Loop Amplitudes from Precision Networks

Simon Badger[1], Anja Butter[2,3], Michel Luchmann[2], Sebastian Pitz[2] and Tilman Plehn[2]

1 Physics Department, Torino University and INFN Torino, Torino, Italy
2 Institut für Theoretische Physik, Universität Heidelberg, Germany
3 LPNHE, Sorbonne Université, Université Paris Cité, CNRS/IN2P3, Paris, France

January 11, 2023

Abstract

Evaluating loop amplitudes is a time-consuming part of LHC event generation. For di-photon production with jets we show that simple, Bayesian networks can learn such amplitudes and model their uncertainties reliably. A boosted training of the Bayesian network further improves the uncertainty estimate and the network precision in critical phase space regions. In general, boosted network training of Bayesian networks allows us to move between fit-like and interpolation-like regimes of network training.

Contents

1 Introduction

Combining our expectations of a vastly increased dataset from the upcoming LHC runs with novel analysis methods and ever-improving theory predictions, we are looking at exciting times for particle physics. One of the keys to make optimal use of the LHC data is to consistently employ modern techniques, inspired by data science and further developed for particle physics application [1]. Inference based on precision predictions from first principles critically rests on the assumption that we can provide theory predictions over the full phase space fast, precisely, and with flexible model assumptions. To meet the speed and precision expectations from HL-LHC we can use modern machine learning (ML) throughout the event generation and simulation chain [2].

A straightforward ML-task is regression of loop amplitudes, represented as a smooth scalar function over a relatively simple phase space. For simple $(2 \to 2)$-processes learning a non-divergent loop amplitude does not even require deep networks [3] and has been achieved with conventional interpolation methods [4,5] as well. For higher final-state multiplicities [6] precision turns into a challenge, which we can try to meet by separating phase space into finite and divergent regions [7] or physics-inspired channels combined with very large training samples [8]. For our di-photon benchmark process at one-loop order current methods have shown to work well, but with limited precision especially in challenging regions of phase space [9].

Like in many physics applications, we would like to complement precision predictions of amplitudes with a reliable uncertainty estimate. Amplitudes are a simple problem because the training data consists of arbitrarily precise numerical values for well-defined phase space points. Once the network has learned all relevant features, we expect the leading uncertainties to reflect a possible local sparsity of the training data. Bayesian networks are perfectly suited to track training-related uncertainties [10]. In LHC physics they have been applied to regression [11], classification [12], ensembling [13], and generation [14,15]. We will use Bayesian networks as a surrogate for ML-amplitudes because they learn amplitude values together with an uncertainty, and because we can use their likelihood loss to improve the network training.

When we want to train a network on amplitudes over phase space, with the additional condition that large amplitude values should be reproduced well, we need to re-think our training strategy. While usual NN-applications can be viewed as a non-parametric fit, we want to precisely reproduce individual amplitudes in the spirit of an interpolation [16]. We can force the network to reproduce certain amplitudes by boosting these amplitudes in the training. To decide which amplitudes need boosting, we use a Bayesian network with its point-wise uncertainty estimate. We find that moving freely between fit-like and the interpolation-like tasks allows us to improve the uncertainty estimate through a loss-based boosting and the precision though a process-specific performance boosting.

In Sec. 2 we introduce our dataset and the benchmark results, before introducing the Bayesian network in Sec. 3. The improved training through the two boosting strategies is illustrated for the $\gamma\gamma g$ channel in Sec. 4. Finally, we compare a set of 1-dimension kinematic distributions for the training data and the NN-amplitudes including uncertainties and with the different boosting strategies, before we provide an Outlook. In the Appendix we show the corresponding results for the $\gamma\gamma gg$ final state, based on the same concepts and architectures, but using a larger network and more training data.

2 Dataset and benchmark results

As an example process for our surrogate NN-amplitudes we use the partonic one-loop process [9]

$$gg \to \gamma\gamma g(g)\,, \tag{1}$$

generated with SHERPA [17] and the NJET amplitude library [18][*]. We apply a basic set of detector-inspired cuts on the partons in the final state,

$$\begin{aligned} &p_{T,j} > 20\text{ GeV} \qquad && |\eta_j| < 5 \qquad && R_{jj,j\gamma,\gamma\gamma} > 0.4 \\ &p_{T,\gamma} > 40,30\text{ GeV} && |\eta_\gamma| < 2.37\,. && \end{aligned} \tag{2}$$

They cut off collinear and infrared divergences, simplifying our network task. However, we have checked that the method developed in this paper can also be applied to squared amplitudes with less restrictive cuts. The kinematic cuts reduce the originally produced dataset from 1.1M squared $(2 \to 3)$-amplitudes of Ref. [9] to roughly 960k valid phase space points.

Our goal is to learn the transition amplitudes precisely and with minimal pre-processing. Throughout the paper we refer to 'amplitudes' as the real values of the complete squared transition amplitude for the process given in Eq.(1) as a function of the phase space coordinates. This means, at face value, each training data point is a single real number as a function of the external 4-momenta, defining a 20-dimensional phase space. Reducing the number of dimensions by requiring on-shell partons or enforcing momentum conservation is expected to help the ultimate precision [15, 19], especially for the two-jet process discussed in the Appendix. For the one-jet process we divide our dataset into 90k training amplitudes and 870k independent test amplitudes acting as a high-statistics truth.

Transition amplitudes play a special role in the applications of neural networks to LHC physics, because they can be computed as functions of phase space with essentially arbitrary precision. The combination of high precision with limited training data is the challenge for the corresponding regression networks. Implicitly, it is assumed that the NN-amplitude networks will be faster than even the evaluation of leading-order amplitudes with state-of-the-art methods. The main figure of merit compares the true and the NN-amplitudes for a set of training or test data points,

$$\Delta_j^{(\text{train})} = \frac{A_{j,\text{NN}}}{A_{j,\text{train}}} - 1 \qquad \text{or} \qquad \Delta_j^{(\text{test})} = \frac{A_{j,\text{NN}}}{A_{j,\text{test}}} - 1\,, \tag{3}$$

where j runs over amplitude data points and we subtract 1 compared with the original paper [9]. Typical distributions of these Δ for existing calculations come with a width of 10% or more for the one-jet process of Eq.(1) [9]. For the tree-level process $e^+e^- \to q\bar{q}g$ the width of the Δ-distribution can be reduced to the per-mille level for a training dataset of 500k amplitudes, or to a hundredth of a per-mille for 40M training amplitudes, with the help of a comparably complex, physics-inspired architecture of networks [8].

Our approach follows a different strategy from the physics-inspired architectures mentioned above. We will use a relatively simple and small network, enhanced by a Bayesian network structure, and target the precision requirements with a new training strategy. The goal is to show that small and hence fast networks are expressive enough to describe a scattering amplitude over phase space. Enforcing and controlling the required precision leads us to, essentially, an appropriate loss function and the corresponding network training strategy.

[*]In the main body of the paper we work with the $(2 \to 3)$-process, the corresponding results for the 4-body final state are presented in the Appendix. The interface between NJET and SHERPA is provided with Ref. [9] and available at https://github.com/JosephPB/n3jet.

3 Bayesian network

Deterministic networks, usually trained by minimizing an MSE loss function, exhibit several weaknesses when it comes to LHC applications and controlled precision predictions. First, they only learn the amplitude value over phase space, without any information on if they have learned all features and how precise their estimate is. Second, the definition of the MSE loss function as an χ^2-approximation with a constant uncertainty limits their performance for regression tasks with a wide range of functional values of the target function. We will show how a Bayesian network with a likelihood loss comes with a whole range of conceptual and practical benefits.

Bayesian networks and uncertainties

In contrast to standard, deterministic networks, Bayesian neural networks (BNNs) learn distributions of network parameters or weights ω [10,20]. Sampling over the weight distributions gives us an uncertainty in the network output. At the end of this introduction we will approximate each weight distribution by a Gaussian, which does not limit the expressivity of a deep Bayesian network, but means that the Bayesian network requires only twice as many parameters as its deterministic counterpart [10]. We will see that the relation between the usual deterministic networks and our Bayesian network setup can be understood as a generalized dropout and an explicit regularization term in the loss, which stabilize the training. For a pedagogical introduction to this application of Bayesian networks we refer to Ref. [1].

