Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Sophia Vent

born in Backnang (Germany) 1999

September 2021

Expressive Uncertainties for Generated LHC Events

Introducing Bayesian invertible neural networks to generate a spectrum of LHC event samples with expressive uncertainties

This Bachelor Thesis has been carried out by Sophia Vent at the
Institute for Theoretical Physics in Heidelberg
under the supervision of
Prof. Tilman Plehn

Abstract

Using machine learning to generate LHC events showed huge success for data analyses. This thesis introduces Bayesian invertible neural networks (BINN) allowing us to compute uncertainties on the generated event samples. Keeping the focus on creating a pipeline for the BINN and observing the properties of the Uncertainties for realistic LHC events to manipulate them through introducing systematic uncertainties.

Zusammenfassung

Machine Learning Techniken zeigten große Erfolge für LHC Event Generatoren. In dieser Thesis werden Bayesian invertierbare Neuronale Netze (BINN) eingeführt um Unsicherheiten für die Event samples zu berechnen. Der Fokus liegt dabei darauf eine funktionierende Pipeline für das BINN zu erstellen, das Verhalten der Fehler für realistische LHC Prozesse zu untersuchen und die Fehler im Sinne von systematischen Unsicherheiten zu manipulieren.

Contents

1. Introduction

Driven by the curiosity to understand the fundamental laws of physics, high energy particle collisions provide information about the underlying theories and properties of the fundamental particles and interactions. Scattering processes, the decay of different particles and discovering new particles reveal the connection between different forces and particle features. However, to test theories with experimental data collected at particle colliders like the Large Hadron Collider (LHC) in CERN, Switzerland we need a simulation to generate events based on a theory that should be verified. Monte Carlo techniques have been around for decades and tools like Madgraph [1] or SHERPA [2] are incredibly powerful tools to generate events and compare the Standard Model to experimental data collected at the LHC. Taking into account that the current collision energy is around $\sqrt{s}$ =13 TeV, Monte Carlo generated events can model a wide variety of different processes. But as there are already new plans for new colliders such as the FCC (Future Circular Collider) to reach collision energies of 100 TeV, and the upcoming Run 3 at the LHC increasing the luminosity by far, Monte Carlo generations become computationally expensive and time consuming. To match the number of events produced at the upcoming Run 3 at the LHC, the need to find different methods for event generation has increased significantly. Different types of neural networks are showing promising results that machine learning approaches will keep up with the ever growing event rate at the LHC and are convenient for a variety of purposes besides event generation such as jet-tagging and anomaly detection [3, 4]. Generative adversarial networks (GAN) [5–7], variational auto encoders (VAE) [8, 9], and Invertible Neural Networks (INN) [10–15] are already applicable. Using a neural network allows to learn a given phase space density, generated by Monte Carlo techniques [16, 17], and generate a multitude of events once the network is fully trained.

Uncertainties are a major factor in data analysis, whether that being on the experimental side, looking at uncertainties given by detectors or the analysis itself or on the theory side. Usual neural networks can only account for a statistical error, for example given by the Poisson statistic. Introducing Bayesian neural networks allows to produce uncertainties on the generated events. Computing uncertainties that go beyond Poisson statistics, showcasing the uncertainties produced through theory or the training itself, as the neural network is not trained to perfection. The aim of this thesis is to introduce these Bayesian invertible neural networks (BINN)[18, 19] for LHC events and especially the uncertainties given by the BINN and to observe how training statistics, training limitations and systematic errors contribute to the generated uncertainties. It is an attempt to demystifying the "black box" through understanding the uncertainties.

The last part of the thesis is about going beyond the Standard Model, and trying to manipulate the uncertainties by introducing new (semi-realistic) theory uncertainties motivated by the Standard Model effective field theory predictions leading to a contribution to the uncertainties.

2. Physics

2.1. Standard Model

The *Standard Model* is one of the best tested theories known to date. The Standard Model can be described as a gauge quantum field theory containing all the fundamental particles- the leptons, quarks, gluons and the bosons.

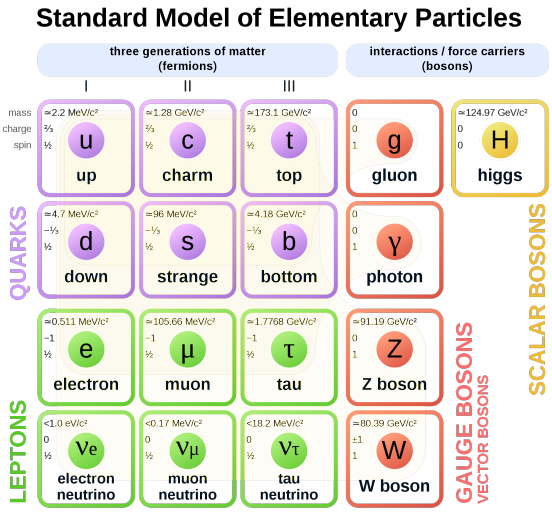

Fig. 1: Fundamental Particles of the Standard Model *taken from [20]*

Being mathematically self-consistent, the Standard Model explains three of the four fundamental forces and incorporates special relativity. Using the Lagrangian of the Standard Model allows to make predictions for particle collisions in search for new particles.

2.2. Jets

Analysing jets at the LHC provides information about the hard scattering process. A large number of particles can be created by the collision of protons, such as quarks or gluons. Each quark or gluon carries a color charge. However, because of the confinement, they can not exists individually as the potential energy increases rapidly between two quarks with the distance. Therefore, they only exist in a bound state with a neutral color charge. Which means they appear in sets of three, with each quark carrying a different color or in pairs of two, with one quark carrying the color charge and the other the corresponding ”anti color”. To form this colorless state, quark-antiquark pairs undergo the process of hadronization. A quark or antiquark produced through the collision carries a fragment of the color charge. To obey the confinement they produce other colored objects around them from the vacuum, which happens several times. This ensemble of hadrons tend to travel in the same direction and can be clustered into one cone, which is the jet. There are many jet-tagging algorithms like the k_T or $anti - k_T$ [21] algorithm. Analysing the jets provides information about the particles produced at the hard scattering

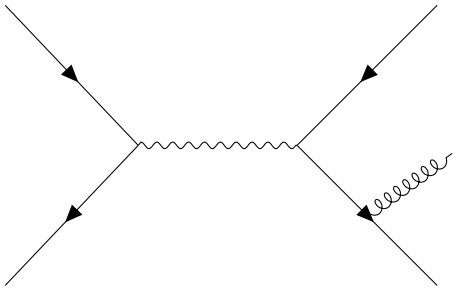

Fig. 2: Feynman Diagramm for a 3 jet final state

and is therefore used to find new fundamental particles. The gluon for example was discovered by jet tagging. As quarks are only produced in pairs there was no explanation for a 3 jet final state. This lead to the discovery of the gluon. The main data set consists of a three jet final state. The collsion beeing

$$pp \longrightarrow jjj \tag{2.1}$$

where j can be quarks or gluons. Figure 2 shows a quark-antiquark pair annihilating into a intermediate particle which is decaying into a quark-antiquark pair again, with one of them radiating a Gluon.

2.3. Z Production

The Z boson is the neutral intermediate vector boson mediating the weak force. During the proton-proton collision it is created due to a quark-anti quark annihilation. It can decay into different particle-antiparticle pairs. In this data the decay into a muon-antimuon pair was chosen. The simple Feynman diagram is given by Figure 3:

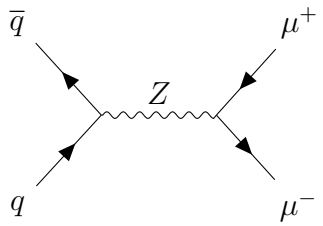

Fig. 3: Feynman Diagram for the Z Production

During this process jets are also produced. The number of jets can vary. In this particular case we allowed for final states with a variable jet number between 1 and 3. The process then becomes:

$$pp \to \mu\mu^- j, pp \to \mu\mu^- jj, pp \to \mu\mu^- jjj \tag{2.2}$$

Again, j can be quarks or gluons.
There are many possibilites of decays and annihilation processes in order to produce

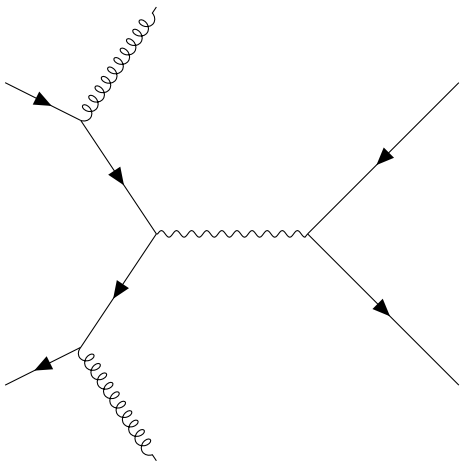

Fig. 4: Feynman Diagramm for the Production of the Z Boson decaying in a muon-antimuon pair. 2 addtional jets in the final state..

final states with a different number of jets in the final state. The different processes rely on the initial state particles, as they can be quarks or gluons. One possible Feynman Diagram could look like Figure 4. The final state being Z + 2 jets. The two jets come from the radiated gluons.

Data Set

Both data sets were generated using SHERPA, at a collision energy of E_{CMS} = 13TeV. After generating the process the anti-k_T[21, 22] algorithm was used to cluster the jets. The end result being
3 Jets : 5.4 million events
Z + Multijet: 5.4 million Events

- Number of 1 jet-events 4 million events
- Number of 2 jet-events 1.1 million events
- Number of 3 jet-events 0.3 million events

The data set is split into 2 parts. One being the data which is used for the training process *(Train)* , and the other part being used for testing *(True)* . For the Z + multijet data set, the ratio between test and train was chosen to be 0.5. For the 3 jet data set it was 0.82. In general, large training sets increase the duration of training but difficult phasespace densities profit from larger training sets.

Change of Variables

The data set provided by SHERPA consists of four-vectors. However, a change of variables is useful to keep the symmetries of the Data collected at the LHC. First we change from cartesian coordinates to spherical coordinates. As the experiments at the LHC measure the transversal momentum we train our model on the p_T

$$p_T = \sqrt{p_x^2 + p_y^2} \tag{2.3}$$

Furthermore, we do not train our model on the Azimutal angle

$$\phi = \arctan\left(\frac{p_y}{p_x}\right) \tag{2.4}$$

but rather on the difference of the Azimutal angle between particles. As for the 3 jet events :

$$\Delta\phi_{i,j} = \phi_i - \phi_j \tag{2.5}$$

Since it is enough to know 2 distributions of $\Delta\phi_{i,j}$ the last can be calculated as $\Delta\phi_{i,k} = \Delta\phi_{i,j} + \Delta\phi_{j,k}$. Taking out a degree of freedom, which translates into less trainable parameters, speeding up the training process and the performance. The last parameter, that is also used in LHC analysis, is the *pseudo rapidity* η, which is calculated by using the polar angle θ

$$\eta = -\ln[\tan\left(\frac{\theta}{2}\right)] \tag{2.6}$$

The four vector is then given by:

$$P = \begin{bmatrix} \mathrm{E} \\ \sin(\phi)p_T \\ \cos(\phi)p_T \\ \sinh(\eta)p_T \end{bmatrix} \tag{2.7}$$

Last but not least, we train our model on the invariant mass m rather than on the Energy.

The Standard Model as an EFT

Despite the huge success of the Standard Model, there are some phenomena unexplainable by it, for example incorporating general relativity or dark matter. In particular the failing at energies or distances where the gravitation is expected to emerge proves that there is a theory beyond the Standard Model. Nowadays, the Standard Model can be treated as an Effective Field Theory [23] .
The Standard Model can be described with a Lagrangian.

$$\mathcal{L} = \mathcal{L}_{SM} \tag{2.8}$$

Constructing the Standard Model Effective Field Theory (SMEFT) out of higher dimensional operators gives a consistent EFT generalisation of the Standard Model. The Lagrangian is then given by:

$$\mathcal{L} = \mathcal{L}_{SM} + \mathcal{L}^{(5)} + \mathcal{L}^{(6)} + \mathcal{L}^{(7)} + ... \tag{2.9}$$

For most analyses at the LHC, the additional terms in the Lagrangian can be neglected, as the deviation from the Standard Model can not be observed within the precision that is possible. Future runs at the LHC might make it possible to observe the properties of the additional Lagrangian terms. Consider the process:

$$\psi\bar{\psi} \to V \to \psi\bar{\psi} \tag{2.10}$$

At higher energies, some S matrix elements receive contributions from boson fields. These off-shell vertices have a non-trivial relation to LHC observables. One of the expansions in the dimension 6 operators being proportional to p^2, with p^2 being the general kinematic invariant. Observing the deviation from the Standard Model prediction given by $\mathcal{L}_{SM}$ in future LHC runs can verify or disprove the current theories of the SMEFT.

3. Machine Learning

Using neural networks allows to learn a given phase space density generated using Monte Carlo techniques. Once the model is trained it is possible to generate a multitude of events at a very high rate. For example it took several days to produce 5.4 million events for the data set consisting of Z + multijets. Our baseline model is able to generate 5 million events in less than an hour. This chapter gives a short introduction to neural networks and is then progressing to the pipeline for the Bayesian neural network, allowing to computing expressive uncertainties for the generated events.

3.1. Introducing Neural Networks

First a basic neural network (NN) is set up. The idea is, that we feed the neural network an input and the neural network finds the transformation that maps the input layer $\vec{\mathbf{x}}$ to an output vector $\vec{\mathbf{y}}$. In the simplest form of linear layer:

$$\vec{y} = f(\vec{x}, \theta^T, \vec{b}) = \theta^T \cdot \vec{x} \ + \ b \tag{3.1}$$

Each input x_i is parametrisized by a weight θ_i then added a bias vector $\vec{b}$. During training, the neural net finds the optimal values for θ_i and the bias term b. A neural net consists of multiple transformations, which are summed up into a layer. In this case, the weights θ are given by a matrix. Figure 5 shows the basic set up for a neural net. The layers can be divided into three parts. The input layer, the output layer, and the so called hidden layers in between. *Deep neural nets* use multiple hidden layers to model complex distributions.

3.2. Training the Neural Network

The network has to find the best values for the parameter weights θ in order to create the most suitable output. In the best case, the output generated by the neural net is equal to the test data set. The optimal weights are found by minimizing the so called loss. The loss function is a tool to estimate an error between the desired output and the output given by the NN. There are many ways to construct a loss function according to each specific problem. Let y be desired output and $\hat{y}$ the output given by the NN. The loss is for example given by the mean squared error:

$$\mathcal{L}_{MSE} = \sum_{n=1}^{N} (y - \hat{y})^2 \tag{3.2}$$

The network learns by changing the weights θ according to the loss function. Most optimizers are derived from the *gradient decent* algorithm, which evaluates the gradient of the loss at the current state and then changes the weights in the direction of the steepest descent. Sophisticated optimizer like ADAM [24, 25] perfect the change of the weight according to the learning rate, which determines how much the weights change in one epoch.

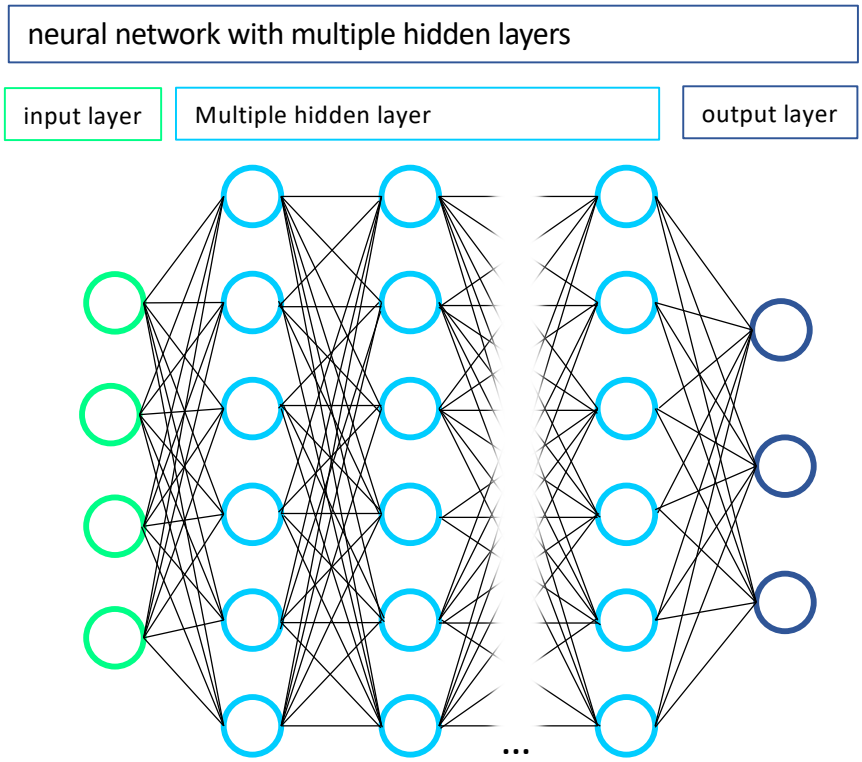

Fig. 5: Basic Deep Neural Net

3.3. Invertible Neural Networks

Introducing invertible neural networks[13] allows to model complex distributions as given by the LHC phase space densities. Mapping the complex LHC phase space densities via invertible transformations to a simpler distribution in the so called *latent space* allows for a simpler training process. Training the neural net in the forward direction and then for the event generation sampling from the latent space in the inverse direction. Let X be the LHC input data in the *phase space* and Z the *latent space*. During training X is mapped to Z. The idea behind it is to take a random variable $z \sim p_Z(z)$ from the latent space and perform invertible transformations on it. Let x be a random variable of our target density p_X in the phase space and a f_i bijective transformation.

$$x = f_1 \circ f_2 \circ ... f_N(z) \tag{3.3}$$

Let f be $f(z) = f_1 \circ f_2 \circ ... f_N(z)$. f is bijective as a composition of bijective transformations. Therefore we can map $f : Z \longrightarrow X \ , z \mapsto x$ with the density

$$p_X(x) = p_Z(z) \det \left| \frac{\partial f(z)}{\partial z} \right|^{-1} = p_Z(f^{-1}(x)) \det \left| \frac{\partial f^{-1}(x)}{\partial x} \right| \tag{3.4}$$

In order to have an effective generation, p_Z should be simple. A standard approach

is to choose a multivariate gaussian with mean value zero and a covariance as the identity matrix. Now f has to be flexible to perform nontrivial transformations but with a rather quick computable jacobian. This is fulfilled as we can decompose f in a set of simple bijective maps f_i (Eq 3.3). The jacobian is then given as :

$$\det \left| \frac{\partial f(z)}{\partial z}^{-1} \right| = \prod_{i=1}^{n} \det \left| \frac{\partial f_i(z)}{\partial z}^{-1} \right| \tag{3.5}$$

As f_i are simple mappings, each determinant is easily to compute. After training in the forward direction, the events are generated by sampling from the latent space and passing it through the inverse direction, allowing for a limitless generation of the events, as long as the local GPU allows it. As of today, computational boundaries are still a hurdle limiting the abilities of a neural net.

Coupling Layers

An INN composes multiple transformation maps into coupling layers [26]. The structure of a coupling layer is given in the following:

1. Split the input into two parts $x = [x_1, x_2], x_1 = (x_1, ..., x_d), x_2 = (x_{d+1}, ..., x_D)$
2. compute the weights θ through the NN using x_1
3. compute $y_{2i} = g_{\theta_i}(x_{2i})$ and set $y_1 = x_1$. f being an arbitrary invertible function
4. return the output y as y= $[y_1, y_2]$

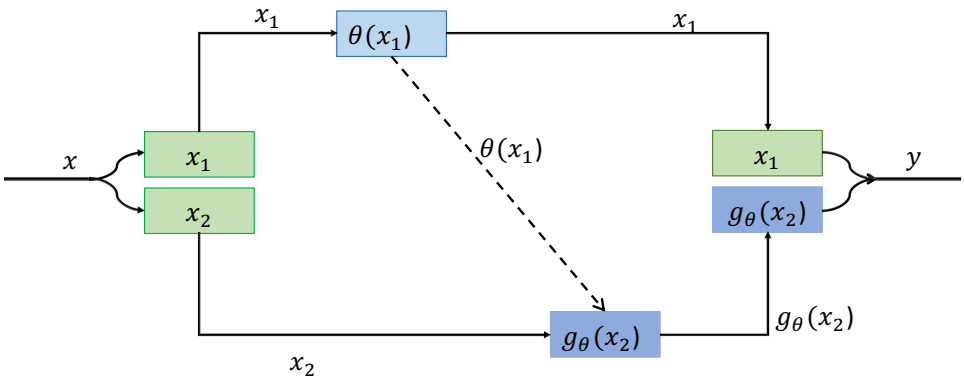

Fig. 6: Structure of Coupling layers

This process is illustrated in Figure 6 .Note that d is in most cases chosen to be $\frac{D}{2}$, but that does not have to be the case necessarily. Since $y_1 = x_1$ and y_1 is given by transforming x_1 element wise the determinant Jacobian of the coupling transform Φ becomes:

$$\det\left(\frac{\partial \Phi}{\partial x}\right) = \prod_{i=d}^{D} \frac{\partial g_{\theta_i}}{\partial x_i} \tag{3.6}$$

a lower triangular matrix. Fulfilling the requirements stated before, being fast to compute and being invertible in a single pass. To ensure that every point of the input data is being used, rotation matrices from the $SO(d)$ are introduced in between the coupling blocks. These are randomly generated.

Cubic Spline Blocks

Choosing the form of g for the coupling blocks is an important part of the training. The *affine coupling blocks* use additive or affine transformations.

$$g_{\theta i}(x_i) = \alpha_i \cdot x_i + \beta_i \tag{3.7}$$

Where $\theta_i = \{\alpha_i, \beta_i\}$. They are easy to invert however lack in flexibility. Moving to a more sophisticated block the *cubic spline block*[26]. In this case $g_{\theta i}$ are composed of monotonic increasing cubic polynomials. They are still easily invertible but enhance the flexibility in our model. $g_{\theta i}$ map [0,1] $\mapsto$[0,1] [26] ensuring the first proposition of being invertible. During training the parameters θ_i are optimized by maximizing the Log Likelihood .Which is equivalent to minimzing the *Kullbeck Leibler Divergence*. Given a dataset of N samples, the loss function is given by:

$$\mathcal{L}_{ML} = -\sum_{i=1}^{N} \log(p_X(x_i;\theta)) \tag{3.8}$$

Let f denote the overall mapping. We derive

$$\mathcal{L}_{ML} = -\sum_{i=1}^{N} \log(p_Z(f^{-1}(x_i;\theta)) + \log\left|\det \frac{\partial f^{-1}(x_i;\theta)}{\partial x_i}\right| \tag{3.9}$$

5tt

3.4. Uncertainties via Bayesian INNs

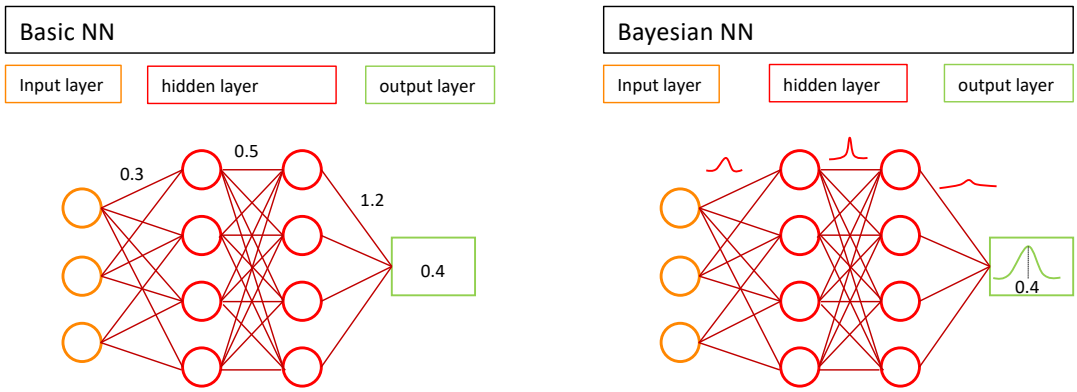

Fig. 7: Comparison between a Neural Net and a Bayesian Neural Net

INNs showed promising results on various different data sets. However, once the network is fully trained we only get access to a statistical uncertainty given by the square root of events in a given bin. The INNs give us an understanding of the properties of the underlying data but don't provide a tool to compute expressive uncertainties caused by the network itself. Of course one could train multiple INNs and compare the results to get an understanding of the uncertainties and fluctuations between different trainings, however this is time consuming and computationally expensive. Therefore, we propose a different approach using Bayesian INNs. This chapter gives an intuitive approach to Bayesian Neural Networks (BNN) and then evolves into the Bayesian Invertible Neural Network set up.

3.4.1. Bayes Neural Network

To get an intuition for what a Bayesian neural network [19, 27–29] actually is, let us consider a very basic neural net for a linear layer as described in chapter 3.1. Now θ_i are the parameters learned during training. In our normal neural network we would get discrete values for θ_i. However, going to a Bayesian setup we assume that each parameter θ is not a discrete value but rather given by a probability distribution. In general the distribution p_θ could have any shape. Since this setup calls for over 30 million trainable parameters it is close to impossible to learn complex distributions for all of them. This is why we propose that each weight θ is modeled by a gaussian distribution $\theta_i \sim \mathcal{N}(\mu_i, \sigma_i)$. For every θ_i the network learns a mean value μ_i and a standard deviation σ_i. Going back to equation 3.1 our output y is no longer a discrete value but also a given by a probability distribution $\mathbf{y} \sim f(\vec{x}, \vec{\theta}, b)$. The uncertainties propagate through the neural network giving us a probabilistic output. As for simple problems like the linear regression, Bayesian neural nets provide a direct estimation in the output.

Let $\mathcal{D}$ be a dataset consisting of N pairs of observations (x_i, y_i).
$\mathcal{D} = \{(x_1, y_2), ..., (x_N, y_N)\}$ and f_θ the mapping between x_i and y_i. Placing a gaussian prior over the weights with some observation noise gives us:

$$\theta \sim p(\theta) \tag{3.10}$$
$$y_i|\theta, x_i \sim p(y_i|\theta, x_i) \tag{3.11}$$

In most cases the posterior $p(\theta|\mathcal{D})$ is highly intractable. For a BNN a variational inference [12] can approximate the posterior with a tractable family of distributions $q_\phi(\theta)$. The parameters θ are optimized by minimizing the *Kullback-Leibler-Divergence*:

$$\min_\phi \mathrm{KL}(q_\phi(\theta), p(\theta|\mathcal{D})) \tag{3.12}$$

Since the posterior is intractable, we use Bayes Theorem to reformulate equation 3.12

$$\mathrm{KL}(q_\phi(\theta), p(\theta|\mathcal{D})) = -\int d\theta q_\phi(\theta) \log \frac{p(\mathcal{D}|\theta)p(\theta)/p(\mathcal{D})}{q_\phi(\theta)} \tag{3.13}$$

$$= -\int d\theta q_\phi(\theta) \log p(\mathcal{D}|\theta) - \int d\theta q_\phi(\theta) \log \frac{p(\theta)}{q_\phi(\theta} + \log p(\mathcal{D}) \tag{3.14}$$

Solving for $\log(p_T)$, we find a lower bound:

$$\log p(\mathcal{D}) = \mathrm{KL}(q_\phi(\theta), p(\theta|\mathcal{D})) + \int d\theta q_\phi(\theta) \log p(\mathcal{D}|\theta) - \mathrm{KL}(q_\phi(\theta), p(\theta)) \tag{3.15}$$

$$\geq \int d\theta q_\phi(\theta) \log p(\mathcal{D}|\theta) - \mathrm{KL}(q_\phi(\theta), p(\theta)) \tag{3.16}$$

Maximizing this evidence lower bound (ELBO) then is equivalent to minimizing equation 3.12, giving us as the objective without the intractable posterior:

$$\mathcal{L}_{\mathrm{ELBO}} = \sum_{i=1}^{N} \langle \log p(y_i|\theta, x_i) \rangle_{\theta \sim q_\phi(\theta)} - \mathrm{KL}(q_\phi(\theta), p\theta) \tag{3.17}$$

Turning the inference problem into a optimization problem, allowing us to use gradient decent techniques.

3.4.2. Bayesian INN

Increasing the complexity to high dimensional data sets makes it impossible to trace the uncertainties propagating through the network. Replacing every deterministic function g_θ as described in subsection 3.3 to a probabilistic function and placing a prior over the weights with some observation noise

$$\theta \sim p(\theta) \tag{3.18}$$

gives us the generative pipeline

$$x|\theta \sim p_X(x|\theta) = p_Z(f^{-1}(x;\theta)) \det \left| \frac{\partial f^{-1}(x;\theta)}{\partial x} \right| \tag{3.19}$$

Combining equation 3.9 and 3.17 we can derive the optimization Problem:

$$\mathcal{L} = \sum_{i=1}^{N} \langle \log p_X(x_i|\theta) \rangle_{\theta \sim q_\phi(\theta)} - \mathrm{KL}(q_\phi(\theta), p(\theta)) \tag{3.20}$$

$$= \sum_{i=1}^{N} \langle \log(p_Z(f^{-1}(x_i;\theta)) + \log \left| \det \frac{\partial f^{-1}(x_i;\theta)}{\partial x_i} \right| \rangle - \mathrm{KL}(q_\phi(\theta), p\theta) \tag{3.21}$$

Every part of the lossfunction is by design easy to compute.

Translating equation 3.19 into practice means, we generate events in the inverse direction. The events generated depend on θ. In the normal INN case, we would generate events knowing each weight θ_i as a fixed value. Now for every θ_i we draw a number from the learned normal distribution $\mathcal{N}(\mu_i, \sigma_i)$. Each giving a slightly different output. First we generate a given number of events and bin the events into a histogram. This process is done multiple times, for example 50 times. We than build the mean value of these histograms and calculate the standard deviation. Huge deviations in the parameter space lead to bigger uncertainties. It could be seen as equivalent to training 50 networks at the same time and comparing the output.

4. Testing the BINN on LHC Events

4.1. Baseline

This chapter is about generating events of a realistic data set as described in chapter 2. Using BINNs involves a lot of fine tuning regarding the hyperparameters. The BINN is a lot more reactive to changes in the hyperparameters compared to the INN. Although I will not discuss every hyperparameter, a short overview of the hyperparameters used for the baseline is shown in table 1. These are the parameters for the multijet data set as it was the main focus of this thesis. The hyperparameters for the Z+ multijet data set remain the same except for the number of trainable parameters.

Hyperparameter	Value
learning rate	2e -5
optimizer	ADAM [24]
bin number	240
number of blocks	20
internal size	265
layers per block	6
learning rate scheduler	one cycle [30]
coupling block	cubic[26]
batch size	1024
number of trainable parameters	30 Million

Tab. 1: Hyperparameters for the BINN

Like the INN the BINN has no difficulties learning the p_T distributions 8 and also the η distributions look reasonable. Because the BINN models the distribution so effectively, the uncertainties are rather small. Looking at the 4^{th} row each plots in Figure **??** we can observe that the relative error is rather small. The relative error is small for bins with a large number of events, like in the bulk of the p_T or the peak in the η distributions. The relative error increases, as one moves to areas where the statistic becomes rather low like in the tails of the distributions. The data set fluctuates more in these regions and therefore the generated events fluctuate more. The BINN does exactly what it should do, increasing the uncertainties. However, the BINN has difficulties in other distributions such as the $\Delta\Phi_{i,j},\ i,j \in \{1,2,3\}\ i \neq j$ 9. The network has trouble learning the smaller peaks and just flattens the curve. This could not be improved further with hyperparameter tuning. Looking at the relative errors, they do not seem to be affected by the failure of the network. The error does not increase, even though the performance lacks in multiple areas. This could also be observed in the $\Delta\eta_{i,j}$ distributions. The network has trouble learning the dip in the middle, but is sure that the parameters are right, leading to small relative uncertainties. Which was the expected behaviour since the model is not

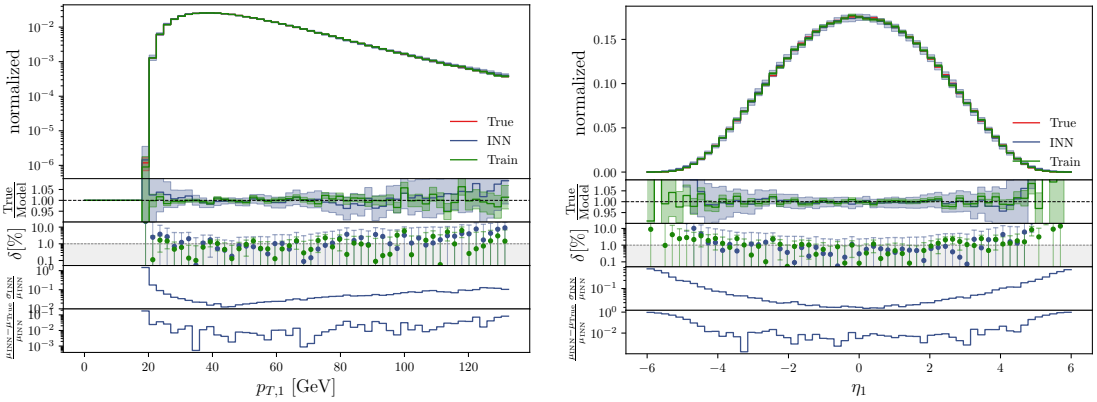

Fig. 8: BINN Baseline for 3 jet events. Top row shows the Histogramm, comparing the BINN, the true data and the test Data. Below is the ratio between the true data and the data generated by the BINN. The errorbars for the true data are given by the squareroot of the Number of events per bin. The fourth row shows the relative error and the last row a normed deviation between the true and model distribution

able to learn these observables correctly in the first place using an INN.
Looking at the correlations between two observables shows that the BINN has a lot of room for improvement (Figure 10). Especially the topological problems like the "hole" like distributions highlight the weaknesses of the BINN. Since the focus is on the properties of the uncertainties, we declared the baseline as good enough to study the uncertainty quantities further. Working with this data set allowed to get an understanding of the hyperparameters.

Similar problems occur in the second dataset. Due to it consisting of of three different types of events, the additional challenge to distinguish the different jets adds to the difficulty and complexity of the model. The network architecture is slightly different since one network is trained to generate μ_1, μ_2, j_1 and is given the number of jets as a condition. For each additional jet, another network is trained. It is conditioned on the training observables of the previous networks and the number of jets. Essentially it could be seen as training 3 networks at once, one for each possible number of final state particles. The network learns the p_T distributions shown in Figure 11 very well. Going to more difficult distributions like the $\Delta\phi_{i,j}$ shown in Figure 12 shows, that the network has again difficulties getting the Distributions right. Figure 12 shows the effect of low generated statistics well. When the events are generated, the ratio between the different number of events for each number of final state particle should be equivalent to the ratio given by the

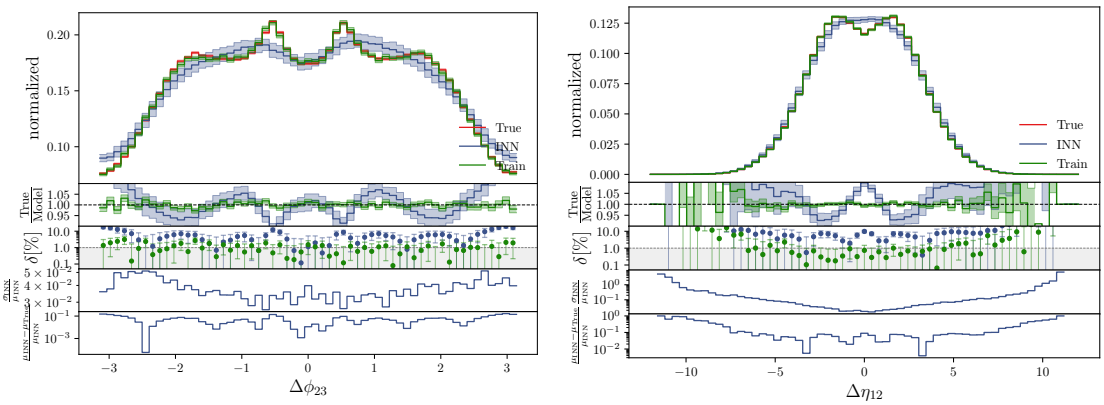

Fig. 9: BINN Baseline for 3 jets one dimensional observables

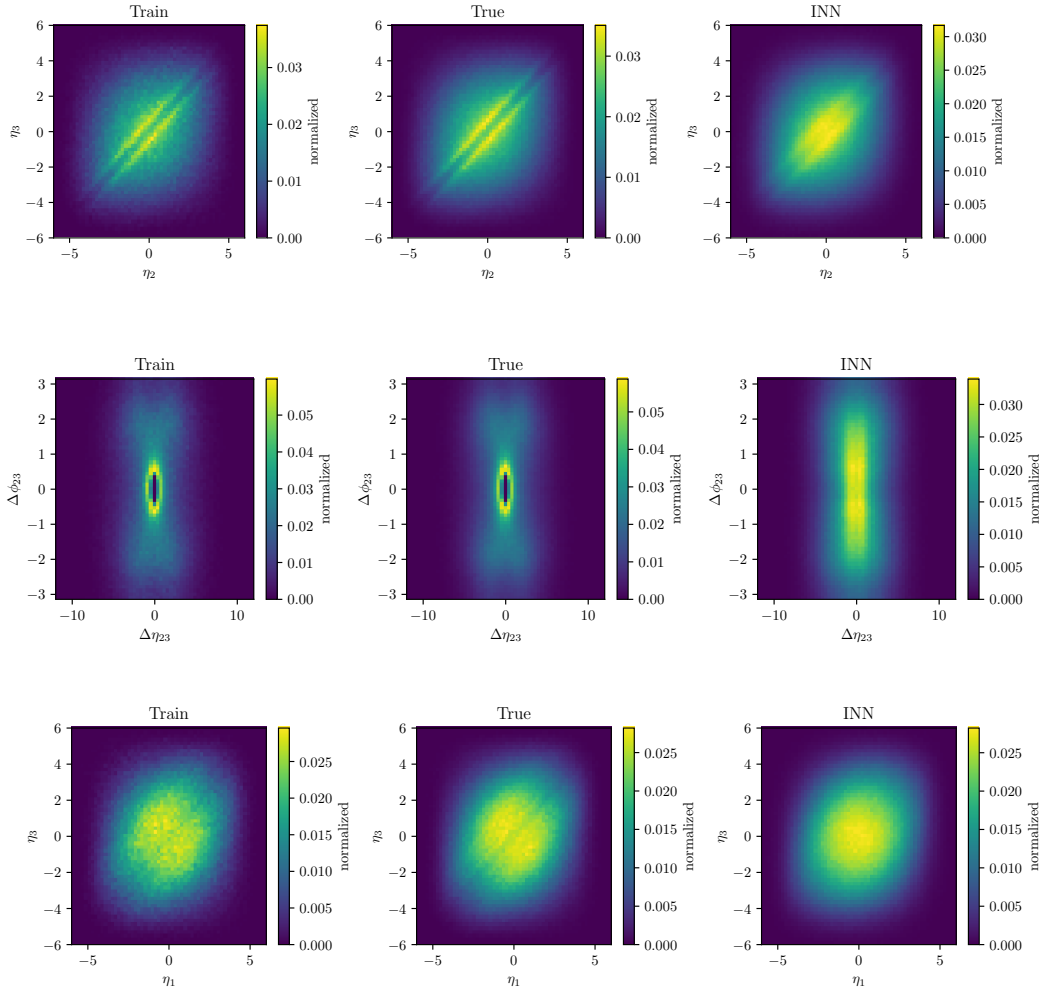

Fig. 10: BINN jets baseline correlations

dataset. Since we only generate 100000 Events this means we generate 74 thousand for the 3 particle final state, 20 thousand for the for the 4 particle final state and only 5.5 thousand for the 3 particle final state. The fluctuations are stronger for

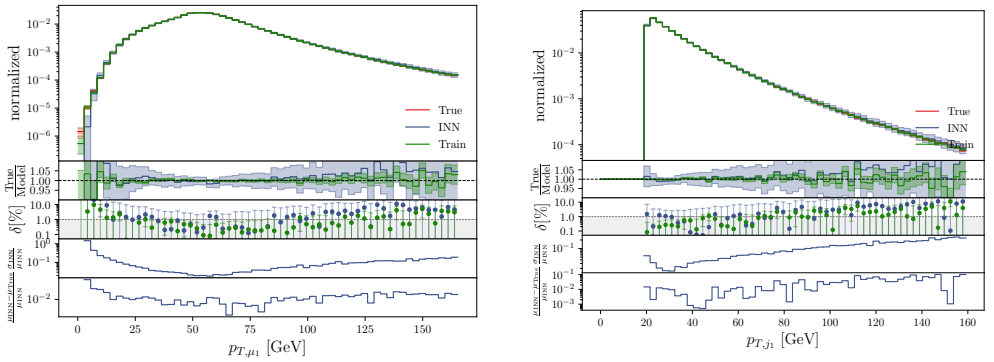

Fig. 11: Z + jets Baseline p_T Distribtutions for the 3 particle final state

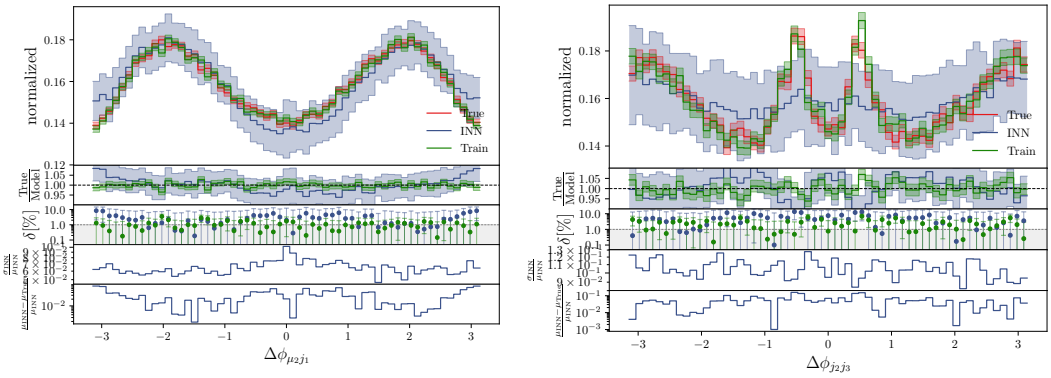

Fig. 12: Z + jets Baseline $\Delta\phi_{i,j}$ distributions. *left: $\Delta\phi$ between the first jet and the first muon for the 4 particle final state. right: $\Delta\phi$ between the second and third jet for the 5 particle final state.*

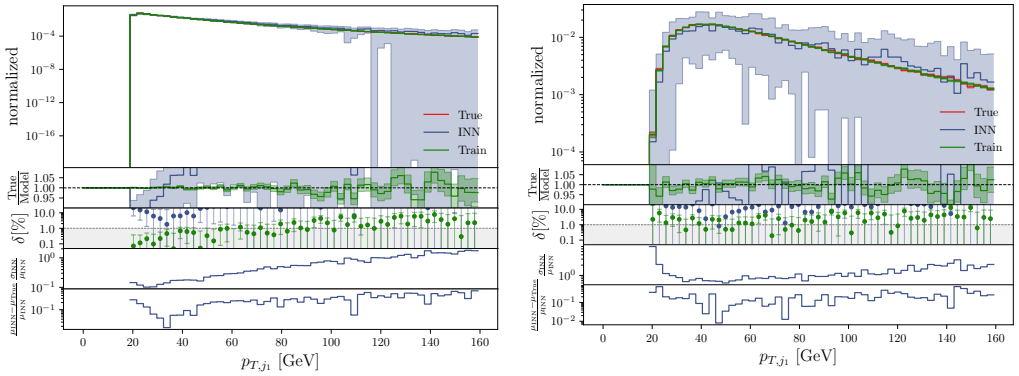

Fig. 13: Z + jets baseline p_T distributions with 1000 generated events. *left: the p_T distributions for the first jet in the 3 particle final state. right: the p_T distributions for the first jet in the 5 particle final state.*

lower statistics and therefore the uncertainties increase. To take things one step further, Figure 11 displays the p_T distributions for only 1000 generated counts. Especially the plot on the left for the 3 particle final state shows that the tails fluctuate more when compared to the tails. This effect is even more drastic in the plot on the left in Figure 13. This behavoiur is exactly what we would expect.

In conclusion once the BINN learns the distributions, the generated uncertainties are reasonable. However they are not a tool to assess the networks performance, as the network can be far off , but being so confident that it learned the right distribution, that the uncertainties get rather small. Seemingly, precision networks have to be the foundation for expressive uncertainties. A well learned baseline is crucial to quantify the uncertainties.

5. Quantifying Uncertainties

Building up from this foundation, the question arises which technicalities have an impact on the uncertainties. The next chapter is about quantifying the uncertainties. Every result in the following chapter was made using the 3 jet data set.

5.1. Error Decomposition

5.1.1. Theoretical Proof

According to the law of total variance, it is possible to decompose the uncertainty given by the BINN into two parts. Looking at a single bin in a histogram of some marginalized distribution from the INN output. We are interested in the number of events n in that bin when N events are sampled using the INN. Let ρ_ω be the probability that an event is inside the bin for given network weights $\omega \sim q(\omega)$. Then n has a binomial distribution,

$$p(n|\omega) = \binom{N}{n} \rho_\omega^n (1-\rho_\omega)^{N-n} \tag{5.1}$$

The expectation value and variance are given by

$$\langle n \rangle_\omega = \sum_{n=0}^{N} p(n|\omega) n = N\rho_\omega \tag{5.2}$$

$$\mathrm{Var}(n)_\omega = \langle n^2 \rangle_\omega - \langle n \rangle_\omega^2 = N\rho_\omega(1-\rho_\omega)\ . \tag{5.3}$$

The standard deviation of n can be decomposed as follows:

$$\begin{aligned}
\sigma_{\mathrm{tot}}^2 &= \langle (n - \langle n \rangle)^2 \rangle && (5.4)\\
&= \int \mathrm{d}\omega\ q(\omega) \langle (n - n\langle n \rangle)^2 \rangle_\omega && (5.5)\\
&= \int \mathrm{d}\omega\ q(\omega) \langle n^2 - 2n\langle n \rangle + \langle n \rangle^2 \rangle_\omega && (5.6)\\
&= \int \mathrm{d}\omega\ q(\omega) \left[\langle n^2 \rangle_\omega - 2\langle n \rangle_\omega \langle n \rangle + \langle n \rangle^2 \right] && (5.7)\\
&= \int \mathrm{d}\omega\ q(\omega) \left[\langle n^2 \rangle_\omega - \langle n \rangle_\omega^2 + (\langle n \rangle_\omega - \langle n \rangle)^2 \right] && (5.8)\\
&= \sigma_{\mathrm{stoch}}^2 + \sigma_{\mathrm{pred}}^2 && (5.9)
\end{aligned}$$

with

$$\sigma_{\mathrm{stoch}}^2 = \int \mathrm{d}\omega\ q(\omega) (\langle n^2 \rangle_\omega - \langle n \rangle_\omega^2) \tag{5.10}$$

$$\sigma_{\mathrm{pred}}^2 = \int \mathrm{d}\omega\ q(\omega) (\langle n \rangle_\omega - \langle n \rangle)^2\ . \tag{5.11}$$

To get an estimate of σ_{tot}, we draw samples (ω, n) from the underlying distributions and calculate the standard deviation of the n. In practice, this can be done by

drawing weights ω, generating N events using the INN with those weights. Then we can make a histogram for the observable of interest and get n for the bin of interest from it. This process is repeated and the standard deviation is calculated. Therefore, σ_{tot} can be identified with the usual BINN uncertainty estimate. However, this is only true when the latent space samples of the INN are drawn again for each weights.

We can derive an analytical result for σ^2_{stoch} using equation 5.3,

$$\sigma^2_{\text{stoch}} = \int \mathrm{d}\omega\, q(\omega)(\langle n^2\rangle_\omega - \langle n\rangle^2_\omega) \tag{5.12}$$

$$= \int \mathrm{d}\omega\, q(\omega) N\rho_\omega(1-\rho_\omega) \approx \int \mathrm{d}\omega\, q(\omega) N\rho_\omega \tag{5.13}$$

$$= \int \mathrm{d}\omega\, q(\omega)\langle n\rangle_\omega = \langle n\rangle\ , \tag{5.14}$$

where we assumed $\rho_\omega \ll 1$ which is reasonable for a small enough bin. This is the Poisson error from binning and matches our observations for the behaviour of the BINN uncertainties.

5.1.2. Observed Decomposition

Identifying σ_{tot} as the error given by the BINN, and σ_{stoch} as the statistical Poisson error, we observed the error decomposition, σ_{pred} is inaccessible due to the complexity of the model. Using again just subsets of the data, we observed the error decomposition. The data was split into fractions as described earlier. We run 3 trainings per fraction, using a randomised split, so every training used a different fraction of the given data using the same length. For each run the average error was taken and the standard deviation calculated. σ_{pred} was calculated taking the quadratic difference:

$$\sigma^2_{pred} = \sigma^2_{tot} - \sigma^2_{stoch} \tag{5.15}$$

σ_{stoch} tends to be uncorrelated using different training data lengths as expected since it only depends on the number of generated events, whereas σ_{tot} and therefore inevitably σ_{pred} seem to decrease with an increasing number of training events. Having a look at the error bars in Figure 14 one can clearly see that the errors fluctuate pretty heavily when a limited training data is used. The trend is visible however there is no direct connection besides a decrease.

5.2. Effect of the Number of Training Epochs

Theoretically speaking, if we would have a perfect model, it can learn any distribution perfectly as long as it is trained long enough. The question is only how long it may take. The time could range between hours and years. Unfortunately, our model is not perfect and training it for hundreds of hours requires resources, such as time and computational capacity, which are usually unavailable. Therefore, we

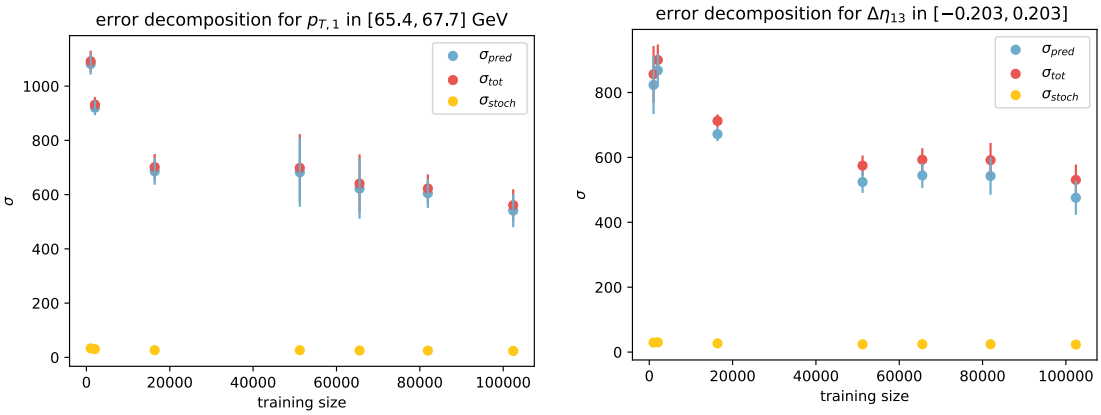

Fig. 14: Error decomposition dependant on the number of training events for an arbitrary bin

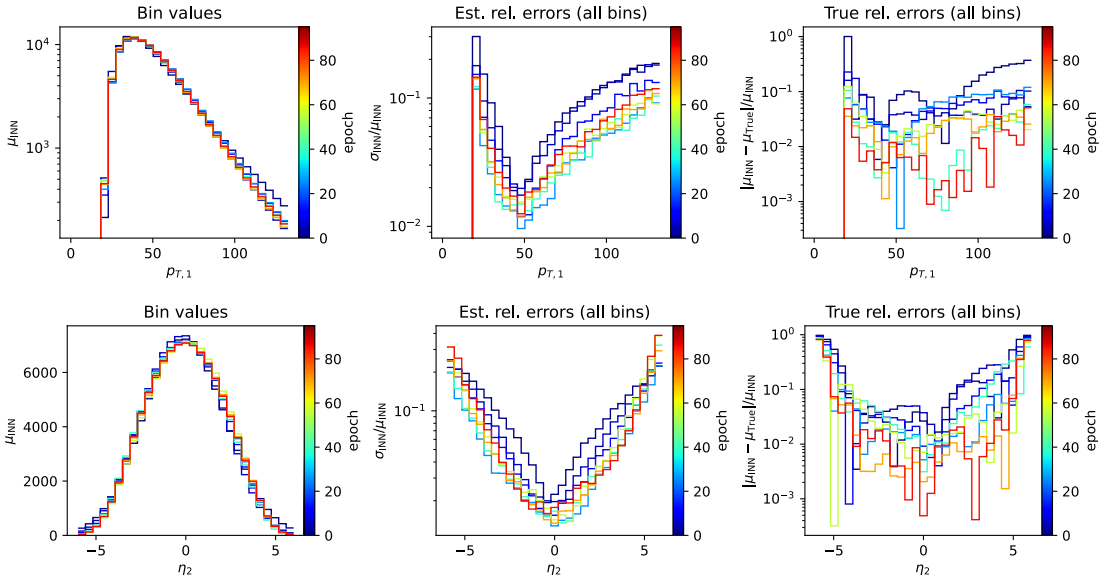

Fig. 15: Uncertainties and learned Distribtions dependent on the training duration

tried to improve our training in a shorter period of time. For reference, in this set up 10 epochs take approximately an hour. We aimed to study the effect of the number of epochs and equivalently the effect of the training duration on the uncertainties. As the BINN improves the performance with the training duration we would expect a decrease in the uncertainties. To study this behaviour we trained our model for 100 epochs, saving the data after every 10 epochs. As expected, the training performance increases and the uncertainties decrease with an increase of the number of epochs (Figure 15). Note that it is astonishing that the BINN learns the shape of the uncertainties rather quick. (After 1 Epoch).

5.3. Effect of the Training data length

The data generated by SHERPA consists of 5.4 million events. This data is was split into a training and a test set, using 82 % of it as training data for the BINN.

To observe the effect of a limited training data, the data was cut and shortened, where we run multiple trainings using fractions of the initial training data. The first fraction being equal to one batch size consisting of 1024 data points, then going up by powers of two.

$$N_i = 1024 \cdot 2^i \tag{5.16}$$

N is the number of training points, i $\in\{0,9\}$ A decrease in the number of training points leads to a decrease in the relative error. Additionally, more training data leads also to a improvement of the learned distribution.

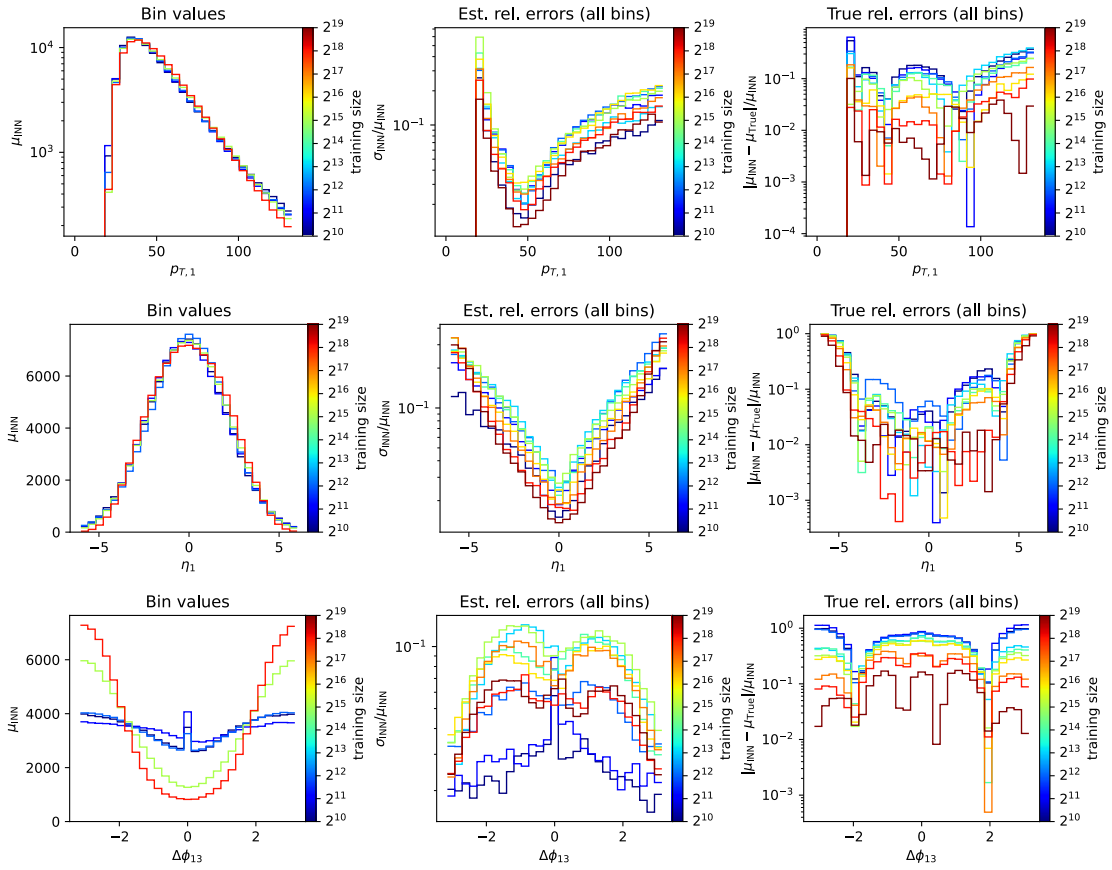

Fig. 16: Uncertainties and learned Distribtions dependent on the training statistic

Figure 16 shows a decrease in the errors. It is somehow amazing that the BINN is able to learn the distributions of the p_{Ti} and η_i so well even for very small data sets of 0.02 % of the original dataset. However more difficult distributions like the $\Delta\Phi_{i,j}$ show how important a sufficiently large data set is. In the last row of Figure 16, it is clearly visible that the network can not learn the density correctly. Because of that the corresponding uncertainties are rather small. Once the network has enough datapoints to outline the correct density, the errors increase as they now have a meaning. Then with a increase of data points, the networks performance improves and the uncertainties decrease further. In the limit of a endless training duration and a massive data set the uncertainties should should result in a minimized statistical uncertainty.

5.4. Effect of the generated statistic

Getting uncertainties beyond the statistical error is the reason to introduce BINN in the first place. Plotting the mean value against the uncertainty displays the scale of the uncertainty. If we were only dealing with statistical uncertainties from binning

$$\sigma = \sqrt{(\mu)} \tag{5.17}$$

every data point would be found on this curve, as seen in Figure 10 representing the black line. For a better visualization a double logarithmic scale was chosen. The points are clearly above the black line representing equation 5.17. In an attempt to quantify the uncertainties, a power law fit was chosen. But as the majority of the points are not well represented by the fit, this proposition can be disregarded. The

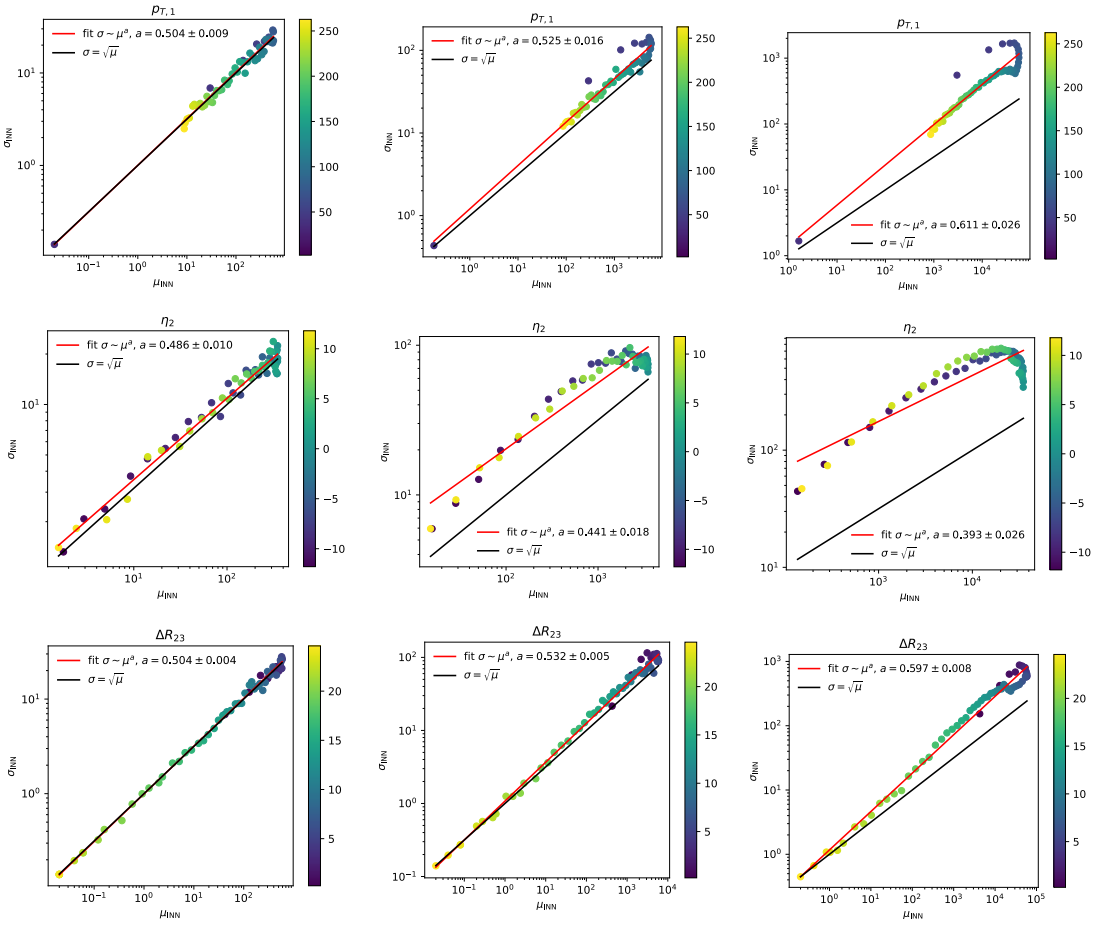

Fig. 17: Effect of the number of generated uncertainties on the uncertainties. *left column 10k generated events, middle column 100k, right column 1 million*

mean value of events per bin also influences the uncertainties. Generating 10k, 100k or 1 million events has a significant influence on the uncertainties. Starting at 10k generated events, most of the uncertainties behave like the statistical Poisson error. Moving to higher values of generated events shows that the uncertainties are not well modeled by the Poisson error and must have an additional contribution. The error increases with increasing generated statistic, suggesting there has to be some

sort of correlation between the uncertainties given by the BINN and the number of events per bin as expected in Chapter 5.1.

6. Systematic Uncertainties

The main goal of simulations is to show the agreement of theory and nature. Going back to chapter 2, let us assume we want to prove or disprove the expansion terms that we calculated for the S matrix elements. Of course, the contributions are only relevant for high energy physics, like the creation or decay of the Higgs boson. However, this is just a semi realistic toy model exaggerating the theory, and choosing arbitrary parameters along the way. Only the general idea is motivated by real assumptions and predictions given by SMEFT.

Motivated by EFT let us assume we have a new theory with some kind of additional term in the p_T distributions for high energies. p_{Told} is the distribution now known. The new theory proposes $p_T = p_{Told} + f(p_T)$ with f being some sort of transformation. In the following chapter we want to treat this additional term as a systematic uncertainty and build a tool to train a model conditionally on the function.

Weighted Events

The first step is to go from unweighted events to weighted events [31]. Every event has a given weight now and for the unaltered distribution set to one. This leads to an additional term in the loss function.

$$\mathcal{L} = \sum_{i=1}^{B} \hat{w_i} \cdot (\frac{{z_i}^2}{2} - J_i) \tag{6.1}$$

Here z_i are the latent space vectors, J_i are the corresponding logarithms of the jacobian determinant and B is the batch size. The weights $\hat{w_i}$ are normed:

$$\hat{w_i} = w_i \cdot (\sum_{i=1}^{B} w_i)^{-1} \cdot B \tag{6.2}$$

We assume that the weights w_i can vary with $log(p_T)$ for example of the first jet. The weights are then:

$$w = 1 + a \cdot \log(p_T, j_1) \tag{6.3}$$

The variation is given by the parameter a. If a would be known exactly we could simply reweight our distribution and there would be no contribution to the error given by the BINN. Therefore, we assume that a is unknown to some degree. We sample a from a normal distribution of an arbitrary standard deviation σ and the mean value $\mu = a$.

6.1. Improvements through Manipulated Weights

In the beginning of the training a, and therefore the weights, were computed and used as train weights. To get an increase in the uncertainty, the weights were

computed and again drawn from of a normal distribution.

$$w \sim \mathcal{N}(\mu_w, \sigma_w) \tag{6.4}$$

$$\mu = 1 + a \cdot \log(p_T) \tag{6.5}$$

Note that this was not necessary at all and more of a bug but it lead to a happy

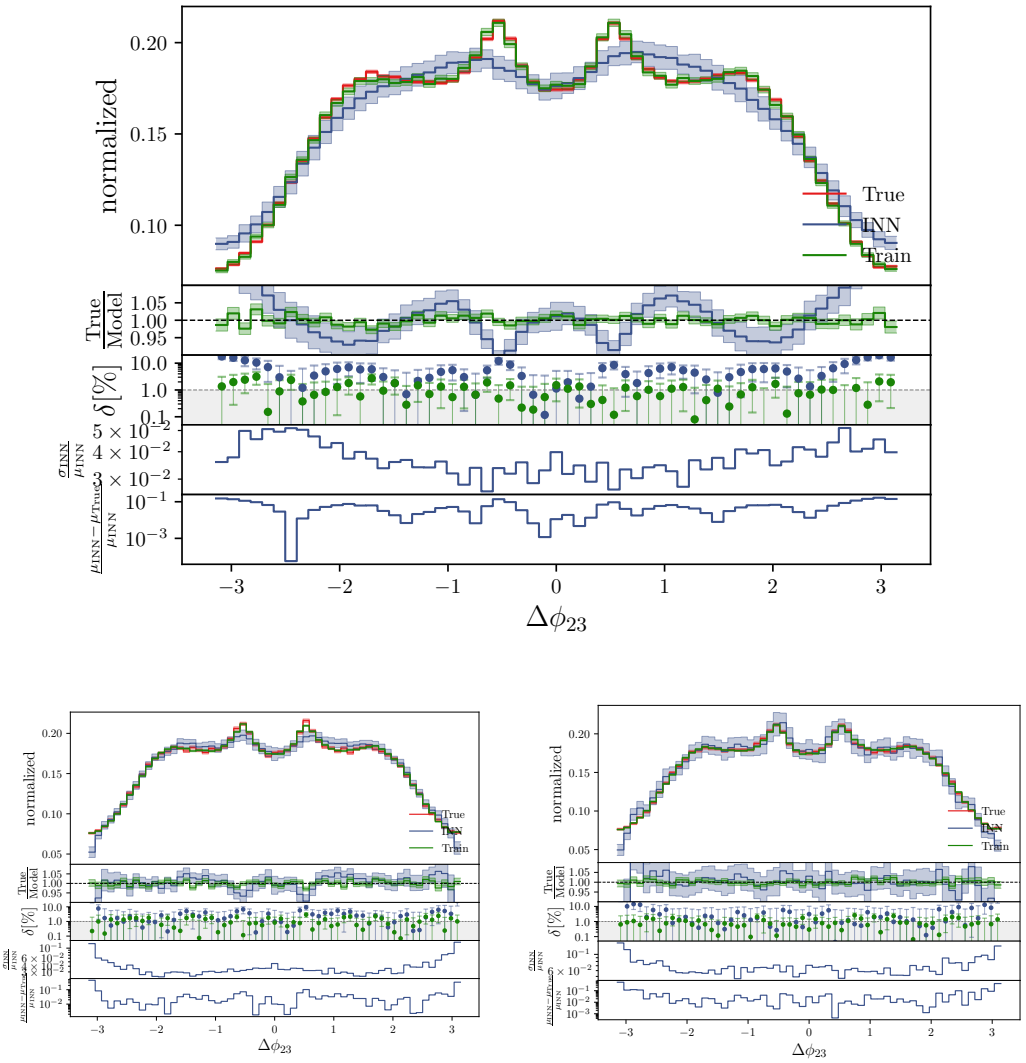

Fig. 18: Top: Unweighted distribution for $\Delta\phi_{2,3}$ Bottom row: increasing the performance via introducing noise on weights for the three jet dataset. Left a=0.02; right a=1

accident. Most of the noise averages because of the high statistic. Therefore, the uncertainties in the tails of the p_T distributions do not change at all. Unexpectedly, we observed something interesting. As shown in chapter 4 the BINN has trouble learning the $\Delta\Phi_{2,3}$ distribution, as it flattens the peaks. If we introduced the noise described in equation 6.5 the BINN performance in $\Delta\Phi_{2,3}$ increased significantly without destroying the other observables.The network seems to learn even the "holes" in the correlations (Figure 19). Former research in image processing has

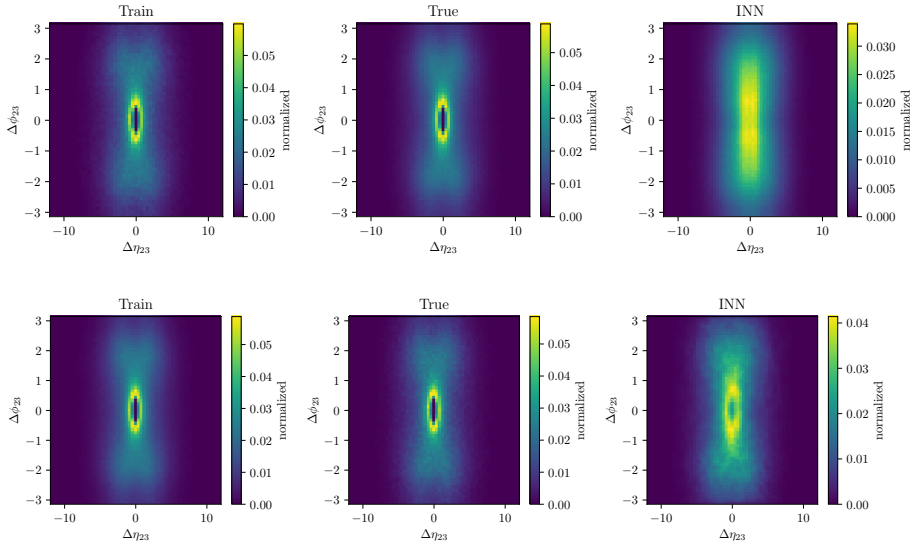

Fig. 19: top row showing the Baseline correlations and the bottom row the correlations after introducing noise on the weights

shown that INNs profit from noise of the input data. However this improvement could only be observed in this specific data set. Trying the same setup in the Z plus multijet data set showed no improvement at all. Noting that this might be because the $\Delta\Phi_{2,3}$ are not learned well at all without the weights (Figure 12. Therefore,this manipulation should be repeated once the data set is optimized. Another reason why we couldn't observe an effect in the Z + mulitjet Dataset might be because of the network different architecture. As the Network consists of 3 smaller networks, one for each additional jet. It would be interesting to try the same setup on a different dataset, of similar complexity, using the same architecture as for the 3 jet dataset. Due to time limitations this remains an open question on how noise can improve the network perfomance. In general it showed that a specific manipulation of the weights can lead to an increase in the performance. Instead of just a random manipulation via introducing noise the performance can be increased through concrete manipulation, pushing the weights in a distinct direction as proposed by Theo Heimel [32].

6.2. Introducing Systematic Uncertainties through a Conditional BINN

In the next part we want to construct a tool to allow for a variation in the p_T tails after the training is completed. Using a *conditional neural network* [33] allows us to train different distributions at once. As an input we do not only give the INN the latent space variables, but also an external parameter . The set up is illustrated in Figure 14. For this process we moved on to the second data set consisting of a decaying Z boson and a variable jet number. The weights differ by

$$w = 1 + a \cdot f(p_T) \tag{6.6}$$

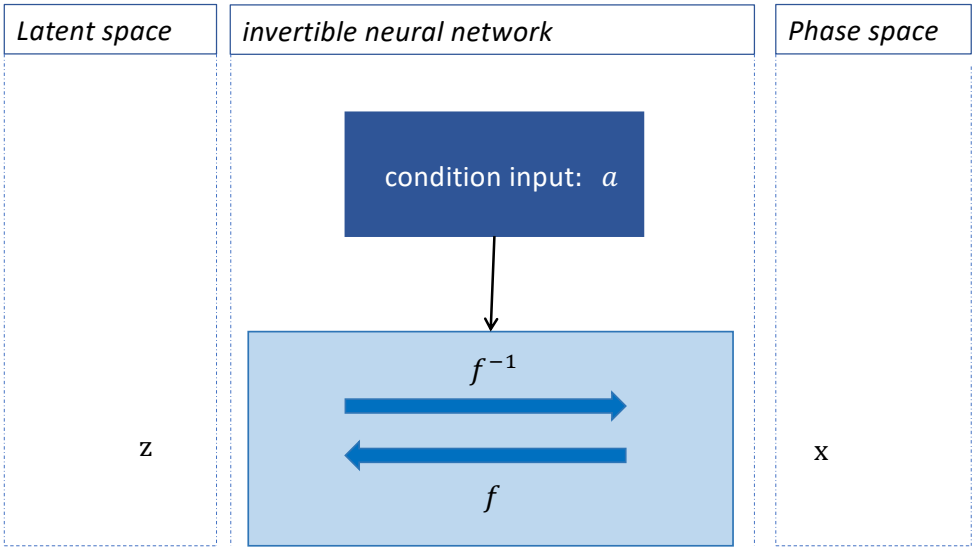

Fig. 20: Conditional Neural Network

where f is a more or less arbitrary function motivated by EFT and a a nuisance parameter and the condition input. During training we learn different distributions by varying a. In the first approach we drew a from a uniform distribution. This was unsuccessful, since the INN could not distinguish between the different curves parameterized by a. To make a drastic change a was chosen to be either 1 or 0. The network had to learn two distributions. The original one where the weights are just one and the other, were the weights are given by $w = 1 + f(p_T)$. When the events were generated we drew a from a normal distribution of mean value μ either being 0 or 1 for each histogram. This gives us slightly different outcome for each histogram and adding a systematic error to the uncertainty. During training, some functional of f has to be chosen. The goal is to keep the bulk of the p_T unvaried and only change the weights in the p_T tails. Motivated by EFT, either a term with the logarithm of p_T or with a quadratic expansion. Looking for a function that is approximately zero for low values of p_T and increases when going to higher values of p_T. Additionally, we want the function f to be continuous. Keeping in mind, even though EFT is the motivation behind this tool, it is still a toy example with arbitrary functions and arbitrary values. The best guess is a quadratic function.

$$f = 0.0003 \cdot (p_T - 15)^2 \tag{6.7}$$

For comparison I also tried a function which is

$$f(x) = \begin{cases} p_T & p_T \leq 73.9 \\ \log(p_T) & p_T \geq 73.9 \end{cases} \tag{6.8}$$

The comparison between equation 6.8 and equation 6.7 is shown in Figure 21. Equation 6.8 increases faster than equation 6.7. However 6.8 is not physical since it is not differentiable at $p_T = 15$. Of course, we could model a more complex function where the weights in the bulk remain one and increase in the tails of the

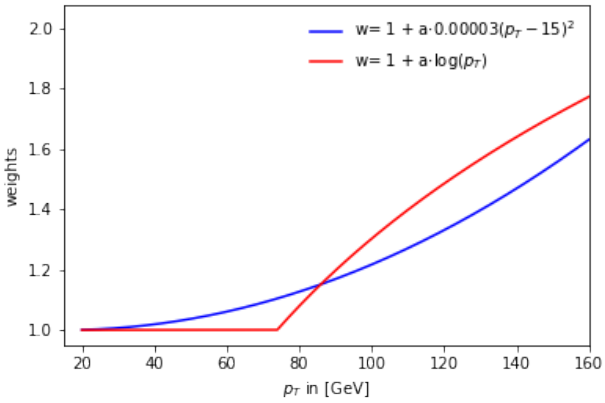

Fig. 21: Comparison of different weight distributions

p_T. However we want to stay semi realistic to EFT predictions. The results for the conditional BINN are shown in Figure 22. The BINN is trained once per function and then evaluated 2 times, once for $a = 0$ and once for $a = 1$.

When the events were predicted, a was drawn from a normal distribution. As for the parameter weights θ each sampling leads to a slightly different outcome, increasing the standard deviation. In Figure 23 shows the comparison between the relative uncertainty comparing a training without a conditioning and 2 trainings using different conditioning. While the events were predicted, a was chosen to be 0. The outcome should be the normal unweighted distribution. Showing that the conditional theory uncertainty leads to a contribution to the uncertainty given by the BINN. This is now a successful tool to alter the uncertainties and add a systematic uncertainty. Additionally we built a mechanism that allowed us to test different nuisance parameters for a new theory.

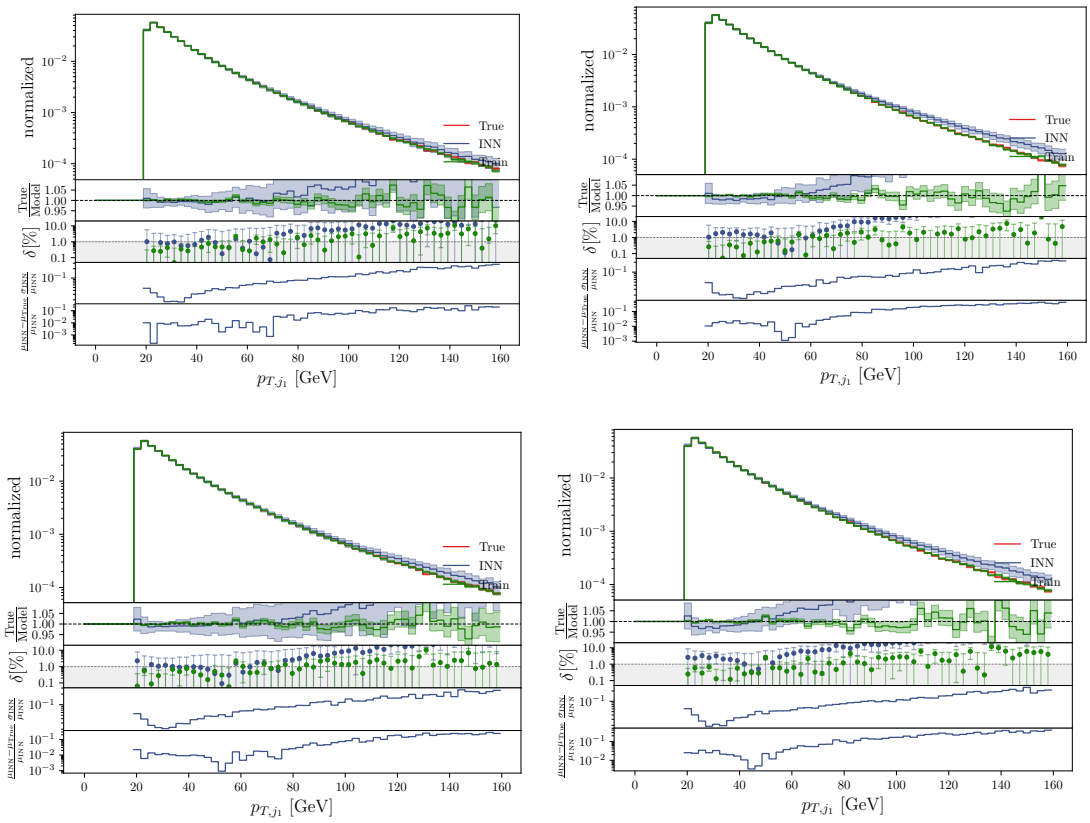

Fig. 22: Comparison between predictions for different values of a. left: a=0; right: a=1 ; top $w = 1 + a \cdot \log(p_T)$, bottom $w = 1 + a \cdot 0.0003 \log(p_T)$

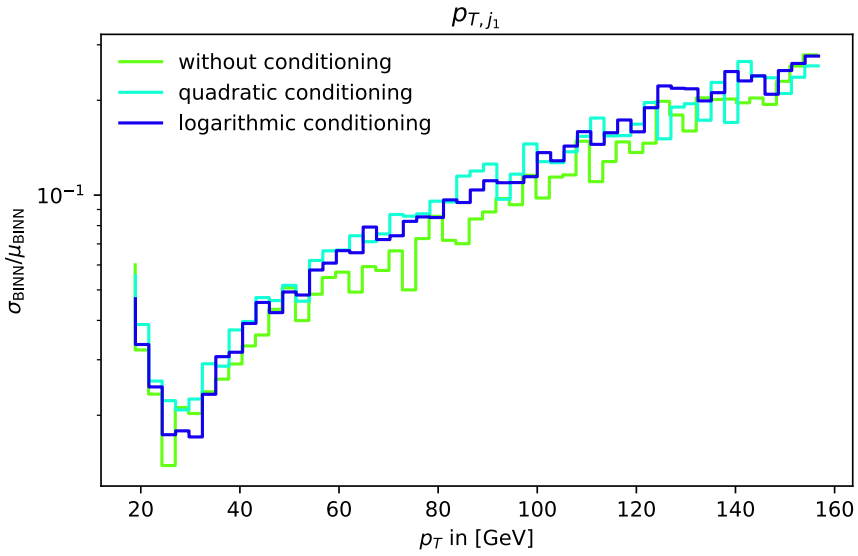

Fig. 23: Comparison of relative uncertainties between a conditional BINN and the standard BINN

7. Conclusion

This thesis introduced a setup to generate uncertainties for event generation via invertible neural networks. An attempt at demystifying the uncertainties via quantifying the errors was made. The most important points were:

- The uncertainties can be manipulated by the size of the training data set and the length of the training.
- The uncertainties are correlated with the number of generated events.
- The uncertainties can be split into a statistical Poisson error σ_{stoch} and a second part σ_{pred}. We have yet to find a way to calculate σ_{pred} other than via taking the quadratic difference. This would be great to verify our theory, however it remains difficult.
- The statistical error is not correlated with the length of the training data set, however σ_{pred} and σ_{tot} decrease with an increasing number of training events.
- The performance of the BINN can be enhanced by training on weighted events and a manipulation of the weights. This is more of a trial and error procedure according to the specific Data set. Introducing noise to the weights worked great for a three jet data set, but did nothing for the Z+ multijet data set. Testing the same setup on a different architecture would be interesting project.
- Lastly, we build a tool to allow for a theory uncertainty. Allowing us to test different EFT theories, and adding a systematic error contribution to the uncertainties.

Bayesian neural networks will become one of the most important tools in LHC data analysis in the future. Increasing the overall performance via introducing a discriminator[34] or an additional MMD[35] loss could build a powerful analysis tool. Progressing into a precision network with expressive uncertainties will justify using machine learning for data analysis and convince sceptics of the seemingly endless possibilities.

Gaining more control over the uncertainties would be a great way to have some sort of feedback if the network is learning the right densities. For future projects it would be useful, if the uncertainties would increase when the model is off from the target density, for example in the $\Delta\phi_{i,j}$ distributions. Having a tool to control the uncertainties bound to the performance would be very helpful. Unfortunately, as of today this is not yet possible.

To put everything in a nutshell, Bayesian INNs are another useful trick data analysts can add to their toolbox to improve the final analysis and justify new theories.

8. References

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

[2] E. Bothmann et al., "Event Generation with Sherpa 2.2", SciPost Phys. **7**, 034 (2019).

[3] O. Knapp, G. Dissertori, O. Cerri, T. Q. Nguyen, J.-R. Vlimant, and M. Pierini, "Adversarially Learned Anomaly Detection on CMS Open Data: re-discovering the top quark", (2020).

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

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets", in Proceedings of the 27th international conference on neural information processing systems - volume 2, NIPS'14 (2014), pp. 2672–2680.

[6] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: an overview", IEEE Signal Processing Magazine **35**, 53–65 (2018).

[7] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman, and T. Plehn, "GANplifying Event Samples", SciPost Phys. **10**, 139 (2021).

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes", (2014).

[9] D. P. Kingma and M. Welling, "An introduction to variational autoencoders", Foundations and Trends® in Machine Learning **12**, 307–392 (2019).

[10] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows", in Proceedings of the 32nd international conference on international conference on machine learning - volume 37, ICML'15 (2015), pp. 1530–1538.

[11] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: an introduction and review of current methods", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1–1 (2020).

[12] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows", (2015).

[13] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks", (2018).

[14] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp", (2016).

[15] D. P. Kingma and P. Dhariwal, "Glow: generative flow with invertible 1x1 convolutions", (2018).

[16] M. D. Klimek and M. Perelstein, "Neural Network-Based Approach to Phase Space Integration", (2018).

[17]J. Bendavid, "Efficient Monte Carlo Integration Using Boosted Decision Trees and Generative Deep Neural Networks", (2017).

[18]S. Bollweg, M. Haußmann, G. Kasieczka, M. Luchmann, T. Plehn, and J. Thompson, "Deep-Learning Jets with Uncertainties and More", SciPost Phys. **8**, 006 (2020).

[19]M. Bellagente, M. Haußmann, M. Luchmann, and T. Plehn, "Understanding Event-Generation Networks via Uncertainties", (2021).

[20]Wikimedia, *Standard model*, July 2006.

[21]M. Cacciari, G. P. Salam, and G. Soyez, "The anti-k_t jet clustering algorithm", JHEP **04**, 063 (2008).

[22]M. Cacciari, G. P. Salam, and G. Soyez, "The anti-k_t jet clustering algorithm", JHEP **04**, 063 (2008).

[23]I. Brivio and M. Trott, "The Standard Model as an Effective Field Theory", Phys. Rept. **793**, 1–98 (2019).

[24]D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", (2014).

[25]D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization", (2017).

[26]C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Cubic-spline flows", (2019).

[27]D. MacKay, "Probable Networks and Plausible Predictions – A Review of Practical Bayesian Methods for Supervised Neural Networks", Comp. in Neural Systems **6**, 4679 (1995).

[28]P. C. Bhat and H. B. Prosper, "Bayesian neural networks", Conf. Proc. C **050912**, 151 (2005).

[29]Y. Gal, "Uncertainty in Deep Learning", PhD thesis (Cambridge, 2016).

[30]L. N. Smith and N. Topin, "Super-convergence: very fast training of neural networks using large learning rates", in Artificial intelligence and machine learning for multi-domain operations applications, Vol. 11006 (International Society for Optics and Photonics, 2019), p. 1100612.

[31]M. Backes, A. Butter, T. Plehn, and R. Winterhalder, "How to GAN Event Unweighting", SciPost Phys. **10**, 089 (2021).

[32]A. Butter, T. Heimel, S. Hummerich, T.Krebs, T. Plehn, A. Rousselot, S. Vent, *Controlled event generation with precision networks*, Oct. 2021.

[33]L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, "Guided image generation with conditional invertible neural networks", (2019).

[34]K. Cranmer, J. Pavez, and G. Louppe, "Approximating Likelihood Ratios with Calibrated Discriminative Classifiers", (2015).

[35]A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample problem", CoRR (2008).

List of Figures

List of Tables

9. Acknowledgments

I would like to thank Tilman and Anja for giving me the opportunity to work on such an interesting and future oriented topic. Thank you for your innovative ideas and very helpful guidance during this time. Also I'd like to thank the whole group for the interesting discussions and the nice atmosphere in the garden meetings. Namely i am grateful for Theo, who provided the code basis and found every Bug and Butterfly that i might had, answering every question along the way and helping me out whenever he could. Last but not least i would like to thank Barry, Anja and Theo for proof reading this thesis.

A. How to BINN

A guide to find the best Hyperparameters

The search for the optimal hyperparameters can be a frustrating and time consuming process. This is a quick guidance to find optimal Hyperparameter and spare frustration for future projects. If trained on the same dataset, the usual INN is not as reactive to changes in the hyperparemters as the BINN. Certain hyperparameters lead to instability and failure of the BINN.

Cubic spline blocks

As discussed in the Thesis, cubic spline blocks allow flexibility and a fast computable Jacobian. However they increase the instability compared to the INN. Reducing the number of blocks solves this problem. I found that 22 blocks is the maximal Number of blocks before the network crashes. Note that the BINN was stable when the affine blocks were used, independent of the Number of Blocks

Number of layers

I found that to reproduce a similar baseline result as for the INN i had to double the number of layers. Using a small architecture has the advantage of a shorter Training duration but the BINN really profits from using larger architectures. For exapmle using the INN, i could get away with 3 layers however for the BINN i had to use 6. Somehow 6 seemed to be the magic Number of layers, since the networks performance got worse when more layers were added or taken away.

B.

Erklärung

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

Heidelberg, den 8. September 2021,