathscr{L}_y$ and $\mathscr{L}_x$ could be implemented using the L_2 norm, while $\mathscr{L}_z$ could be realized utilizing MMD, which will be explained in Section 5.4.

5.1.3. Parton-Noise Correlation

The problem of this approach should become apparent when we repeatedly unfold the same detector event. We always get the same output of x_p for a fixed input x_d. This contradicts the probabilistic nature of the physical background described in Chapter 2.

While there exists the possibility of padding both sides with additional gaussian noise, we also need a way to guarantee that this noise correlates with the network output in such a way as to create a meaningful parton-level distribution when we sample from it. In the results Section 6.6 we show that training an INN with a regular MMD loss on the noise does not create a correct distribution when we sample over the latent space. This is due to the fact that the MMD loss marginalizes over a batch of data, with the effect of decorrelating the input noise with the parton-level outputs.

A solution to this problem is to utilize the network Jacobian to ensure a correlation between the input noise and the output four-vectors. However, while we showed that the Jacobian determinants of the whole network are easily tractable, the same does not apply for parts of it. If we want to ensure a correlation between the four-vectors x_p on parton and noise r_d on detector side, we need to know the Jacobian determinant of the first D_p output entries with respect to the last D_{r_d} input entries. This corresponds to the (generally non-quadratic) Jacobian sub-matrix $\hat{J} = J[1 : D_p, D_d : D_d + D_{r_d}]$, which only contains the columns corresponding to the input noise and the lines corresponding to the output four-vectors. While we know $\hat{J} = \frac{\partial x_p}{\partial r}$, the problem lies in acquiring this matrix in the first place. As discussed earlier, we are only able to efficiently propagate the Jacobian determinant through the network, not the whole matrix itself. However, this is required for calculating the sub-matrix determinant. We can look at the simple case of three consecutive operations, which result in the Jacobian $J = J_1 * J_2 * J_3$ to understand this. The sub-matrix determinant follows the formula

$$\begin{aligned} \det(\hat{J}) &= \det\left(J[1 : D_p, D_d : D_d + D_{r_d}]\right) \\ &= \det\left(J_1[1 : D_p] * J_2 * J_3[:, D_d : D_d + D_{r_d}]\right). \end{aligned} \tag{5.10}$$

As we can see, we can not use the factorization rule we used in Equation 5.7 for the determinant since the left- and rightmost matrices are not quadratic. This in turn means that we have to calculate the product of all the matrices inside the determinant of Equation 5.10. Doing this is inefficient, as we loose the triangular form of the matrices during the multiplication. As shown in Equations 5.5 and 5.6, the two Jacobians of a coupling block take an upper right and lower left triangular form respectively. It is easily verified that multiplying two matrices of these forms in general does not lead to a triangular result matrix.

$$\begin{aligned} &\begin{bmatrix} \mathbb{I} & 0 \\ \frac{\partial v_2}{\partial v_1} & \operatorname{diag}(e^{s_2(v_1)}) \end{bmatrix} * \begin{bmatrix} \operatorname{diag}(e^{s_1(u_2)}) & \frac{\partial v_1}{\partial u_2} \\ 0 & \mathbb{I} \end{bmatrix} = \\ &\begin{bmatrix} \operatorname{diag}(e^{s_1(u_2)}) & \frac{\partial v_1}{\partial u_2} \\ \operatorname{diag}(e^{s_1(u_2)}) * \frac{\partial v_2}{\partial v_1} & \frac{\partial v_2}{\partial v_1} * \frac{\partial v_1}{\partial u_2} + \operatorname{diag}(e^{s_2(v_1)}) \end{bmatrix} \end{aligned} \tag{5.11}$$

On top of that, permuting the data vectors in between coupling blocks during the forward pass is often crucial to achieve good results [3]. This permutation matrix would destroy the triangular form, even if we managed to construct the coupling block such that the product of its Jacobians was triangular. Therefore, we can only use the full Jacobian determinant to ensure input and output correlation.

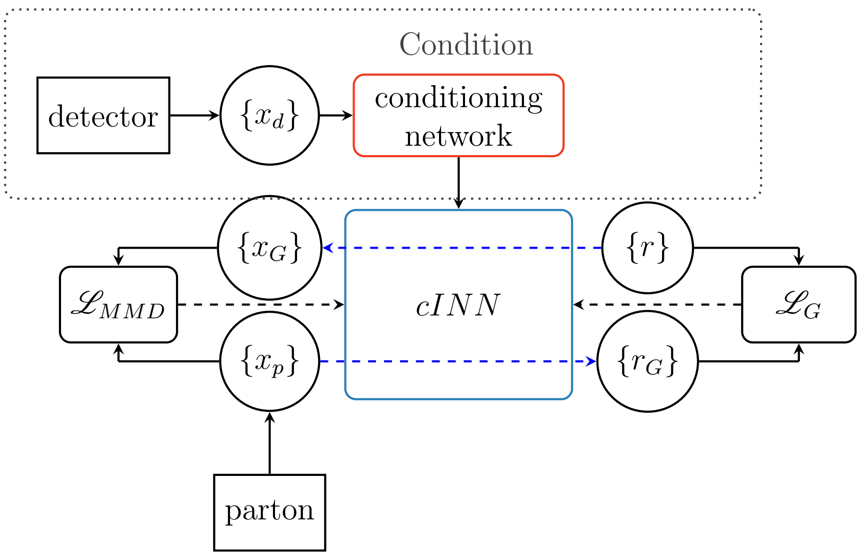

Figure 5.3.: Structure of the conditional INN. The parton-level inputs x_p are transformed into gaussian noise r_G under the condition of the detector event x_d processed by the conditioning network. This noise is the input to the loss function $\mathscr{L}_G$ from Equation 5.15. In the other direction, gaussian noise r is used as input to generate parton events x_G (under the same condition of x_d). Their mass distribution is then compared to the one of x_p via $\mathscr{L}_{MMD}$.

5.2. Conditional INNs

We can utilize a conditional INN (cINN) [3] to achieve a fully probabilistic mapping on parton side. Instead of mapping the detector-level directly to the parton-level, the cINN uses the detector-level data as condition for the subnets in each coupling block. Its inputs consist of the parton-level data on one side and gaussian noise of the same dimensionality on the other side. An overview of the cINN architecture can be found in Figure 5.3

5.2.1. Transformation of Distributions

Our goal while training is to map the parton-level events to a gaussian distribution under the condition of the detector measurements. From the discussion in the previous section we know that we can not just use a MMD loss to match the shape of the distributions. We require the features x_p to be correlated with the noise r, such that sampling over r will represent the full distribution of x_p. In other words, we want to transform the gaussian distributions on one side of the network to the correct parton distributions on the other side. The formula that describes this transformation is the change of variables formula

$$p_{part}(x_p|x_d, \theta) = p_{gauss}(f(x_p|x_d, \theta)) \left|\det\left(\frac{\partial f}{\partial x_p}\right)\right|. \tag{5.12}$$

Where f is the network function $f : x_p \to r$ and θ are the network parameters. Our goal is to fit these network parameters such that they maximize the posterior over the parameter space

$$\max_{\theta} p(\theta|x_p, x_d) \propto p_{part}(x_p|x_d, \theta)p_\theta(\theta) \tag{5.13}$$

via Bayes theorem. As our training set and therefore the number samples from our distributions are finite, we define our objective as minimizing the negative log-likelihood of the expected probability over the whole dataset of size N.

$$\mathscr{L} = \mathbb{E}_{i\in 1,\ldots N}\left[-\log\left(p_{part}(x_{p,i}|x_{d,i}, \theta)\right)\right] - \log\left(p_\theta(\theta)\right) \tag{5.14}$$

This objective is the same as for non-invertible networks. However, we can now use the invertibility of the network to substitute $p_{part}(x_p|x_d, \theta)$ via Equation 5.12, to get an expression for our loss dependent on only the r side output of the network. Additionally, we also assume a gaussian prior $\mathscr{N}(0, \sigma)$ on the network weights θ, which means we can substitute $p_\theta(\theta) \propto \exp\left(-\frac{||\theta||_2^2}{2\sigma^2}\right)$. For notational purposes we introduce $\frac{1}{2\sigma^2} = \tau$. Finally, the substitution for Equation 5.12 introduces the term $\left|\det\left(\frac{\partial f}{\partial x_p}\Big|_{x_{p,i}}\right)\right|$, which we can express using the Jacobian determinant J_i of the network at point $x_{p,i}$. Summing up we get the loss function

$$\mathscr{L} = \mathbb{E}_{i\in 1,\ldots,N}\left[\frac{||f(x_{p,i}|x_{d,i}, \theta)||_2^2}{2} - \log|J_i|\right] + \tau||\theta||_2^2. \tag{5.15}$$

As described in 5.1.1, the Jacobian determinants of the network are easy to compute. Thus, we can easily utilize them for the loss function. The term $\tau||\theta||_2^2$ describes a regular L_2 weight decay with strength τ.

As we can see in Equation 5.15, we include no extra information about the distribution of x_p. Therefore, the big advantage of using a cINN is that we can train to fit the known gaussian distribution of r, for which we can define a loss without having to know anything about the distribution of x_p beforehand. Afterwards, we can still predict in the inverse direction $r \rightarrow x_p$, which would not be possible with other networks.

5.2.2. Conditioning Networks

As all of the subnetworks in the coupling blocks receive the same conditioning input x_d, it is vital that its structure is not too complex for the networks s and t to extract information from. For this reason, it is beneficial to first preprocess the condition input with a conditioning network. This is a small subnetwork that extracts higher level features from x_d, and is trained simultaneously with the whole cINN. It should be noted that, as the condition input for the same data point during the forward and backward pass is identical, the conditioning network can be chosen arbitrarily and is not required to be invertible or fulfill any other special characteristics. A practical problem is that, if the conditioning network architecture is chosen too deep, its output is often degraded. After initialization, the condition features are unusable for the cINN and over the course of the training, they are ignored completely. We can choose two approaches to avoid this problem:

- Keep the conditioning network architecture shallow and wide

- Initialize the conditioning network advantageously for the cINN

The former solution should be self-explanatory, the latter can be achieved by pre-training the conditioning network, e.g. as an Autoencoder. An Autoencoder is a neural network, used to learn an efficient encoding for a certain data domain [23]. It consists of two parts an encoder and a decoder, which are typically structural mirror images of each other. In the center between them, we find an intermediate latent space output z, which is of lower dimensionality than the input. The decoder then decompresses the encoding to match the original input as closely as possible. The network is trained to minimize the reconstruction loss

$$\mathscr{L}_{recon} = ||x_{gen} - x_{real}||_2^2 = ||D(E(x_{real})) - x_{real}||_2^2. \tag{5.16}$$

with D as the decoder and E as the encoder. This combines two goals: We want the encoder to preserve as much information as possible in the latent variable z and the decoder should then be able to reconstruct the compressed information to its original state. Training a network like this can help us to find a good initialization for the conditioning network. By cutting off the encoder right before the final layer, in which the dimensionality is reduced and the information compressed we can use it as our initial conditioning network. An illustration of this can be found in Figure 5.4. This allows us to choose a deeper architecture than before, and thus extract more meaningful correlations from the detector level data x_d, which was necessary to improve the results in Section 6.5.

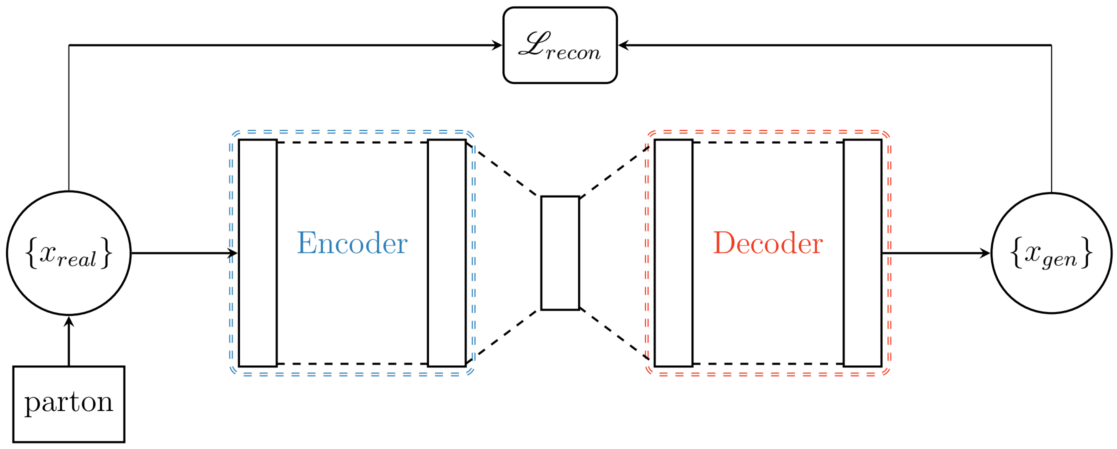

Figure 5.4.: Schematic of an Autoencoder. The input x_{real} is compressed by the encoder to an encoding of lower dimensionality in the central layer. The decoder then decompresses this encoding to create the generated datapoint x_{gen}. The network is trained to minimize the reconstruction loss $\mathscr{L}_{recon}$

5.3. All-In-One Coupling Block

For the specific coupling block architecture we use the All-In-One (AIO) coupling block. This coupling block combines several additional features that aid the training of INNs.

The basic transformation described by Equations 5.3 and 5.4 is now predicted by a single subnet in its entirety. This enables the network to share weights for the prediction of the transformation parameters.

Soft clamping is applied to the exponential scale subnetwork output $s_i(u_2/v_1)$ to prevent instabilities resulting from large values in the exponential function. This clamping takes the form of

$$s_{clamp} = \alpha \tanh(\frac{1}{\gamma}s). \tag{5.17}$$

This restricts the value of $s_{clamp} \in [-\alpha, +\alpha]$. At small $|s|$, tanh is approximately the identity function, meaning $s_{clamp} \approx \frac{\alpha}{\gamma}s$. We find $\alpha = 0.8$ and $\gamma = 10$ to be good values for the clamping.

Only one half of the input is replaced by the coupling block. In terms of the previously introduced Equations 5.3 and 5.4, we instead use $\hat{v} = \begin{pmatrix} v_1 \\ u_2 \end{pmatrix}$ as the coupling block output [14].

A global affine transformation (scaling and bias) is used on the output of each coupling block, which is introduced as actnorm in [21]. In this transformation of the form $\hat{v} = sv + b$ and inverse $v = (\hat{v} - b)s^{-1}$ the parameters s and b are fixed and do not change over the course of the training.

Finally, the coupling block output is permuted to allow the two splits u_1 and u_2 to interact more between coupling blocks. This becomes especially important, as we don't change u_2 during the learned transformation of the AIO coupling block. The permutation is performed by multiplication with fixed unitary matrices, which remain unchanged during training.

$$\hat{v} = Uv \qquad\qquad v = U^T\hat{v} \tag{5.18}$$

It is possible to use any random unitary matrix to perform this permutation, which in general is called "soft permutation" [3]. However, we found that "hard permutation", meaning using shuffled unit matrices, works best for our case.

5.4. Maximum Mean Discrepancy

We often want to compare a true to a generated distribution and measure how closely they match to define our loss function. For example, in the loss of the regular INN in Equation 5.9, for $\mathscr{L}_z$ we want to compute a distance measure between the generated latent space distribution $g_z(x_d)$ and the target gaussian normal distribution $\mathscr{N}^{D_r}(0,1)$. Maximum Mean Discrepancy (MMD) provides a solution for this. Given samples from two distributions $X = \{x_i\}_{i=1}^N \sim P_X$ and $Y = \{y_j\}_{j=1}^M \sim P_Y$ the MMD is defined as [25]

$$\mathscr{L}_{MMD}^2 = \left|\left|\frac{1}{N}\sum_{i=1}^N \phi(x_i) - \frac{1}{M}\sum_{j=1}^M \phi(y_j)\right|\right|_2^2 = \tag{5.19}$$

$$\frac{1}{N^2}\sum_{i=1}^N\sum_{i'=1}^N \phi(x_i)^T\phi(x_{i'}) + \frac{1}{M^2}\sum_{j=1}^M\sum_{j'=1}^M \phi(y_j)^T\phi(y_{j'}) - 2\frac{1}{MN}\sum_{i=1}^N\sum_{j=1}^M \phi(x_i)^T\phi(y_j).$$

Given the right choice of function ϕ, MMD is a metric on the space of probability distributions [17]. This especially means that $\mathscr{L}_{MMD} \geq 0$ and (in the limit of $M, N \to \infty$) $\mathscr{L}_{MMD} = 0 \Leftrightarrow P_X = P_Y$. In essence, this allows us to use it to compare the proximity of any two distributions and minimizing it during training will lead to the two distributions matching exactly. Taking a look at Equation 5.19, we see that all sums involve the inner product of $\phi(\cdot)^T\phi(\cdot)$. We can use the kernel trick to substitute these vector products by a kernel that represents the ϕ induced scalar product, $<\cdot,\cdot>_\phi$.

$$\begin{aligned}\mathscr{L}_{MMD}^2 &= <X,X> + <Y,Y> -2<X,Y> \\ &= \overline{k(X,X)} + \overline{k(Y,Y)} - 2\overline{k(X,Y)}\end{aligned} \tag{5.20}$$

5.4.1. Kernels

There exist a wide variety of established kernel functions to choose from [36]. The restriction for the choice of ϕ, and consequently for our kernel, is that MMD has to be a metric on the space of probability distributions. We have to choose our kernel such that it is positive-definite in order to fulfill this condition. The specific choice of kernel always depends on the distributions we are trying to compare. For example, if we want to make sure the generated latent space distribution of our regular INN follows the target gaussian distribution a gaussian kernel would be a natural first choice, seen in Equation 5.21. While training our cINN to unfold detector events $x_d \to x_p$, like in [7] and [5] we encountered problems with the invariant mass (see Section 2.1) distribution of the partons. As a solution, we use the MMD to help the model learn this correlation. Two choices for the kernel of this MMD compared in the work of [7] are the gaussian kernel and the breit-wigner/cauchy kernel

$$k_{\text{gauss}}(x,y) = \exp\left(-\frac{||x-y||_2^2}{2\sigma^2}\right) \qquad k_{\text{breit-wigner}}(x,y) = \frac{\sigma^2}{\sigma^2 + ||x-y||_2^2}. \tag{5.21}$$

We found the breit-wigner kernel to work slightly better than the gaussian kernel. This is to be expected, as the invariant mass of the partons follows a breit-wigner distribution. We also used summed kernels over different widths σ [5], to ensure the generated distribution would not fall out of the scope of the kernel at any stage of the training. Choosing $\sigma \ll ||x-y||_2$ results in $k_{\text{breit-wigner}} \approx 0$, likewise, choosing a $\sigma \gg ||x-y||_2$ leads to $k_{\text{breit-wigner}} \approx 1$. In both cases, we end up with $\mathscr{L}_{MMD} \approx 0$, despite the distributions not matching up. Using the sum over many different kernel widths ideally ensures that at any point during the training, we have some $\sigma_i \sim ||x-y||_2$.

5.4.2. Conditional MMD

Even though we use MMD to ensure the invariant mass distribution is learned correctly, we always marginalize over the data of a whole training batch. Therefore, the model might only learn to fit the invariant mass correctly under the condition that the input batch represents the full phase space adequately. However, we want to be able to reconstruct the invariant mass of any part of the phase space accurately. This problem becomes apparent in Section 6.2, where we can see deviations in the invariant masses when slicing. Optimally, the MMD loss should therefore not marginalize over the condition but take it into account as well. This idea is called conditional MMD, introduced in [32]. We start with two conditional distributions $P_{X|C_1}$, $P_{Y|C_2}$, which we can sample, resulting in two sets $D_{X|C_1} = \{(x_i, c_{1,i})\}_{i=1}^N$ and $D_{Y|C_2} = \{(y_i, c_{2,i})\}_{i=1}^M$. We define $K_X = k(C_1, C_1)$, $K_Y = k(C_2, C_2)$ and $K_{XY} = k(C_1, C_2)$ as the kernel matrices of the sampled conditions, as well as $L_X = k(X, X)$, $L_Y = k(Y, Y)$ and $L_{XY} = k(X, Y)$ as the kernel matrices of the sampled data points. As the condition kernel matrices need to be inverted, we usually need to add regularization to ensure that the matrices have full rank. We denote this as $\widetilde{K} = K + \lambda\mathbb{I}$. The authors [32] show that the target function of the conditional MMD can now be written as:

$$\mathscr{L}_{cMMD} = K_X \widetilde{K}_X^{-1} L_X \widetilde{K}_X^{-1} + K_Y \widetilde{K}_Y^{-1} L_Y \widetilde{K}_Y^{-1} - 2K_{XY} \widetilde{K}_X^{-1} L_{XY} \widetilde{K}_Y^{-1} \tag{5.22}$$

While this approach should in theory solve the problem that we encountered with the invariant masses when slicing, in practice it turned out that the inversion of the condition kernel matrices is too unstable and requires both a lot of computing time and regularization. Thus, with reasonable training time, we could not achieve results comparable to those of the cINN trained on regular MMD.

5.5. Whitening

When training a model with the loss function of Equation 5.15 it is actually possible to reduce the loss below 0. The minimum that the loss function can reach is dependent on the network Jacobian. This not only depends on the architecture itself but especially on the correlation within the input data. We can understand this by examining the extreme case in which two variables x_i, x_j are correlated completely. Since $\frac{\partial f(x)}{\partial x_i} = \frac{\partial f(x)}{\partial x_j}$, these two columns of the Jacobian are identical, i.e. linearly dependent. Therefore the Jacobian determinant is 0, its logarithm goes towards $-\infty$. If the loss reaches very low values it can lead to issues during training. We can apply whitening to the data to decorrelate its features beforehand in order to combat this problem. Whitening is a linear transformation, given by multiplication with a whitening matrix W, which has the goal to transform a given dataset matrix $X = (x_1, ..., x_N) \in \mathbb{R}^{d\times N}$so that its new covariance matrix is the identity matrix[19].

$$\mathrm{cov}(Z) = \mathrm{cov}(WX) \overset{!}{=} \mathbb{I}_{N\times N} \tag{5.23}$$

Given the covariance matrix $\mathrm{cov}(X) = \Sigma$, any W such that $W^T W = \Sigma^{-1}$ satisfies the condition. We can estimate the covariance matrix of a given dataset as follows [28]:

$$\mathrm{cov}(X)_{jk} = \hat{\Sigma}_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} (X_{ij} - \overline{X}_j)(X_{ik} - \overline{X}_k) \tag{5.24}$$

However, this does not uniquely define the whitening matrix, but we have multiple possibilities to choose from. Two of the most common choices constitute Zero-Phase Component Analysis (ZCA) and Principal Component Analysis (PCA).
ZCA uses the inverse square root of the covariance matrix $W = \Sigma^{-1/2}$. Since Σ is symmetric (and therefore also W) we get $W^T W = W^2 = \left(\Sigma^{-1/2}\right)^2 = \Sigma^{-1}$. The authors of [19] showed that this approach to whitening leaves the whitened data Z as similar as possible to the original data X, in terms of total squared distance.
PCA defines the whitening matrix, using the eigenvalue decomposition of the covariance matrix $\Sigma = U \Lambda U^T$, with unitary eigenvector matrix U and diagonal eigenvalue matrix Λ. Since Σ is symmetric and we can add regularization to ensure the matrix has full rank we know that such a decomposition must exist. The whitening matrix is now given by $W = \Lambda^{-1/2} U^T$. This fulfills the whitening matrix constraint $W^T W = (\Lambda^{-1/2} U^T)^T \Lambda^{-1/2} U^T = U (\Lambda^{-1/2})^T \Lambda^{-1/2} U^T = U \Lambda^{-1} U^T = \Sigma^{-1^T} = \Sigma^{-1}$. The authors of [19] showed that this best encodes the variation from the original data X in the first entries of the whitened data Z in terms of the cross-covariance.

Experimentation has shown that ZCA is better choice for our cINN, which fits with the property of leaving the data as similar as possible to the original. Using whitening we improve the results in Chapter 6, when the data gets too complex in Section 6.5, we found that whitening can not help to improve the results anymore.

6. Results

We compare the cINN to the cGAN used in [5] to evaluate its performance. We make sure to keep the training times and number of parameters on the same order of magnitude for both networks to ensure comparability. Our network is build out of the AIO coupling blocks (Section 5.3). For training, we use a summed MMD loss over 4 widths with a breit-wigner kernel on the generated W and Z Boson masses. We apply ZCA whitening to the data and use a regular shallow but wide conditioning network with no Autoencoder pretraining. The training data is generated using Pythia and Delphes, with no additional disruptive effects like ISR. Pythia creates the parton-level events, which serve as the input during training to be transformed into the gaussian latent distribution. Delphes simulates the showering, creating detector-level events, which are matched to the parton events. The matching detector-level event serves as a conditional input to each parton-level event. When passing samples from the latent distribution back through the network under the condition of the detector events, we get an approximated parton event distribution of the observables shortly after the hard process. A setup comparison of the cGAN and the cINN can be found in Table 6.1.

Parameter	cINN	cGAN
Blocks	24	-
Layers per Block	2	12
Units per Layer	256	512
Trainable weights	~2M	~3M
Condition Subnet Layers	2	0
Whitening	ZCA	-
Batch Size	512	512
Epochs	1000	1200
Number of Training Events	3×10^5	3×10^5
Kernel Widths	~ 2, 8, 25, 67	20, 30, 40
Test/Train Split	10% / 90%	0% / 100%
λ_{MMD}	0.5	1

Table 6.1.: Setup of the cINN compared to the setup of the cGAN

6.1. Full Distributions

Firstly, in Figure 6.1 we take a look at the full phase space to check whether both the cINN and the cGAN can correctly generate events in the full range of the observables

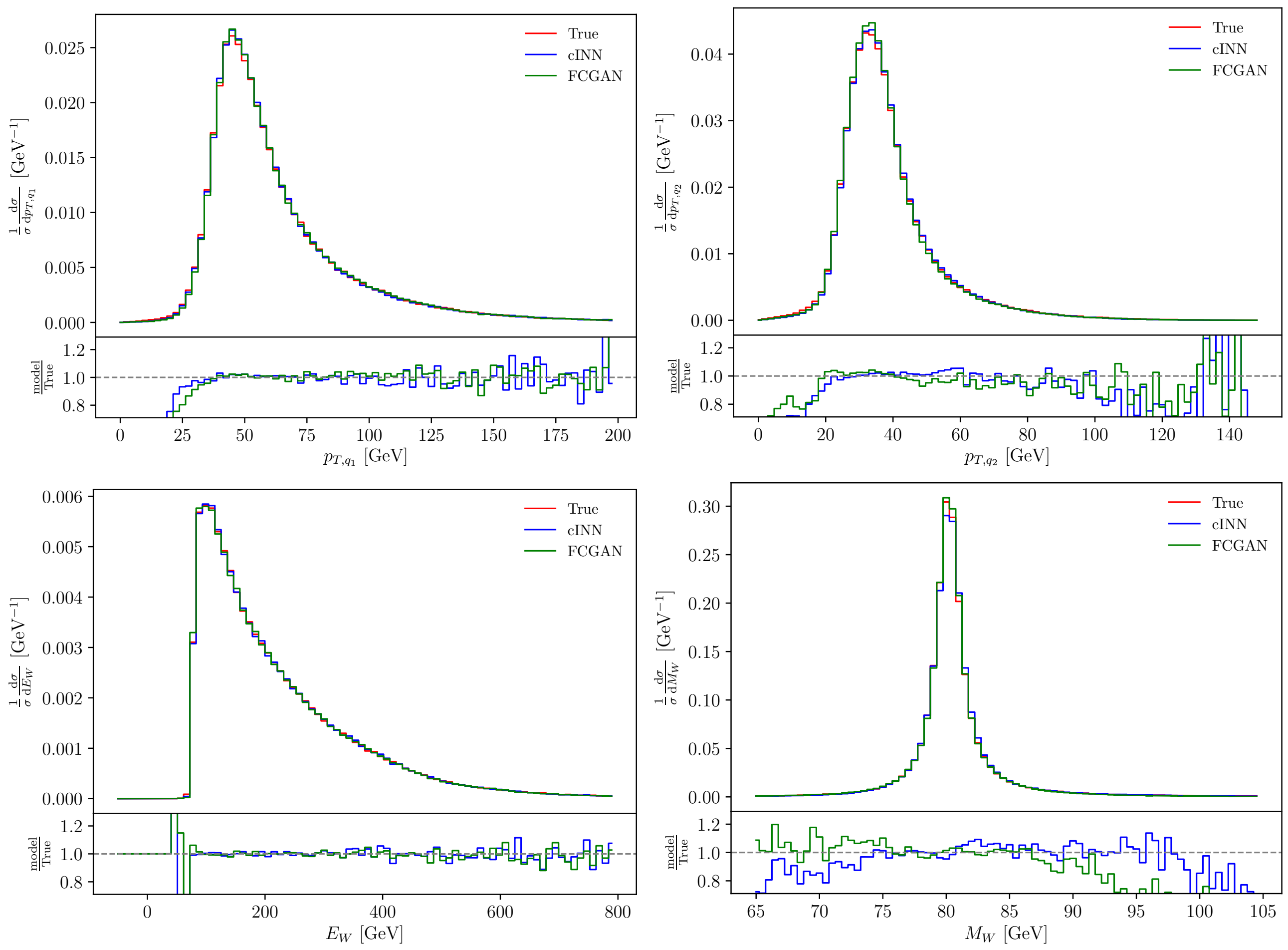

Figure 6.1.: Full Phase Space distributions for the cINN (blue) and cGAN (green). Both models predict the distributions of p_{T,q_1} (upper left), p_{T,q_2} (upper right), E_W (lower left) and M_W (lower right) accurately in the bulk.

and capture their correlations. We include a cINN that has been trained without MMD on the invariant mass in Figure A.1.

As can be seen, the cINN without the additional MMD loss is learning the masses to some extent, but the MMD is very necessary to match the sharp peak. The masses of the particles at the detector are very smeared resulting in a much wider distribution. In contrast, the W Boson mass distribution has a very sharp peak, completely undoing the smearing is very hard for the model to learn. On top of this, the invariant mass is a more complex correlation compared to observables such as momentum or energy of a single jet, thus the difficulty to form said correlation is much higher for the model.

The differences between the cGAN and the cINN trained with MMD lie in the range of statistical fluctuations for the most part. From the ratio plots we can see that there is some systematic underestimation of the events with low p_{T,q_1}, p_{T,q_2} from both networks. These occur in regions of the phase space with very low statistics, which is likely the reason both models are not performing as well as in the bulk. In the invariant mass plot we can see that the cINN manages to follow the tail of the distribution for longer than the cGAN.

In Figure A.2 we provide a correlation plot of the cINN between $P_{T,Z}$ and E_{j_1} that shows the cINN captures the relations correctly. The main differences occur at the outer edges of the correlation, at low $p_{T,Z}$ or low E_{j_1}, which points towards a lack of statistics as the reason for the deviations.

6.2. Slices

One problem that the cGAN exhibited were the mismatched masses when considering only part of the phase space. As indicated by the name, the invariant mass distribution should not change when we only consider particles within a certain range of momentum or energy. If the models build up some correlations between these quantities and the invariant mass they generate for the event, this condition is not met. We evaluate the models' generated distributions on events that fall into specific ranges of detector-level measurements for multiple observables to check for these correlations. We test the following restrictions on events, called slices, in analogy to [5]:

- **Slice I:** $p_{T,j_1} = 30 \ldots 100$ GeV (6.1)
- **Slice II:** $p_{T,j_1} = 30 \ldots 60$ GeV; $p_{T,j_2} = 30 \ldots 50$ GeV (6.2)
- **Slice III:** $p_{T,j_1} = 30 \ldots 50$ GeV; $p_{T,j_2} = 30 \ldots 40$ GeV;
  $p_{T,\ell^-} = 20 \ldots 50$ GeV (6.3)
- **Slice IV:** $p_{T,j_1} > 60$ GeV (6.4)

From Figure A.3-A.6 it is apparent that, while the cINN still does not correctly disjoin the invariant mass from the region of phase space, especially in the harder slices (III and IV), it outperforms the cGAN. Both models always over-/ underestimate the mass distribution in the same direction, but the invariant mass curves of the cINN lie systematically closer to the true peak every time. Interestingly, this deviation in the slices only concerns the masses, as the other observables do not show any sign of degeneration. This means that the models have correctly learned the overall low level observable distribution even for slices of the phase space, but their higher level correlations do not match the truth. This is potentially a byproduct of using the MMD to enforce the mass distribution, as this does not force the model to correctly learn this correlation. Instead, all that is needed to reduce this loss is to match the distributions. If the sampled training data for each batch typically represents the whole phase space to a certain extent, the learned correlation does not have to be independent of the phase space region of the event. We tried using a conditional MMD to address this problem, however we encountered problems with the inversion of the kernel matrices in Equation 5.22. The amounts of regularization we needed to add, as well as the computing power the matrix inversion required made so that this approach was not executable within reasonable training times.

6.3. Calibration Curves

We take a look at calibration curves for some observables in Figure 6.2 to test the integrity of the cINN's predicted distributions. For this, we randomly pick 1500 different events and do inference 60 times for each of them. Since we re-sample from the latent space every time we do inference, in the end we approximate a probability distribution for each event with this method. Consequently, for each of the input events, we can now calculate the quantile that it falls into in its generated probability distribution. For the posterior distributions to be statistically sound, we would expect 10 % of the events to be in the 10 % quantile, 20 % of events to be in the 20 % quantile, etc. Plotting each quantile against the percentage of points that landed in it, the optimal outcome is a diagonal line. In the figure, it becomes pretty clear that the cINN does create statistically meaningful probability distributions in which the points follow the expected quantile frequencies. We see slight deviations in the Z and W mass, which might correspond to the problem with the sliced masses. While the mean of these distributions is correct (the cINN and optimal curves cut in the center), the cINN's distribution is a bit too narrow and consequently more points land in very high/low quantiles.

When looking at the cGAN, we find that the true parton-level event completely falls to the right or left of the distribution most of the time. In other words, while the cGAN's generated distributions might have the correct mean when averaging over many events (the cGAN and optimal curves cut in the center), it is way too narrow to meaningfully approximate the true distribution of the parton level event. It is worth noting that, while calibration curves show that on average the true parton-level events follow the generated distributions, it does not guarantee that overall, they are the true distributions on parton-level. This could only be tested if we could generate multiple different parton-level events, which all lead to exactly the same detector measurement, which is impossible.

6.4. Single Points

The difference we see between the cGAN and the cINN in the calibration curves can be further illustrated by looking at the distributions created for a single point in Figure 6.3. While not as statistically significant, we can get a good look at the problem with the cGAN's predictions.

We also use Approximate Bayesian Computation (ABC) to estimate the true probability distribution of the parton-level, based on the entire dataset [33]. For this, we define a distance measure $d(x_d, x'_d) = ||(p_T, p_x, p_y, p_z)^T_{x_d} - (p_T, p_x, p_y, p_z)^T_{x'_d}||^2_2$, based on the difference between two detector level measurements. We first divide each momentum by its standard deviation to equalize their scales. We then apply ZCA whitening, to better separate the dataset. We also include an additional term in the distance measure with a weak influence $\tilde{d}(x_d, x'_d) = d(x_d, x'_d) + \lambda||((p_T, p_x, p_y, p_z)^T_{x_p} - (p_T, p_x, p_y, p_z)^T_{x_d}) - ((p_T, p_x, p_y, p_z)^T_{x'_p} - (p_T, p_x, p_y, p_z)^T_{x'_d})||^2_2$. This term signifies the proximity of the perturbation each event experienced between parton- and detector-level. We naturally apply the same whitening and scaling to each of these momentum vectors as before. With this

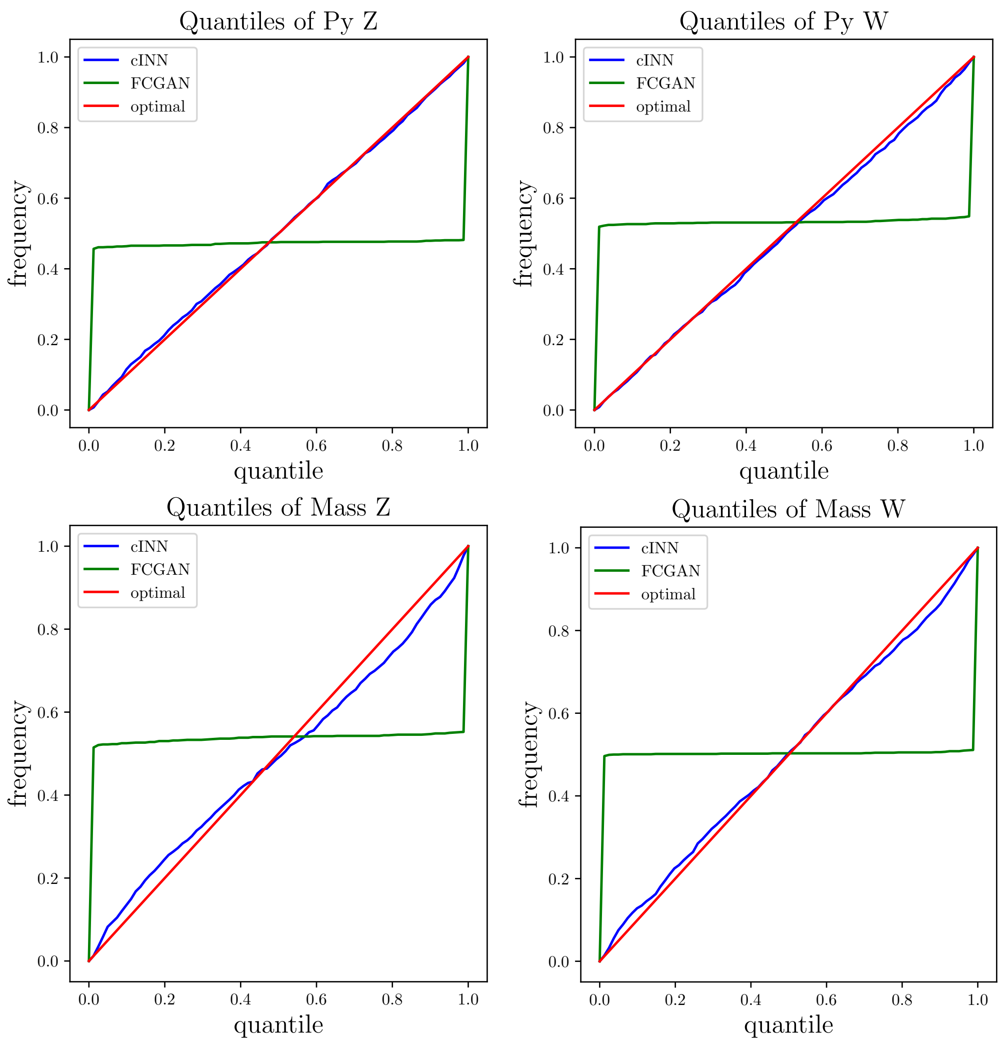

Figure 6.2.: Calibration curves on the full Phase Space for the cINN (blue) and cGAN (green). The cINN generates distributions that create realistic quantiles for the input events. The cGAN predicts distributions that lie either completely to the right or to the left of the points, which shows as exactly half of the points always lie in the 0 % or 100 % quantile respectively.

distance measure we compute the 400 most similar events in the dataset and use their parton-level data to approximate the true distribution.

We stress that this is only one of many choices possible to define such a distance measure. Changing the whitening, scaling or observables we use will considerably change the outcome of the distribution ABC predicts for a single point. The cause of this is an insufficient dataset size for ABC. We can check this by picking a single point from the bulk of the distribution. We then firstly approximate the range of events on parton-level that can lead to a detector-level event in the region of the phase space that our point lies in by looking at the correlations between parton and detector data. Secondly, we compute the standard deviation of the 400 events with the closest parton-level observables to the point we picked. We find that the latter value is usually higher by a factor of $1.5 - 2.5$. This means that the smearing introduced by the detector lies on a smaller scale than what our dataset allows us to unfold via ABC. Therefore, without further verification, it should not be considered the true parton-level distribution of a single point. Nevertheless, it acts as a point of reference for the models.

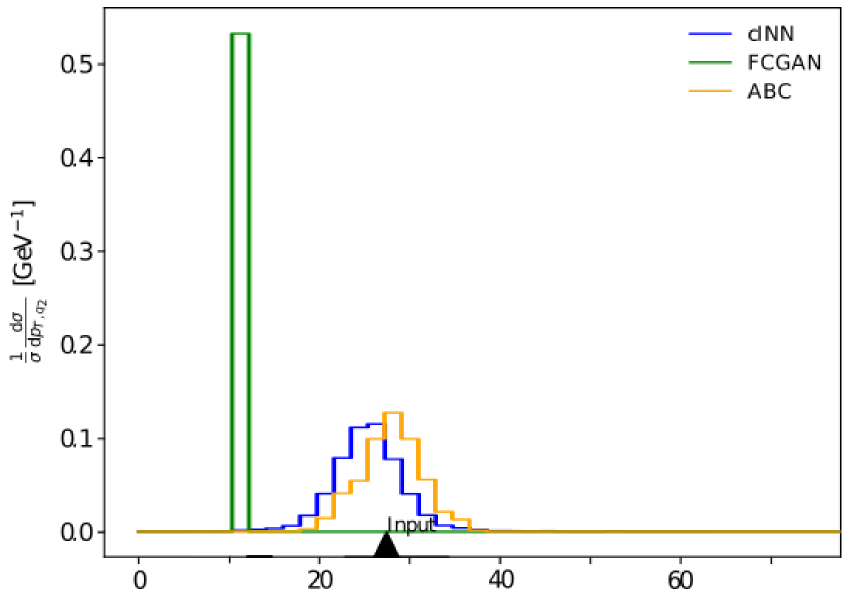

Figure 6.3.: Predictions for a single event's p_{T,q_2} when iterating 3200 times. The cGAN (green) generates a distribution that is too narrow, while the cINN (blue) and ABC (orange) generate a possibly viable distribution around the event.

From the figure we immediately see that the cGAN's predicted distributions are way narrower than the cINN's, which we already expected from the previous section. While the cINN's distribution is shifted towards lower p_{T,q_2} w.r.t. ABC, the shapes and widths created by both are very similar. From the dataset we determined that for the range of p_{T,det,q_2} this point is located at, the parton-level events that can produce these detector measurements lie in the range from $\sim 18 - 42$GeV for $p_{T,part,q_2}$. These widths roughly correspond to the generated distributions, but both distributions estimate the range on the lower side. This could be entirely valid, as the particular event might show some correlations which point towards a lower range of $p_{T,part,q_2}$. While in no way a proof for the correctness of the cINN's parton-level distribution, this puts into contrast the viability of the predictions compared to the cGAN. From this section's and the previous section's results we can assume that the cGAN has experienced mode collapse, described in Chapter 4. As a consequence its generated distribution is extremely narrow. This contradicts the physical nature of the task, as explained in Chapter 2.

6.5. Initial State Radiation

We also test our model on the task of unfolding ISR, which was introduced in Chapter 2. We introduce additional disruptive effects in the form of radiation that is given off before the collision in some events. This might result in events where, instead of only two jets from the W Boson, we measure three or four jets. The parton-level inputs still only consist of the two jets of the W Boson and the two leptons created by the Z Boson, but the detector-level condition inputs now gain two additional four-vector entries. This task required us to modify the model architecture, especially with a focus on increasing the size of the model. The model needs to filter out the disturbances from the condition input, and thus we specifically increased the size of the conditioning network. For this reason we also had to utilize the Autoencoder pretraining trick, mentioned in Section 5.2. Whitening also showed no positive effect on the training anymore, consequently it was turned off. We separate the plots by event type, meaning all events with two, three and four jets are plotted individually. We first trained a regular cINN with no other augmentations than the increased size on unfolding the ISR as a baseline (Figures A.7 - A.9). It can be seen that for four jets, the peaks in p_{T,q_1} and p_{T,q_2} are systematically too low. The same also shows in the invariant mass for four jets, which has too many events in its tails. We can also see in the two jet plot that the model is overcompensating for this to overall match the invariant mass distribution.

Since the cINN's problems mainly lie in the ISR jet events, it might be natural to assume that the difficulty lies within telling the ISR and W jets apart. Having this additional task to solve during training might be too hard for the network to learn. We trained a classifier that labels the jets as either a W or ISR jet to reduce the complexity of the task. Afterwards we can choose the right order for the jets, so that the network will automatically learn that only the first two jets are the ones to reconstruct the W Boson from. Results for the four jet events are provided in Figure A.10. They show that this approach unfortunately did not ease the training for the cINN.

Finally, getting ISR jet events is not as common as normal events ($\sim$ 80k/180k). Getting two ISR jets at the same time is even rarer ($\sim$ 30k/180k). This might also contribute to the cINN not being able to learn this correlation as its occurrence is just too low. For this reason we trained a model on a dataset in which the prior probabilities of each, two, three and four jet events, are adjusted to be identical. It is important to point out that, since we provide the number of jets in the condition for the cINN, predicting data with different priors than the training data should theoretically not pose a problem.

From the two jet plots in Figure A.11 it is obvious that the new model performs similar for two jets to the first model. Notably, the mass distribution is not overestimated anymore to compensate for the low mass peak in the four jet case. We can also see improvements over the first model in Figures A.12 and A.13 where the invariant mass is now considerably better than before. However, this model shows anomalies in p_{x,q_1} among other observables in the form of an asymmetric peak (Figure A.14).

6.6. Non-Conditional INN

While the basic INN does not suffice for our purposes, we can still take a look at its ability to learn the bijective mapping as a way to incorporate both detector unfolding and simulation in a single model. We train two versions of the INN, the first one follows the first structure described in Chapter 5. We add noise to the parton side inputs, to adjust the dimensionality. Then we train the model to reduce the mean-squared distance between the predicted and true events on either side of the network. We found that it is beneficial, to reduce the influence of this loss and additionally introduce an MMD loss on all of the observables. The network (at least in detector to parton direction) has no way of incorporating stochastic processes into its predictions. The MMD allows for some freedom while purely using MSE does not account for this. We also use an MMD loss on all of the masses, as with the cINN before. We show the results for this model in Figure A.15.

Since the missing degrees of freedom on either side limit the network in its predictive power, for the second version of the INN we add additional noise padding of 10 dimensions to either side of the input. We now also include an MMD loss on the noise on either side to ensure the predicted noise follows the input distribution. In Figure A.16 the results for this model are provided.

We can see that the masses are well approximated by both models. The cut we apply in the data creation to the p_{T,j_2} of the detector seems to be very hard for the models to learn. Especially the first model has severe problems learning the detector-level p_{T,j_2}, which is likely impelled by the lack of degrees of freedom. It should also be noted that balancing all of these loss functions adds a considerable amount of hyperparameter search to the training process.

In Figure 6.4 we also show some calibration curves of the INN with added noise, to showcase the issue with a regular INN. Even though the mapping is probabilistic, the distribution we get when sampling from latent space does not take a realistic shape.

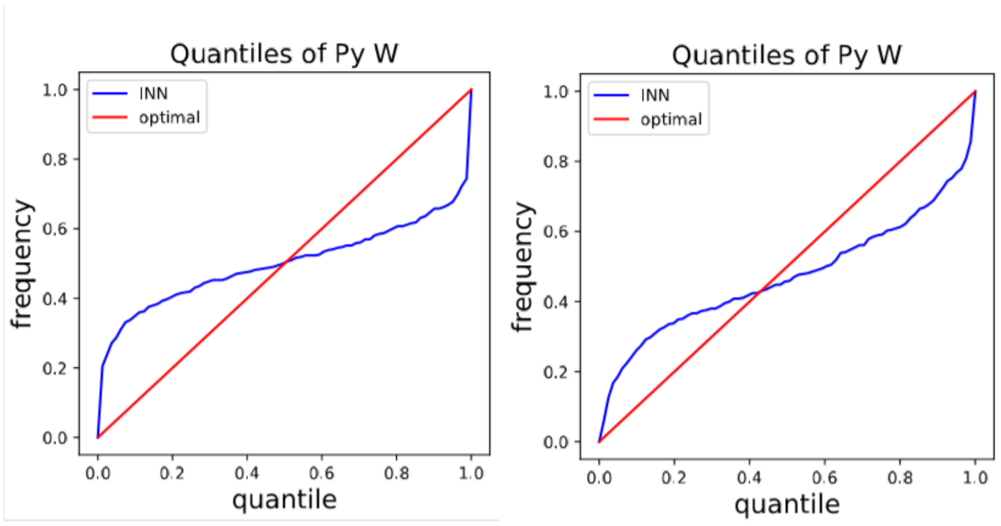

Figure 6.4.: Calibration Curves for the INN with noise on the detector distribution of $p_{y,W}$ (left) and the parton distribution of $p_{y,W}$ (right). These curves show the INN's inability to create meaningful distributions over the events when sampling from its latent space after being trained with MMD.

6.7. Shifted Masses

The behavior of the model is obviously based on the training dataset we use. This simulated dataset on the other hand depends on the simulation and the parameters we use for it. Finally our parameter choice is influenced by our assumptions and intuition for physics. However, this can be misleading in some cases, when our model will have a bias towards predicting the process that we already assumed to be true. We check to see how big the ability of the model is to correctly predict the underlying process, despite being trained on data that assumes a different process. For this, we train a model on the standard dataset and create testing data where the masses of each particle are shifted.

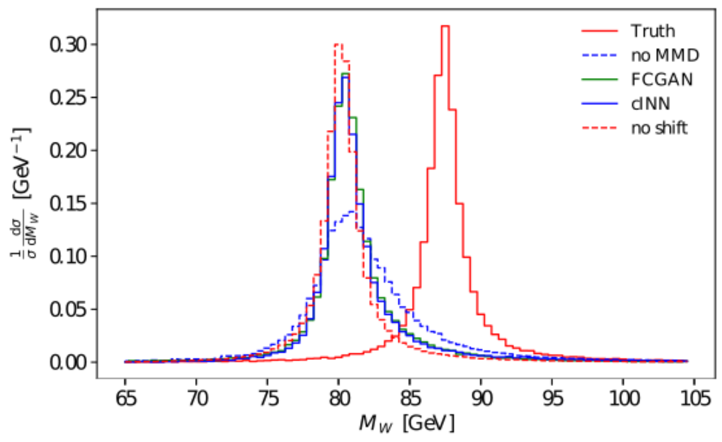

Figure 6.5.: Invariant Masses of the cINN (blue, solid) and the cGAN (green, solid), when the data deviates from the original training data. Both of the models stay very close to the old mass peak (red, dashed). Training a cINN without MMD reduces this bias (blue, dashed). However, it is still very much shifted towards the old mass peak.

As can be seen in Figure 6.5, the invariant mass distribution generated by the cINN and the cGAN very closely follow the unshifted distribution. When we use no MMD to train the model, the peak is still way closer to the old mass peak, but we can see a clear shift towards the new mass peak, which could help spot the shift in distributions. The reason for why the models trained with MMD do not respond to the shifted mass is likely the fact that we marginalize over a batch of events when calculating the MMD during training. By doing this, as explained in Section 5.4, we decorrelate the invariant mass distribution of the particles from the detector-level input. A working conditional MMD could potentially fix this problem.

6.8. Hidden Mother Particle

We simulate a process where sometimes a hidden particle W' is created before the W and Z Bosons to showcase how we could find new particles with the help of the cINN. This W' particle then decays into the W and Z particles, which individually follow the same distributions as before. The difference is the correlation between W and Z, more specifically adding the four-vectors of W and Z back together will result in a very specific mass peak in the cases where the W' was created. We again use a model that has been trained on data without any W' production.

Figure 6.6 shows that both the cINN and cGAN detect the mass peak, the cINN's response is considerably more pronounced. In this case, even though the sharp peak is

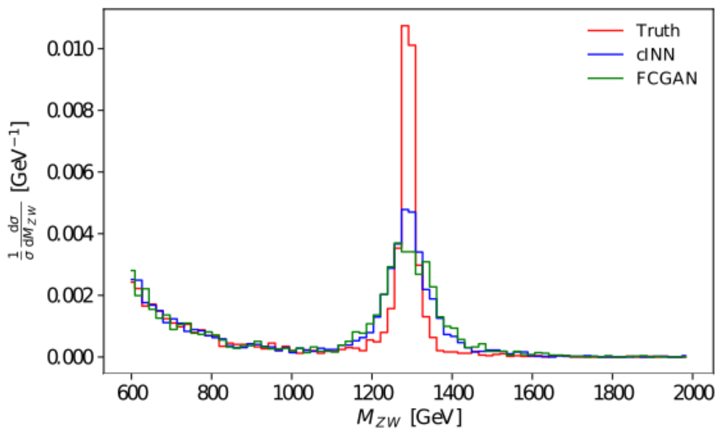

Figure 6.6.: Masses of the sum of the Z and W Boson for the cINN (blue) and the cGAN (green), on a dataset where a W' particle has been created in some events. Both models capture the new mass peak, the cINN predicts the shape more accurately.

not matched exactly, both networks would have shown some anomaly that could help us detect the particle.

7. Conclusion

In this work, we used cINNs to unfold detector effects and showering on simulated data, introduced during the measurement of ZW production. We showed that the cINN outperforms the cGAN when slicing and creates statistically more coherent results. Our model can accurately map distributions of low level observables on the sliced and full phase space, as well as the invariant masses of the full phase space. While the cINN contributes to reducing the error in comparison to the cGAN, the masses of the sliced phase space still show deviations from the true distribution. We show the limits of the new architecture when introducing new disruptive effects in the form of ISR. The bias affinity when training the cINN with MMD was depicted as it was not able to detect a mass shift in a test dataset. We show that the cINN is able to detect a hidden mother particle from collision data, by reconstructing its mass peak, even if it wasn't present in its training data. Employing new techniques from generative tasks will likely be able to improve these results even further. A next step could consist of improving the conditional MMD to address the problems with slicing and possibly the bias we acquired from using regular MMD.

A. Appendix

In the following pages, additional figures for the results from Chapter 6 are provided.

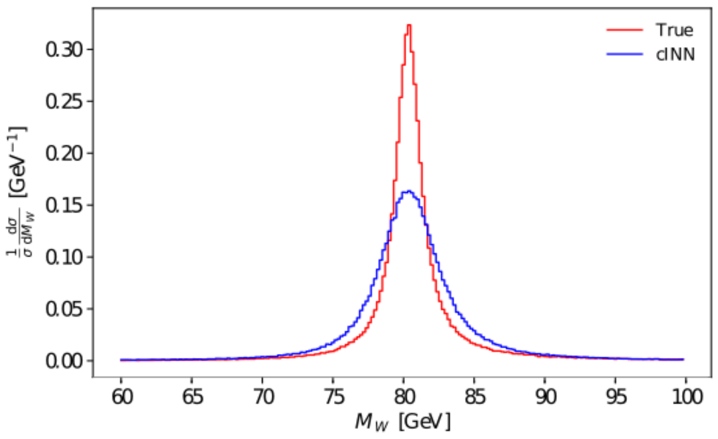

Figure A.1.: Invariant masses of the W Boson generated by a cINN trained without MMD.

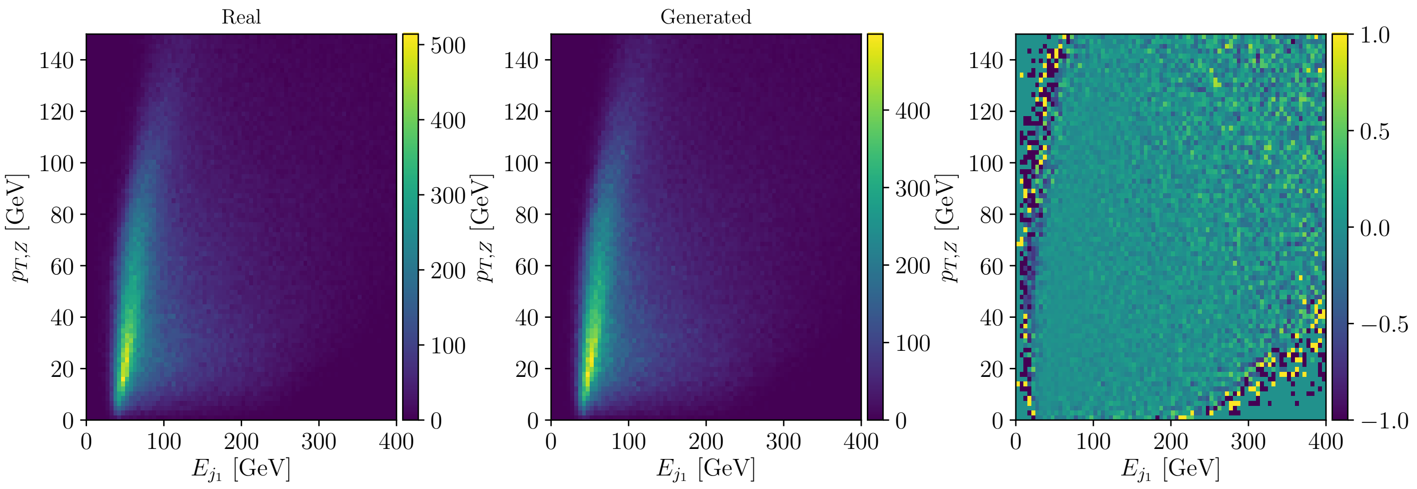

Figure A.2.: Correlation plot of the cINN. The left panel shows the true correlation, the middle panel the generated correlation and the right panel the difference between the two.

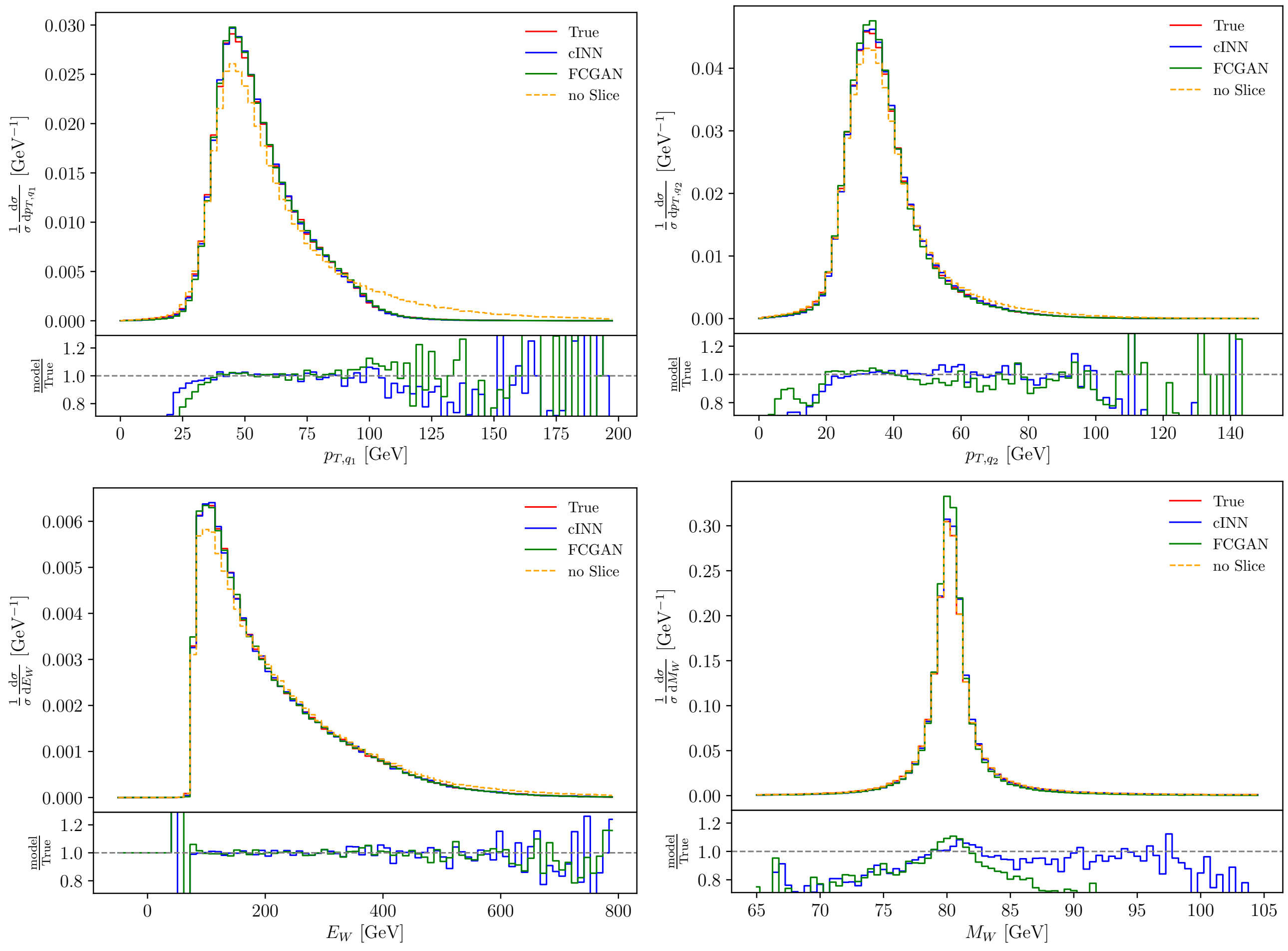

Figure A.3.: Slice I distributions for the cINN (blue) and GAN (green). Only events with the detector measurements in the range of $p_{T,j_1} = 30 \ldots 100$ GeV are used.

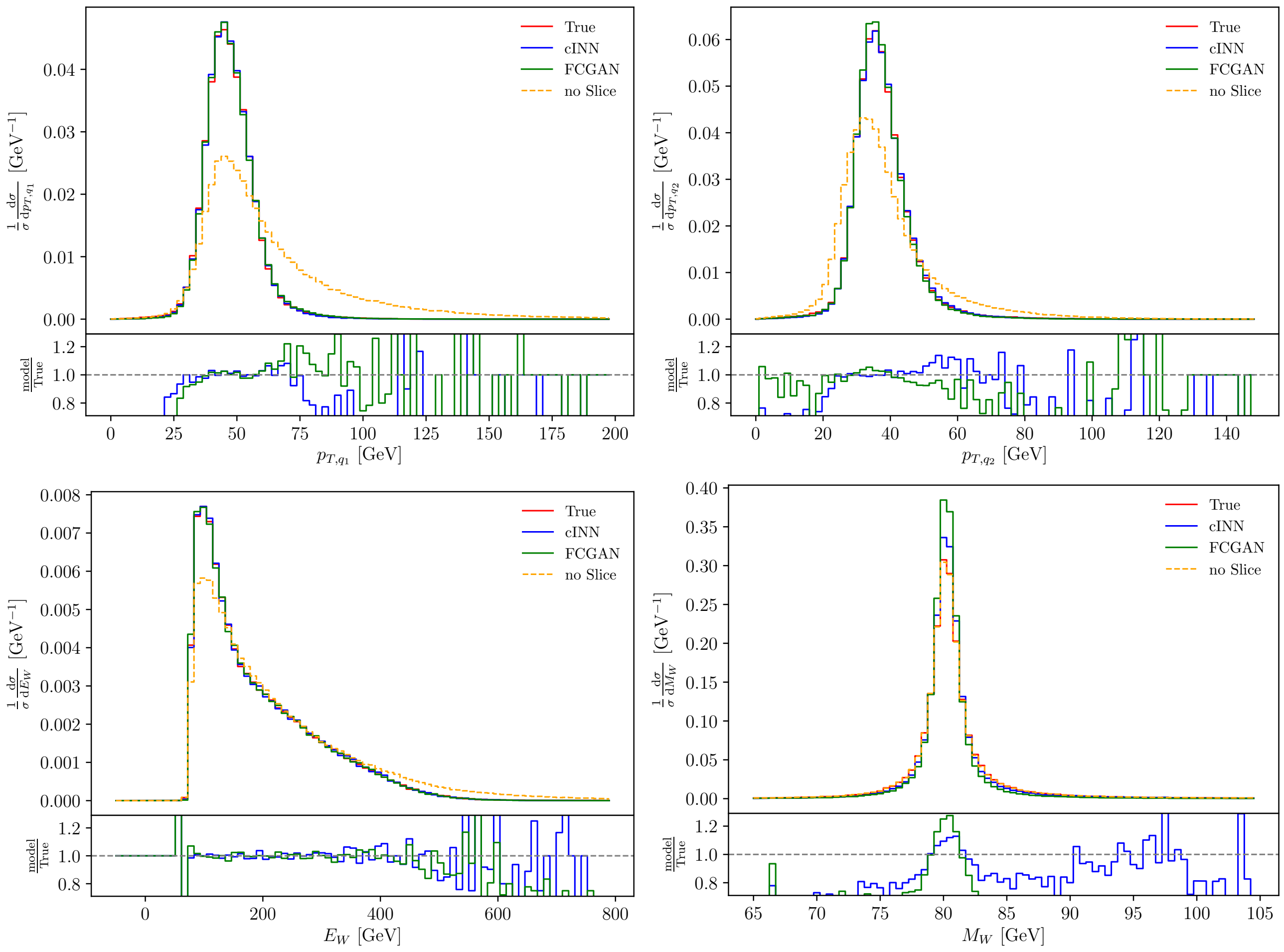

Figure A.4.: Slice II distributions for the cINN (blue) and GAN (green). Only events with the detector measurements in the range of $p_{T,j_1} = 30 \ldots 60$ GeV and $p_{T,j_2} = 30 \ldots 50$ GeV are used.

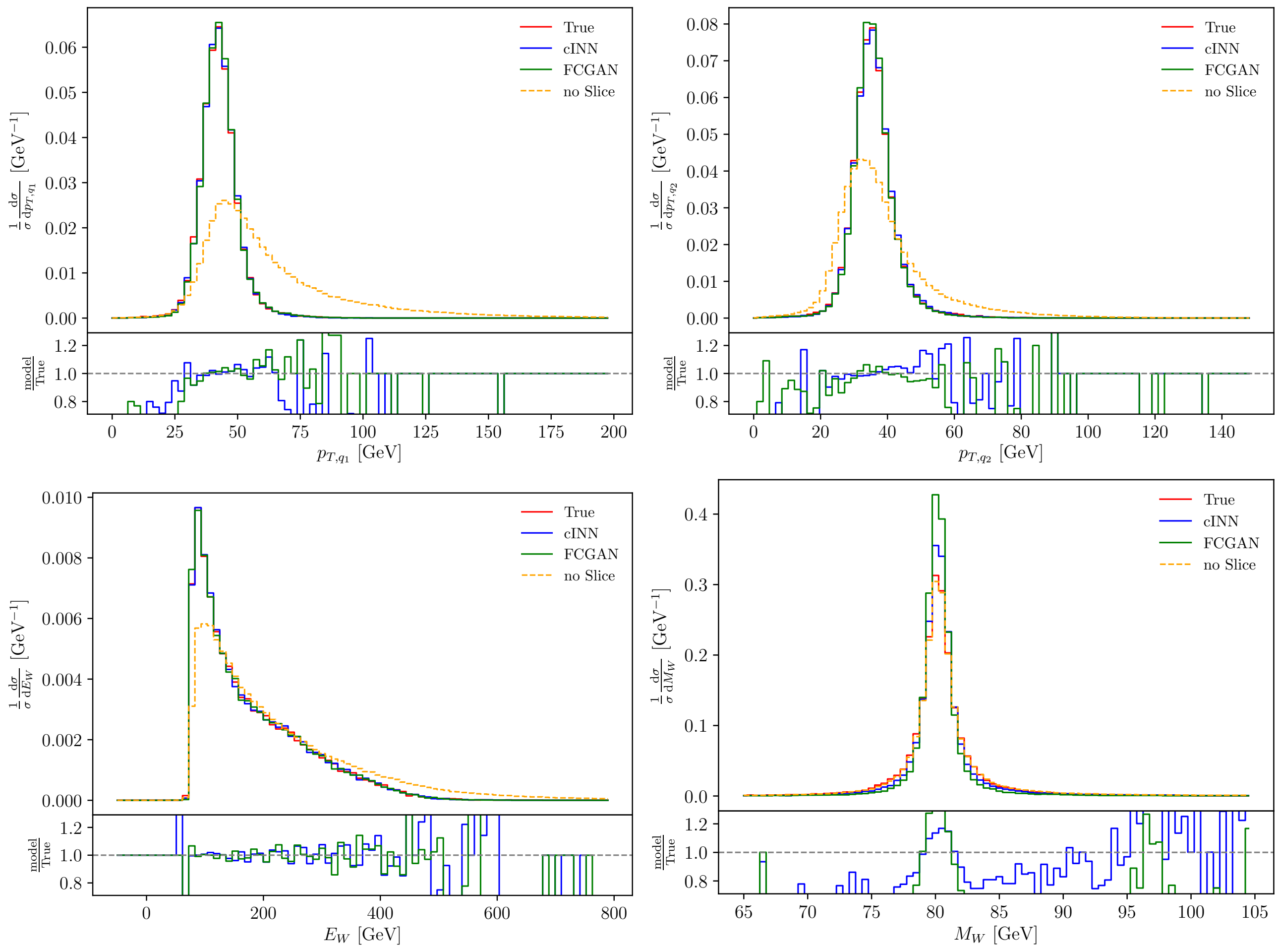

Figure A.5.: Slice III distributions for the cINN (blue) and GAN (green). Only events with the detector measurements in the range of $p_{T,j_1} = 30 \dots 50$ GeV, $p_{T,j_2} = 30 \dots 40$ GeV and $p_{T,\ell^-} = 20 \dots 50$ GeV are used.

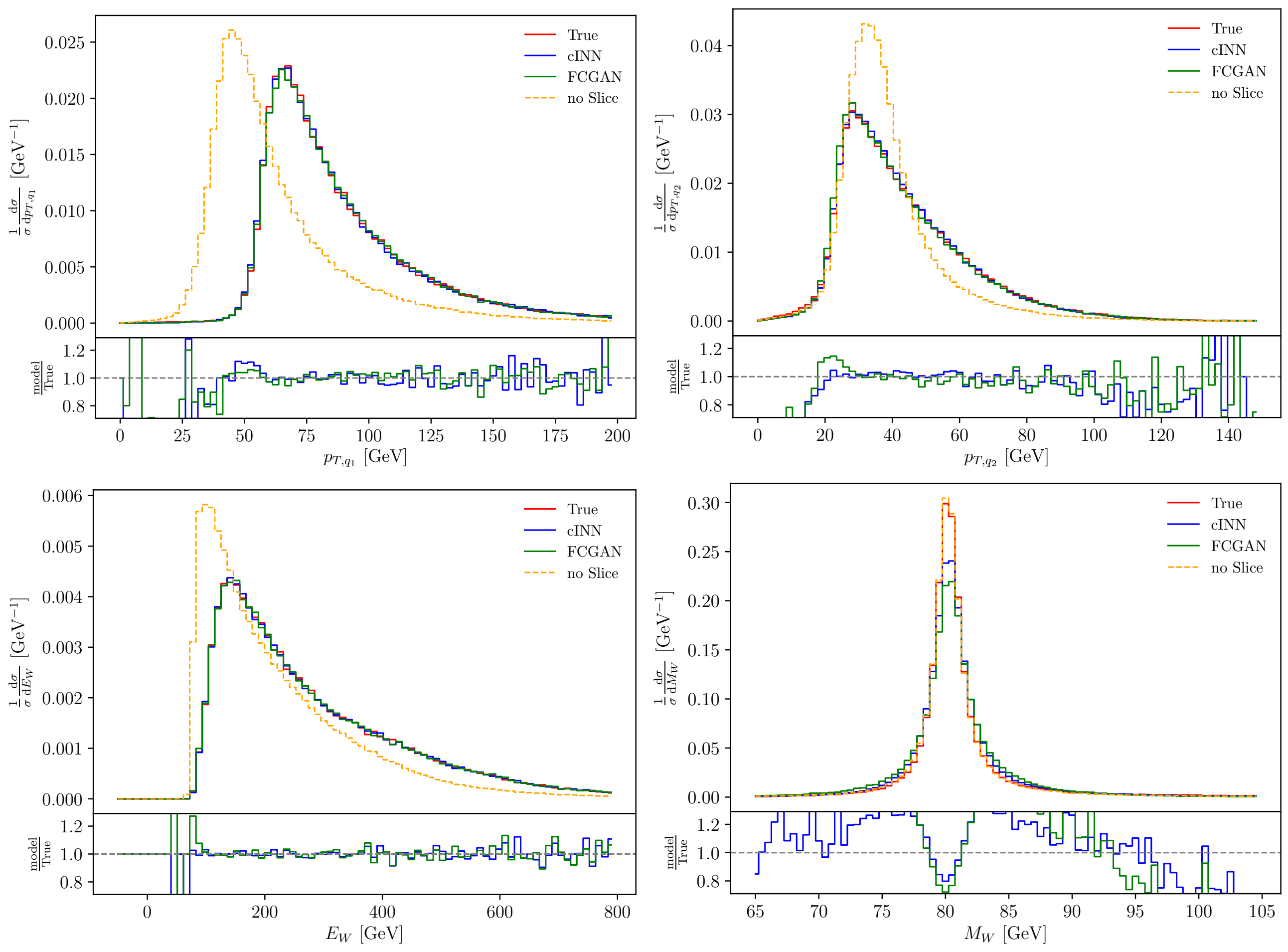

Figure A.6.: Slice IV distributions for the cINN (blue) and GAN (green). Only events with the detector measurements in the range of $p_{T,j_1} > 60$ GeV are used.

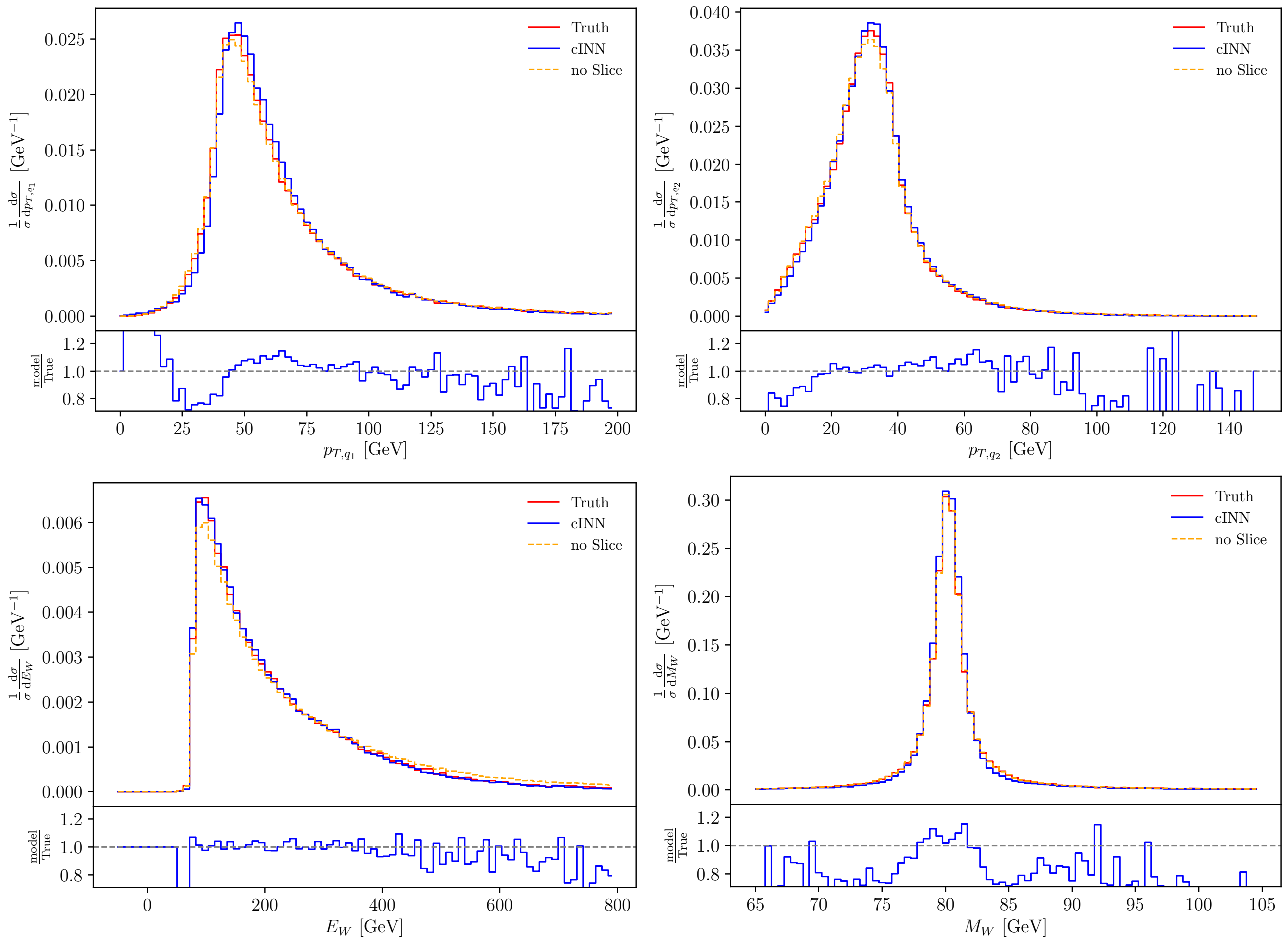

Figure A.7.: Predictions of the cINN trained on a dataset with ISR. Only events where two jets were measured at the detector are shown.

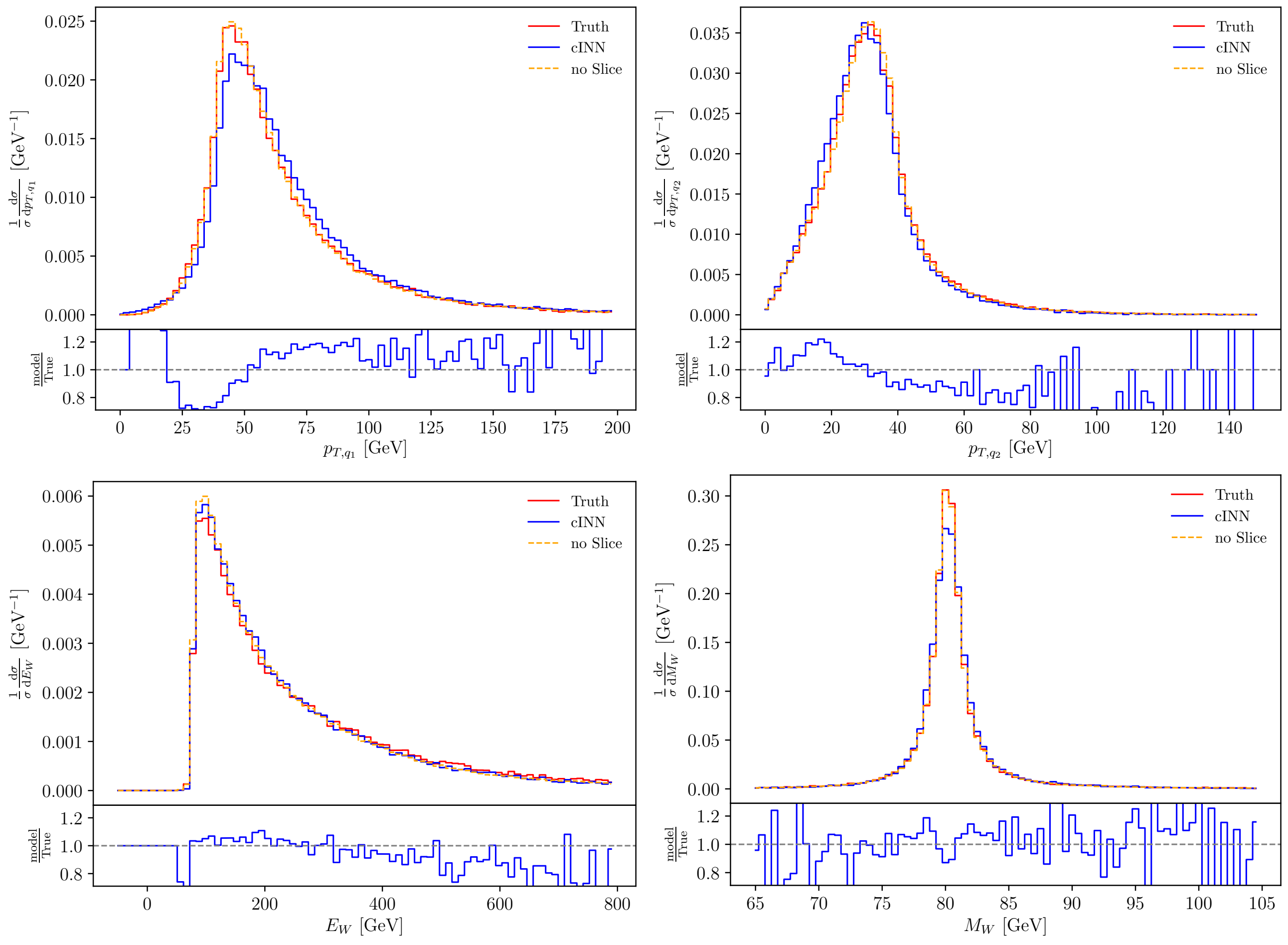

Figure A.8.: Predictions of the cINN trained on a dataset with ISR. Only events where three jets were measured at the detector are shown.

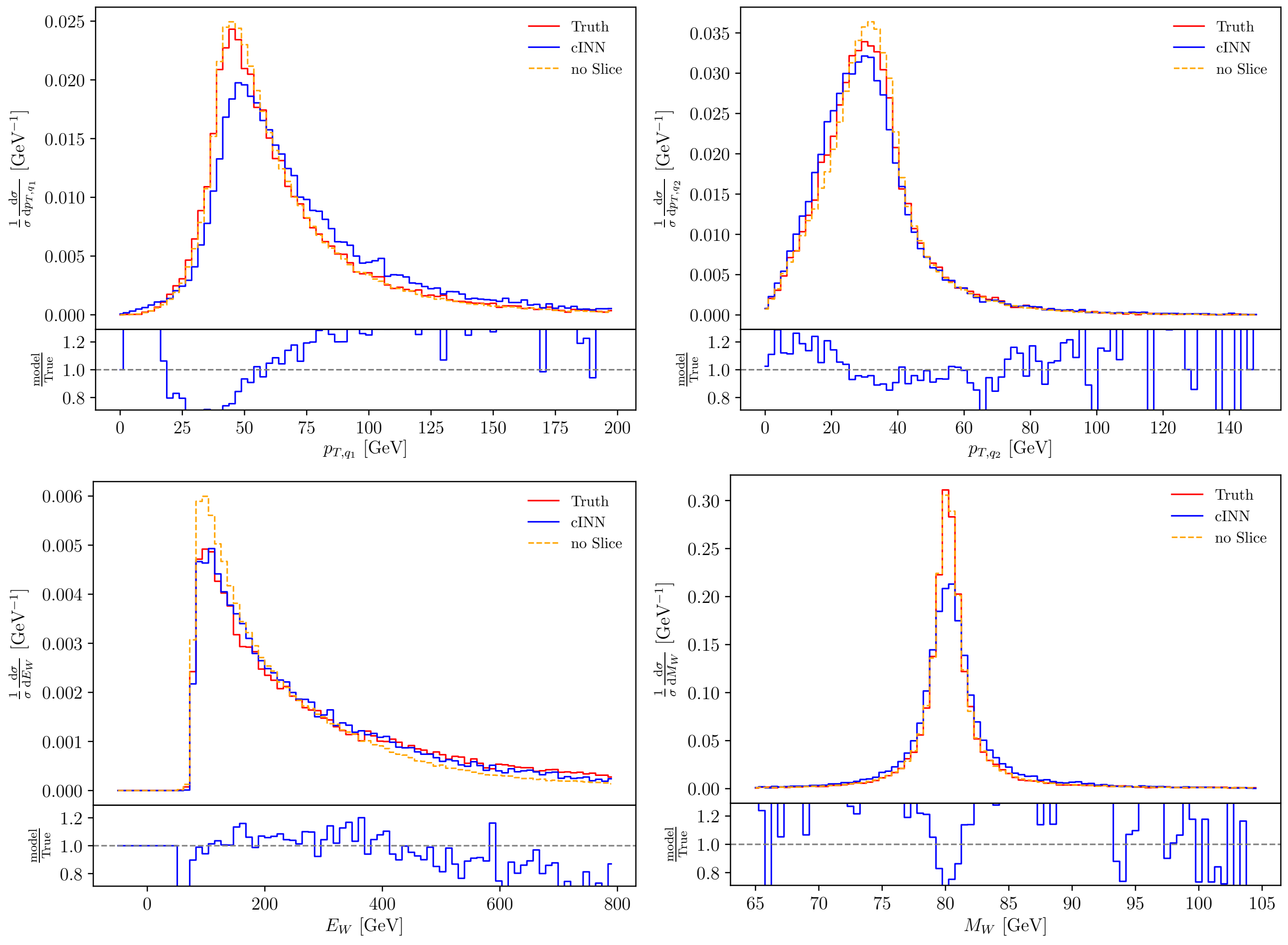

Figure A.9.: Predictions of the cINN trained on a dataset with ISR. Only events where four jets were measured at the detector are shown.

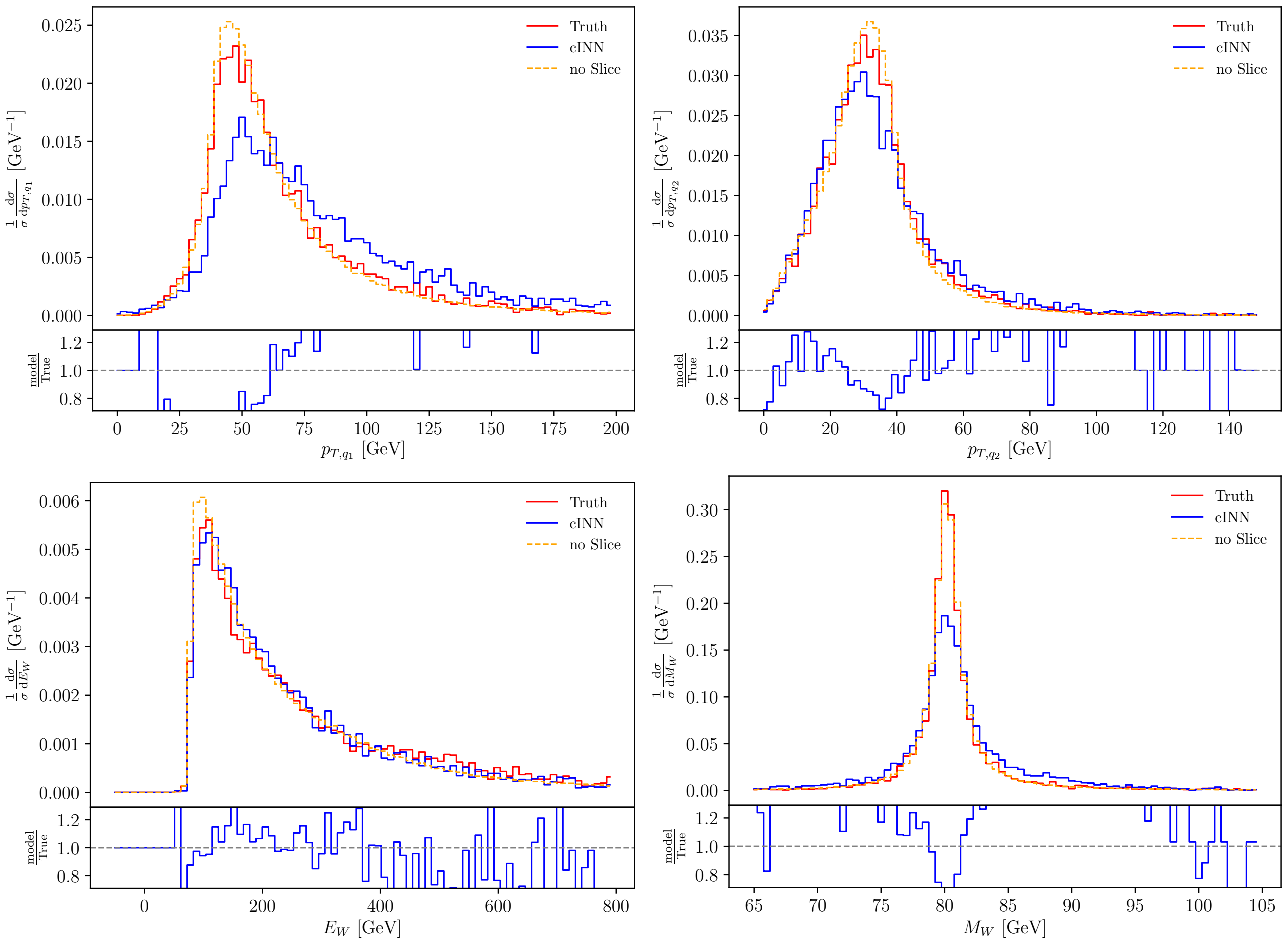

Figure A.10.: Predictions on data with ISR for the events where four jets were measured by a cINN augmented by the classifier labels

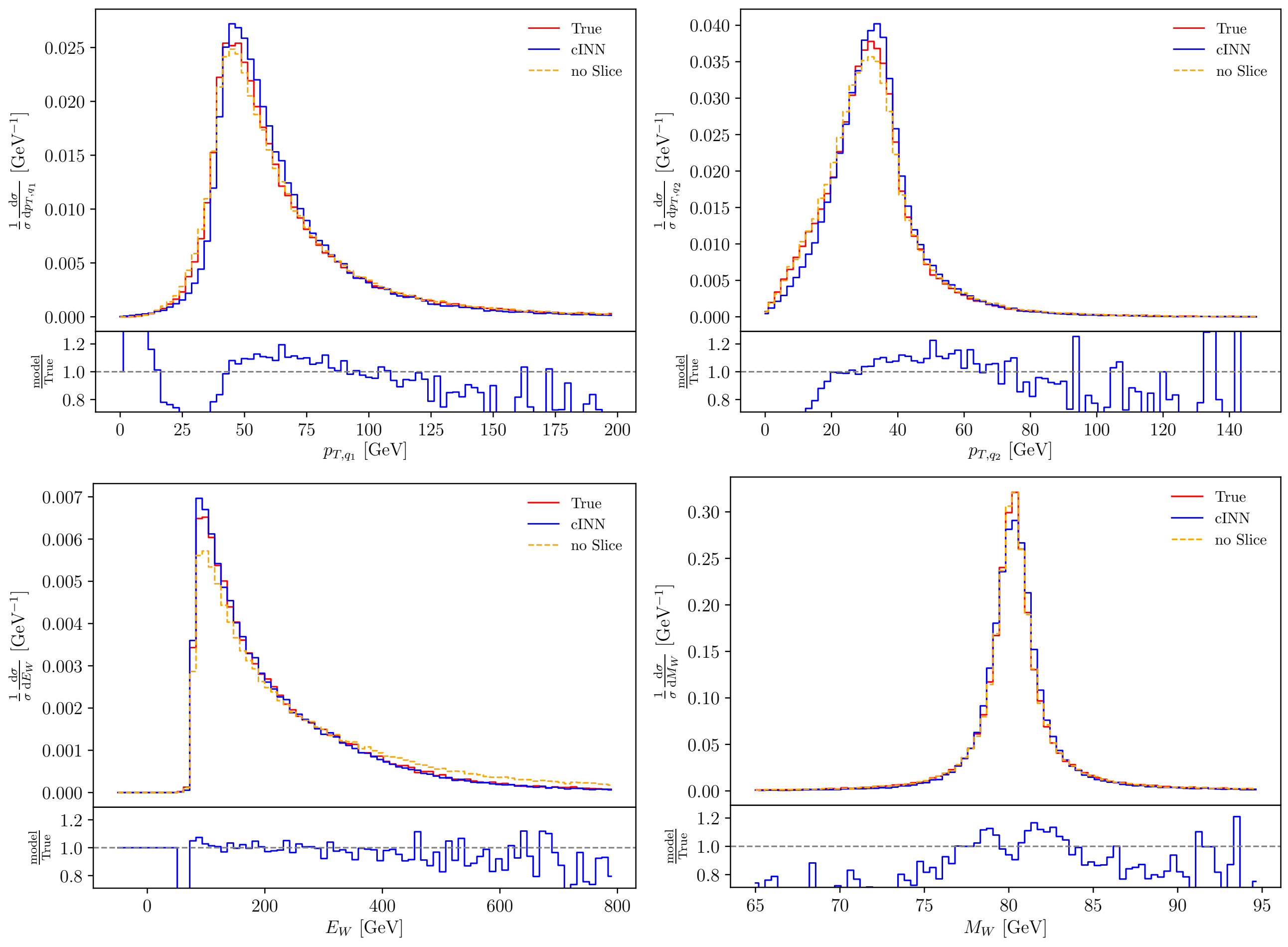

Figure A.11.: Predictions on data with ISR of the cINN with adjusted priors for the events where two jets were measured at the detector

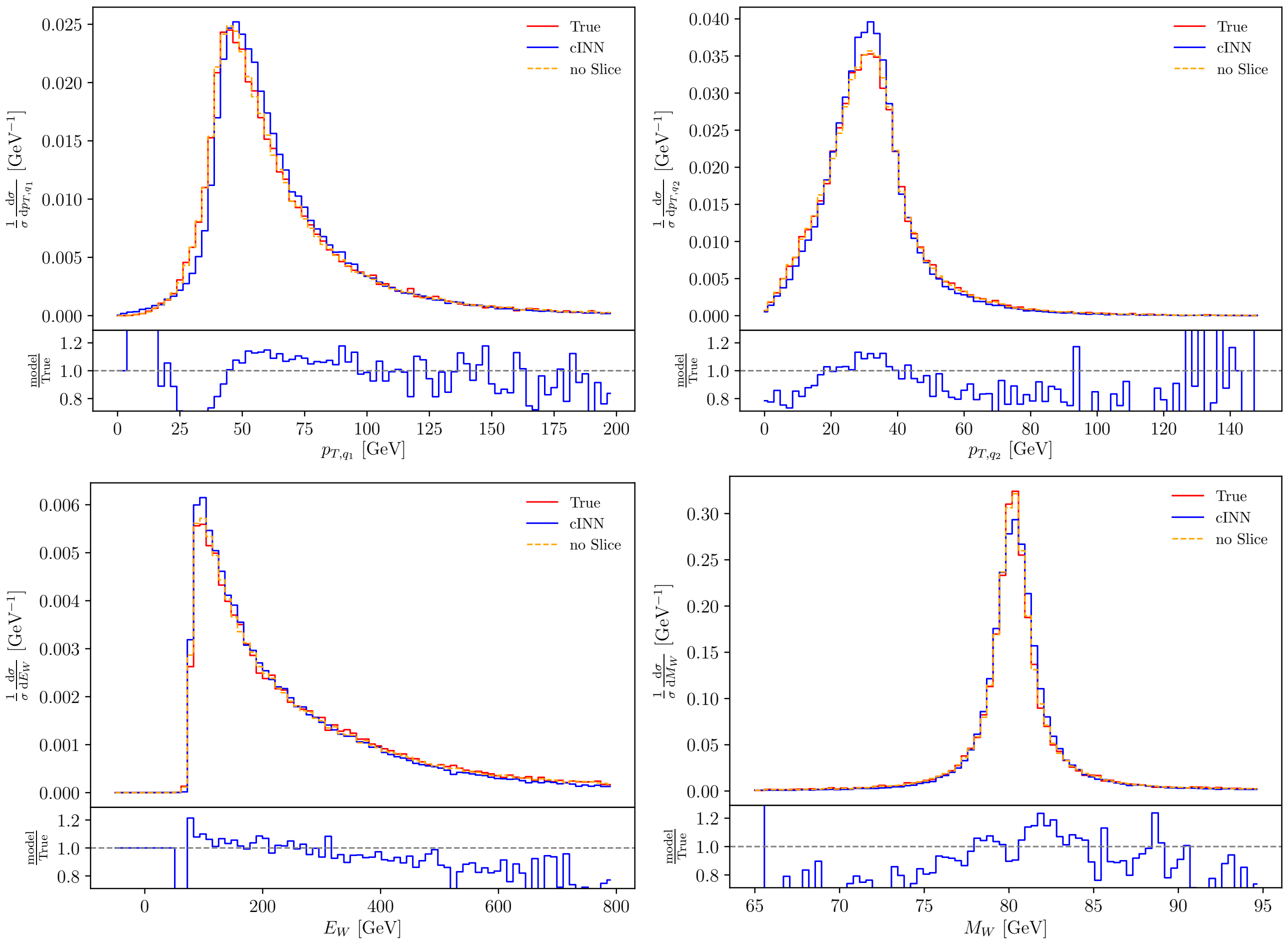

Figure A.12.: Predictions on data with ISR of the cINN with adjusted priors for the events where three jets were measured at the detector

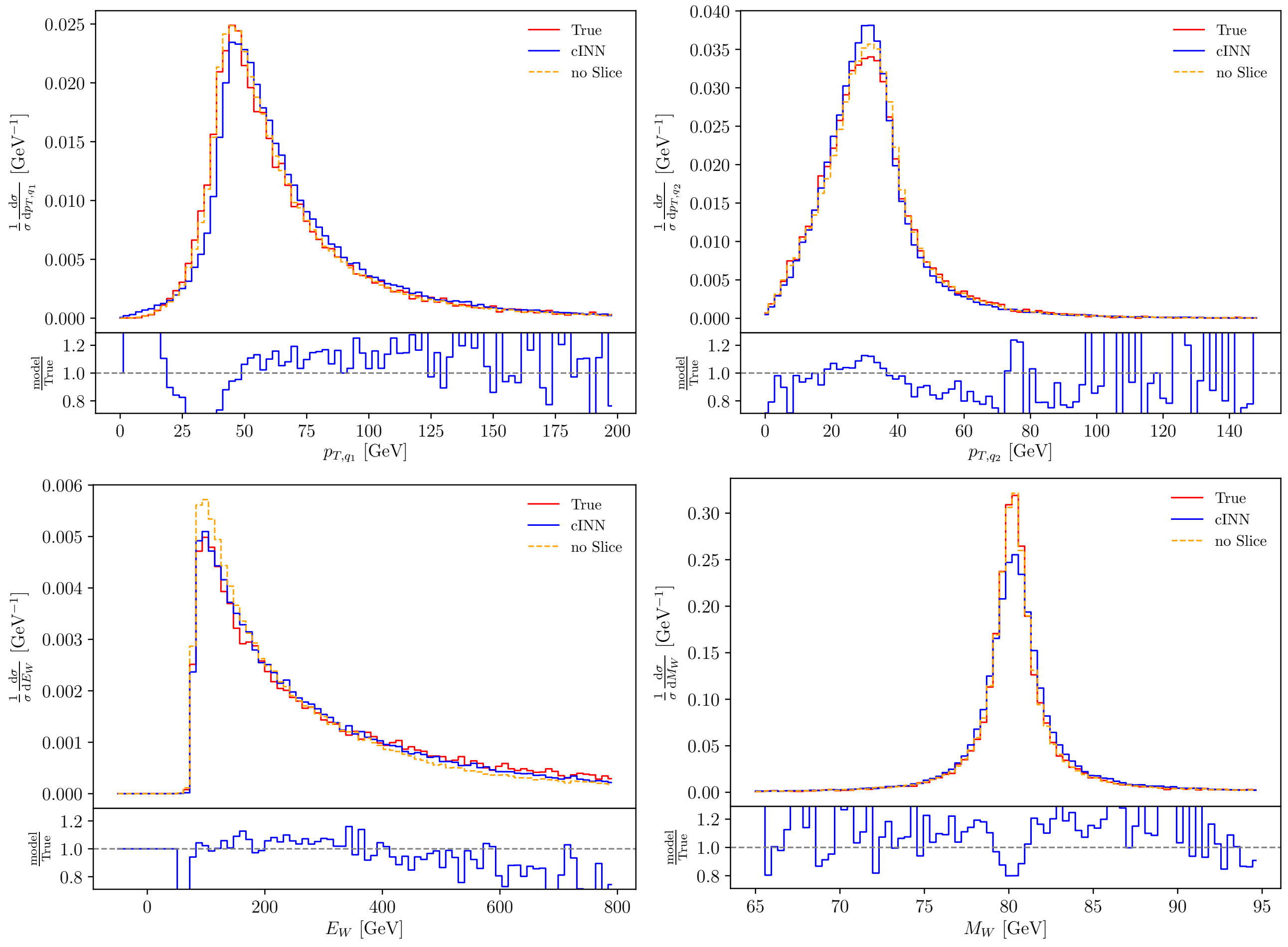

Figure A.13.: Predictions on data with ISR of the cINN with adjusted priors for the events where four jets were measured at the detector

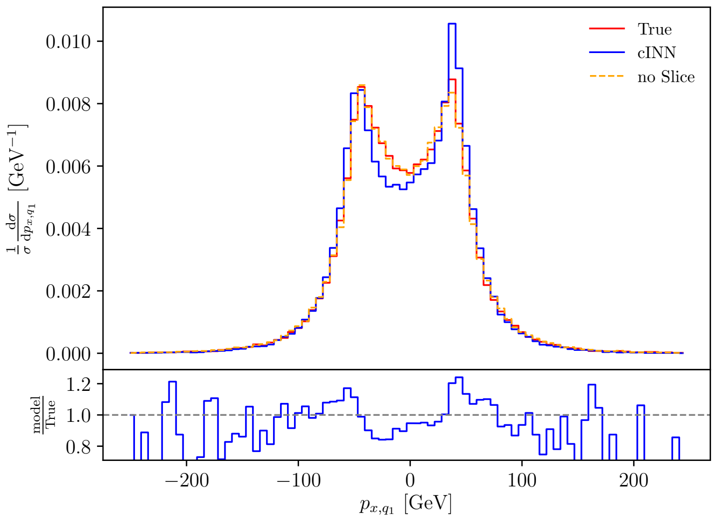

Figure A.14.: Generated distribution by the cINN with adjusted ratios on ISR data of p_{x,q_1} for events where two jets were measured. The model shows an asymmetric peak at the positive bulk of p_{x,q_1}.

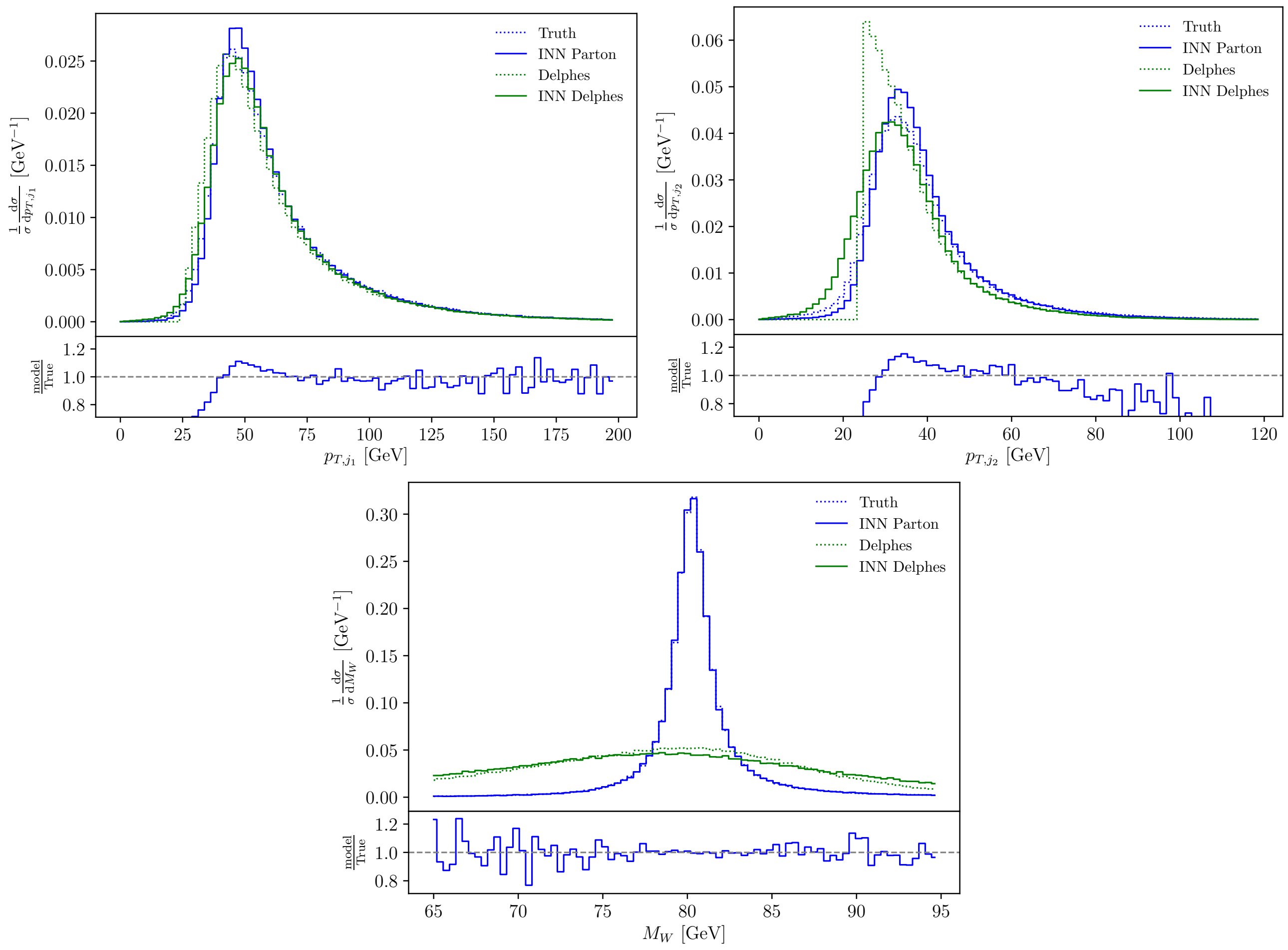

Figure A.15.: Full Phase Space Distributions for the INN without noise. The model does not learn the detector distribution p_{T,j_2} where a cut was applied.

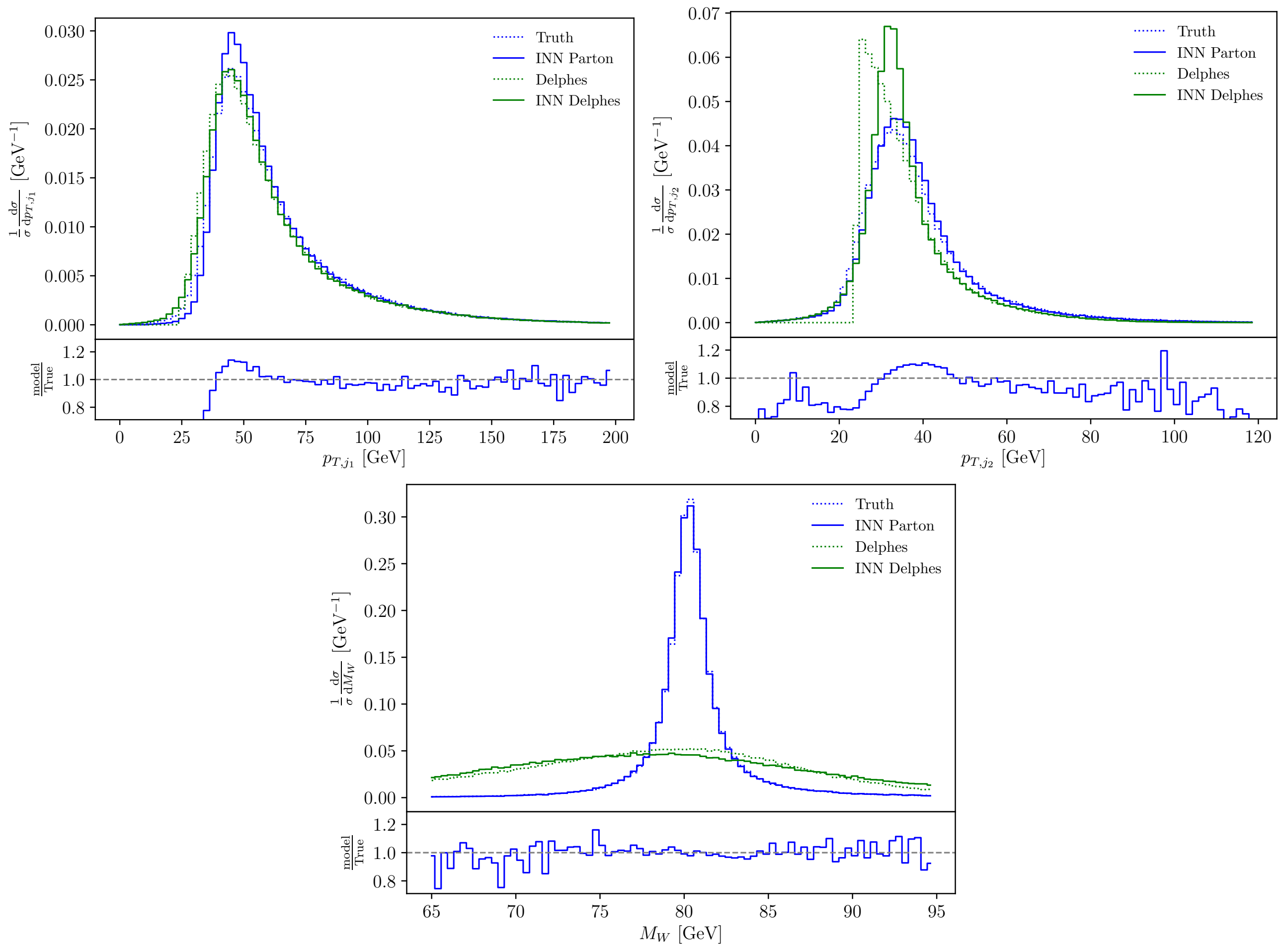

Figure A.16.: Full Phase Space Distributions for the INN with noise. While improvements are made in p_{T,j_2} compared to the INN without noise, the peak still does not fit the sharp cut of the dataset.

Bibliography

[1] Altarelli, G., & Wells, J. (n.d.). Collider physics within the standard model : a primer.

[2] Ardizzone, L., Kruse, J., Wirkert, S., Rahner, D., Pellegrini, E. W., Klessen, R. S., . . . Köthe, U. (2018). Analyzing Inverse Problems with Invertible Neural Networks. Retrieved from http://arxiv.org/abs/1808.04730

[3] Ardizzone, L., Lüth, C., Kruse, J., Rother, C., & Köthe, U. (2019). Guided Image Generation with Conditional Invertible Neural Networks. Retrieved from http://arxiv.org/abs/1907.02392

[4] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. http://arxiv.org/abs/1701.07875

[5] Bellagente, M., Butter, A., Kasieczka, G., Plehn, T., & Winterhalder, R. (2019). How to GAN away Detector Effects. Retrieved from http://arxiv.org/abs/1912.00477

[6] Beringer, J., Arguin, J. F., Barnett, R. M., Copic, K., Dahl, O., Groom, D. E., . . . Schaffner, P. (2012). Review of particle physics. Physical Review D - Particles, Fields, Gravitation and Cosmology, 86(1), 010001. https://doi.org/10.1103/PhysRevD.86.010001

[7] Butter, A., Plehn, T., & Winterhalder, R. (2019). How to GAN LHC Events. SciPost Physics, 7(6). https://doi.org/10.21468/SciPostPhys.7.6.075

[8] Cacciari, M., Salam, G. P., & Soyez, G. (2008). The anti-k_t jet clustering algorithm. https://doi.org/10.1088/1126-6708/2008/04/063

[9] Cern. About — Worldwide LHC Computing Grid. (n.d.). Retrieved February 28, 2020, from https://wlcg-public.web.cern.ch/about

[10] Chen, T., Lucic, M., Houlsby, N., & Gelly, S. (2018). On Self Modulation for Generative Adversarial Networks. 7th International Conference on Learning Representations, ICLR 2019. http://arxiv.org/abs/1810.01365

[11] Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On Empirical Comparisons of Optimizers for Deep Learning. Retrieved from http://arxiv.org/abs/1910.05446

[12] de Favereau, J., Delaere, C., Demin, P., Giammanco, A., Lemaître, V., Mertens, A., & Selvaggi, M. (2013). DELPHES 3, A modular framework for fast simulation of a generic collider experiment. Journal of High Energy Physics, 2014(2). https://doi.org/10.1007/JHEP02(2014)057

[13] de Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O., & Courville, A. (2017). Modulating early visual processing by language. Advances in Neural Information Processing Systems, 2017-December, 6595–6605. http://arxiv.org/abs/1707.00683

[14] Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using Real NVP. Retrieved from http://arxiv.org/abs/1605.08803

[15] Gao, C., Hoeche, S., Isaacson, J., Krause, C., & Schulz, H. (2020). Event Generation with Normalizing Flows. Retrieved from http://arxiv.org/abs/2001.10028

[16] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). Generative adversarial nets. In Advances in Neural Information Processing Systems (Vol. 3, pp. 2672–2680). Neural information processing systems foundation. https://doi.org/10.3156/jsoft.29.5_177_2

[17] Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., & Smola, A. J. (2012). A Kernel Two-Sample Test. The Journal of Machine Learning Research, 13, 723–773.

[18] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. http://arxiv.org/abs/1207.0580

[19] Kessy, A., Lewin, A., & Strimmer, K. (2015). Optimal whitening and decorrelation. American Statistician, 72(4), 309–314. https://doi.org/10.1080/00031305.2016.1277159

[20] Kingma, D. P., & Ba, J. L. (2015, December 22). Adam: A method for stochastic optimization. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings.

[21] Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative Flow with Invertible 1x1 Convolutions. Advances in Neural Information Processing Systems, 2018-December, 10215–10224. Retrieved from http://arxiv.org/abs/1807.03039

[22] Kobyzev, I., Prince, S., & Brubaker, M. A. (2019). Normalizing Flows: An Introduction and Review of Current Methods. Retrieved from http://arxiv.org/abs/1908.09257

[23] Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. AIChE Journal, 37(2), 233–243. https://doi.org/10.1002/aic.690370209

[24] Krogh·, A., & Hertz, J. A. (1992). A Simple Weight Decay Can Improve Generalization.

[25] Li, Y., Swersky, K., & Zemel, R. (2015). Generative Moment Matching Networks. 32nd International Conference on Machine Learning, ICML 2015, 3, 1718–1727. Retrieved from http://arxiv.org/abs/1502.02761

[26] Livan, M., & Wigmans, R. (2019). Calorimetry for Collider Physics, an Introduction. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-23653-3

[27] Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets. Retrieved from http://arxiv.org/abs/1411.1784

[28] Montgomery, D. C., & Runger, G. C. (2011). Applied statistics and probability for engineers. Wiley.

[29] NESTEROV, & E., Y. (1983). A method for solving the convex programming problem with convergence rate O(1/k$\hat{2}$). Dokl. Akad. Nauk SSSR, 269, 543–547.

[30] Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation Functions: Comparison of trends in Practice and Research for Deep Learning. Retrieved from http://arxiv.org/abs/1811.03378

[31] Ragusa, F., & Rolandi, L. (2007). Tracking at LHC. New Journal of Physics, 9(9), 336. https://doi.org/10.1088/1367-2630/9/9/336

[32] Ren, Y., Li, J., Luo, Y., & Zhu, J. (2016). Conditional Generative Moment-Matching Networks. Retrieved from http://arxiv.org/abs/1606.04218

[33] Sisson, S. A., Fan, Y., & Beaumont, M. A. (2018). Overview of Approximate Bayesian Computation. http://arxiv.org/abs/1802.09720

[34] Sjöstrand, T., Lönnblad, L., & Mrenna, S. (2001). PYTHIA 6.2 Physics and Manual. Retrieved from http://arxiv.org/abs/hep-ph/0108264

[35] Song, H., Kim, M., Park, D., & Lee, J.-G. (2019). Prestopping: How Does Early Stopping Help Generalization against Label Noise? http://arxiv.org/abs/1911.08059

[36] Souza, César R. “Kernel Functions for Machine Learning Applications.” 17 Mar. 2010. Web. http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html .

[37] Subramanian, V. (n.d.). Deep learning with PyTorch : a practical approach to building neural network models using PyTorch.

[38] Tsamparlis, M. (2019). Special Relativity. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-27347-7