With our amplitude network we want to predict the transition amplitude A for a phase space point x. If we define $p(A|x) \equiv p(A)$ as the probability distribution for possible amplitudes at a given phase space point x, and omitting the argument x from now on, its mean value is

$$\langle A \rangle = \int dA\, A\, p(A) \qquad \text{with} \qquad p(A) = \int d\omega\, p(A|\omega)\, p(\omega|T)\,, \tag{4}$$

where $p(\omega|T)$ are the network weight distribution and T is the training data. We do not know the closed form of $p(\omega|T)$, but we can approximate it with a distribution $q(\omega)$,

$$p(A) = \int d\omega\, p(A|\omega)\, p(\omega|T) \approx \int d\omega\, p(A|\omega)\, q(\omega)\,, \tag{5}$$

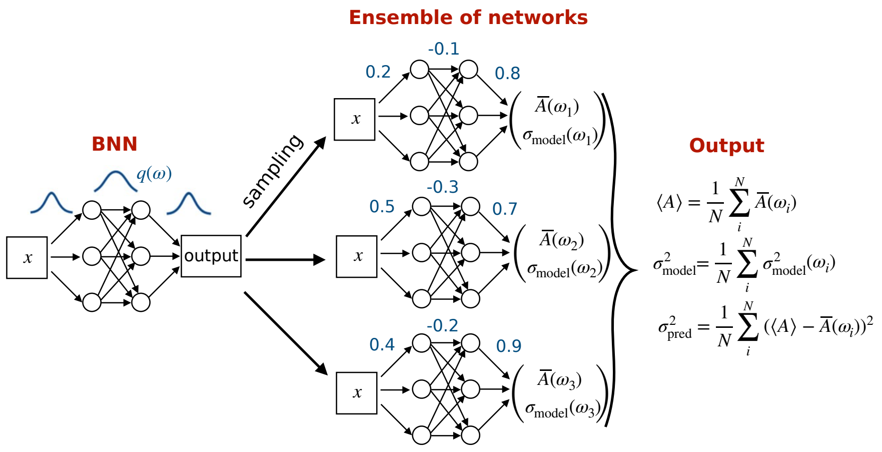
Ensemble of networks
BNN
$q(\omega)$
x
output
sampling
-0.1
0.2
0.8
-0.3
0.5
0.7
-0.2
0.4
0.9
$\bar{A}(\omega_1)$
$\sigma_{\text{model}}(\omega_1)$
$\bar{A}(\omega_2)$
$\sigma_{\text{model}}(\omega_2)$
$\bar{A}(\omega_3)$
$\sigma_{\text{model}}(\omega_3)$
Output
$\langle A \rangle = \frac{1}{N}\sum_i^N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

Figure 1: Illustration of the Bayesian network.

where $q(\omega)$ is encoded in the neural network and replaces the unknown $p(\omega|T)$ in the evaluation, including the information from the training data T. This so-called variational approximation leads us directly to the BNN loss function, which employs the Kullback-Leibler divergence to minimize the difference between the two distributions,

$$\begin{aligned}
\mathrm{KL}[q(\omega), p(\omega|T)] &= \int d\omega\, q(\omega)\, \log \frac{q(\omega)}{p(\omega|T)} \\
&= \int d\omega\, q(\omega)\, \log \frac{q(\omega)p(T)}{p(\omega)p(T|\omega)} \\
&= \mathrm{KL}[q(\omega), p(\omega)] - \int d\omega\, q(\omega)\, \log p(T|\omega) + \log p(T) \int d\omega\, q(\omega)\,. \qquad (6)
\end{aligned}$$

Bayes' theorem gives the corresponding networks their name. The prior $p(\omega)$ describes the model parameters before training. The model evidence $p(T)$ guarantees the correct normalization of $p(\omega|T)$. If we ensure that the weight distributions $q(\omega)$ are normalized by construction, the evidence term does not depend on the network weights, and we can turn the first two terms of Eq.(6) into a loss function

$$\mathcal{L}_{\mathrm{BNN}} = -\int d\omega\, q(\omega)\, \log p(T|\omega) + \mathrm{KL}[q(\omega), p(\omega)]\,. \qquad (7)$$

The combined log-likelihood $\log p(T|\omega)$ implicitly includes the sum over all training points.

To get access to the mean and the uncertainty of the network prediction for A, we exchange the two integrals in Eq.(4) and find

$$\begin{aligned}
\langle A \rangle &= \int dA\, d\omega\, A\, p(A|\omega)\, q(\omega) \\
&\equiv \int d\omega\, q(\omega)\, \overline{A}(\omega) \qquad \text{with} \qquad \overline{A}(\omega) = \int dA\, A\, p(A|\omega)\,. \qquad (8)
\end{aligned}$$

A network with perfect x-resolution and perfect interpolation properties would be described by $q(\omega) = \delta(\omega - \omega_0)$, and $p(A|\omega)$ would simply return the one correct value for the amplitude. For noisy training data $p(A|\omega)$ actually describes a spectrum reflecting the noisy labels [11]. In our case, the amplitudes are exact, but the network will still not interpolate perfectly between the sparse training data points. Corresponding to Eq.(8) the variance of A is

$$\begin{aligned}
\sigma_{\mathrm{tot}}^2 &= \int dA\, d\omega\, (A - \langle A \rangle)^2\, p(A|\omega)\, q(\omega) \\
&= \int d\omega\, q(\omega) \left[\int dA\, A^2\, p(A|\omega) - 2\langle A \rangle \int dA\, A\, p(A|\omega) + \langle A \rangle^2 \int dA\, p(A|\omega) \right] \\
&= \int d\omega\, q(\omega) \left[\overline{A^2}(\omega) - 2\langle A \rangle \overline{A}(\omega) + \langle A \rangle^2 \right] \\
&= \int d\omega\, q(\omega) \left[\overline{A^2}(\omega) - \overline{A}(\omega)^2 + \left(\overline{A}(\omega) - \langle A \rangle \right)^2 \right] \equiv \sigma_{\mathrm{model}}^2 + \sigma_{\mathrm{pred}}^2\,, \qquad (9)
\end{aligned}$$

where $\overline{A^2}(\omega)$ is defined in analogy to $\overline{A}(\omega)$ in Eq.(8). This defines two contributions to the uncertainty which have to be added in quadrature to describe the variance of the Bayesian network prediction σ_{tot}. The two terms can be separated by the way they behave in in the limit of perfect training using an infinitely large training dataset, $q(\omega) \to \delta(\omega - \omega_0)$.

Tracing σ_{tot} as a function of the amount of training data [11, 12] shows how the first contribution, σ_{pred} defined in terms of the ω-integrated expectation value $\langle A \rangle$

$$\sigma^2_{\text{pred}} = \int d\omega \, q(\omega) \left[\overline{A}(\omega) - \langle A \rangle \right]^2 , \tag{10}$$

vanishes in the limit $q(\omega) \to \delta(\omega - \omega_0)$ or for a large training dataset. It can therefore be interpreted as a statistical uncertainty and we expect it to be small for a well-trained network.

In contrast, σ_{model} already occurs without sampling the network parameters,

$$\begin{aligned} \sigma^2_{\text{model}} \equiv \langle \sigma_{\text{model}}(\omega)^2 \rangle &= \int d\omega \, q(\omega) \, \sigma_{\text{model}}(\omega)^2 \\ &= \int d\omega \, q(\omega) \left[\overline{A^2}(\omega) - \overline{A}(\omega)^2 \right] . \end{aligned} \tag{11}$$

It is induced by limitations to the training data, but in the limit of perfect training it approaches a plateau. It can account for a range of issues with the training data, like a non-deterministic or stochastic label, limited expressivity of the network, not-so-smart choices of hyperparameters etc, in the sense of a systematic uncertainty. To avoid mis-understanding we refer to it as a model-related uncertainty rather than the usual σ_{stoch} in case our data is non-stochastic, like the amplitudes in this application.

Looking at Eqs.(8) and (11) we can extract the central value and the systematic contribution to the uncertainty of the network prediction by sampling $\overline{A}(\omega)$ and $\sigma_{\text{model}}(\omega)^2$ over Gaussian network parameter distributions $q(\omega)$ for each phase space point x,

$$\text{BNN} : x, \omega \; \longrightarrow \begin{pmatrix} \overline{A}(\omega) \\ \sigma_{\text{model}}(\omega) \end{pmatrix} , \tag{12}$$

while σ_{pred} is only defined as a function of x and after sampling over ω.

Until now, we have not made any simplifying assumptions about the prior or weight distributions. To start with, in Ref. [12] we have shown that the details of the prior $p(\omega)$ have no visible effect on the network output. If we assume a Gaussian prior and, in addition, a Gaussian weight distribution $q(\omega)$ with the respective means and widths, the regularization term in Eq.(7) turns into

$$\text{KL}[q_{\mu,\sigma}(\omega), p_{\mu,\sigma}(\omega)] = \frac{\sigma_q^2 - \sigma_p^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} + \log \frac{\sigma_p}{\sigma_q} . \tag{13}$$

For this form we can use the reparameterization trick to translate an ω-dependence into a dependence on σ_q and μ_q. Second, we can simplify the loss function by assuming that the ω-dependent network output in Eq.(12) is described by a Gaussian. This allows us to approximate the likelihood $p(T|\omega)$ in Eq.(7) as Gaussian, and the BNN loss function becomes

$$\begin{aligned} \mathcal{L}_{\text{BNN}} = \int d\omega \, q_{\mu,\sigma}(\omega) \sum_{\text{points } j} &\left[\frac{\left| \overline{A}_j(\omega) - A_j^{(\text{truth})} \right|^2}{2\sigma_{\text{model},j}(\omega)^2} + \log \sigma_{\text{model},j}(\omega) \right] \\ &+ \frac{\sigma_q^2 - \sigma_p^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} + \log \frac{\sigma_p}{\sigma_q} . \end{aligned} \tag{14}$$

This loss is minimized with respect to the means and standard deviations of the network weights describing $q_{\mu,\sigma}(\omega)$. In this setup, the log-likelihood term includes a trainable uncertainty $\sigma_{\text{model}}(\omega)$ which is learned by the network in parallel to the amplitudes. When we evaluate the likelihood over a mini-batch rather than the full training dataset, we rescale the normalization of the regularization term to account for the different numbers of data points.

The same heteroscedastic loss [21] can be used in deterministic networks, if we introduce σ_model as a second trained quantity in addition to the amplitude values. The Bayesian network setup guarantees that we really capture all training-related uncertainties correctly, at the expense of splitting the uncertainty measures σ_model and σ_pred. It also does not make assumptions about a Gaussian uncertainty of the network output, so we stick to the more general BNN, even though it might well be possible to use a deterministic network for similar applications.

Network architecture

We use one Bayesian network trained on the entire training dataset. We train on amplitudes as a function of phase space with logarithmic preprocessing,

$$A_j \;\to\; \log\left(1+\frac{A_j}{\sigma_A}\right) , \tag{15}$$

where σ_A is given by the distribution of the amplitude values. In addition, all phase space directions are preprocessed by subtracting the respective mean and dividing by the respective standard deviation.

The network describing the $(2 \to 3)$-part of the reference process in Eq.(1) consists of four hidden layers with 20 kinematic input dimensions, $\{20, 20, 30, 40\}$ nodes, and two output dimensions corresponding to the amplitude and its uncertainty, as illustrated in Eq.(12). The network has around 6k parameters.

Between the hidden layers we use a tanh activation function, while for the last layer we use a SoftPlus activation for one of the two output dimensions. This ensures positive values for the amplitudes. For the other dimension responsible for the uncertainty we do not use any activation function. Instead, we learn the parameterization $\log\sigma^2$ which is numerically more stable than directly learning σ. Our tests revealed that the SoftPlus activation function outperforms GeLU slightly and ReLU significantly. The network is trained on 90k amplitudes for 400000 epochs with a batch size of 8192 and learning rate of 10^{-4}, after which we observe no significant improvement in the loss. We use the Adam optimizer [22] with standard parameters.

BNN performance

As a first test of our BNN, we check the precision with which it approximates the true amplitudes in the training and test datasets, as defined in Eq.(3). For Fig. 2 we split the amplitudes by their absolute values, to see the effect of the limited training statistics in the collinear phase space regions. For the complete set of amplitudes the precision follows an approximate Gaussian with a width of a few per-mille, for the training and for the test data. This matches the best available performance from the literature [8], but with a very compact and fast network.

In the logarithmic panels of Fig. 2 we see that the tails of the Δ-distributions for the full datasets are clearly enhanced. The picture changes when we only consider the phase space points with large amplitudes. For the 0.1% largest amplitudes the network is consistently less accurate with a slight tendency of underestimating the amplitudes. This is the motivation for training a separate network on the divergent phase space region(s) [9]. As we will see, the BNN offers an alternative approach which allows the full amplitude to be accurately described by a single network.

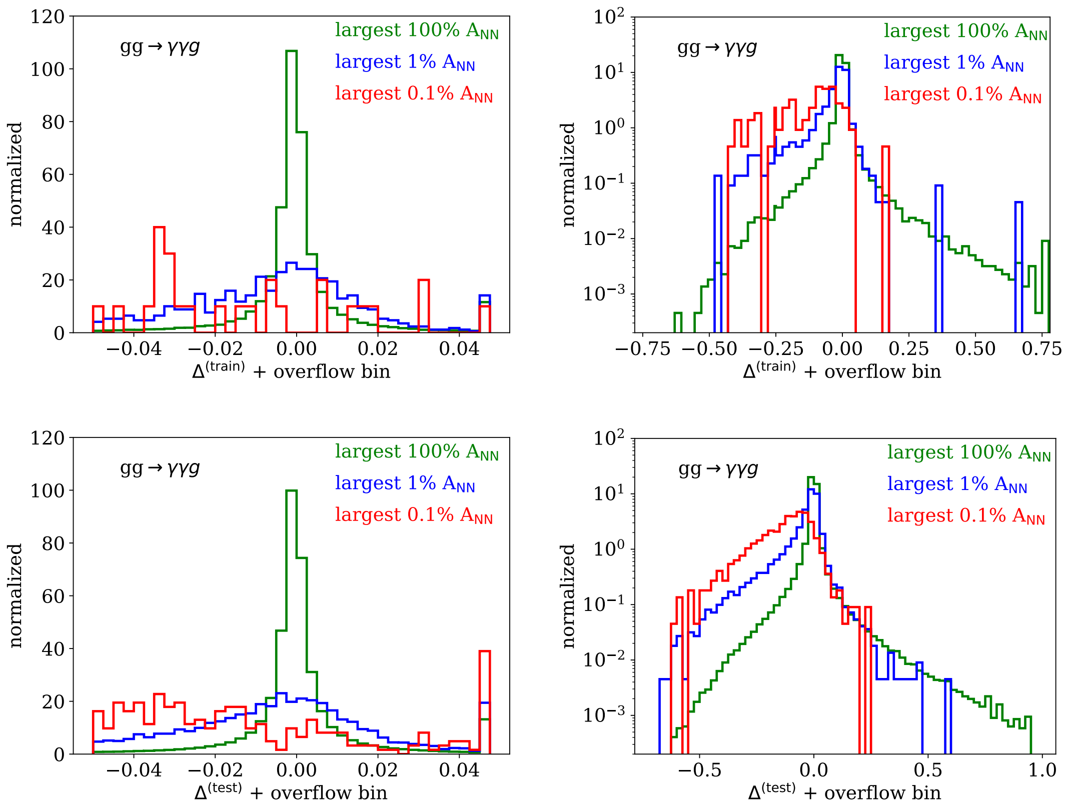

Figure 2: Performance of the BNN in terms of the precision of the generated amplitudes as defined in Eq.(3). In the upper panels we evaluate the training datasets, while in the lower panels we evaluate the independent test datasets. The left and right panels show the same information on linear and logarithmic scales.

Because the BNN provides us with an uncertainty estimate for the NN-amplitude, we can define pull variables after integrating over the weight distributions,

$$t_{\text{model},j} = \frac{\langle A \rangle_j - A_j^{(\text{truth})}}{\sigma_{\text{model},j}} \qquad \text{or} \qquad t_{\text{pred},j} = \frac{\langle A \rangle_j - A_j^{(\text{truth})}}{\sigma_{\text{pred},j}} , \tag{16}$$

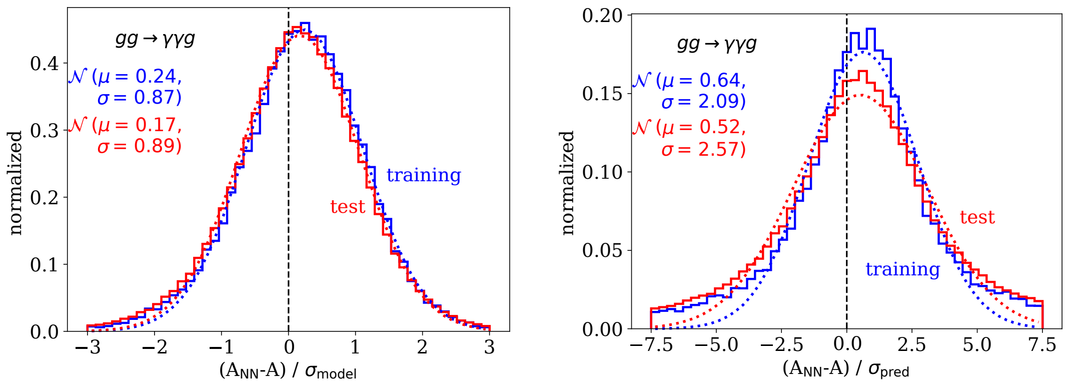

Figure 3: Pulls for the BNN, defined in Eq.(17) and evaluated on the training and test data.

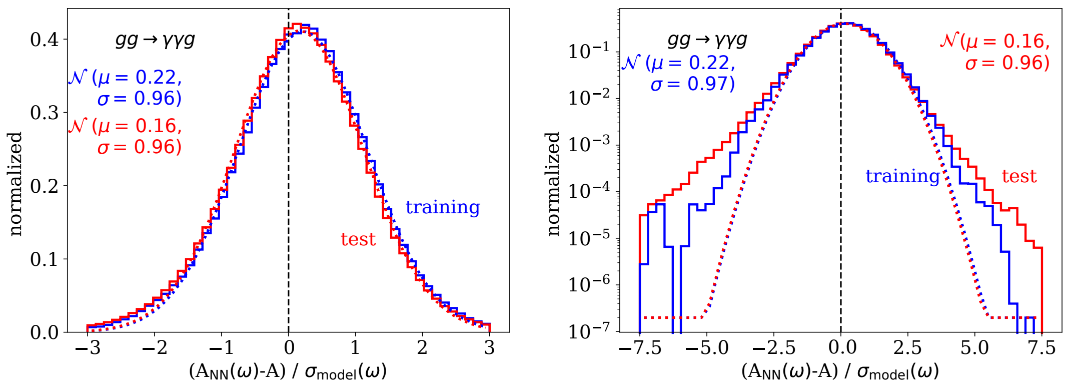

Figure 4: Weight-dependent pulls for the BNN, defined in Eq.(17) and evaluated on the training and test data. The two panels show the same curve on a linear and a logarithmic axis.

where the point-wise 'truth' refers to the training or test datasets we use to evaluate the pulls. Neither of these pulls have an ω-dependent counterpart, because their numerators and denominators are sampled over the network weights independently. In Fig. 3 we see that the two pulls follow an approximate Gaussian shape, but with a much broader distribution for the σ_pred-based pull because of the smaller estimated uncertainty. We note that because of the log-rescaling of Eq.(15) it is not actually the amplitudes A which should define Gaussian pulls, but their logarithms. We have explicitly checked that indeed the $\log A$ lead to a Gaussian, but that given our limited range of relevant amplitudes, the Gaussian shape translates into an approximately Gaussian shape for the amplitudes themselves.

Making use of the Gaussian likelihood loss of the BNN, Eq.(14), we can also define the weight-dependent pull

$$t_{\text{model},j}(\omega) = \frac{\overline{A}_j(\omega) - A_j^{(\text{truth})}}{\sigma_{\text{model},j}(\omega)} \; . \tag{17}$$

As part of the loss we can use its distribution as a consistency condition during network training. Given the Gaussian likelihood loss we expect a Gaussian distribution of $t_{\text{model},j}(\omega)$, sampled over ω according to the Gaussian $q(\omega)$ and over phase space points x. In Fig. 4 we see that, again, the pull distribution is Gaussian in the center, but develops symmetric, enhanced tails roughly two standard deviations from the mean.

Finally, we need to go back to the definition of the network uncertainties and understand how the split $\sigma_\text{tot}^2 = \sigma_\text{model}^2 + \sigma_\text{pred}^2$ can affect improved ways of training the amplitude network. From Fig. 3 we know that the two sampled pulls defined in Eq.(16) are approximately Gaussian, and the width of the $t_{\text{model},j}$ distribution is much smaller than for $t_{\text{pred},j}$. This is an effect of the general observation that for a well-trained model

$$\sigma_\text{tot} \approx \sigma_\text{model} > \sigma_\text{pred} \; . \tag{18}$$

Figure 5 shows very strong correlations between the two pulls defined in Eq. (16). Both pulls correctly identify the training data points which are not described by the network well. For our regression tasks with exact amplitudes both uncertainties are largely induced by the lack of training statistics especially in the divergent phase space regions, so this correlation is expected.

The only issue with all pulls shown in Fig. 3 is that they come with a slight bias towards positive values, which means the network slightly overestimates the amplitude values as a whole. The bias is small in comparison to the estimated uncertainties, and it is in contrast to the systematic underestimation of the 0.1% largest amplitudes observed in Fig. 2. These effects indicate a limitation in the network architecture and training, and we will target the bias, the precision on the largest amplitudes, and Gaussian shape of the pulls by improving the network training.

4 Network boosting

While the BNN-amplitude results described in the previous section are promising, the distribution of amplitudes and the pull distributions indicate potential improvements. We know that for generative networks we can employ an additional discriminator network for reweighting [23, 24] and identify and correct poorly learned phase space regions [15], the solution is much simpler for a regression network. In the BNN loss we can compute the relative deviations between data and network output, or large pulls, and target these amplitudes directly. Once we control the network and its uncertainties, we can even think about further enhancing the training in the direction of an interpolation.

4.1 Loss-based boosting

Because the BNN loss in Eq. (14) represents a Gaussian log-likelihood, we can modify it and require a higher precision for those phase space points which according to the BNN uncertainty are not yet learned well. In practice, this is equivalent to feeding these training data points n_j times into the computation of the BNN loss

$$\begin{aligned}\mathcal{L}_\text{Boosted BNN} = \int d\omega\, q_{\mu,\sigma}(\omega) \sum_{\text{points } j} n_j \times \left[\frac{\left| \bar{A}_j(\omega) - A_j^\text{(truth)} \right|^2}{2\sigma_{\text{model},j}(\omega)^2} + \log \sigma_{\text{model},j}(\omega) \right] \\ + \frac{\sigma_q^2 - \sigma_p^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} + \log \frac{\sigma_p}{\sigma_q} \,. \qquad (19)\end{aligned}$$

As mentioned for Eq. (14), the regularization has to be adjusted for the additional amplitudes in the boosted training sample. This feedback training is similar to simple boosting algorithms for decision trees, where amplitudes for which the decision tree gives a wrong result are enhanced

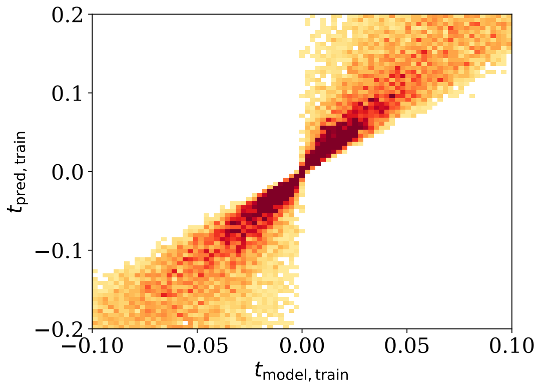

Figure 5: Correlation between the two pulls for the BNN, evaluated on the training data.

with additional weights. In our simple approach we duplicate some training amplitudes, or equivalently increases their weights in discrete steps.

In a first stage, we improve the self-consistency of the network with the initial assumptions and boost the network training for amplitudes with a large ω-dependent pull, Eq.(17). In five iterations we identify the amplitudes with values of $t_{\text{model},j}(\omega)$, which are more than two standard deviations away from the mean and increase their contribution to the loss function in Eq.(19) by values $n_j = 5$. This is done four times. After adding the weights we continue the training on the enlarged datasets. For the next training cycle we again add weights to the amplitudes which now have pulls more than two standard deviations away. Each training ends when we see no more significant change to the loss which usually takes around 2000 epochs. This boosting forces the network towards a more self-consistent description of the tails of the pull distributions. We checked that small variations of n_j or the number of cycles do not have a significant impact on these improvements.

In Fig. 6 we show the pulls from the boosted Bayesian neural network, boosted based on the self-consistency of the loss measured by the pull. We see a significant improvement for $t_{\text{model}}(\omega)$, the parameter we target with our boosting. One would naively expect the corresponding distribution to assume a Gaussian shape with unit width. However, first of all our loss-based boosting only moves amplitudes from the tails into the bulk, which means that the tails of the boosted pull distributions should be low. Second, the pulls entering the loss and the pulls shown in Fig. 6 are different because the loss includes weights for high-pull amplitudes. In combination, both effects explain the narrower Gaussian for $t_{\text{model}}(\omega)$. In the logarithmic version we also see a visible over-training through loss-boosting.

Moving on to the precision for the amplitudes, we see in Fig. 7 that the loss-boosting only has a mild impact on the Δ-distributions. It does not significantly improve the precision of the amplitudes compared to Fig. 2, it even increases the fraction of the largest 1% of the amplitudes in the overflow, indicating that we need a better, second boosting step.

4.2 Performance boosting

Given that the loss-boosting in the previous section worked for the uncertainty estimate but only had a modest effect on the performance of our amplitude network, we proceed to a more powerful boosting strategy. Independent of the self-consistency of the network, we know at the training level which amplitudes challenge the network. This means we can select them

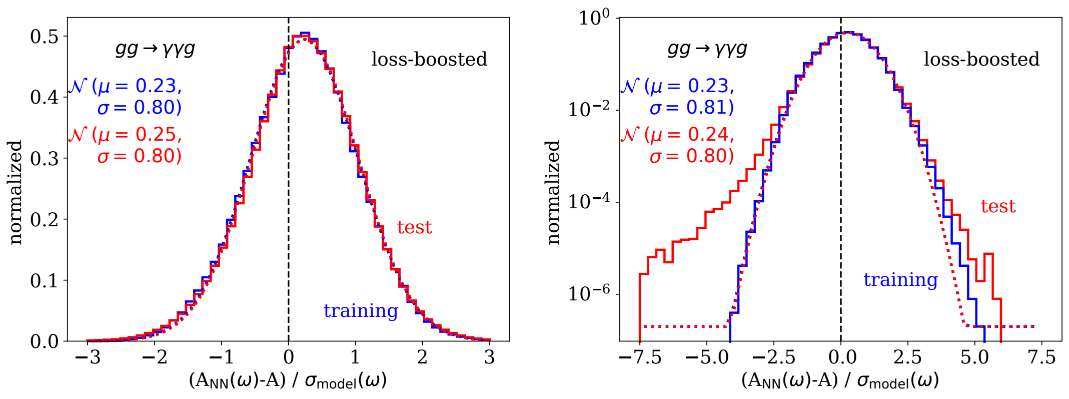

Figure 6: Pulls for the loss-boosted BNN, defined in Eq.(17) and evaluated on the training and test data. The two panels show the same results on a linear and a logarithmic axis. All curves can be compared to the BNN results without boosting in Fig. 4.

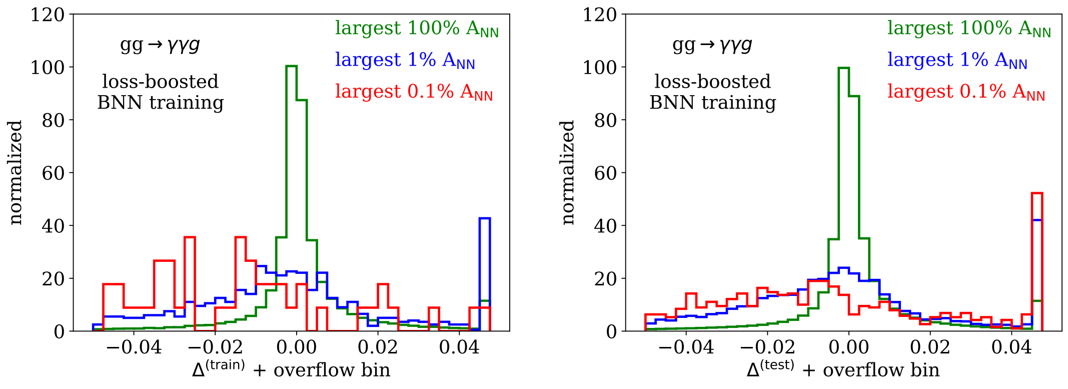

Figure 7: Performance of the loss-boosted BNN in terms of the precision of the generated amplitudes, Eq.(3), evaluated on the training and test datasets on a linear (left) and logarithmic (right) axis. The curves can be compared to the BNN results without boosting in Fig. 2.

with the goal of improving the training for the largest amplitudes. The difference between a general loss boosting and this process-dependent strategy is that now we target the largest and most poorly learned amplitudes by selecting them based on σ_{tot}. We choose the 200 amplitudes with the largest uncertainty σ_{tot} and add three additional copies to the training dataset. This process is repeated 20 times, where each training ends when we see no more significant change to the loss which is usually around 2000 epochs.

In Fig. 8 we first see that the process-specific performance boosting broadens the pull distributions and this way reverses some of the beneficial effects of the loss-boosting on $t_{model}(\omega)$. However, the widths of the boost distributions remains below one, and the bias towards larger amplitudes is removed. This is true for the training data and for the test data. In addition, the consistency with the Gaussian shape is broken symmetrically for too small and too large amplitudes, again consistently for training and test data. Given that the two boostings target different amplitudes and effectively compete with each other, this pattern is expected.

The positive impact on the large amplitudes can be seen more clearly in Fig. 9. First,

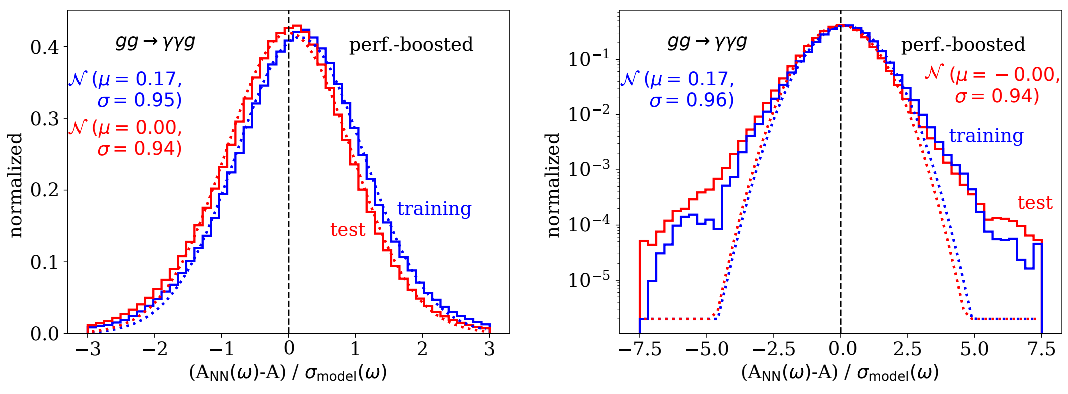

Figure 8: Pulls for the performance-boosted BNN, defined in Eq.(17) and evaluated on the training and test data. The two panels show the same results on a linear and a logarithmic axis. All curves can be compared to the BNN results without boosting in Fig. 4 and the loss-boosted results in Fig. 6.

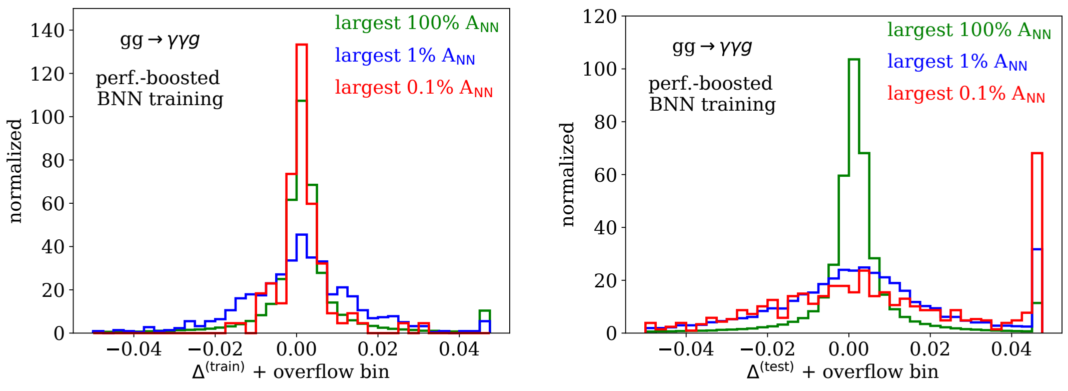

Figure 9: Performance of the performance-boosted BNN in terms of the precision of the generated amplitudes, Eq.(3), evaluated on the training (left) and test datasets (right). All curves can be compared to the BNN results without boosting in Fig. 2 and the loss-boosted results in Fig. 7.

for the full set of amplitudes the differences between the already successful BNN training in Fig. 2, the loss boosting in Fig. 7, and the performance boosting are minor. For the 1% largest amplitudes the performance boosting beats the loss boosting for the training data, some of this improvement is lost for the test data, but the generic precision of the network for these amplitudes is stable at the sub-percent level. The big improvement happens for the otherwise poorly learned 0.1% largest amplitudes, which now show a clear peak at small Δ^{train}, consistent with all other amplitudes. This means the network has learned all amplitudes in the training dataset equally well. This effect translates to the test sample qualitatively, so the performance on the test data improves after performance-boosting, but this improvement is less pronounced than for the training data. This means that, at the expense of an overtraining, we have improved our network from a fit-like description to an interpolation-like description of the largest amplitudes.

The pattern observed by performance-boosting points to a conceptual weakness of standard network training when it comes to precision applications. If we stop the network training at the point where the performance on a training sample exceeds the performance on the test

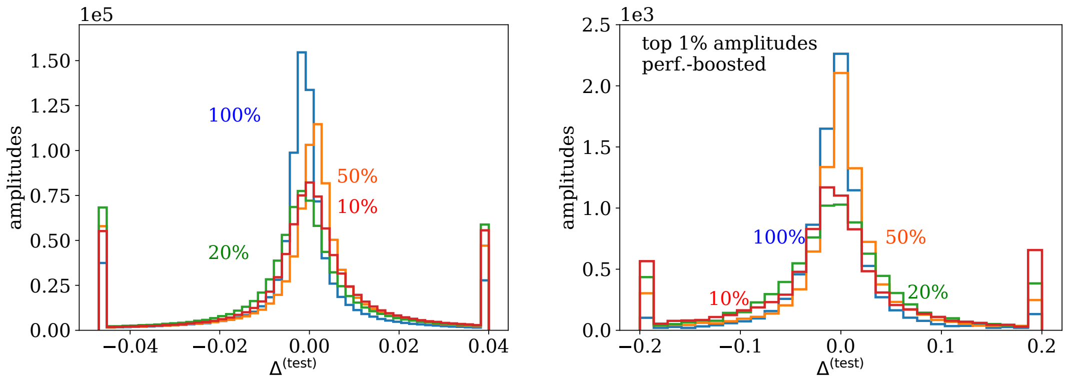

Figure 10: Performance of the BBN for all amplitudes (left) and a performance-boosted BNN for the largest 1% of all amplitudes (right), after training on different fractions of the full training dataset.

sample, we miss the opportunity of improving the network on the test and training data, but at a different rate. Overtraining is, per se, not a problem, as we know from applications of interpolation to describe data. The only challenge from such a network overtraining is a reliable uncertainty estimate from the generalization, for which we propose an appropriate scheduling of loss-boosting and performance-boosting.

4.3 Effect of training statistics

Given that our amplitude-BNN has successfully learned the amplitudes for the partonic process $gg \to \gamma\gamma g$ well below the percent level, with a small and simple network and 90k training points, we can ask the question how much training data we actually need for a precision amplitude network. For this study we use the same BNN as before, including loss-boosting and performance-boosting, but trained on a reduced dataset of

$$10\% \quad (9.000 \text{ amplitudes}) \quad \cdots \quad 100\% \quad (90.000 \text{ amplitudes})\,. \tag{20}$$

In Fig. 10 we show the corresponding Δ-distributions for the test dataset. Our smallest training dataset contains 9000 amplitudes, which turn out sufficient to train our network with its 6192 parameters. The corresponding network reproduces the test data well, albeit with sizeable overflow bins. Increasing the amount of training data improves the precision of the network, but relatively slowly. We observe the same level of improvement for all amplitudes and for the 1% largest amplitudes. For the latter we only show results after process boosting, without any boosting the quality of the low-statistics training is comparably poor.

5 Kinematic distributions

After illustrating the performance of the amplitude network in a somewhat abstract manner, we can also show 1-dimensional kinematic distributions. The integration of the remaining phase space dimension requires a little care, because we cannot just integrate the uncertainties together with the central values for the amplitudes.

For the central values we combine the amplitudes with phase space sampling. For example applying the simple Rambo [25] algorithm we identify the phase space weights with A. A 1-dimensional distribution is generated through bins which collect the sum of the amplitudes in the remaining phase space directions. The histogram value for a bin k is

$$h_k = \sum_{j=1}^{N} A_j\,. \tag{21}$$

To use the amplitudes predicted by the BNN we have to add the sampling over the weights ω. By replacing the truth amplitudes with the NN-amplitudes we can target the uncertainties from the modelling of the amplitudes through the BNN. In analogy to Eq.(8) and omitting the index k for the histogram we first extract a central histogram value as

$$\begin{aligned}\langle h\rangle &= \int d\omega\, q(\omega) \sum_j \bar{A}_i(\omega) \\ &= \int d\omega\, q(\omega)\,\bar{h}(\omega) \qquad \text{with} \qquad \bar{h}(\omega) = \sum_j \bar{A}_j(\omega)\,.\end{aligned} \tag{22}$$

Again in analogy to Eqs.(10) and (11) we define the absolute uncertainties on the bin entry as

$$\begin{aligned}\sigma^2_{h,\text{pred}} &= \int d\omega\, q(\omega) \left[\overline{h}(\omega) - \langle h \rangle \right]^2 \\ \sigma^2_{h,\text{model}} &= \int d\omega\, q(\omega) \left[\overline{h^2}(\omega) - \overline{h}(\omega)^2 \right] . \end{aligned} \tag{23}$$

The total uncertainty is again $\sigma^2_{h,\text{tot}} = \sigma^2_{h,\text{model}} + \sigma^2_{h,\text{pred}}$. We can simplify $\sigma_{h,\text{model}}$ further. In all of the above formulas h is just a sum over amplitudes. If we assume that the corresponding σ_model values are uncorrelated, we can relate $\sigma_{h,\text{model}}$ to σ_model by exchanging the sum and the variance,

$$\begin{aligned}\sigma^2_{h,\text{model}} &= \int d\omega\, q(\omega) \operatorname{Var}(\overline{h}(\omega)) \\ &= \int d\omega\, q(\omega) \operatorname{Var}\left(\sum_j A_j(\omega) \right) = \int d\omega\, q(\omega) \sum_j \operatorname{Var}\left(A_j(\omega) \right) \\ &= \int d\omega\, q(\omega) \sum_j \sigma^2_{\text{model},j}(\omega) \equiv \left\langle \sum_j \sigma^2_{\text{model},j}(\omega) \right\rangle . \end{aligned} \tag{24}$$

While we assume uncorrelated uncertainties for σ_model we cannot do the same for σ_pred. To compute σ_pred we first sample a set of weight configurations, which turns our BNN into an ensemble of neural networks, and then use each of these neural networks to compute the corresponding histogram value. Computing the standard deviation of these values gives us an estimate for $\sigma_{h,\text{pred}}$. By sampling from the weight distributions we change the neural network itself and all of its predictions. To assume that these changes are uncorrelated for different amplitudes seems not exceptionally well justified.

Based on this procedure we show BNN-amplitude results for a set of kinematic distributions in the upper panels of Fig. 11. We see the effect of limited training data towards the end of the different kinematic distributions, where the agreement between the NN-amplitudes and truth deteriorates. For our reference process this happens for $|\eta_g| \gtrsim 2.5$ or $|\eta_\gamma| \gtrsim 1.5$. Still, the BNN uncertainty estimate covers the deviation from the truth reliably.

In the lower panels of Fig. 11 we see that after performance-boosting the BNN predictions agree with the training data spectacularly well. This is the goal of the boosting and leads to the network learning all features in the training data extremely well. In the phase space regions where the regular BNN precision is limited by sparse and large training amplitudes, the improved agreement between NN-amplitudes and the training data carries over to the test data at a level that the network prediction is significantly improved. The uncertainties for the training data still cover the deviations from the truth, but unlike the central values this uncertainty estimate does not generalize correctly to the test data. This structural issue with process boosting could be ameliorated by alternating between loss-boosting and performance-boosting, until the specific requirements of a given analysis in precision and uncertainty estimates are met.

6 Outlook

Learning loop-amplitudes for LHC simulations is a classic ML-problem, because we need to train a precision network only once to provide a much faster simulation tool which can be used many times. In this application neural networks really work like better fits to the training data. Unlike for many other network applications, the training amplitudes are not noisy,

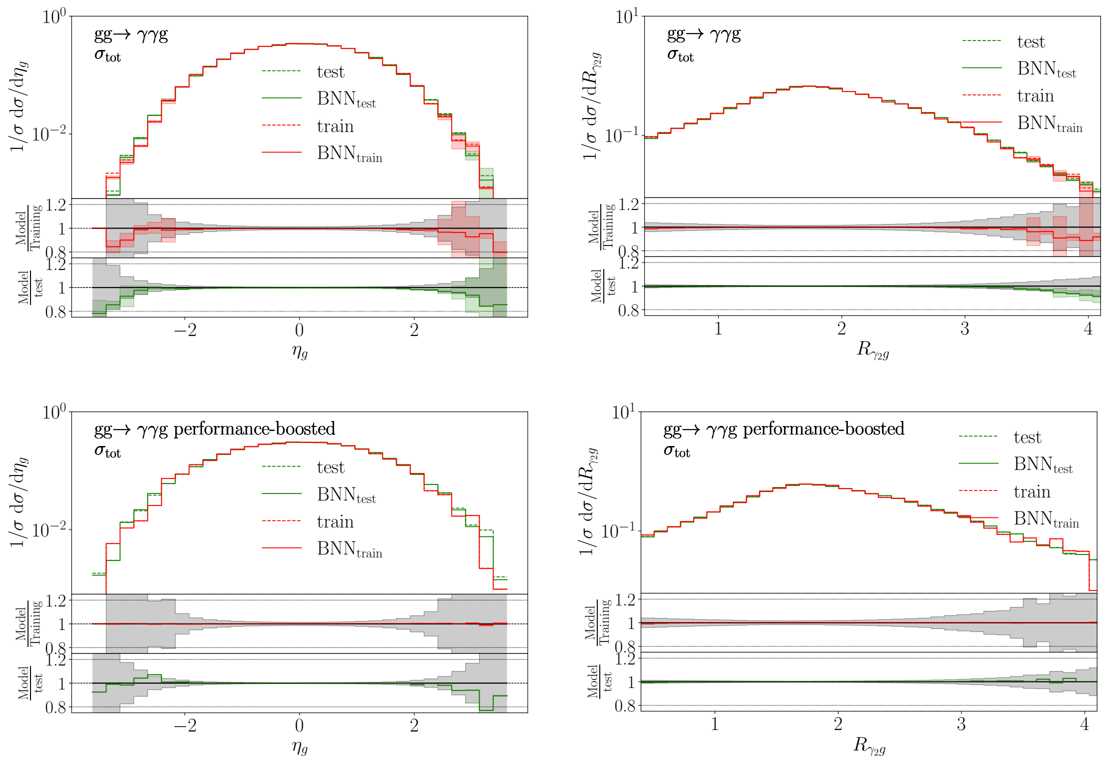

Figure 11: Kinematic distributions from the BNN without boosting (upper) and after performance-boosting (lower). The grey error bars in the lower panels indicate the statistical limitation of the training and test data.

which means we would like to reproduce the training amplitudes exactly, supplemented with a controlled uncertainty over all of phase space. To provide a reliable uncertainty map over phase space, we can rely on Bayesian regression networks [11].

The precision task reminds us of an interpolation rather than a fit, which means we need modify our ML-approach conceptually. If we are willing to accept a certain amount of overtraining, we can significantly improve the network training through boosting certain amplitudes. Because the Bayesian network provides a reliable uncertainty estimate, we can select the to-be-boosted amplitudes based on their deviation from the training data in units of the uncertainty. This loss-based boosting simply improves the self-consistency of the Bayesian network training. In a second step, we can boost training amplitudes just based on their absolute uncertainty. This selection helps with the precision for a given process, and because we use the absolute uncertainty we typically focus on the largest amplitude values.

We have applied Bayesian network training and the two strategies of amplitude boosting to the partonic process $gg \to \gamma\gamma g$ [9]. We have first found that the network amplitudes agree with the true amplitudes at the sub-percent level, for the training data and for a test dataset. For the 1% largest amplitudes an agreement at the percent level required process-specific performance boosting. For 1-dimensional kinematic distributions we have seen that the performance-boosting allows for extremely precise predictions in kinematic tails, albeit with a somewhat reduced performance in the uncertainty estimate for the test dataset. This can be improved by alternating between process and loss boosting in order to retain improved uncertainty estimation and increased performance which will be subject of future studies.

Finally, we have checked what happens with our boosted Bayesian network training when we reduce the number of training amplitudes from 90k to 9k and found that thanks to the boosting this only leads to a mild decrease in the network precision. This leaves us confident that boosted amplitude training with its shift from a fit-like to interpolation-like objective provides us with highly efficient surrogate models whenever the generation of training data is CPU-intensive.

Acknowledgments

First, we would like to thank Manuel Haußmann for introducing us to Bayesian networks. We thank Steffen Schumann for many enlightening discussions and Frank Krauss for regularizing any unreasonable enthusiasm. We are also grateful to Joseph Aylett-Bullock and Ryan Moodie for helpful conversations. This research is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 – TRR 257: *Particle Physics Phenomenology after the Higgs Discovery*, through Germany's Excellence Strategy EXC 2181/1 - 390900948 (the Heidelberg STRUCTURES Excellence Cluster) and the European Union's Horizon 2020 research and innovation programmes *High precision multi-jet dynamics at the LHC* (consolidator grant agreement No 772009).

A The $(2 \to 4)$-process

We know that that increasing the number of particles in the final state leads to a significant drop in network performance [7–9, 26, 27]. We illustrate how we can use the boosted BNN training to improve network predictions for the $(2 \to 4)$-process

$$gg \to \gamma\gamma gg \, . \tag{25}$$

For this process we use a network with seven hidden layers, 24 kinematic input dimensions, $\{32, 64, 256, 512, 128, 64, 32\}$ nodes, and two output dimensions corresponding to the amplitude and its uncertainty. This larger network has around 600k parameters. The training dataset contains around 90k amplitudes. Aside from these changes, we apply the same basic BNN training with two levels of loss boosting and process-specific performance boosting.

As for the $(2 \to 3)$-process we first show the performance of the network training in Fig. 12. While the overall scale of the agreement has increased for the sub-percent level to the percent level, we still see that the network has learned the largest amplitudes extremely well after process boosting. Unlike for the standard BNN, there is a certain amount of overtraining after performance-boosting, indicating the shift from a fit-like network training to an interpolation-like training.

Next, we check the consistency of the network output by looking at the ω-dependent pull distribution defined Eq.(17). We see that especially for large amplitudes the loss-boosting guarantees a well-behaved, consistent network, while the additional process boosting reverses

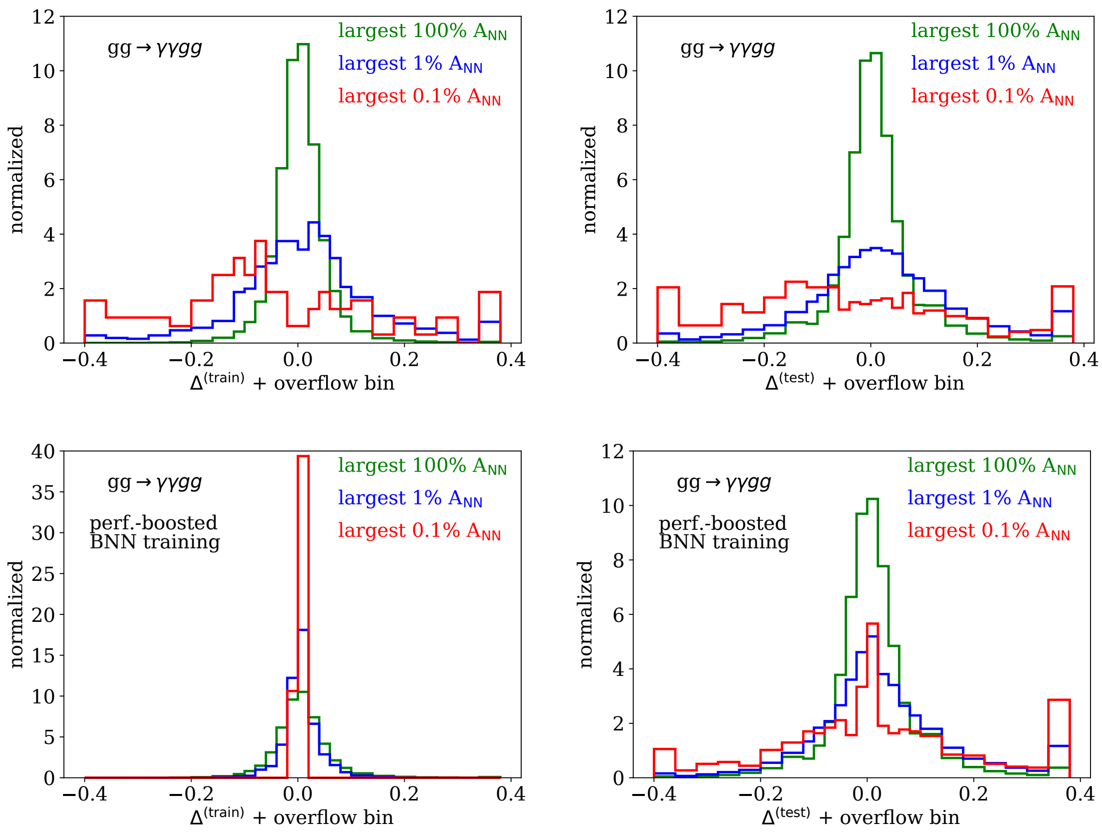

Figure 12: Performance of the basic (upper) and performance-boosted (lower) BNNs for the $(2 \to 4)$-process in terms of the precision of the generated amplitudes, Eq.(3), evaluated on the training (left) and test datasets (right).

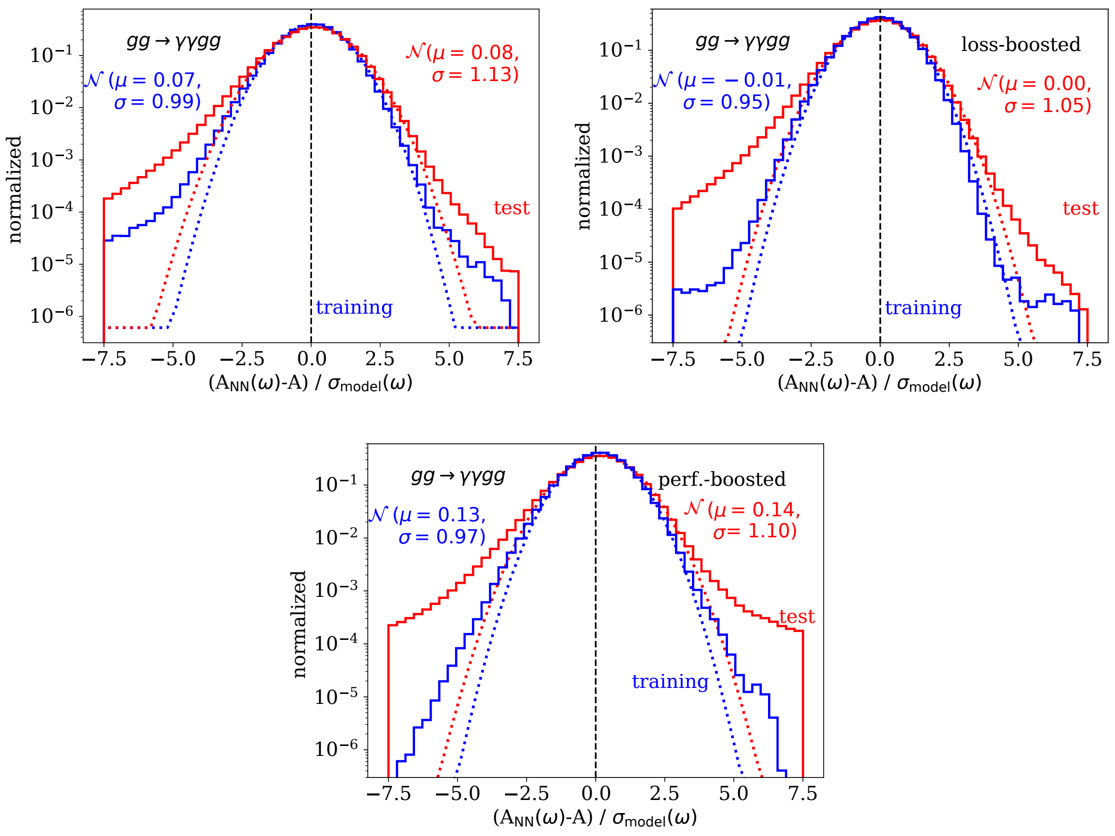

Figure 13: Pulls of the loss-bossted (left), and performance-boosted (right) BNN for the $(2 \to 4)$-process, defined in Eq.(17) and evaluated on the training and test data.

some of the beneficial effects of the loss-boosting. This effect was already observed for the $(2 \to 3)$-process.

Finally, we show the 1-dimensional kinematic distributions for the basic BNN and for the performance-boosted BNN. As for the $(2 \to 3)$-process the boosting step has a spectacula effect on the training data in the poorly learned kinematic tails. After integrating over the additional phase space directions this improvement translates well into the test dataset, but at the expense of the uncertainty estimate on the training data.

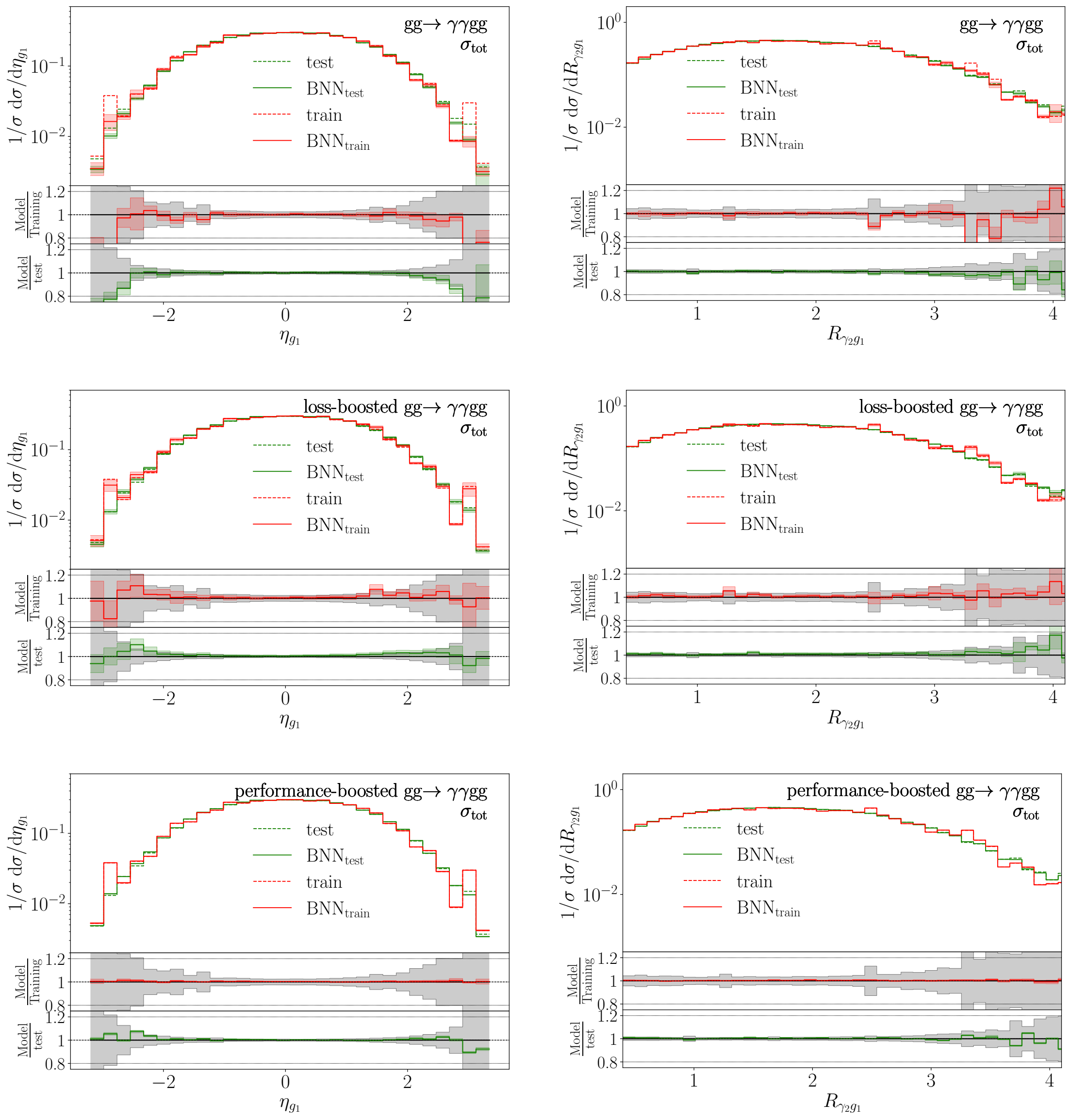

Figure 14: Kinematic distribution for the $(2 \to 4)$-process without boosting (upper), after loss boosting (center), and after process boosting (lower). The grey error bars in the lower panels indicate the statistical limitation of the training and test data.

References

[1] T. Plehn, A. Butter, B. Dillon and C. Krause, *Modern Machine Learning for LHC Physicists* (2022), arXiv:2211.01421.

[2] A. Butter, T. Plehn, S. Schumann *et al.*, *Machine Learning and LHC Event Generation* (2022), arXiv:2203.07460.

[3] F. Bishara and M. Montull, *(Machine) Learning Amplitudes for Faster Event Generation* (2019), arXiv:1912.11055.

[4] M. