Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Sebastian Pitz

born in Heppenheim (Germany)

2000

Using Bayesian Neural Networks (BNN) to predict scattering amplitudes

This Bachelor Thesis has been carried out by Sebastian Pitz at the
Institute for Theoretical Physics in Heidelberg
under the supervision of
Michel Luchmann & Prof. Tilman Plehn

Abstract

Following the idea proposed by S.Badger and J. Bullock to train neural networks on scattering amplitudes to optimise LHC simulations we investigate how the approach can be further improved by making use of the uncertainty estimates provided by Bayesian neural networks. We optimize the models performance by finding the best architecture and analyse the behavior of uncertainties obtained by the network. Finally, we use a "feedback" algorithm to increase the performance on problematic, divergent phase-space regions.

Zusammenfassung

Aufbauend auf der Idee von S.Badger und J.Bullock, zur Optimisierung von LHC Simulationen neuronale Netzwerke auf Streuamplituden zu trainieren, untersuchen wir, wie dieser Ansatz mittels der Unsicherheiten von Bayesian Neural Networks verbessert werden kann. Wir optimieren die Leistung dieses Models durch Finden der besten Architektur und analysieren das Verhalten der Unsicherheiten des Netzwerks. Schlussendlich benutzen wir einen "feedback" Algorithmus um die Aussagekraft auf problematischen, divergenten Phasenraumregionen zu verbessern.

Contents

1 Introduction

In the field of theoretical particle physics and with higher energy runs at LHC, a substantial effort is devoted to the increasingly analytically complex calculation of scattering amplitudes of high multiplicity final states. These amplitudes play significant roles in the generation of collider events via interfacing into Monte Carlo event generators. To compute these values, numerous numerical methods already exist [1, 2, 3, 4, 5]. While these methods allow for powerful predictions, they are limited by the numerical workload required when upscaling to large datasets. Machine Learning methods do not encounter this problem by employing a separate training and prediction process.
High energy physics offers a large number of possible applications for ML, such as classification tasks using auto-encoders [6], capsule networks [7] or top tagging with uncertainties [8]. Additionally phase space integration and sampling [9, 10, 11] have been shown to be accelerated and improved by ML, while event generation is being improved by using generative Networks with uncertainties [12] or unweighting events with Generative Adversarial Networks (GANs) [13, 14]. The prediction of cross-sections [15] and scattering amplitudes for specific processes with Boosted Decision Trees (BDTs) [16] on the loop-induced $gg \to ZZ$ or an ensemble of networks for divergent and non-divergent phase space regions on $e^-e^+ \to <6$ jets (see Ref. [17]) and on diphoton processes $gg \to \gamma\gamma$+n(g) (see Ref. [18]) predicted with an uncertainty have been proven to show reliable and good approximations.
In this thesis, I will further investigate the relevant $gg \to \gamma\gamma$+n(g) process, which is loop-induced and has relevant contributions from high multiplicity matrix elements, making it a challenging and interesting process to study. Concretely, I will begin to firstly examine performances of different architectures and approaches, concretely also examining the possible advantages and disadvantages of phase space partitioning and the Frixione, Kunszt, and Signer (FKS) [19] [20] pair methods. To do this, I will use Bayesian Neural Networks (BNNs) [21] [12] [8] and in particular exploit their feature to predict scattering amplitudes using fully connected network layers with learned Gaussian weight distributions, leading to a detailed exploration of uncertainties whilst avoiding the training of an ensemble of networks. I will further explore the impacts of decreased training data and duplication of features and targets in critical, divergent phase space regions to increase performance. This allows us to compute kinematic distributions, compute the cross section and lays the foundation to an efficient reweighting / resampling.

2 Basics

2.1 Scattering Amplitudes, Cross Sections, Divergent Structures

In this chapter I will introduce the concepts of scattering amplitudes (matrix elements) and cross sections in order to give a mathematical and physical background to the different concepts we will be using during the thesis. This chapter is based on [22] [9] [23] [24] [25].

2.1.1 Scattering Amplitudes

Since, in particle physics, we usually examine scattering processes with distinct initial and final states, it makes sense to define a $\mathcal{S}$-matrix, which encodes the relation between the incoming and outgoing state which come from isomorphic Fock spaces:

$$|out\rangle = \mathcal{S}\,|in\rangle \tag{1}$$

Alternatively, in the interaction picture one defines it as the limit of the time-evolution operator $U(t', t)$ assuming the particles do not interact at very early or very late times. This gives us a definition in relation to the interaction Lagrangian if it can be split up into a interaction and non-interaction part:

$$S = \lim_{t\to-\infty t'\to\infty} U(t', t) = T\exp\left\{\left(i\int_{\infty}^{\infty} d^4x\mathcal{L}^I_{int}(x)\right)\right\} \tag{2}$$

In order to properly justify this limit, one uses LSZ-formalism to understand how to describe single particles in the interacting theory(for a more detailed derivation see for instance Ref. [22]).. If the particles do not interact at all, the $\mathcal{S}$-matrix becomes the identity matrix, such that is makes sense to split it up into the identity matrix and a transition matrix $\mathcal{T}$ resulting from the interactions in our scattering process:

$$\mathcal{S} = \mathbf{1} + i\mathcal{T} \tag{3}$$

where the imaginary unit is added by convention. For a process with incoming particles with momenta $p_1, ..., p_i$ and outgoing momenta $p_{i+1}, ..., p_n$ this then means:

$$\langle out|\, i\mathcal{T}|in = (2\pi)\delta^{(4)}(p_1 + ... + p_i - p_{i+1} - ... - p_n)i\mathcal{M}(p_1, ..., p_n) \tag{4}$$

with the δ function ensuring conservation of momentum. $\mathcal{M}$ is then defined as the matrix elements or scattering amplitudes and allow to compute observables in particle scattering processes and e.g. the cross-section of the process. One can now use LSZ-Formalism to compute the time ordered correlation functions of the interaction theory with only the free field commutation relations using:

$$\langle 0|\, T\varphi(x_1)\cdots\varphi(x_n)\,|0\rangle \tag{5}$$

$$= \frac{{}_0 < T\varphi_I(x_1)\cdots\varphi(x_n)\exp\left\{\left(-i\int_{-\infty}^{\infty} d\tau H_{int}(\varphi_I(\tau,\vec{x}))\right)\right\}|0\rangle_0}{{}_0\langle 0|\, T\exp\{(d\tau H_{int}(\varphi_I(\tau,\vec{x}))\}\,|0\rangle_0} \tag{6}$$

with the free field vacuum state $|0\rangle_0$ and the particle field φ_I in the interaction picture. Additionally we use the Wick theorem to incorporate all contractions into the the correlation function to finally calculate by expanding around the interaction constant, which in the case of strong gluon coupling is given by:

$$\alpha_s = \frac{g_s^2}{4\pi} \tag{7}$$

with the strong coupling g_s, which relies on the assumption that $\alpha_s << 1$, which turns out to be valid, especially at high energies achieved in a particle collider.

After using a Fourier transformation we then get the Feynman rules which can be used to compute all possible scattering channels which together make up the matrix element of processes by using them as "building blocks" which together make up the entire process. The squared matrix elements are then used for the calculation of cross-sections.

2.1.2 Divergent Structures

When constructing the Feynman diagrams and the writing down the according equations for the scattering amplitude with the Feynman rules, one finds that the integral over loops do not disappear and together with other phenomenones (e.g. soft / collinear particles) create divergences. Generally one decides between two types of divergences: ultraviolet (UV) and infrared (IR), corresponding to high energy and low energy limits of the propagator integrals: While we can deal ultraviolet divergences by using regularization and renormalization which become correction terms in the QCD Lagrangian, the infrared divergences are more difficult to deal with but usually not as problematic since all observable stay finite. Infrared divergences appear in theories with massless particles (like the photon in our scattering process). Since the networks have problems with the training of IR-divergencies however, I will further examine them: As previously mentioned, there are several cases that can create IR-divergencies: one is loop integrals which do not disappear by using Fourier transformation as other propagators do, but remain and can diverge (virtual IR divergences) and divergences resulting from soft and collinear emissions (real IR divergences). The latter result from Matrix elements that have the invariant mass of the particles in their denominator. In general, these two types of IR-divergences would cancel another out. [26][27][28]. Since numerical methods (Monte Carlo integration techniques are used to integrate the final states over the entire phase space, one needs a different methodology for the calculations of infrared terms. In this theses I will be using the Frixione, Kinszt, and Signer (FKS) subtraction [19] [20]. We are specifically using these methods to isolate divergent regions in one-loop processes so that the split up data can be given to separate networks for training.

2.1.3 Scattering Cross section

This chapter is based on lecture notes [29]. The cross section is similar to the chance of an event to happen and as such can be derived in the following way: Consider a fixed target experiment, with N$_B$ particles coming in over a area F while target A is hit with N$_{\text{events}}$ particles. The cross section is then intuitively given as:

$$\frac{N_{\text{events}}}{N_B} = \frac{\sigma}{F}. \tag{8}$$

Now we consider the more realistic case of incoming wave packages

$$|f_{\vec{p}}\rangle = \int d\tilde{k} f_{\vec{p}}(\vec{k}) \, |k\rangle \tag{9}$$

where $f_{\vec{p}}$ is localized around $\vec{p}$ and the Fourier transform is localized in real space around $\vec{x} = 0$. We can then define a state where the target A and the incoming particle B are both localized around $\vec{x} = 0$ for t=0:

$$|i\rangle = \int d\tilde{k}_A d\tilde{k}_A f_{\vec{p}A}(\vec{k}_A) f_{\vec{p}B}(\vec{k}_B) \, |k_A k_B\rangle \tag{10}$$

now we want to account for the spread of incoming particles and define a impact parameter $\vec{b}$ and, as in quantum mechanics, the momentum operator $\hat{P}$ which generates the shift and satisfies:

$$\hat{P} \, |k_B\rangle = k_B, \tag{11}$$

such that a shifted state is given by

$$e^{-i\hat{\vec{P}}\vec{b}} \int d\tilde{k}_B f_{\vec{p}B}(\vec{k}_B) \, |k_B\rangle \tag{12}$$

and therefore

$$|i_b\rangle = \int d\tilde{k}_B d\tilde{k}_A f_{\vec{p}A}(\vec{k}_A) f_{\vec{p}B}(\vec{k}_B) e^{-i\vec{k}_B \vec{b}} \, |k_A k_B\rangle \, . \tag{13}$$

A particle approaching from far away at t=$-\infty$ will then reach the desired state $|p_1 p_2\rangle$ then N$_{\text{events}}$ times:

$$N_{\text{events}} = \sum_{\vec{b}} \langle p_1 p_2| \, S \, |i_b\rangle \, |^2 \approx \frac{N_B}{F} \int_F d^2b | \, \langle p_1 p_2| \, S \, |i_b\rangle \, |^2 \tag{14}$$

where the approximation holds for a homogeneous, transverse distribution of N$_B$ particles in the area F, and therefore:

$$\sigma(\vec{p}_1, \vec{p}_2) = \frac{N_{events}}{(N_B/F)} = \int_F d^2b | \, \langle p_1 p_2| \, S \, |i_b\rangle \, |^2. \tag{15}$$

Finally, if we want to consider a larger phase space area, we get the following cross-section:

$$\sigma(V_f) = \int_{V_f} d\tilde{p}_1 d\tilde{p}_1 \sigma(\vec{p}_1, \vec{p}_2), \tag{16}$$

which can be rewritten as

$$d\sigma = \frac{1}{2s} |\mathcal{M}_{fi}|^2 dX^{(n)}, \qquad dX^{(n)} = (2\pi)^4 \delta^4(p_f - p_i) d\tilde{p}_1 ... d\tilde{p}_n \tag{17}$$

which allows us to partially integrate the above expression and obtain a differential cross-section.

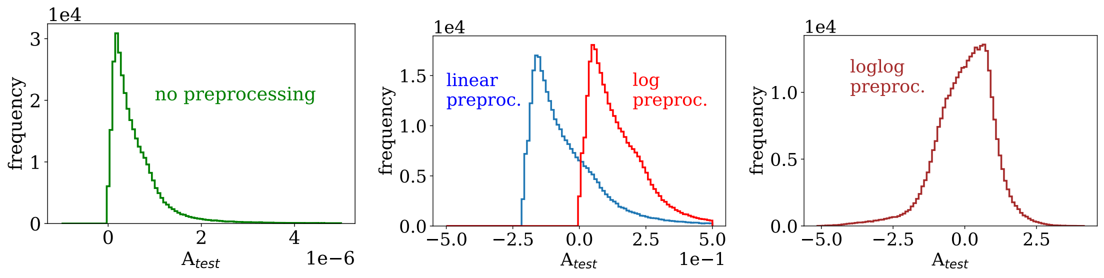

Fig. 1: Amplitude distribution before and after preprocessing

2.1.4 $gg \to \gamma\gamma g$

This chapter is based on [30].The process gg $\to \gamma\gamma + n(g)$ is a one-loop induced process and has relevant contributions from high-multiplicity matrix elements. The process $pp \to \gamma\gamma+ \leq 3$ has been the object of current research[31] [32] [33] and as such this process and the similar process with a second gluon in the final state have become increasingly relevant. For the examined process of $gg \to \gamma\gamma g$ there are analytic solutions available. However, this is not the case for the more difficult process of two gluons in the final state, such that we aim for our network to later be able to generalize to this scattering process. These analytic amplitudes entail a fermion loop (compare Ref. [18]). The existence of a analytic solution allows us to use use NJet [34] to compute the amplitudes for events we generate with RAMBO, to then use supervised training. The advantage to the analytic methods and even the numeric methods, which exist for the $2 \to 4$ process, is the better scaling of the neural networks, which has been shown in Ref. [18]. As mentioned in the Introduction, we are examining the process $gg \to \gamma\gamma$+n(g), in our case specifically $gg \to \gamma\gamma g$ to find the scattering amplitudes in different phase space regions. In the following thesis I will use "g" and "j" (for *jet*) analogously which corresponds to the physical measurable quantity also stresses the fact that this network can in general be used for more general processes, although specifics like cuts would have to be changed. Our features consist of the four-momenta $x \in \mathcal{R}^4$ the five particles each, creating a total 20 dimensional input size, while the targets are the scattering amplitudes $y \in \mathcal{R}$ of each event. The amplitudes are computed by the NJet3 algorithm [34] using a rambo sampling algorithm [35] to generate flat phase-space points. Until otherwise specified, the same integrator is used for training, validation and testing. The validation set consists of 10% of the training set. We use a training set of inital size 100k events and a test set of a size 1M events before cuts. The size of these datasets will however be significantly shrinked to about one-third of its original size by the physical cuts that have to be applied. We will use cuts on the transverse momentum, the angular distance and the pseudorapidity. In order to sort the data further, we make sure that the first photon 4-momenta in the dataset is always corresponding to the photon with the higher transverse momentum. This data is then split up into a divergent and a non-divergent part, using the quantity

y_{ij}:

$$s_{ij} = (p_i + p_j)^2 \tag{18}$$

$$y_{ij} = \frac{s_ij}{s_12} \tag{19}$$

where s_{12} is the center-of-mass energy by definition. The numbering of the particles corresponds to the following one: $gg \to \gamma\gamma g$. The divergent part corresponds to phase space region we assume to contain divergent i.e. soft or collinear particles. The data will be further split up to isolate specific divergent structures as will be shown in the following chapter. This will lead to a weighting of the events in the divergent phase-space regions. Each such region will then be used as the input for a BNN, such that there will be a total of N+1 networks where N is the number of divergent phase-space regions and we use one additional network for the non-divergent phase space region.

This approach is supposed to increase the network performance, since the divergent phase space regions often have significantly different properties and as such are harder to train on for conventional networks [34, 18]. One of our result will be that BNNs are able to handle these phase space regions better and even achieve better result when only using a single network for the entire training set. Since the amplitudes are not Gaussian distributed and relatively small, we will use a preprocessing on both the data as well as the amplitudes to avoid exploding/vanishing gradients as is customary in ML literature. For the 4-momenta, we standardize them by subtracting the mean value of each of the 20-dimensions and then dividing by its standard deviation, leading to a mean value of zero and a standard deviation of one (compare figures 1, 2). We will try out several different preprocessing methods for the scattering amplitudes, since the performance does vary significantly (see results). The normal standardisation to mean value zero and standard deviation one works works not as well here when considering the distribution of the amplitudes. One finds there are naturally a lot of smaller amplitudes, with several single outliers being significantly larger. When we apply linear preprocessing:

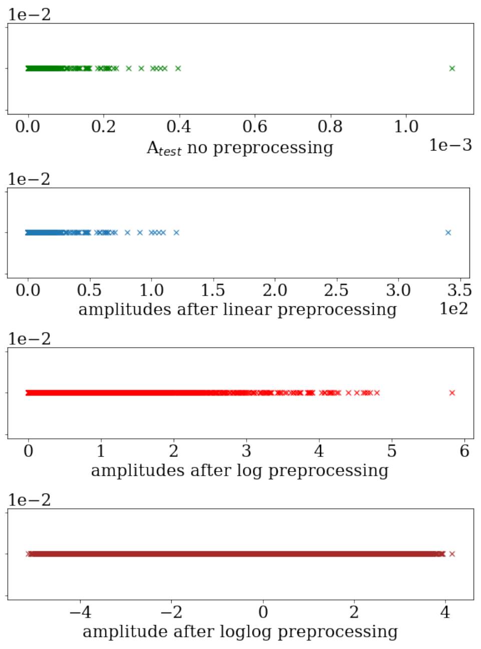

Fig. 2: Amplitude distribution before and after preprocessing

$$A \mapsto \frac{(A - \langle A|A \rangle)}{\sigma_A} \tag{20}$$

We can see that the scale of amplitude magnitude is significantly lower, thus reducing the risk of exploding/vanishing gradients, but still has outliers in high amplitude regions. This can be reduced, if we use a log preprocessing:

$$A \mapsto \log\left(1+\frac{A}{\sigma_A}\right) \tag{21}$$

We essentially are able to differentiate more clearly amplitudes with lower magnitude by splitting them up and decrease the extend of large amplitude outliers. Additionally we can see that when we use a second log, this effect is further increased. We do not use this last preprocessing method however, since it leads to large numeric instability, when the amplitudes have to be exponentiated when transforming back, once the network has predicted the preprocessed amplitudes.

2.2 Cuts and Phase Space Partitioning

For the partonic one-loop process we are trying to find the scattering amplitudes for we use the following basic cuts:

$$\begin{aligned} &p_{T,j} > 20 \;\; \text{GeV} && |\eta_j| > 5 && R_{jj,j\gamma,\gamma\gamma} > 0.4 \\ &p_{T,\gamma} > 40, 30 \text{ GeV} && |\eta_j| > 2.37\,. && \end{aligned} \tag{22}$$

where p_T is the transverse momentum of the particles, i.e. the momentum orthogonal to the beam axis:

$$p_T = \sqrt{p_x^2 + p_y^2} \tag{23}$$

eta is the pseudorapidity which is a spartial coordinate describing the angle of the particle relative to the beam axis θ, defined as:

$$\eta = -\ln\tan\frac{\theta}{2} \tag{24}$$

R$_{j}j$ is the angular distance and is defined as:

$$R_{jj} = \sqrt{(\Delta\eta)^2 + (\Delta\varphi)^2} \tag{25}$$

where $\Delta\varphi$ marks the relative azimuthal angle between the two jets. These cuts stem from physical backgrounds of the scattering process and the detector. For example: the cut in the angular distance ensures that the jets have a minimal angle between them. This makes sure that the detector is not interpreting a single jet (or shower) as two separate events as easily. As previously mentioned, we use the quantity y_{ij} is used to split the data into a divergent and a non-divergent part. This y_{ij} is computed for both incoming, outgoing as well as mixed particle pairs:

$$\mathcal{R}_{div} = \{p|\min(y_{ij}) < y_{\text{cut}}, p = (p_1, ..., p_n), i, j \in \{1, ..., n\}\} \tag{26}$$

$$\mathcal{R}_{non-div} = \{p|\min(y_{ij}) > y_{\text{cut}}, p = (p_1, ..., p_n), i, j \in \{1, ..., n\}\} \tag{27}$$

where p is an event, consisting of the two incoming gluons four-momenta $\{p_1, p_2\}$ and the outgoing momenta $\{p_3, ..., p_n\}$, where in our case $n = 5$. The value of y_{cut} is a hyperparameter and has further been examined in [34]. We use y_{cut}=0.02 for our

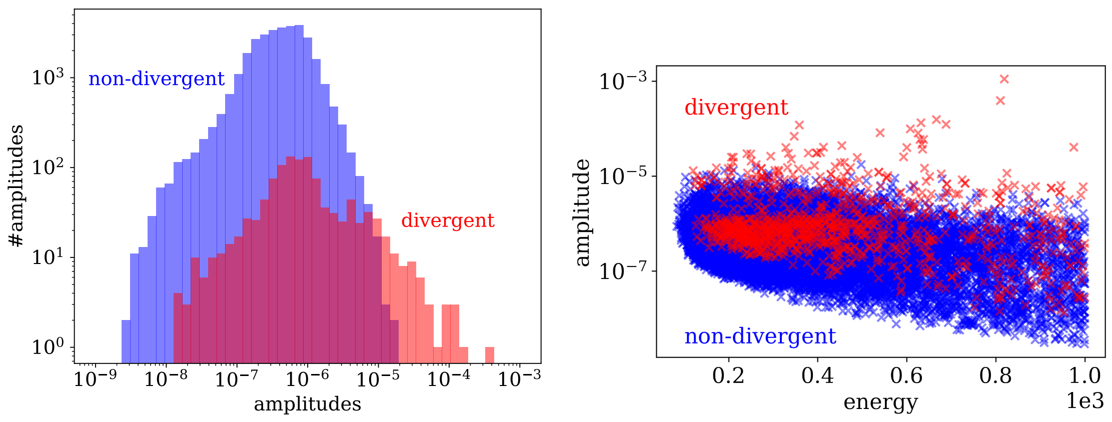

Fig. 3: Left: Amplitudes split up into divergent and non-divergent phase space regions; although there are also smaller amplitudes in the "divergent" area, all high amplitudes are exclusively in it; Right: Energy of all particles over their amplitudes; the highest amplitudes are all contained in the divergent area

networks. It should be mentioned here, that these divergent regions also include points which are not divergent and even particle pairs that can not be soft. This is done to keep the uses general and not fit them to this very specific purpose, thereby making it less appliable. Now we want to isolate IR-divergences, which appear from soft or collinear real emissions and integrals over massless partons, by using FKS subtraction [19] [20] as follows:

$$\mathcal{P}_{\text{FKS}} = \{(i,j)|1 \leq i \leq n, 2 \leq j \leq n, i \neq j,\\ \mathcal{M}^{(\text{n},0)} \text{ or } \mathcal{M}^{(\text{n},1)} \to \infty \text{ if } p_i^0 \to 0 or p_j^0 \to 0 \text{ or } \vec{p}_i || \vec{p}_j\} \tag{28}$$

With the partition sums:

$$S_{i,j} = \frac{1}{D_1 s_{ij}}, D_1 = \sum_{i,j \in \mathcal{P}_{FKS}} \frac{1}{s_{ij}} \tag{29}$$

such that

$$d\sigma = \sum_{i,j} S_{i,j} d\sigma \tag{30}$$

where σ is the cross section of the process. In our process this leads to ten regions, where the scattering amplitudes in each region are then weighted by $S_{i,j}$ (for example figure 4) and trained with a seperate network. Including even pairs that do not produce singilarity structures helps with generalizability and has been shown in Ref. [18] to not have damaging effects on the networks performance, nonetheless adding redundancy.

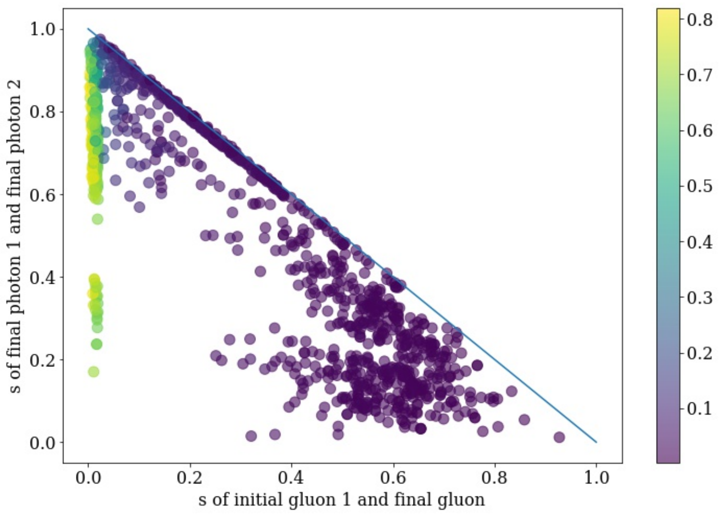

Fig. 4: Examplatory partition/weighing function $S_{g,g}$ for first initial gluon and final gluon: regions with low value of s (indicative of singularity structure) are weighted higher, while other regions are suppressed; this network would mainly train on the regions where g, g would indicate singularities

2.3 Neural Networks, Optimizers, Activation functions

2.3.1 Neural Networks

Neural Networks describe a collection of algorithms which are able to approximate highly non-linear functions though fitting. Here I will give a short introduction to Multi-Layer Perceptrons (MLP), comprised of a series of layers called Perceptrons combined with a optimizer. The perceptron is a simple building clock for neural netoworks [36]. These layers take as a input the point $\mathrm{x} = (\mathrm{x}_1,\ldots,\mathrm{x}_n)$ and return:

$$f(x) = a\left(\sum_{k=0}^{d} x_k \cdot \omega_k\right) \tag{31}$$

where by definition $\mathrm{x}_0 = 0$ and $\mathrm{x}_0 \cdot \omega_k$ gives a bias, while the other ω_i give a weighting. a is a activation function, which is used to introduce non-linearity into the network. This is necessary since the output of the previous layer is always given to the next layer and a series of linear layers would only have the expressiveness of a single linear layer. In regression problems we will lay specific importance on the last layer, since it has to be able to map onto our entire range of targets y. Often times this means that the last layer is only a linear layer or no layer at all. Other than conventional analytic algorithm, the use of a neural network is comprised of two distinct phases: a training and a test phase. During supervised training, the network is given a training data set comprised of features x and targets y. It then uses a Optimizer (see next chapter) to approximate the training data i.e. attempt to align f(x) to y. To achieve this, we need to define the Loss (next chapter) between f(x) and y and minimize it through the use of gradients and chain rule. Depending on the amount of layers and the weights per layer, the network will be able to "learn" a underlying relation between the features and the targets if such a relation exists. However there is no reason to believe that there will only be one minima in the loss function and the network might also simply "remember" the targets to the features in the training set. This phenomenon is called overfitting and leads to a minimal loss of zero, but at the same time to a complete lack of expressiveness on new data. To avoid this, there exist several methods (dropout, early stopping), but the detection can simply be achieved by designating a portion of the training set to be a validation set, which the network then makes prediction on, but does not use for training, allowing insight into the expressiveness of the network. Additionally to this "validation loss", one might also define a "training loss", i.e. the loss produced by the prediction on the training set and a "test loss" which is indicative on the performance on a test set if the correct targets are available. During the test phase, the weights ω are then held constant and the network is able to make prediction on given data. As the name suggest, MLP use several layers where each layer takes a input the output of the previous layer:

$$f(x) = f^{(o)}(f^{(h)}(f^{(h-1)}(...(f^(1)(x))...))), \tag{32}$$

where $f^{(i)}$ are the inner layers, $f^{(1)}$ is the input and $f^{(o)}$ is the output layer. Usually the choice is made to use the same activation function for all inner layers to reduce the number of possible hyperparameter combinations. The striking advantage of this method is the speed up in the required time during prediction, which has been shown to speed up e.g. the integration of functions. This has made NN a attractive method to use for phase-space sampling and integration [37][38]. Recent work [39][40][41] have used difference achitectures such as renormalizing flows Netoworks and Generative Adversarial Networks (GANs) to achive better result than classical algorithms.

2.3.2 Optimizers

To train a neural network, one needs a meassure of the performance or the lack of performance of the network, given the available training data. With this "Loss Function", one can then use gradients analysis to compute the required change of the network parameters to minimize this this loss, thus maximizing the network performance. This computation of the optimal change of network weights and biases is done by a optimizer. We decided to use the commonly used Adam [42], adaptive moment estimation, optimizer , which combines the adaptive learning rate from RMSprop [43] and the use of a momentum implementation. This momentum method is given by an additional term in the step equation of the optimizers, which discourages sudden changes in the direction of the previous updates of weights and biases, similar to the force a rolling sphere would experience in a physical system if it would be pushed slightly in a direction different from its previous direction of momentum. This helps to archive a smoother convergence, especially when dealing with outliers, which can introduce false directions into the gradient and therefore hinder the progress of optimization. Using this method to take into consideration the previous gradients has been shown to lead to high performance SGD if used correctly [44]. The step function is then given as:

$$\theta_{t+1} = \theta_t - \eta\nabla L(\theta_t) + m_t \tag{33}$$

$$m_t = \nu\nabla L(\theta_{t-1}) + m_{t-1} \tag{34}$$

where θ are the learning parameter, η is a given hyperparameter, the learning rate, $L(\theta)$ is the loss function, m_t is the momentum and ν is another hyperparameter corresponding to the influence previous gradiens have on the calculation of the current gradient. Notice that the momentum at time t always carries the exponentially decaying momentum of all previous momentum. When adding adaptive learning rate / adaptive gradients, we aim to deal with the problem that different weights often have very different gradient magnitudes, which makes the choice of a global learning rate difficult. To solve this, we carry a moving average of the squared average of each weight and then divide the gradient by the square root of it. This leads to RMSprop, with the step function:

$$\theta_{t+1} = \theta_t - \frac{\eta\nabla L(\theta_t)}{\sqrt{v_t} + \epsilon} \tag{35}$$

$$v_t = (1-\mu)(\nabla L(\theta_t))^2 + \mu v_{t-1} \tag{36}$$

where the term ϵ is small and has been added for numerical stability, v_t is the exponentially decaying average of squared gradients and μ is a hyperparameter corresponding to the weight previous parameters have on the current computed gradient. Adam combines these two concepts into the following step function:

$$\theta_{t+1} = \theta_t - \frac{\eta\hat{m}_t}{\sqrt{v_t} + \epsilon} \tag{37}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{38}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{39}$$

$$m_t = (1-\beta_1)\nabla L(\theta_t) + \beta_1 m_{t-1} \tag{40}$$

$$v_t = (1-\beta_2)(\nabla L(\theta_t))^2 + \beta_2 v_{t-1} \tag{41}$$

with hyperparameters β and one division to remove biases.

2.3.3 Activation Functions

In the use of regression appliances it is crucial to use appropriate final activation functions which are able to cover the entire realm of targets. Since we want to predict scattering amplitudes, we need to use activation function with positive output in the case of log preprocessing, and a activation function with positive and negative outputs in the case of linear preprocessing. During our training we mainly use three different final activation functions: ReLU [45], GeLU[46] and Softplus, of which only GeLU is able to allow negative values. GeLU and Softplus are approximately linear for large input values are approach zero for small values, mainly differing in the region around zero. Softplus and GeLU are differentiable while ReLU is not at $x = 0$, which usually does not pose a problem. Between the inner layers we use tanh activation function, as is done in Ref. [18].

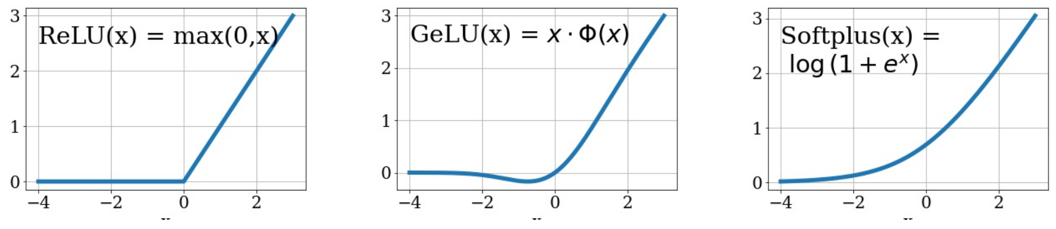

Fig. 5: Activation functions

2.4 Bayesian Neural Networks

Generally Bayesian Neural Networks (BNN) work similar to normal fully connected linear network layers with two output dimensions - one dimension for the output and an accompanying uncertainty. Additionally the BNN does not only learn the weights ω, but rather a distribution for each network weight[47], which will be approximated as a Gaussian, allowing for reduction to two parameters for each network weight. For the prediction of scattering amplitudes of each phase space point we want to sample the amplitudes and the uncertainty over the weight distributions assuming p(A | ω) to be a Gaussian, such that the network output is given as:

$$\text{BNN} : x, \omega \rightarrow \begin{pmatrix} \overline{A}(\omega) \\ \sigma_{\text{stoch}}(\omega) \end{pmatrix} . \tag{42}$$

where we define $\overline{A}(\omega)$ and $\sigma_{\text{stoch}}(\omega)$ are the mean and the standard deviation of the Gaussian and defined as follows:

$$\overline{A}(\omega) = \int dA \; A \; p(A|\omega) \tag{43}$$

$$\sigma_{\text{stoch}}(\omega)^2 = \left(\overline{A^2}(\omega) - \overline{A}(\omega)^2\right) . \tag{44}$$

This is then the output of a single network with sampled weights ω and since the networks with different networks will, in general not have the same output, it is clear that we will obtain another error corresponding to the weight distributions. The output can then be sampled over the different weight distributions to obtain:

$$\langle A \rangle = \int d\omega \; p(\omega|T)\overline{A}(\omega) \tag{45}$$

$$\sigma^2_{\text{stoch}} = \langle \sigma^2_{\text{stoch}} \rangle = \int d\omega \; p(\omega|T)(\overline{A^2}(\omega) - \overline{A}(\omega)^2), \tag{46}$$

where T is the training data. In reality this will be done by sampling from the network weight distributions to get distinct networks and then averaging over the output of these networks (compare figure 6). However, we do not have a closed form for the network weight distributions $p(\omega|T)$, which we solve by using the variational ansatz do approximate it, such that the following holds true:

$$p(A|x) = \int d\omega \; p(A|\omega)p(\omega|T) \approx \int d\omega \; p(A|\omega)q(\omega), \tag{47}$$

for the probability distributions for possible amplitudes (the dependence on the phase-space point is implicitly included whenever A appears). Since the performance of the network depends on how accurate this approximation is, we will define the loss as the Kullback-Leibler divergence of the learned and the true probability distribution:

$$\begin{aligned}\mathcal{L}_{\text{BNN}} &= \text{KL}[q(\omega), p(\omega|T)] \\ &= \int d\omega \; \log \frac{q(\omega)}{p(\omega|T)} \\ &= \int d\omega q(\omega) \log \frac{q(\omega)p(T)}{p(\omega)p(T|\omega)} \\ &= \text{KL}[q(\omega), p(\omega)] - \int d\omega \; q(\omega) \log p(T|\omega) + \log p(T) \int d\omega \; q(\omega)\end{aligned} \tag{48}$$

in the transformation from the second to the third term we use Bayes' theorem, and p(ω) is the prior distribution which we will later decide to set as Gaussian. In general however, one could also choose different distributions here. The first loss term acts as a regularization to $q(\omega)$ to p(ω) which essentially ensures that the weight distribution keeps the given shape of the prior distribution, while the second term is the expected likelihood which can be used to work in a frequentist sense. We can disregard the last term since it does not depend on the network training and will just add a constant loss term, leading to the final loss:

$$\mathcal{L}_{\text{BNN}} = \text{KL}[q(\omega), p(\omega)] - \int d\omega \; \log p(T|\omega), \tag{49}$$

where the second term already includes the sum over all training points that are being used in the batch. If the batch is smaller than the total training size, it is important to scale the regularization term accordingly. Now we can make the assumption that the weight distributions are Gaussians to further expand the KL term:

$$\text{KL}[q(\omega), p(\omega)] = \frac{\sigma_q^2 - \sigma_p^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \log \sigma_q + \log \sigma_p \;, \tag{50}$$

where we can once again disregard the last term, since the standard deviation of the prior is fixed and therefore not relevant in the loss. Finally, we can find the total error of the amplitudes by applying the Law of total variance:

$$
\begin{aligned}
\sigma_{\text{tot}}^2 &= \langle (A - \langle A \rangle)^2 \rangle \\
&= \int dA \;\left(A - \langle A \rangle\right)^2 \; p(A|T) \\
&= \int dA d\omega \;\left(A - \langle A \rangle\right)^2 \; p(A|\omega, T)\; q(\omega) \\
&= \int dA d\omega \;\left(A^2 - 2A\langle A \rangle + \langle A \rangle^2\right) \; p(A|\omega, T)\; q(\omega) \\
&= \int d\omega \; q(\omega) \left[\int dA \; A^2 \; p(A|\omega, T) - 2 \int dA \; A \langle A \rangle \; p(A|\omega, T) + \int dA \; \langle A \rangle^2 \; p(A|\omega, T) \right] \\
&= \int d\omega \; q(\omega) \left[\overline{A^2}(\omega) - 2\langle A \rangle \overline{A}(\omega) + \langle A \rangle^2 \right] \\
&= \int d\omega \; q(\omega) \left[\overline{A^2}(\omega) - \overline{A}(\omega)^2 + \overline{A}(\omega)^2 - 2\langle A \rangle \overline{A}(\omega) + \langle A \rangle^2 \right] \\
&= \int d\omega \; q(\omega) \left[\overline{A^2}(\omega) - \overline{A}(\omega)^2 + \left(\overline{A}(\omega) - \langle A \rangle\right)^2 \right] \equiv \sigma_{\text{stoch}}^2 + \sigma_{\text{pred}}^2 \, . \qquad (51)
\end{aligned}
$$

here we can identify the stochastic uncertainty which the network predicts since it appears in the second term of the loss in equation 49, when we assume that p$(A|\omega)$ is a Gaussian. This uncertainty appears even if we do not sample from the weight distributions more than once, leading to the interpretation as a uncertainty of not the weight distributions, but the output of a single sampled network. Since our training and test data will be exact, one might assume that σ_{stoch} will vanish, but this would only be the case if the network could also represent/interpolate the data perfectly, which does not happen since the network has limited resolution and capability to find the perfect loss minimum. The additional uncertainty, σ_{pred} does however depend on $\langle A \rangle$ and therefore several sampled networks. σ_{pred} will approach zero for increasing training data since the weight distributions of the BNN will become delta distributions which leads to identical networks in the sampling process. Accordingly, σ_{pred} can be interpreted as a uncertainty resulting from the predictions of different sampled networks and therefore from the weight distributions. In contrast σ_{stoch} approaches a constant value for unlimited training data. It is nice to note here, that the calculation in equation 51 does neither assume Gaussian weight distributions, nor Gaussian ω-dependent network output. If one would use a different output however, σ_{stoch} might not be so readily available.

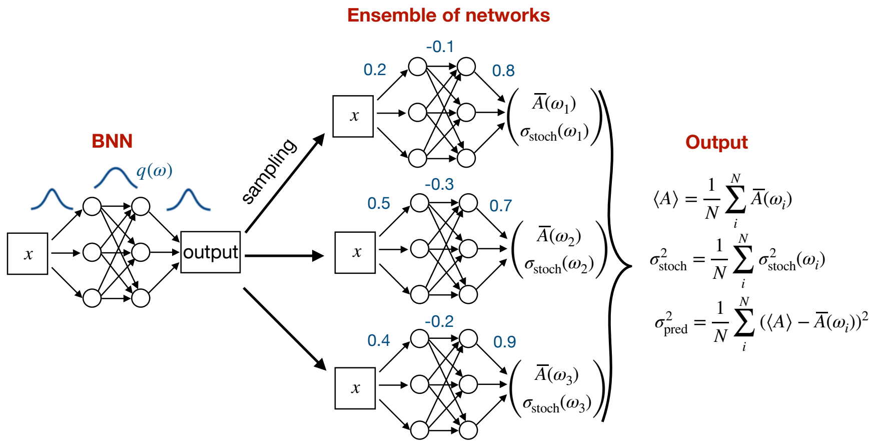
Ensemble of networks
BNN
$q(\omega)$
x
output
sampling
0.2
-0.1
0.8
0.5
-0.3
0.7
0.4
-0.2
0.9
$\bar{A}(\omega_1)$
$\sigma_{\text{stoch}}(\omega_1)$
$\bar{A}(\omega_2)$
$\sigma_{\text{stoch}}(\omega_2)$
$\bar{A}(\omega_3)$
$\sigma_{\text{stoch}}(\omega_3)$
Output
$\langle A \rangle = \frac{1}{N}\sum_i^N \bar{A}(\omega_i)$
$\sigma^2_{\text{stoch}} = \frac{1}{N}\sum_i^N \sigma^2_{\text{stoch}}(\omega_i)$
$\sigma^2_{\text{pred}} = \frac{1}{N}\sum_i^N (\langle A \rangle - \bar{A}(\omega_i))^2$

Fig. 6: Bayesian Neural Networks: by sampling from the weighting distributions of the network weights several conventional networks can be obtained. The resulting outputs give a uncertainty of each network itself σ_{stoch} and a uncertainty from the difference of the network predictions A: σ_{pred} (graph from [7])

2.4.1 Statistical quantity: "pull"

Another way to gauge the performance of the network is to examine the "pull", where we define as the possible "pull" quantities:

$$\begin{aligned} t_i^{\text{stoch}} &= \frac{\langle A_i \rangle - A_i^{\text{truth}}}{\sigma_{\text{stoch},i}} \\ t_i^{\text{pred}} &= \frac{\langle A_i \rangle - A_i^{\text{truth}}}{\sigma_{\text{pred},i}} \\ t_i^{\text{tot}} &= \frac{\langle A_i \rangle - A_i^{\text{truth}}}{\sigma_{\text{tot},i}} \end{aligned} \tag{52}$$

We will naturally expect these quantities to be centered around zero and Gaussian distributed, such that the total uncertainty should agree with the standard deviation of the Gaussian. We can now compute the variance of these pulls:

$$\text{Var}(t) = \langle t^2 \rangle - \underbrace{\langle t \rangle^2}_{\approx 0} \tag{53}$$

$$= \langle t^2 \rangle \tag{54}$$

$$= \frac{1}{N}\sum_{i=1}^{N} t_i^2 \tag{55}$$

$$= \frac{1}{N}\sum_{i=1}^{N} \frac{\left(\langle A_i \rangle - A_i^{\text{truth}}\right)^2}{\sigma^2_{\text{x},i}} \tag{56}$$

We can define another alternative pull, by not averaging over the network weight samples to receive a term included in the loss function:

$$t_i^{\text{alt}} = \frac{\bar{A}_i(\omega) - A_i^{\text{truth}}}{\sigma_{\text{stoch},i}(\omega)} \tag{57}$$

with a variance under the assumption of approximately mean value zero of:

$$\mathrm{Var}(t^{\mathrm{alt}})(\omega) = \frac{1}{N}\sum_{i=1}^{N} \frac{\left(\bar{A}_i(\omega) - A_i^{\mathrm{truth}}\right)^2}{\sigma^2_{\mathrm{stoch},i}(\omega)} \tag{58}$$

$$\langle \mathrm{Var}(t^{\mathrm{alt}})\rangle = \int d\omega\, q(\omega) \mathrm{Var}(t_i^{\mathrm{alt}})(\omega) \tag{59}$$

$$= \underbrace{\int d\omega\, q(\omega) \frac{1}{N}\sum_{i=1}^{N} \frac{\left(\bar{A}_i(\omega) - A_i^{\mathrm{truth}}\right)^2}{\sigma^2_{\mathrm{stoch},i}(\omega)}}_{\text{part of the loss function}} \tag{60}$$

Since this variance is directly included in the loss term, we would expect this quantity to directly correlate with the performance of the BNN. For a well trained network, we would expect the variance to come close to 1. We will investigate this in chapter 3.2.

3 Results

3.1 Previous results

Since this thesis heavily depends on the work previously done in Ref. [18], I will begin by summarizing the results of this paper and the results achieved, we well as the advantages our method could provide to the solution of this problem. This paper used a similar approach, but instead of using Bayesian Neural Networks, used a several different fully connected neural networks for the different divergent regions using FKS-subtraction and additionally another ensemble in order to achieve a description of a uncertainty. While this method certainly has been shown to work well and is able to both predict the amplitudes as well as the corresponding uncertainties, this has some aspect which are problematic. One disadvantage stems from the fact that the uncertainties will inevitably also contain a part which depends on the different trainings of the networks. This makes it hard to distinguish between the actutal physical errors of the test data and uncertainties which are created through different initial weights and convergence behavior. This was actually addressed in the paper and shown to be negligible, but this might not be as small of an effect of different data, decreasing the generalizability of the model. Most important however are the long times required to train the large amount of networks. In the paper the authors opted for an ensamble of 20 netwoks for each of the ten divergent plus one non-divergent region. This means that one has to train a total of 220 networks, which is even significantly larger for more complex processes since the number of divergent regions scales with n^2, where n are the number of total particles. This happens because the divergent numbers are always all combinations of two particles of initial and final state:

$$\#\text{divergent regions} = \binom{n}{2} = \frac{n \cdot (n-1)}{2} = O(n^2). \tag{61}$$

Even if the training of a single network does not require much time, this is not very scalable to scattering processes with higher number of particles involved. The performance of the network is shown in figure 7 on the test set. We can see that that the distribution is centered around zero and has only a slight asymmetry into positive direction. The largest values seem to be around $\ln \Delta = 1$ which corresponds to a factor of 2.7 between the targets and predicted values for the amplitudes. We will later show on our own results, that the events with the largest deviation are most likely also part of the divergent areas of the phase-space. Additionally, the we can calculate the differential cross-sections with respect to certain key observables. This can show the robustness of the predictions and the uncertainties, as we are bound to find certain divergent areas in these distributions, which will proof to be the most challenging to predict amplitudes for. However, one should not use these as a primary scale for performance, as the individual binning can lead to vastly different results depending on which events are binned together. If all bins which overestimate the true amplitudes and all events which underestimate the amplitudes are binned together the resulting diagram will look very shaky. However if the events will be binned together in such a way that over- and underestimation cancel each other out, even relatively bad performances can seem very stable. Additionally,

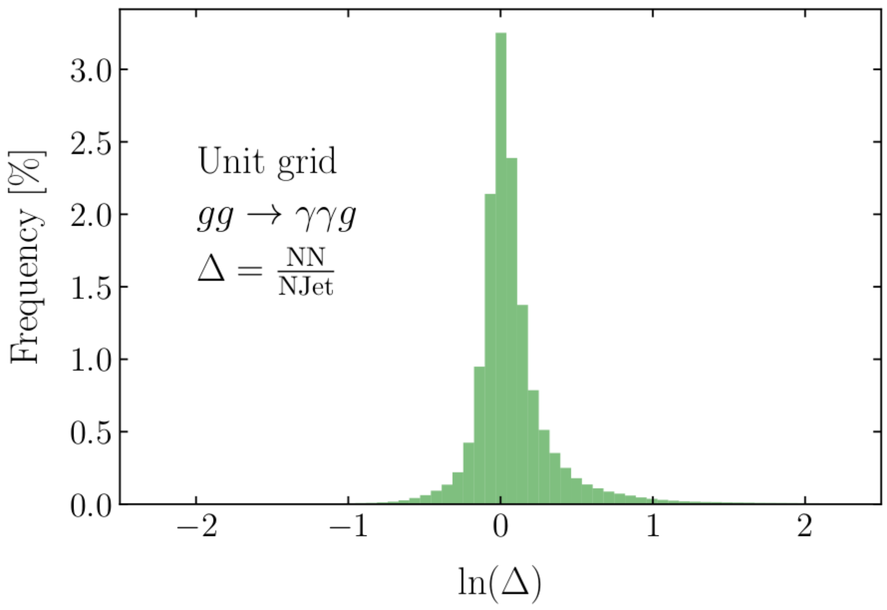

Fig. 7: Performance of paper [18] on the test set

the number of events in a bin strongly changes the total error of the bin, since a single event with very high error has only negligible effect on the total uncertainty, if combined with a large amount of events with low uncertainty. Given all this, the differential cross-sections of the paper are given in figure 8. The paper mentioned two jets in this figure: j_1, j_2 which is due to the fact, that in this paper a anti-k$_T$ jet detection algorithm is used as part of the implementation in a complete framework, which is able to detect several jets from the output data. In our data, this will not happen, such that our "jet" is always just the gluon. In Figure 8 we can see larger errors in regions of different phase differential cross sections. Specifically, we can observe, that the uncertainties and shaking difference of prediction and targets grows larger in the regions of high transverse momenta and low $\Delta\phi_{j_1 j_2}$. This is usually due to few data points in these regions as well as the fact that these regions are often part of the divergent phase space regions. This usually means that in these regions, few events with very large amplitudes make up the majority of the bins, while in other regions, the bins are filled with more, but smaller amplitudes.

3.1.1 Activation functions

As previously mentioned in the chapter 2.3.3 there exist various activation function which can be used for our BNNs. We start by using the splitting of the data which cuts have been applied to into a divergent and non-divergent region according to equation 262728. We then train our non-divergent network with batches of size 4096 and with 100000 epochs. During the training we use ten percent of the training data to compute the validation loss and use this to make sure no convergence is happening. The networks on the divergent phase-space regions are trained only

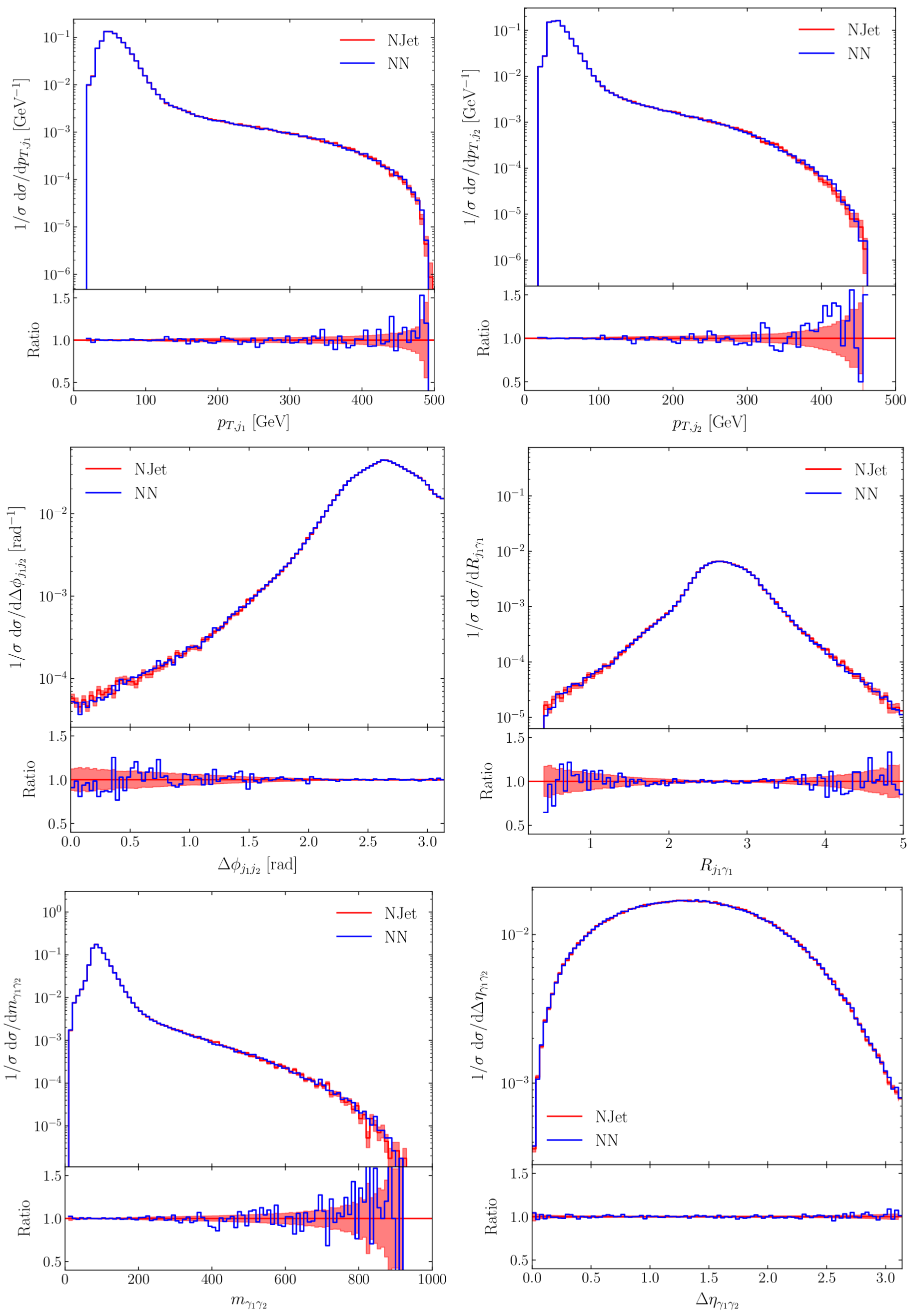

Fig. 8: Kinematic distributions from our benchmark network, Fig. 3 from Ref. [18]

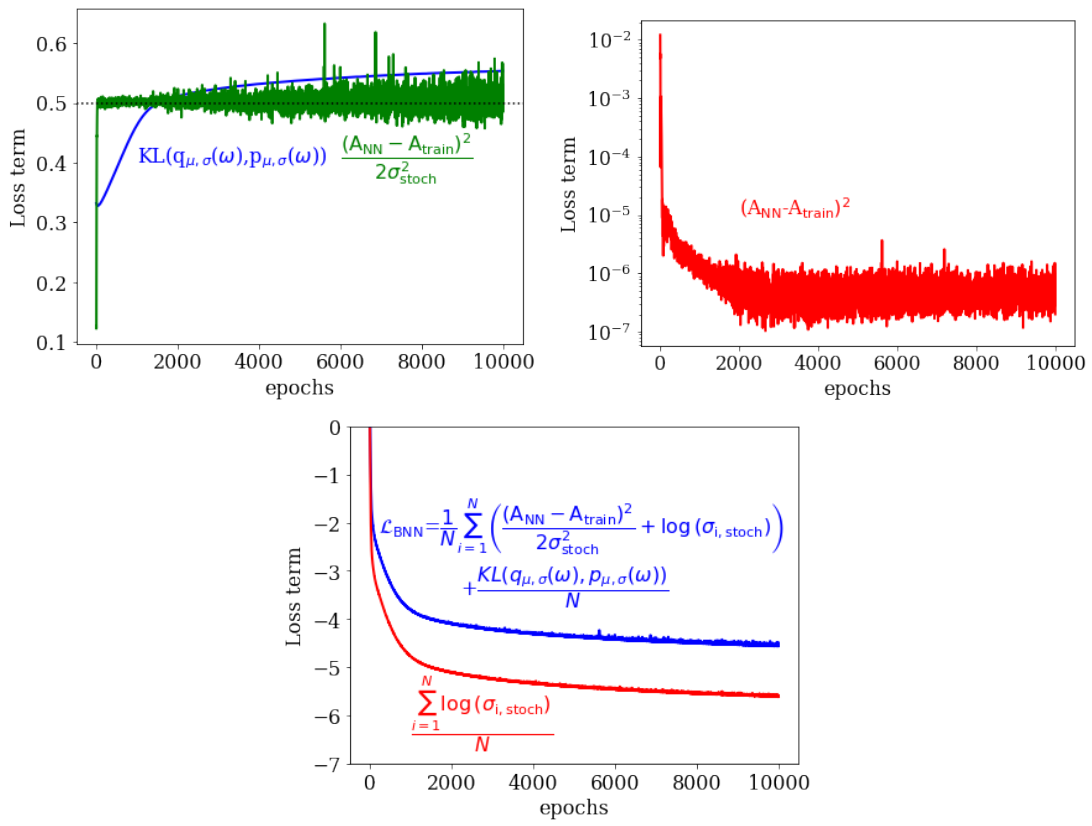

Fig. 9: Exemplary validation loss terms for non divergent BNN with log-preprocessing and Softplus activation function split up into different loss terms. The sum over the batchsize is implicitly contained in terms which contain the the amplitudes, except in the entire loss term, where the sum is explicitly written out

30000 epochs, since the training split after cuts is

$$y_{\text{cut}} = 0.02 \qquad N^{\text{train}}_{\text{div}} = 1099 \qquad N^{\text{train}}_{\text{non-div}} = 31652\ . \tag{62}$$

This means that is is reasonable to assume that the training on the divergent regions will be quicker in comparison. For each of the 11 networks we use the same final activation functions and log-preprocessing to get the following results for the quantity $\Delta^{(\text{train})}$ defined as follows:

$$\Delta^{(\text{train})} = \frac{A_{\text{NN}}}{A_{\text{train}}} - 1 \tag{63}$$

The loss in figure 9 aligns with our expectations: the green line is indicative of the training behavior, that the network very early on begins to match the difference of target and prediction to the according error, such that:

$$\frac{(A_{\text{NN}} - A_{\text{train}})^2}{2\sigma^2_{\text{stoch}}} \to 0.5 \implies |A_{\text{NN}} - A_{\text{stoch}}| \to \sigma_{\text{stoch}}. \tag{64}$$

The total loss term shows that the network will, after fitting the amplitudes to a satisfactory degree, then concentrate on instead training with then intention to decrease the loss. None of these plots show any sign of overfitting.

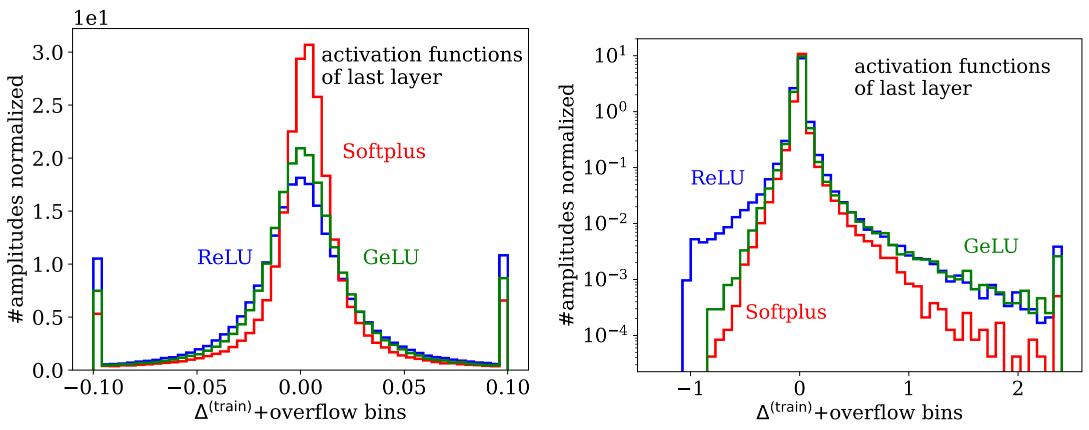

Fig. 10: Performance of BNN networks with different final activation functions with FKS-subtraction method; Left: Linear scale, Right: logarithmic scale

We can see that the performance plots 10 are in general centered around zero, but are asymmetric towards overestimating the amplitudes. This might very well be a artifact of the log-preprocessing, which we will discuss further in the following section. If we instead plot the quantity which as given in the original paper (see figure 7), we get the resulting distribution of figure 11.

Similar to the plot in the original paper, these distributions seem to be more symmetric, apart from the model using ReLU. In the chapter on performance 3.2 we will further investigate why this log-preprocessing is nonetheless preferably compared to a linear preprocessing when considering the gain in performance over the entire phase-space. Both the above plots show us however that the Softplus activation function seems to give the best results and a increase in predictive accuracy as compared to the networks used on the paper (see figure 7). Accordingly we will continue to use in all following models. The ReLU activation function seems to have more occurrences of events being predicted to low, which might be due to the fact that ReLU, other than the other two activation functions has a zero-gradient for negative output. While the amplitudes should all be positive, this might mean that amplitudes which, by initialization, fall into negative areas are difficult to predict properly.

3.1.2 Architecture & Preprocessing

For the networks we are given a input array of 5 particles, each with a 4-momentum, giving us a total of 20 dimensions. We flatten this input and for the inner layers use [20,30,40] dimensions similar to the previous networks in the paper. As discussed in the previous chapter, we will use the Softplus activation function in the last layer, whenever possible. For the preprocessing, we will now examine the difference between the log-preprocessing (equation 21) and the linear preprocessing (equation 20) as seen in figure 2 and figure 5. Since we technically do not learn the σ_{stoch}, but instead $\ln \sigma^2$ because this term is directly contained in the loss function, we only use the final activation function on the amplitude output, A_{NN} to avoid evaluating in on a negative uncertainty. With the linear preprocessing, however, the normalized

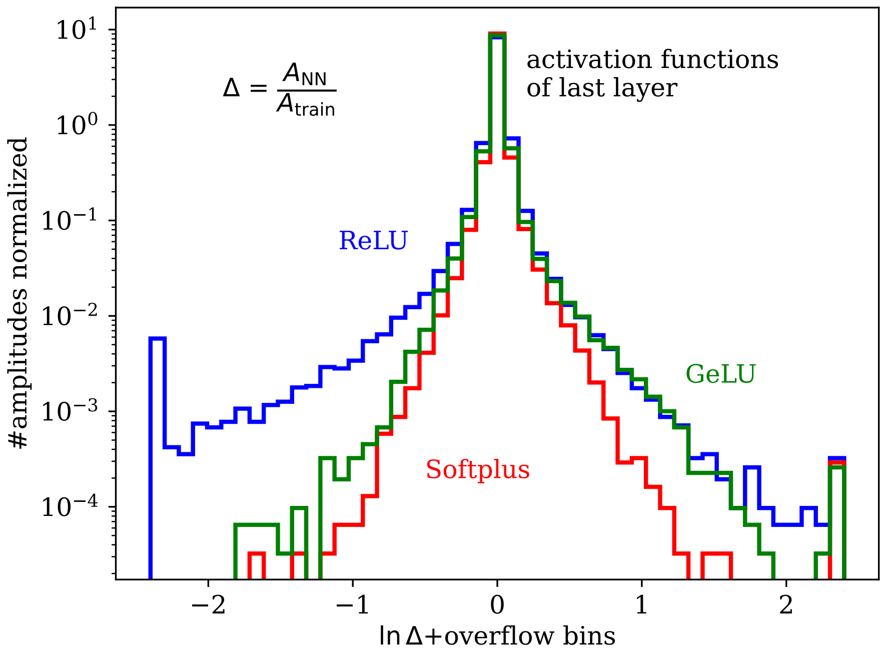

Fig. 11: $\ln\Delta$ Performance of BNN networks with different final activation functions with FKS-subtraction method; same data as in figure 10

amplitudes can also become negative, such that we do not use the activation function on either output. Additionally, we can use two separate networks for the divergent and non-divergent part but do not split up the divergent area using FKS subtraction or we can even just use a single network for both the divergent and non-divergent data. For the latter we use the expanded architecture of size [20,20,30,40] to make sure the network is able to for the now more complex data. This gives the following results of figure 12. As we can clearly see in the figure 12, the architecture and the preprocessing have significant influence on the final performance of the BNN. It would be expected for the network which simply split the data into a divergent and a non-divergent part to perform badly, since the regions of different divergences are in general disconnected and one would expect a fit to be rather difficult, but the method of using FKS pairs seems to be even worse in comparison. Perhaps one could further investigate this with different parameters for the hyperparameter y_{cut}, since this parameters is essential for the training of the divergent area. If it is larger, the the divergent training set is larger, but might include points which do not technically belong in this area. Additionally, with increasing size, the influence on the final performance is of course also increased. If y_{cut} is smaller, however, it has less influence on the final performance of the network, but the quality of the predictions is also decreased since the available training set is smaller. In this thesis I did not investigate this further however, since the topic and the accompanying optimization of this parameter has already been done in Ref. [18] for conventional networks, which lead to a performance which is still worse than the performance on on our BNN training on all data. This network seems to provide the best performance on both the divergent as well as the non-divergent part of the test data. A possible interpretation for the more significant increase in performance on

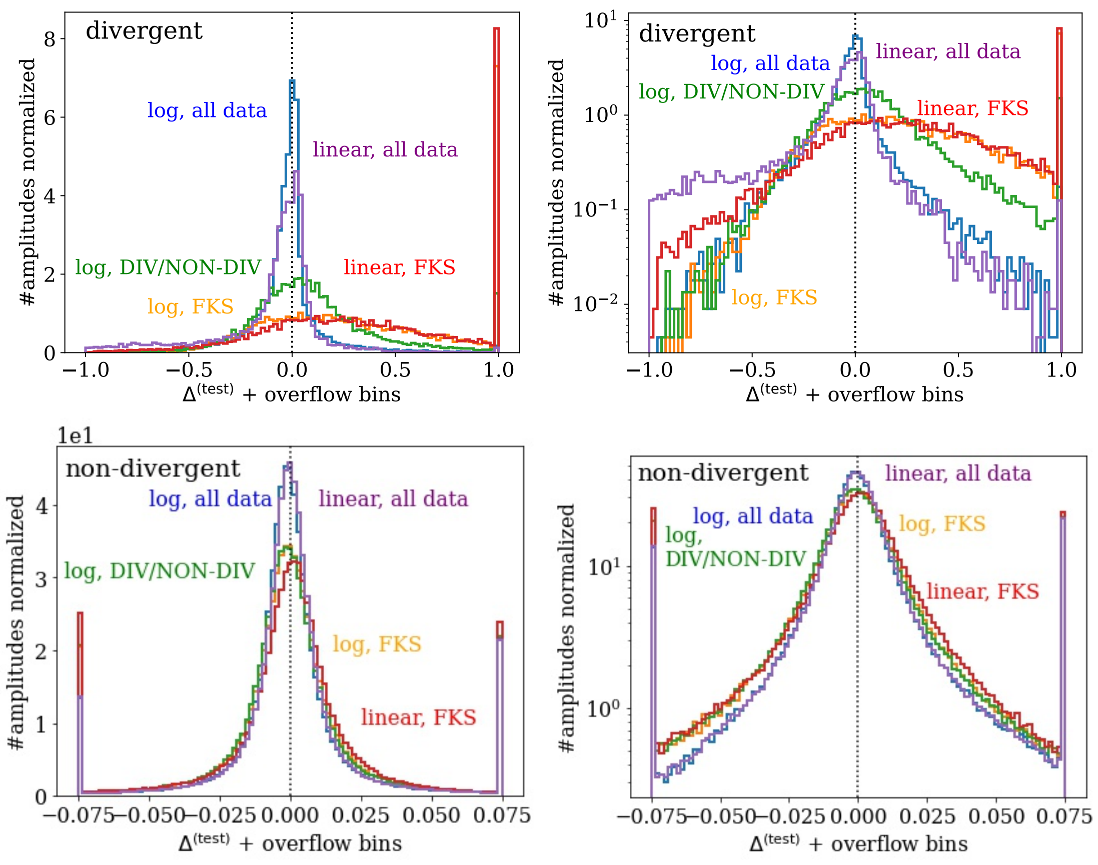

Fig. 12: Performance of BNN networks with different architecture and preprocessing: "log, all data" and "linear, all data" are networks of size [20,20,30,40] hidden layers respectively with a log-preprocessing and a linear preprocessing; "log, FKS" and "linear, FKS" are networks of size [20,30,40] with a log preprocessing and a linear preprocessing respectively and "log, DIV/NON-DIV" and "linear, DIV/NON-DIV" are networks with separate networks for divergent and non-divergent phase-space regions but no FKS-subtraction and log preprocessing and linear preprocessing respectively

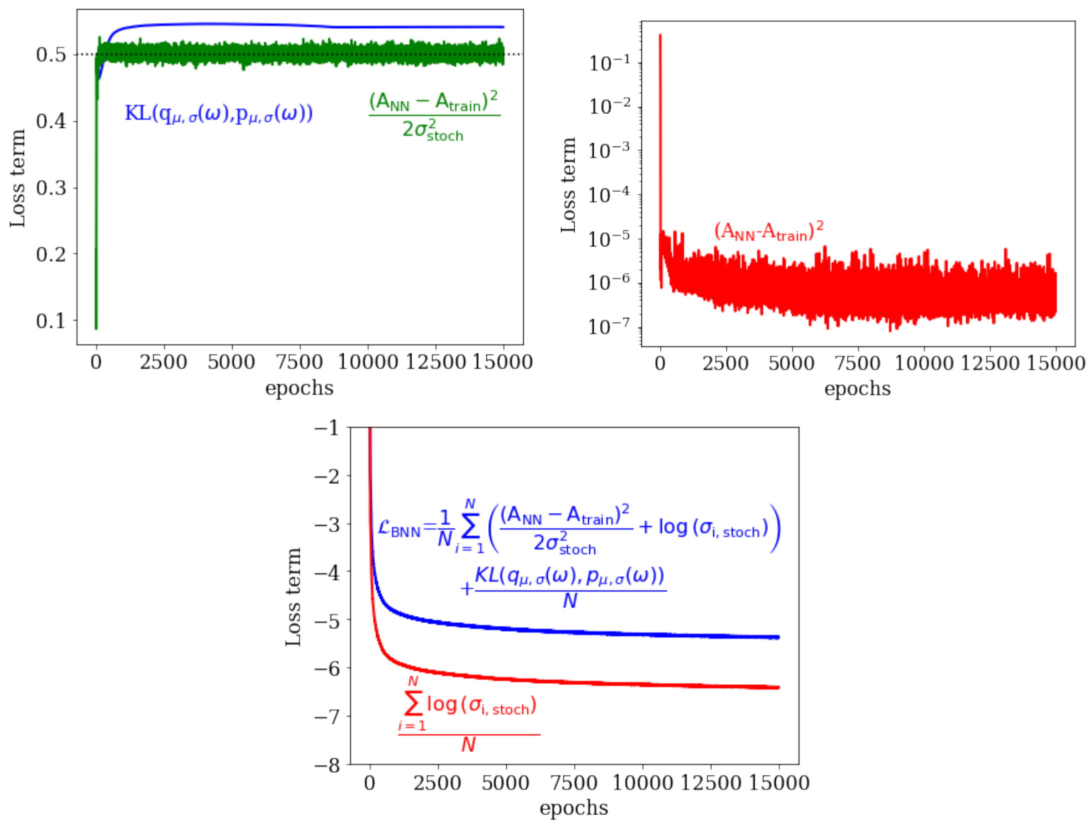

Fig. 13: Validation loss of BNN trained on total training data split up into several loss terms

the divergent data would be, that this network was able to predict these amplitudes better using the context of the non-divergent amplitudes. This would imply that the networks which only had the divergent data to train on might had been better is the hyperparameter y_{cut} would had been chosen larger and therefore less restrictive with respect to the data that would be classified as "divergent". The difference between linear (equation 20) and logarithmic (equation 21) preprocessing is smaller on the non-divergent data, which aligns with our expectations, since the advantage of log-preprocessing is to bring very large amplitudes, which are usually in the divergent region (see figure 3), close to the mean value of amplitudes and therefore make it easier to predict. Since this means that the network on all data is not only better with respect to training time, but also with respect to performance, we opt to use this architecture from not on, paired with log-preprocessing.

3.2 Performance

Our final used network is a single network trained on the entire training data with hidden layers [20,20,30,40], 20 input dimensions, two output dimensions corresponding to the amplitude $A_{\mathrm{NN,i}}$ and the corresponding stochastic uncertainty $\sigma_{\mathrm{stoch,i}}$ and log preprocessing (equation 21). This network has been trained for 150000 epochs with a batch size of 4096, after which no significant further improvement in the training is visible (compare with figure 13). The training is similar to figure 9 and once again it is visible, that the saturation in prediction is reached significantly

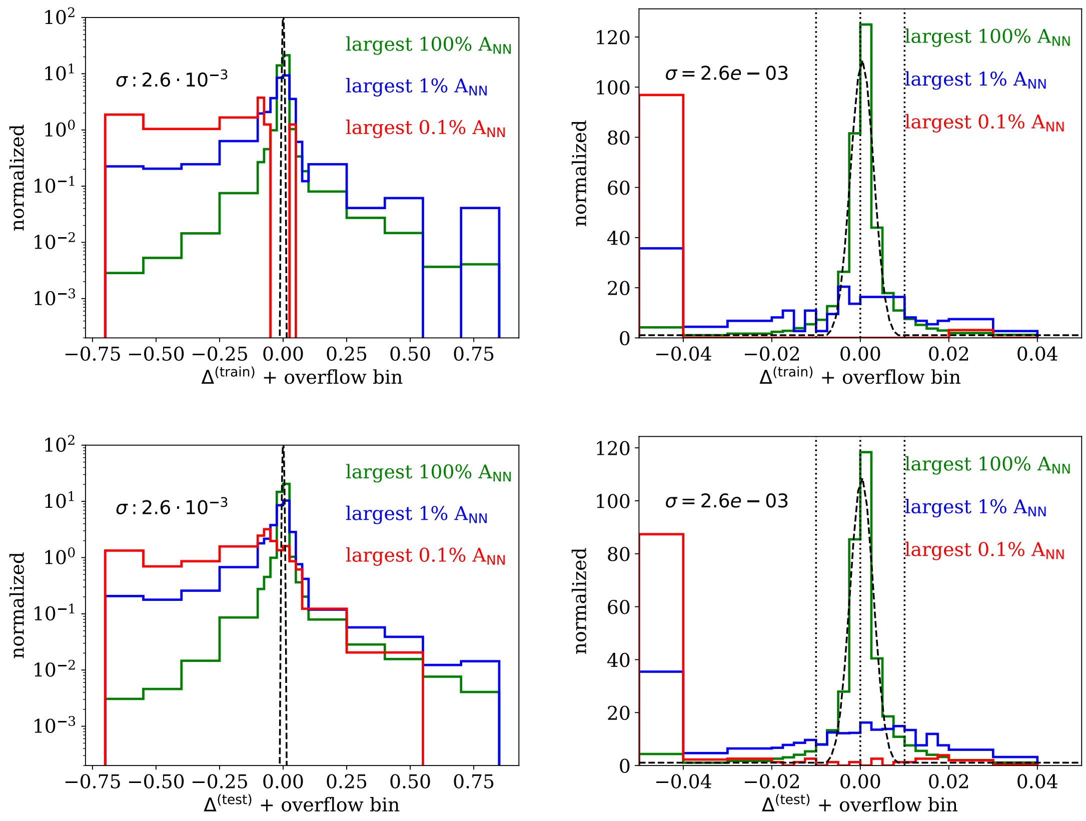

Fig. 14: Performance of the BNN on all data on amplitudes of different size; in the plots there is a added Gaussian distribution for the 100% histogram; the network performs significantly worse on the largest data, which correspond to the part of the data that is contained in the divergent regions

faster than the saturation in uncertainty. To get further insight into the performance of the networks, it makes sense to take a look on the predictive capabilities of the largest amplitudes, as shown in figure 14. We can clearly see, that the performance for larger amplitudes is worse than for smaller amplitudes, which is to be expected, since the large amplitudes are usually then ones in the divergent region which are harder to fit. This is also the time to take a closer look at the pull (see chapter 2.4.1) of the BNN as shown in figure 15 We can see that there seems to be systematic upwards shift which corresponds to a tendency of the network to slightly underestimate the amplitudes, although the execution over several runs shows that this shift can change from run to run and does not seem to have a have a physical background. As we would expect, the alternative pull has standard deviation of approximately one, but deviates from a Gaussian distribution by having tails both in positive as well as in negative direction. This means that there are still events which are not well enough contained in the uncertainties given by the network. Similar effects apply to the total and predictive uncertainty, although neither the predictive uncertainty, nor the stochastic uncertainty are expected to contain the entire uncertainty. The shift in the fitted Gaussian on the predictive uncertainty is due to the fitting of the large width of it, there is no reason to expect a shift in

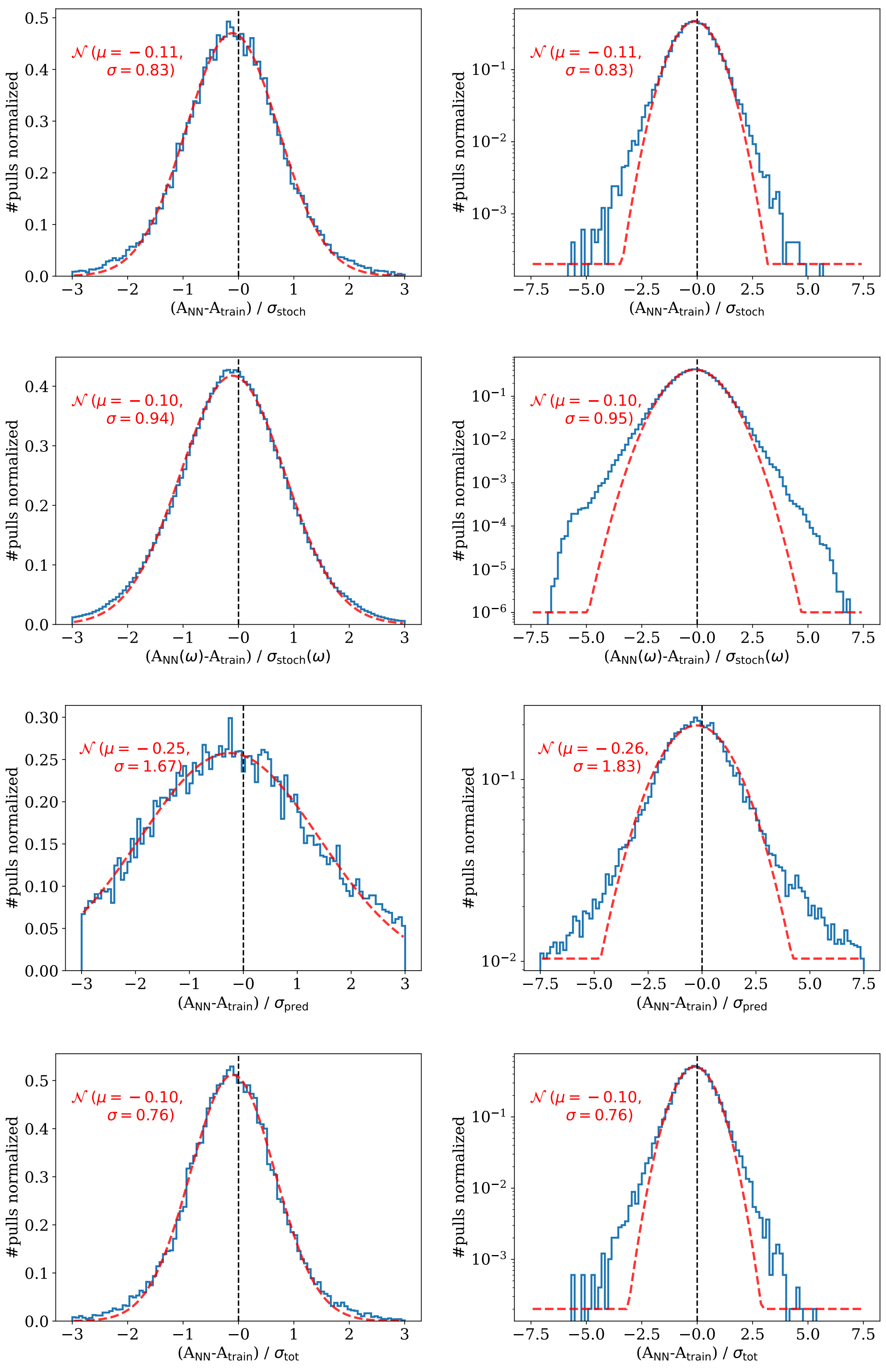

Fig. 15: Different pull quantities of according to chapter 2.4.1; the second row with explicit ω dependence correspond to the alternative pull definition of equation 57, while the other rows correspond to the pull definitions in equation 52

this distribution. The total uncertainty σ_{tot} is larger than it needs to be for the majority of events, such that the standard deviation of the pull is smaller than one. In the chapter on feedback training, 3.5 we will attempt to explicit deal with the tails in these distributions.

3.3 Uncertainties & less training data

As we have already discussed before in chapter 2.4, we get two different uncertainties, σ_{stoch} and σ_{pred}. We will take a closer look at them in this chapter.

3.3.1 Correlations

We can firstly take a look at the correlation between σ_{stoch}, σ_{pred} and the pull defined in equation 52. We can see that in figure 16 that there is a correlation between the uncertainties, although not a strict one. We can see, for example, that in the left correlation plot there are no points where σ_{pred} would be very large while σ_{stoch} is very small. This shows that the uncertainties do not seem to be entirely independent and most likely both depend at least partly on the only source of statistical uncertainty, the statistics of a limited training set. Additionally, we can see that neither error goes all the way to zero. This is due to the clipping we use on the standard deviations of the weight distributions for increased numeric stability. The fact that the predictive uncertainty is significantly smaller than the stochastical uncertainty is not uncommon for BNNs which have been sufficiently trained. We will see a similar result in chapter 3.3.2. While there is definitely some correlation visible in figure 17, we can see that there is a large portion of events which do not follow a linear correlation. This is particularly clear when looking at the stochastical uncertainty with respect to the amplitudes. In this case the correlation is rather a condition which prohibits points in certain areas, than a strict linear condition. This will be important in chapter 3.5, since we use σ_{stoch} there to find points to duplicate/weight more in the loss function. If there was a strict linear relation between this uncertainty and the amplitude, we would not need the stochastical uncertainty at all and could just take the highest amplitudes

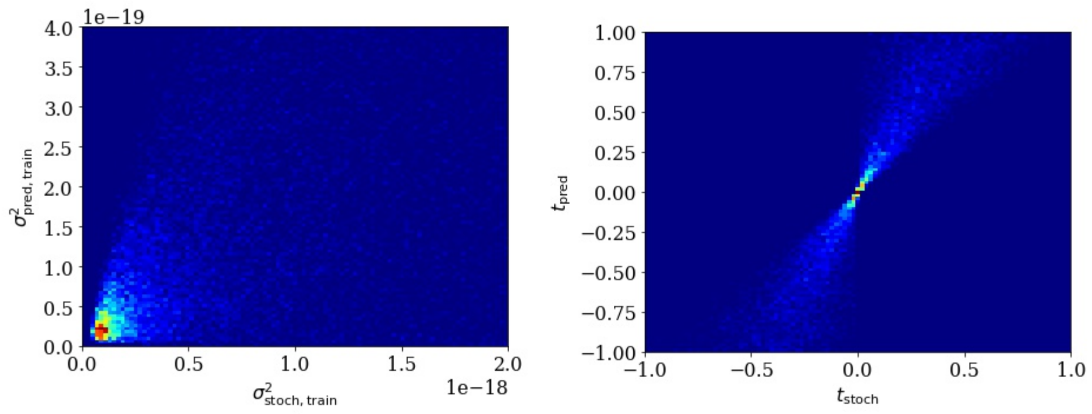

Fig. 16: Left: Correlation of σ_{stoch} and σ_{pred}; Right: Correlation of pulls defined in equation 52

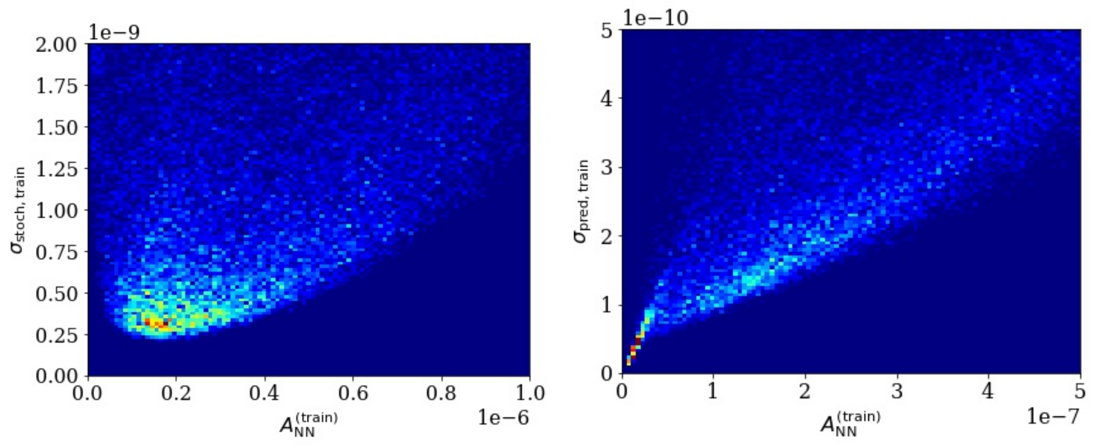

Fig. 17: Left: Correlation of σ_{stoch} and training amplitudes of the BNN; Right: Correlation of σ_{pred} and training amplitudes of the BNN

instead.

3.3.2 Less Data

We have already mentioned, that σ_{pred} is defined in such a way that is is become smaller as the size of training data is increased, since the weight distributions will ultimately become delta distributions. We can explicitly examine this when we use only a part of the training data for the training of the network. Of course there will be a certain bias, depending how we choose this data, since some areas are harder or easier to fit. To decrease the dependence on the amount of hard-to-learn amplitudes, we are only using data from the non-divergent area here. Nonetheless, a clear tendency is visible in figure 18. The first couple values of σ_{stoch} in this figure are significantly higher than the others, but it should be considered, that the performance in these regions already heavily depends on the specific subset of data that was used (see figure 19). This is most abundant for the BNNs with less than 5% training data. The quadratic sum of the uncertainties, i.e. σ_{tot}, then adds up to the MSE, while we can clearly see that the prediction uncertainty decreases as expected, while the stochastic uncertainty stay approximately constant with smaller training size.

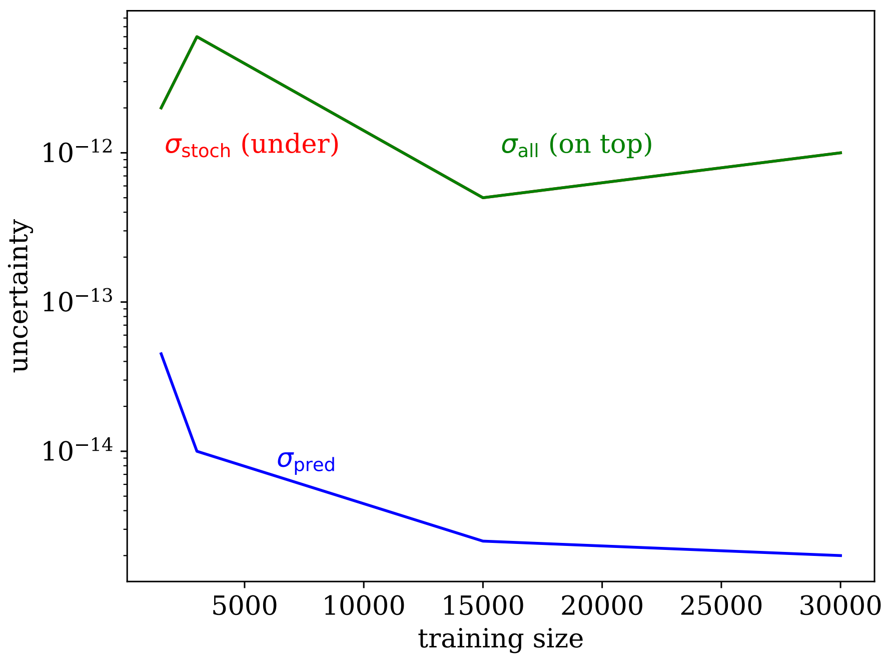

Fig. 18: Mean uncertainty of a BNN trained on non-divergent phase-space region when training with smaller training dataset; the total error is equal to the MSE

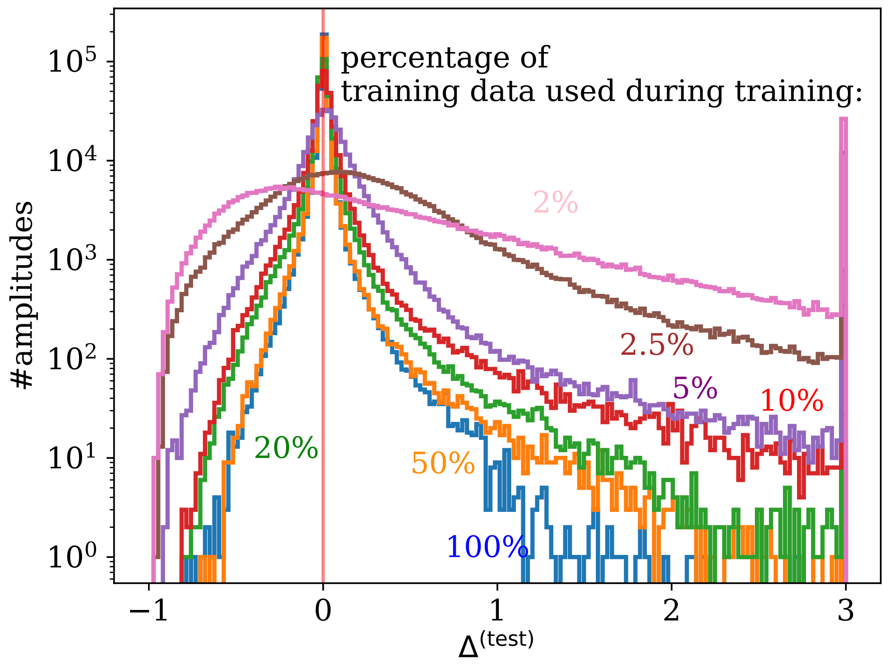

Fig. 19: Performance of a BBN trained on non-divergent phase-space region with lower training size

3.4 Kinematic distributions

While the kinematic distributions i.e. the differential cross-sections can sometimes be misleading due to the strong dependence on chosen binning and the amount of events per bin, they still give insight about critical phase-space regions. The deviations in specific regions of chosen physical observables can furthermore line out the divergent regions and predictions which need can be improved by feedback training (see. chapter 3.5). We will see, that the prediction around $\eta_j \approx 0$ are decent, however the predictions in the outer region of this plot are significantly worse. This is partly due to the few points in these regions (as can be seen by the grey bars in the second and third panel), and also the worse predictions of the large amplitudes in these areas. We have already shown this effect in the performance plots (compare figure 14). Next, we can examine the distributions for the network, which was trained on all data: There are still some outliers which have a strong impact on the quality of the distributions. In the further plots, we will only show the total uncertainty σ_{all} for better visibility. Apart from the peaks, one can also observe some systematic differences between prediction and targets, most notably the region of small η_j and the region of large $R_{\gamma 1,j}$ in which both training and test predictions are systematically lower than the targets. These areas are part of the divergent regions and one of the goals of the feedback training in chapter 3.5 will be to deal with these regions. We can furthermore take a closer look at the prediction on the divergent and non-divergent area alone in figure 21, 22. We can see that the peaks come mainly from the divergent, but partly also from the non-divergent regions. In both regions, there the total error, however, is able to cover large deviations of the prediction to the targets. Additionally, we are able to see some interesting areas: In the non-divergent distributions (see figure 21), the systematic difference in the area of low η_{j} is not there anymore, the targets now align significantly better with the predictions of the network. This shows clearly, that the distribution which is fitted by the network in figure 20 are resembling the non-divergent distributions. This will be addressed in chapter 3.5. We can now try to isolate and examine the events which cause the peaks to do further analysis on them and the reasons for the caused peaks. If we filter only 16 events of the test data (we choose the events with highest σ_{stoch}), we get the distributions in figure 23. Here we are able to decrease the amount of peaks significantly, however, this also means, that problematic kinematic regions, like the low η_j or the high $R_{\gamma 1,j}$ areas are still not good fits. The filter we used here was simply to filter out the events with the highest σ_{stoch} which is risky, since the correlation between large amplitudes and large errors means that the events with large uncertainties often also have high amplitudes, making then particularly important in the kinematic distributions. Instead of filtering events, we can rather use the fact, that the training data does not have a physical uncertainty, such that σ_{stoch} can theoretically become zero, if the BNN manages to fit them perfectly (interpolation). Since the network does instead fits them, however, we can increase the weight of these events in the loss function and thereby make the network put more priority on fitting these events more precisely then other points i.e. decrease σ_{stoch}. This will be done in chapter 3.5.

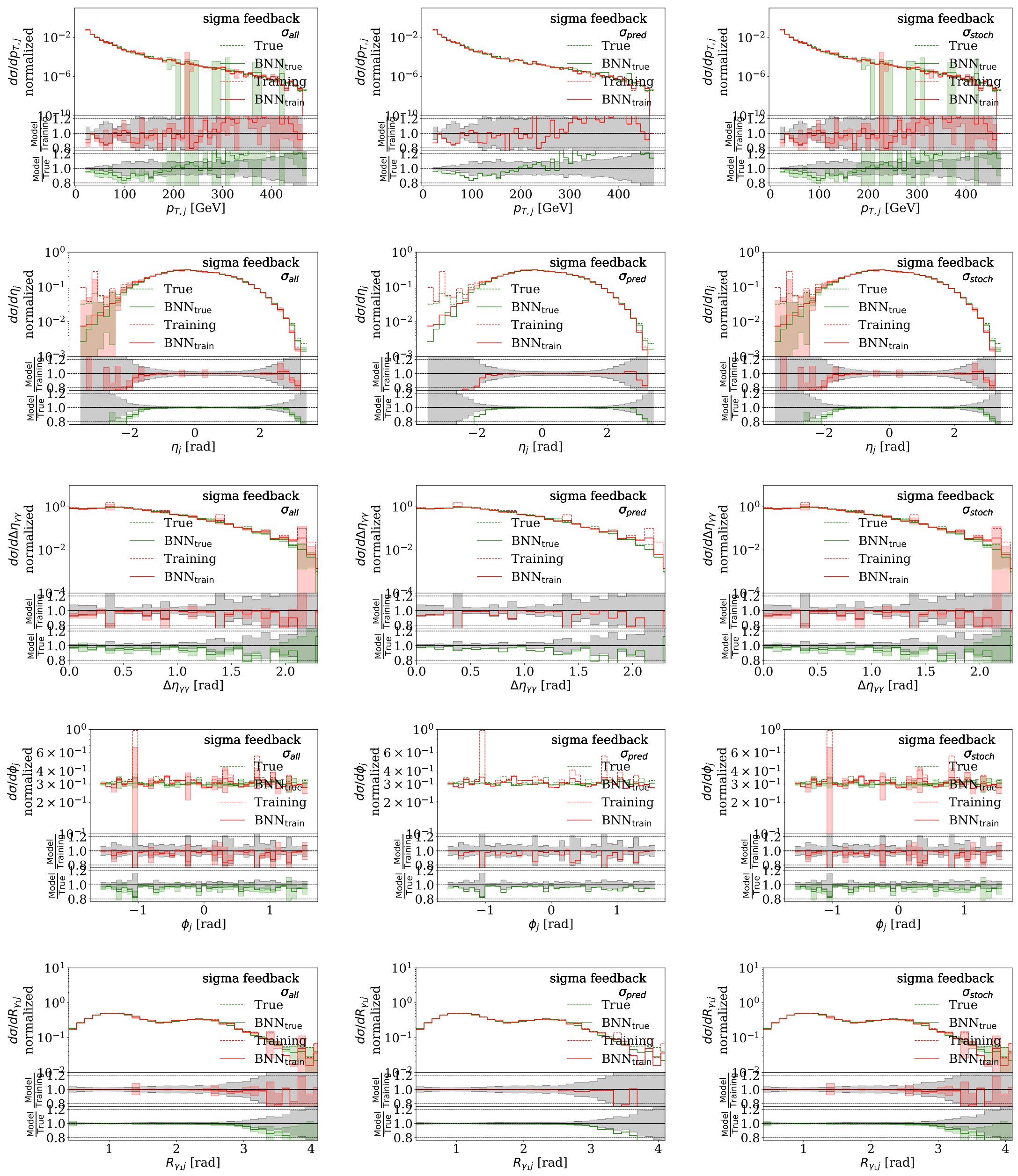

Fig. 20: Kinematic distributions of BNN with all data used for training; left column: σ_{all}, middle column σ_{pred} , right column: σ_{stoch}; first panel shows the histograms of cross-sections of the testing sample ("True"), the training sample "Training"), the BNN evaluated on the Training Sample ("BNN$_{\text{train}}$") and the BNN evaluated on the test sample ("BNN$_{\text{test}}$"); the next two panels show the predictions on training and test set respectively, normalized to the targets. The gray bars denote the statistical error of the targets

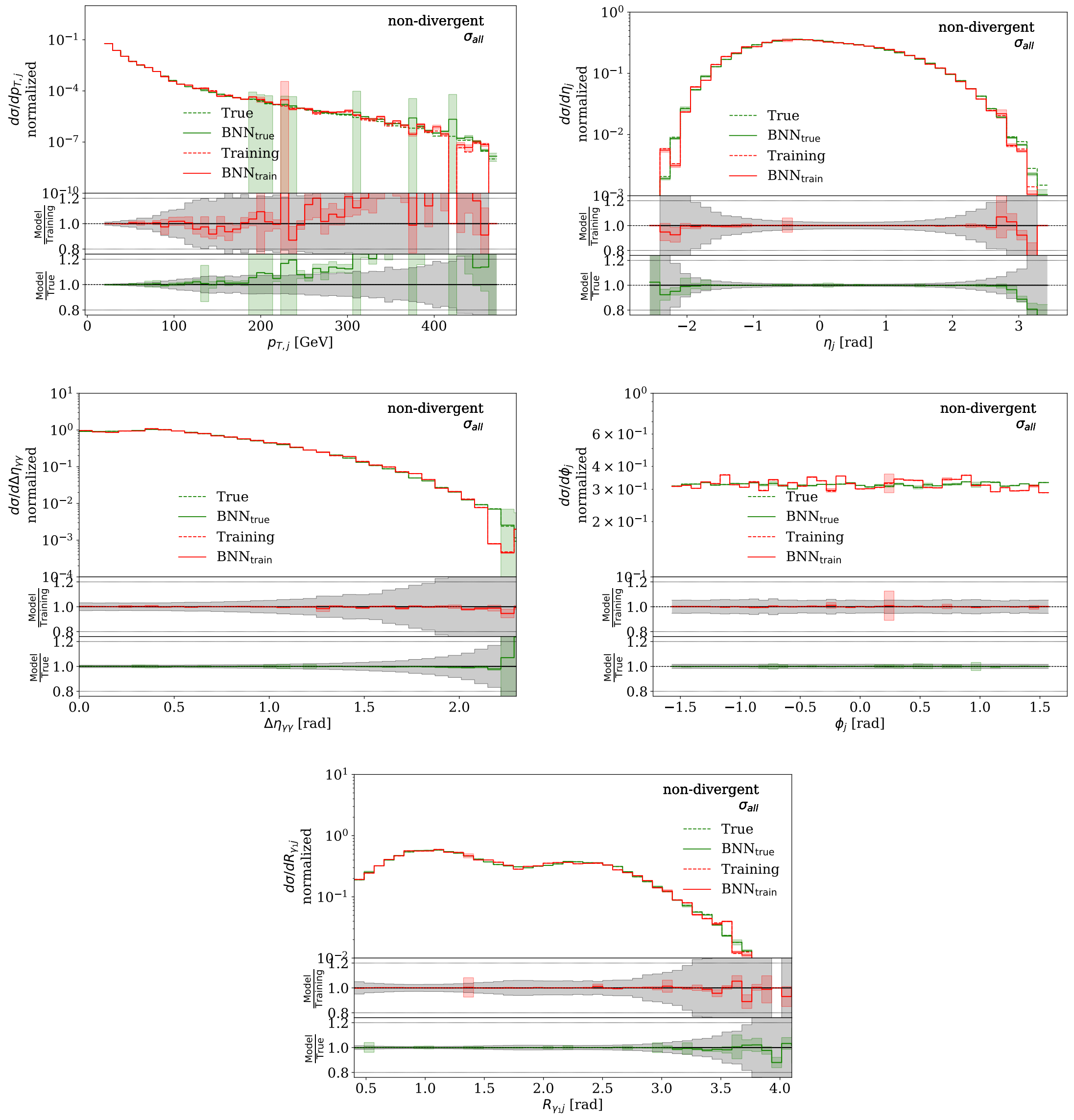

Fig. 21: Kinematic distributions of BNN with all data used for training; evaluated only on non-divergent data; first panel shows the histograms of cross-sections of the testing sample ("True"), the training sample "Training"), the BNN evaluated on the Training Sample ("BNN$_{\text{train}}$") and the BNN evaluated on the test sample ("BNN$_{\text{test}}$"); the next two panels show the predictions on training and test set respectively, normalized to the targets. The gray bars denote the statistical error of the targets

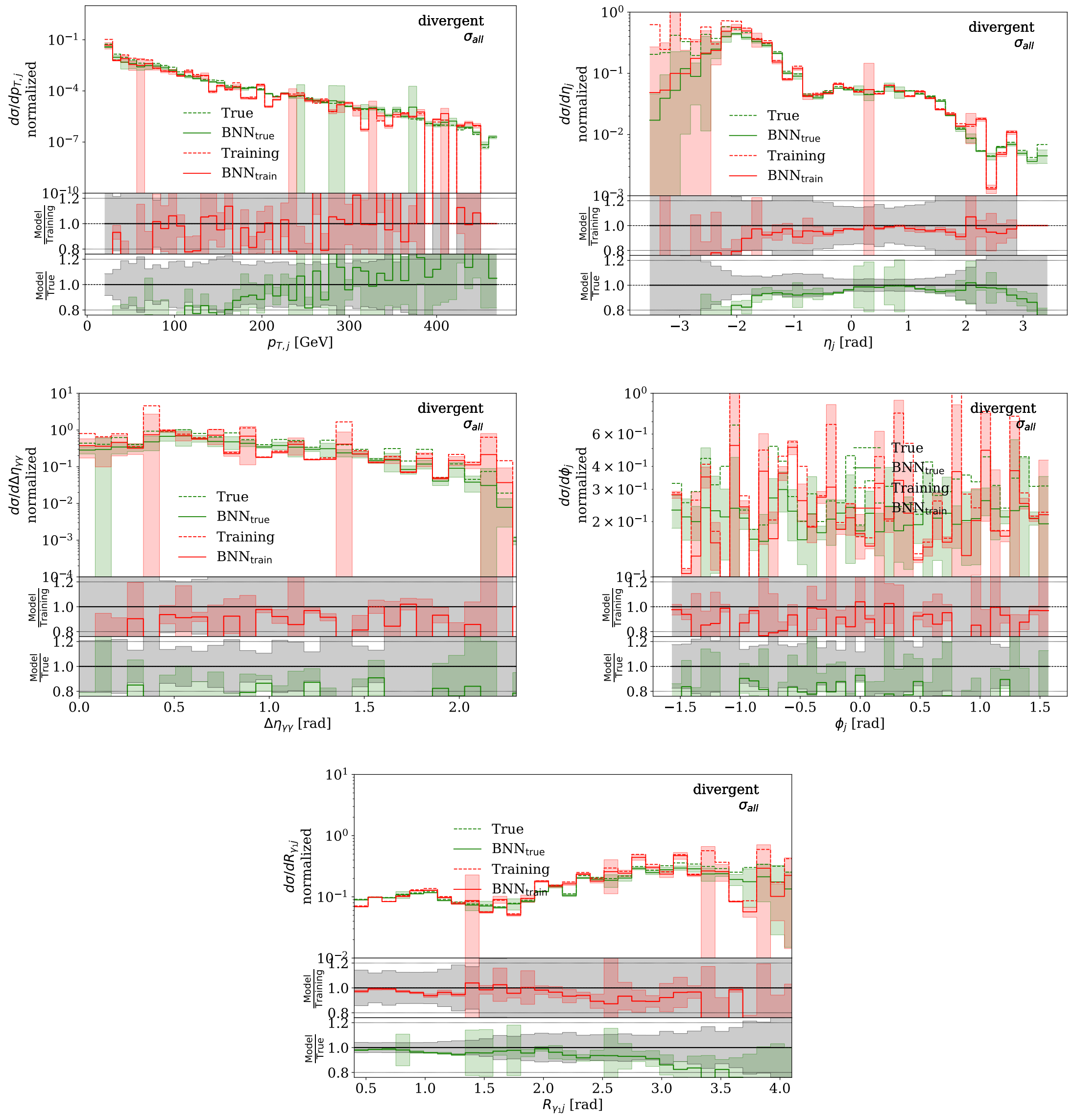

Fig. 22: Kinematic distributions of BNN with all data used for training; evaluated only on divergent data; first panel shows the histograms of cross-sections of the testing sample ("True"), the training sample "Training"), the BNN evaluated on the Training Sample ("BNN$_{\text{train}}$") and the BNN evaluated on the test sample ("BNN$_{\text{test}}$"); the next two panels show the predictions on training and test set respectively, normalized to the targets. The gray bars denote the statistical error of the targets

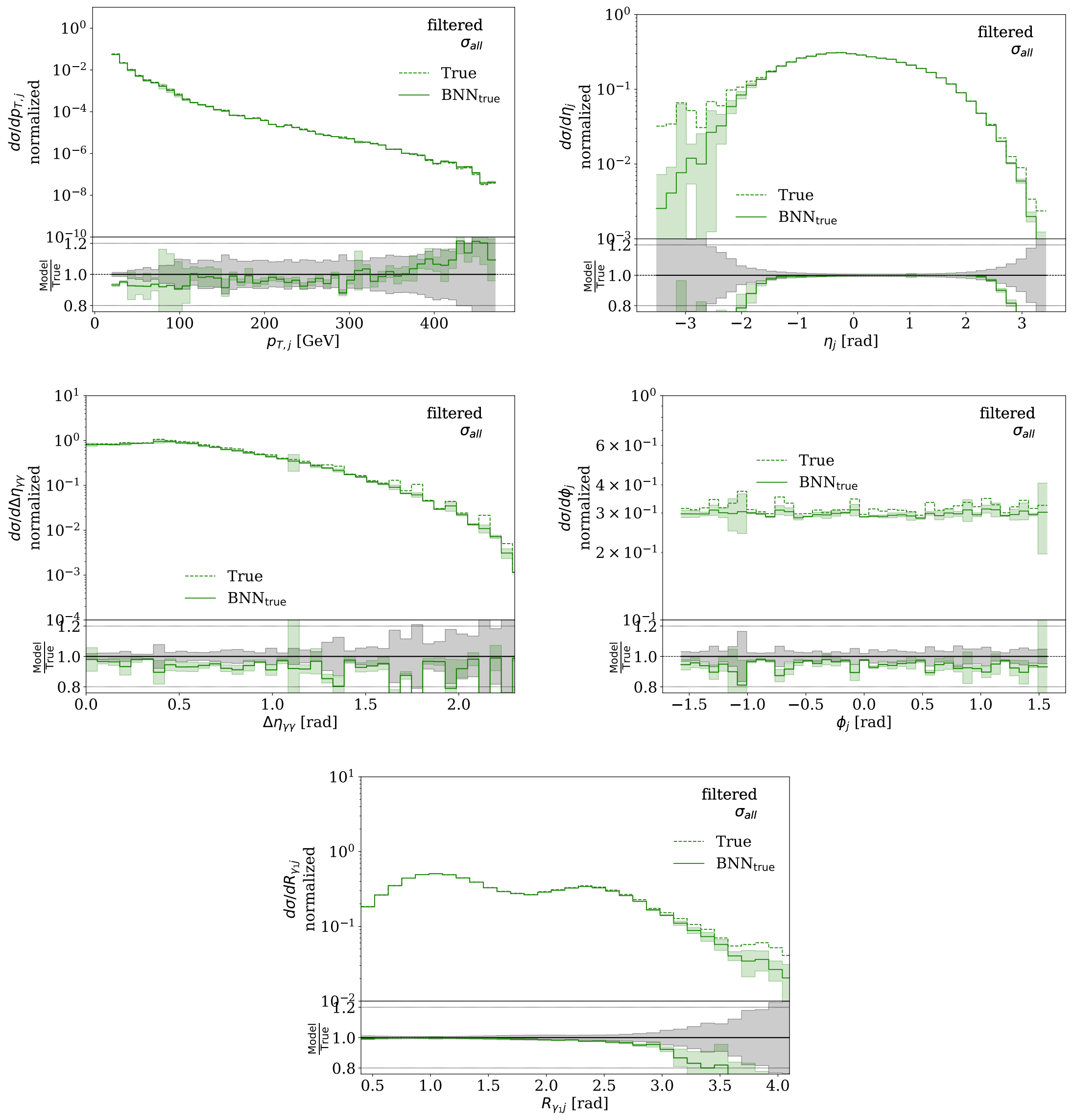

Fig. 23: Kinematic distributions of test set of BNN with all data used for training; first panel shows the histograms of cross-sections of the testing sample ("True"), the training sample "Training"), the BNN evaluated on the Training Sample ("BNN$_{\mathrm{train}}$") and the BNN evaluated on the test sample ("BNN$_{\mathrm{test}}$"); the next two panels show the predictions on training and test set respectively, normalized to the targets. The gray bars denote the statistical error of the targets

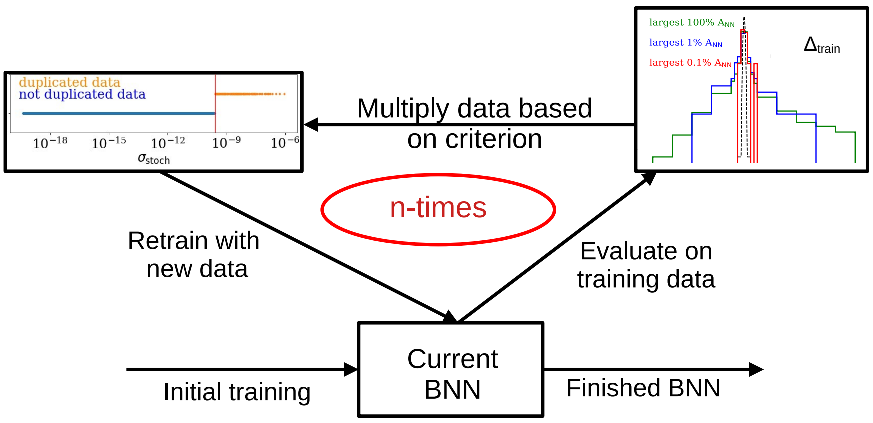

Fig. 24: Feedback loop method; duplicated data increases the weight of the data in the loss function, the criterion can be chosen freely (e.g. the training points with the largest σ_stoch)

3.5 Boosted/Feedback Training

Following chapter 3.4, we can retrain events with high σ_stoch with increased weight in the loss function to get rid of both the peaks, as well as the differences of predictions and targets of large amplitudes, which are the cause of systematic problems in the kinematic distributions (see figure 3.5). For this we retrain the network 20 times, each time evaluating it on the training data and reweighting the 200 events with the highest σ_stoch five times. The trainings consist of 2000 epochs with a batch size of 4096 each. In practice, the chosen data points are simply added several times back into the training set. Therefore they will appear multiple times in the loss function, giving them increased weight in the computation of the gradient. This method is similar to methods (AdaBoost) used for boosted decision trees (see Ref. [48]). This allows us to create the following plots in figure 25. In figure 25 we can see that all almost all peaks have been removed, as well as the systematic differences between prediction and targets. The first kinematic distribution has the worst performance, which in part might be due to the fact that in this distribution the large majority of points are contained in the first couple bins, such that the statistical error of the training data is fairly big for the bins with larger amplitudes. We also see, that the performance on the training sample is significantly better, while the performance on the test set still has some areas of larger deviation, while still being significantly better than on the previous set. It is also notable that the performance is well within the statistical uncertainty of the training data and even the test data. This also means that the strong increase in performance on the test set does not imply overfitting, but rather simply a better fit that is closer to the interpolation. The same result can be obtained by taking a closer look at the performance on high amplitudes in the test and the training set: We can clearly see that in comparison to figure 14, the distributions in figure 26 of the training sample have a significantly narrower peak, such that we can even observe the 0.1% highest amplitudes. In comparison, the performance on the test data does not undergo a increase of the sample magnitude, but is nonetheless better. Particularly, we can observe that the 0.1 % highest amplitudes are not significantly underestimated anymore, which

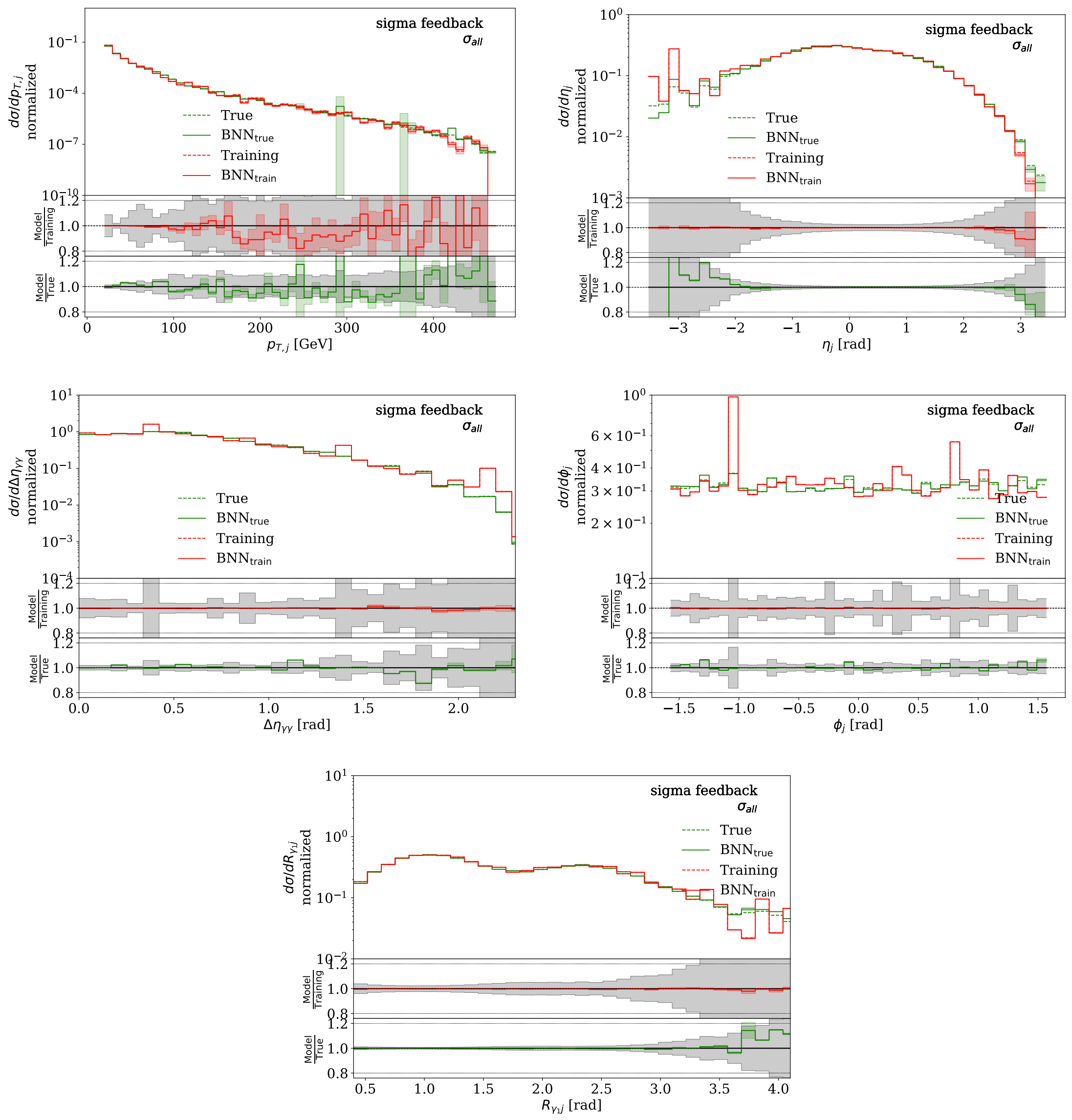

Fig. 25: Kinematic distributions of test set of BNN with all data used for training after feedback on 200 highest σ_{stoch}; first panel shows the histograms of cross-sections of the testing sample ("True"), the training sample "Training"), the BNN evaluated on the Training Sample ("BNN$_{\mathrm{train}}$") and the BNN evaluated on the test sample ("BNN$_{\mathrm{test}}$"); the next two panels show the predictions on training and test set respectively, normalized to the targets. The gray bars denote the statistical error of the targets

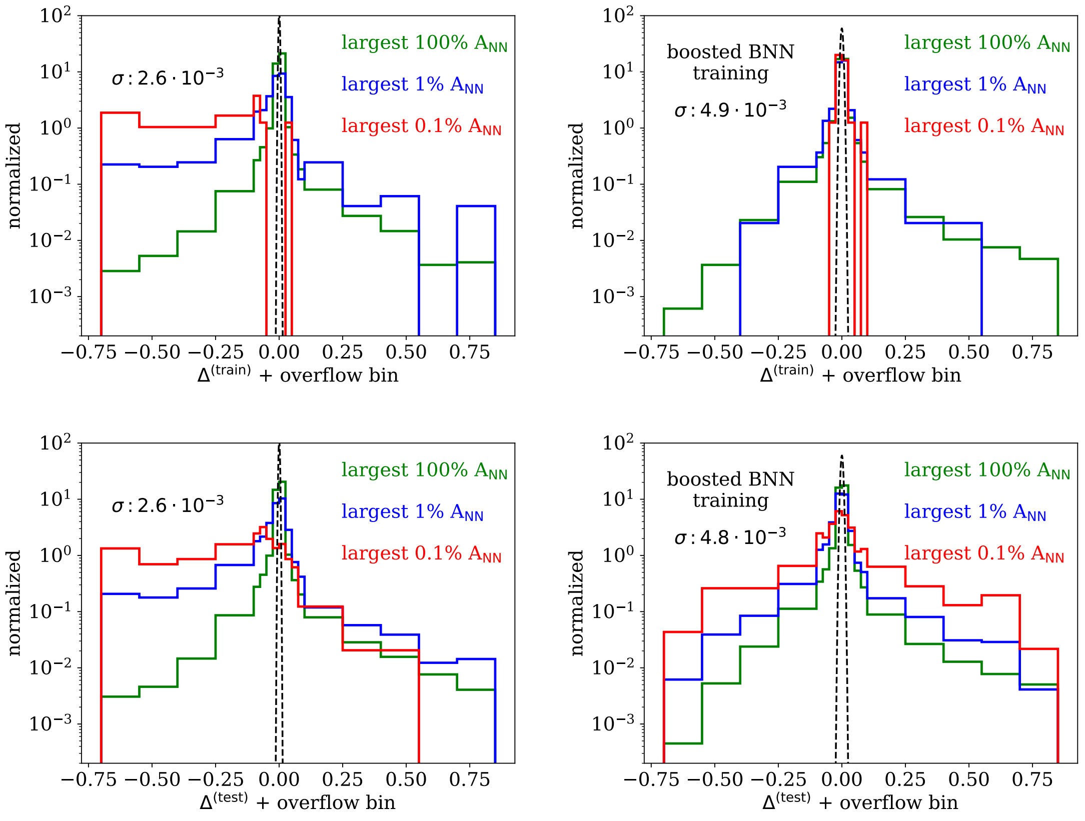

Fig. 26: Performance of a BNN trained on all data (left) and after a feedback training with the events of the highest σ_{stoch} weighted 5 times as much in each run for 20 runs with 2000 epochs of batch size 4096 each (right); Gaussian fitted to 100% data histogram (left plots from 14)

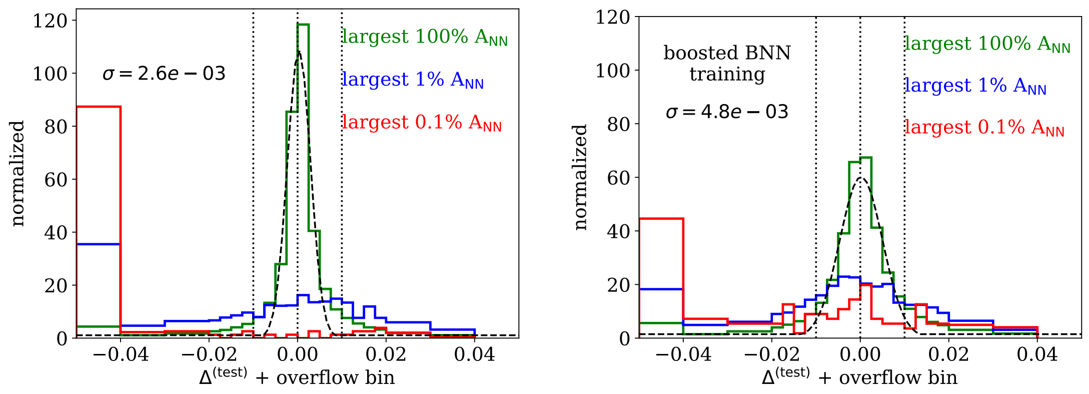

Fig. 27: same as 2nd row in figure 26 with linear scale: Performance of a BNN trained on all +data (left) and after a feedback training with the events of the highest σ_{stoch} weighted 5 times as much in each run for 20 runs with 2000 epochs of batch size 4096 each (right); Gaussian fitted to 100% data histogram (left plots from 14)

corresponds to the better performance in the problematic divergent regions in the kinematic distributions. The peak of the 0.1% highest data is not also more defined (compare figure 27), which seems to come at the cost of decreasing the peak of total data. This can be interpreted as the fact, that the network lays more weight on fitting the high amplitudes and thereby decreases the accuracy of the other predictions. We have chosen 20 loops, since the performance did not get better after this point. With this feedback training, there is no visible advantage in the "pull" distributions compared to 15. Another approach is given, when we do not multiply the training data with the highest σ_{stoch}, but instead the events in the tails of the σ_{stoch}-pull distribution (second row in figure 52, equation 57). The resulting pull distributions have a shape significantly more accurate to a Gaussian distribution (see figure 28). Here we multiply each events will more than two standard deviations from the mean value of t_i^{alt} (from equation 57) five times for a total of 10 iterations.
Most notably seen in the logarithmic plots, the distributions have become more in accordance with the Gaussian distribution. The predictive distribution has become wider, while the other distributions have become less wide. The total pull (last row in figure 28) has also become more similar to a Gaussian distribution, with a standard deviation smaller than one. This means, that the total error is larger than it needs to be, when considering the difference between prediction and targets. While this is not perfect, it is certainly better than the opposite. It should once again be mentioned here, that neither the stochastical, nor the predictive uncertainty are expected to cover the total difference between prediction and targets, such that it is no problem, that their standard deviation is unequal one. This feedback training has no influence on the performance of the network (e.g. distributions as in figure 26), however. The naive approach of simply doing a feedback training of the first kind (multiply points with high σ_{stoch} to increase performance and then on the second kind (multiply points in the tails of the pull distribution) to make the pull more Gaussian does not work, since the second feedback training will undo the progress of the first.

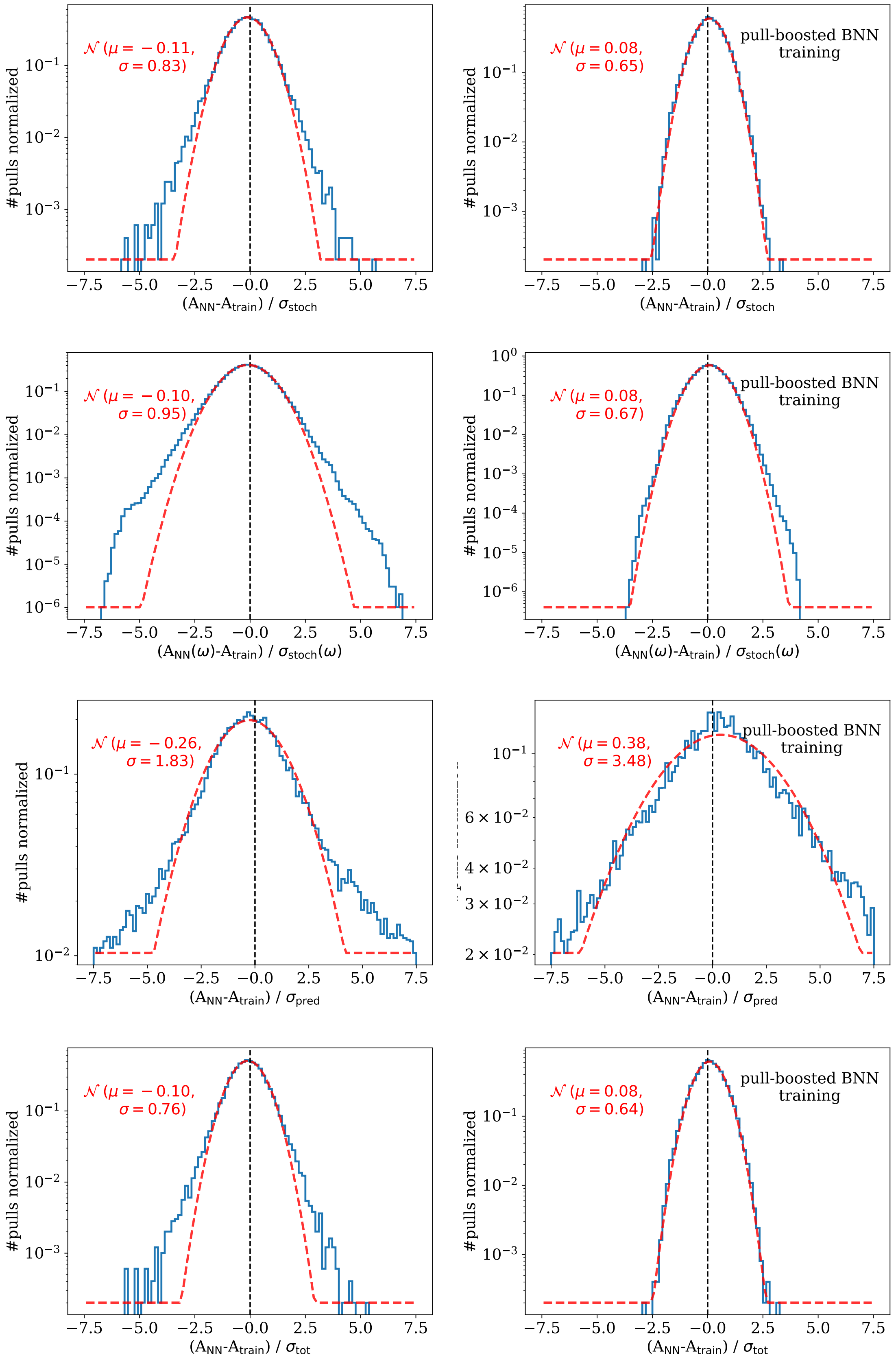

Fig. 28: Different pull quantities of according to chapter 2.4.1 before (left) and after (right) a feedback training has been applied where each event of the training set with a $\sigma_{\rm stoch}(\omega)$-pull deviating two standard deviations from the mean value are multiplied five times each feedback loop iteration for 10 iterations; the second row with explicit ω dependence correspond to the alternative pull definition of equation 57, while the other rows correspond to the pull definitions in equation 52. (same distributions without feedback in figure 15)

4 Conclusions

In this thesis, we firstly began by expanding on the network architecture of Ref. [18] (results in chapter 3.1), which is used to predict scattering amplitudes (see chapter. 2.1.1). Specifically, we were able to replace the original costly ensemble of conventional neural networks by a Bayesian Neural Network (see chapter 2.4). We tested the performance on this new approach with various different final activation functions in chapter 2.3.3 and different architectures in chapter 3.1.2 to find the optimal model for our purpose. This led us from the approach of a dataset, which is being split into regions of distinct divergences (see chapter 2.2) to a BNN which is able to train on the entire dataset (compare chapter 3.2). The performance of this network could be measured both in the network output over the targets as well as in the defined quantity "pull" (definition in chapter 2.4.1). In this setup, we made different observations on the correlation of the uncertainties and their behavior when decreasing the training size (compare chapter 3.3). As an additional measure of performance, we took a look at the kinematic distributions of our network (see chapter 3.4), which allowed us to identify problematic areas. When compared with the performance of Ref. [18] our results were more accurate, but had problems fitting the highest amplitudes, which constitute the divergent regions (see chapter 2.1.2) in large parts. To fix this, we introduced a recursive feedback loop, a method used during training, in which certain training data is chosen and weighted stronger in the loss function. The network is then retrained with this new weighting until it is again evaluated and problematic data is once more reweighted/duplicated. This process is repeated, until better performance is reached (see chapter 3.5). The results showed the impact of this method: The training data can be fitted very accurately and significantly better than the fit before the feedback training was applied. On the test set, we found that the increase in performance is less than on the training set, but the problematic divergent regions nonetheless have significantly improved results. Alternatively, we can use the feedback method to make the pull-quantity more consistent with a Gaussian distribution, but this method did not increase the performance of the BNN.

4.1 Perspectives

The conditions, on which the events to reweight during the feedback training are chosen, can be varied. Possible candidates are the different pulls, the relative uncertainties etc. We have tried out some of them, which showed that they only had mediocre impact on the total performance. Further analysis could be done on the effects any of them (or the combinations of multiple) have on the performance. To give a example: If we apply feedback training on the by duplicating then points of large σ_{stoch}-pull, we bring the pull distributions in better accordance with a Gaussian distribution, however, we do not increase the performance of the network. On the other hand, using the current feedback training, where points with large σ_{stoch} are weighted stronger in the loss function, does increase the performance of the network, however, leaves the pull distributions at their previous shape. Logically, one would now try to combine these, so as to first make the pull distributions more Gaussian and then increase the performance. We have tried this, but the second feedback training undoes the effects of the first one (regardless of the order). One perspective idea could now be to use these multiplication criteria alternatively. Another interesting topic is the further analysis of the performance of the BNN on the test data after the feedback training: while the performance on the training data has increased significantly, the performance on the test data has increased less and still has some regions which are not perfect. We could investigate how much this is limited by the statistics of the training set size and adopt our feedback training to not only emphasize existing training data, but include new training data points in problematic regions. A big and necessary step is also the widening of the scope to the process $gg \to \gamma\gamma + gg$ for which no analytic method to calculate the amplitudes is available. Another interesting idea would be to incorporate a better way to find the point, where the feedback training does not bring any further increase in performance, since the amount of loops is currently a hyperparameter.

5 Acknowledgements

I would like to thank Prof. Tilman Plehn for the opportunity to work on this interesting topic and the immensely helpful discussions and advice along the way. Furthermore, I would also like to thank Michel Luchmann for patiently supervising this theses and for being continuously and readily available for questions of any kind. Additionally, I am grateful to the group for the nice working atmosphere and good company.

List of Figures

6 References

[1] C. F. Berger et al. "Automated implementation of on-shell methods for one-loop amplitudes". In: *Physical Review D* 78.3 (2008). ISSN: 1550-2368. DOI: 10.1103/physrevd.78.036003. URL: http://dx.doi.org/10.1103/PhysRevD.78.036003.

[2] G. Bevilacqua et al. "HELAC-NLO". In: *Computer Physics Communications* 184.3 (2013), 986–997. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2012.10.033. URL: http://dx.doi.org/10.1016/j.cpc.2012.10.033.

[3] Gavin Cullen et al. "GoSam-2.0: a tool for automated one-loop calculations within the Standard Model and beyond". In: *The European Physical Journal C* 74.8 (2014). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-014-3001-5. URL: http://dx.doi.org/10.1140/epjc/s10052-014-3001-5.

[4] Ansgar Denner, Jean-Nicolas Lang, and Sandro Uccirati. "RECOLA2: REcursive Computation of One-Loop Amplitudes 2". In: *Computer Physics Communications* 224 (2018), 346–361. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2017.11.013. URL: http://dx.doi.org/10.1016/j.cpc.2017.11.013.

[5] Jakub Kryś. *Recent progress for five-particle two-loop scattering amplitudes with an off-shell leg.* 2022. arXiv: 2202.06653 [hep-ph].

[6] Barry M. Dillon et al. "Better Latent Spaces for Better Autoencoders". In: *SciPost Phys.* 11 (2021), p. 061. DOI: 10.21468/SciPostPhys.11.3.061. arXiv: 2104.08291 [hep-ph].

[7] Gregor Kasieczka et al. "Deep-learning Top Taggers or The End of QCD?" In: *JHEP* 05 (2017), p. 006. DOI: 10.1007/JHEP05(2017)006. arXiv: 1701.08784 [hep-ph].

[8] Sven Bollweg et al. "Deep-Learning Jets with Uncertainties and More". In: *SciPost Phys.* 8.1 (2020), p. 006. DOI: 10.21468/SciPostPhys.8.1.006. arXiv: 1904.10004 [hep-ph].

[9] Joshua Bendavid. *Efficient Monte Carlo Integration Using Boosted Decision Trees and Generative Deep Neural Networks.* 2017. arXiv: 1707.00028 [hep-ph].

[10] Matthew Klimek and Maxim Perelstein. "Neural network-based approach to phase space integration". In: *SciPost Physics* 9.4 (2020). ISSN: 2542-4653. DOI: 10.21468/scipostphys.9.4.053. URL: http://dx.doi.org/10.21468/SciPostPhys.9.4.053.

[11] I-Kai Chen, Matthew Klimek, and Maxim Perelstein. "Improved neural network Monte Carlo simulation". In: *SciPost Physics* 10.1 (2021). ISSN: 2542-4653. DOI: 10.21468/scipostphys.10.1.023. URL: http://dx.doi.org/10.21468/SciPostPhys.10.1.023.

[12] Marco Bellagente et al. "Understanding Event-Generation Networks via Uncertainties". In: (Apr. 2021). arXiv: 2104.04543 [hep-ph].

[13] Mathias Backes et al. "How to GAN Event Unweighting". In: *SciPost Phys.* 10.4 (2021), p. 089. DOI: 10.21468/SciPostPhys.10.4.089. arXiv: 2012.07873 [hep-ph].

[14] Anja Butter and Tilman Plehn. “Generative Networks for LHC events”. In: (Aug. 2020). arXiv: 2008.08558 [hep-ph].

[15] Sydney Otten et al. “DeepXS: Fast approximation of MSSM electroweak cross sections at NLO”. In: (Oct. 2018).

[16] Fady Bishara and Marc Montull. *(Machine) Learning amplitudes for faster event generation.* 2020. arXiv: 1912.11055 [hep-ph].

[17] Simon Badger and Joseph Bullock. “Using neural networks for efficient evaluation of high multiplicity scattering amplitudes”. In: *JHEP* 06 (2020), p. 114. DOI: 10.1007/JHEP06(2020)114. arXiv: 2002.07516 [hep-ph].

[18] Joseph Aylett-Bullock, Simon Badger, and Ryan Moodie. “Optimising simulations for diphoton production at hadron colliders using amplitude neural networks”. In: *Journal of High Energy Physics* 2021.8 (2021). ISSN: 1029-8479. DOI: 10.1007/jhep08(2021)066. URL: http://dx.doi.org/10.1007/JHEP08(2021)066.

[19] Rikkert Frederix et al. “Automation of next-to-leading order computations in QCD: the FKS subtraction”. In: *Journal of High Energy Physics* 2009.10 (2009), 003–003. ISSN: 1029-8479. DOI: 10.1088/1126-6708/2009/10/003. URL: http://dx.doi.org/10.1088/1126-6708/2009/10/003.

[20] S. Frixione, Z. Kunszt, and A. Signer. “Three-jet cross sections to next-to-leading order”. In: *Nuclear Physics B* 467.3 (1996), pp. 399–442. ISSN: 0550-3213. DOI: https://doi.org/10.1016/0550-3213(96)00110-1. URL: https://www.sciencedirect.com/science/article/pii/0550321396001101.

[21] Yarin Gal. “Uncertainty in Deep Learning”. PhD thesis. 2016.

[22] Michael Edward Peskin and Daniel V. Schroeder. *An Introduction to Quantum Field Theory.* Reading, USA: Addison-Wesley (1995) 842 p. Westview Press, 1995.

[23] R. Keith Ellis, W. James Stirling, and B. R. Webber. *QCD and collider physics.* Vol. 8. Cambridge University Press, Feb. 2011. ISBN: 978-0-511-82328-2, 978-0-521-54589-1. DOI: 10.1017/CBO9780511628788.

[24] P. Skands. “Introduction to QCD”. In: *Searching for New Physics at Small and Large Scales* (2013). DOI: 10.1142/9789814525220_0008. URL: http://dx.doi.org/10.1142/9789814525220_0008.

[25] Michelangelo L. Mangano. “Introduction to QCD”. In: *1998 European School of High-Energy Physics.* 1998, pp. 53–97.

[26] T. Kinoshita. “Mass singularities of Feynman amplitudes”. In: *J. Math. Phys.* 3 (1962), pp. 650–677. DOI: 10.1063/1.1724268.

[27] T. D. Lee and M. Nauenberg. “Degenerate Systems and Mass Singularities”. In: *Phys. Rev.* 133 (6B 1964), B1549–B1562. DOI: 10.1103/PhysRev.133.B1549. URL: https://link.aps.org/doi/10.1103/PhysRev.133.B1549.

[28] F. Bloch and A. Nordsieck. “Note on the Radiation Field of the Electron”. In: *Phys. Rev.* 52 (2 1937), pp. 54–59. DOI: 10.1103/PhysRev.52.54. URL: https://link.aps.org/doi/10.1103/PhysRev.52.54.

[29] A. Hebecker. *Lecture notes in Quantum Field Theory.* 2021.

[30] Joseph Peter Aylett-Bullock. "Colliding worlds: Modern computational methods for scattering amplitude calculations and responding to crisis situations". PhD thesis.

[31] T. Gehrmann, N. Greiner, and G. Heinrich. "Precise QCD Predictions for the Production of a Photon Pair in Association with Two Jets". In: *Phys. Rev. Lett.* 111 (22 2013), p. 222002. DOI: 10.1103/PhysRevLett.111.222002. URL: https://link.aps.org/doi/10.1103/PhysRevLett.111.222002.

[32] S. D. Badger et al. "Five jet production at next-to-leading order QCD". In: *Nucl. Part. Phys. Proc.* 273-275 (2016). Ed. by M Aguilar-Benítez et al., pp. 2066–2072. DOI: 10.1016/j.nuclphysbps.2015.09.334.

[33] Z. Bern et al. "Next-to-leading order+2jetproduction at the LHC". In: *Physical Review D* 90.5 (2014). ISSN: 1550-2368. DOI: 10.1103/physrevd.90.054004. URL: http://dx.doi.org/10.1103/PhysRevD.90.054004.

[34] Simon Badger et al. "Numerical evaluation of virtual corrections to multi-jet production in massless QCD". In: *Computer Physics Communications* 184.8 (2013), 1981–1998. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2013.03.018. URL: http://dx.doi.org/10.1016/j.cpc.2013.03.018.

[35] Simon Plätzer. *RAMBO on diet.* 2013. arXiv: 1308.2922 [hep-ph].

[36] A. M. Andrew. "Perceptrons, Marvin Minsky, Seymour Papert (Eds.). M.I.T. Press, Cambridge, Mass. (1969), 258 pp. 112s". In: *International Journal of Human-computer Studies International Journal of Man-machine Studies* 2 (1970), pp. 314–316.

[37] Matthew Klimek and Maxim Perelstein. "Neural network-based approach to phase space integration". In: *SciPost Physics* 9.4 (2020). ISSN: 2542-4653. DOI: 10.21468/scipostphys.9.4.053. URL: http://dx.doi.org/10.21468/SciPostPhys.9.4.053.

[38] I. Chen, Matthew D. Klimek, and Maxim Perelstein. "Improved neural network Monte Carlo simulation". In: *arXiv: High Energy Physics - Phenomenology* (2020).

[39] Christina Gao, Joshua Isaacson, and Claudius Krause. "i- flow: High-dimensional integration and sampling with normalizing flows". In: *Machine Learning: Science and Technology* 1.4 (2020), p. 045023. ISSN: 2632-2153. DOI: 10.1088/2632-2153/abab62. URL: http://dx.doi.org/10.1088/2632-2153/abab62.

[40] Enrico Bothmann et al. "Exploring phase space with Neural Importance Sampling". In: *SciPost Physics* 8.4 (2020). ISSN: 2542-4653. DOI: 10.21468/scipostphys.8.4.069. URL: http://dx.doi.org/10.21468/SciPostPhys.8.4.069.

[41] Christina Gao et al. "Event generation with normalizing flows". In: *Physical Review D* 101.7 (2020). ISSN: 2470-0029. DOI: 10.1103/physrevd.101.076002. URL: http://dx.doi.org/10.1103/PhysRevD.101.076002.

[42] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014.

[43] Geoffrey Hinton. *Neural Networks for Machine Learning.*

[44] Ilya Sutskever et al. “Proceedings of the 30th International Conference on Machine Learning”. In: ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1139–1147.

[45] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU).* 2019. arXiv: `1803.08375 [cs.NE]`.

[46] Dan Hendrycks and Kevin Gimpel. “Gaussian Error Linear Units (GELUs)”. In: *arXiv: Learning* (2016).

[47] David J. C. Mackay. “Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks”. In: *Network: Computation In Neural Systems* 6 (1995), pp. 469–505.

[48] Yoav Freund and Robert E. Schapire. “Experiments with a New Boosting Algorithm”. In: *IN PROCEEDINGS OF THE THIRTEENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING.* Morgan Kaufmann, 1996, pp. 148–156.

7 Erklärung

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

S. Pitz

Heidelberg, den 16. März 2022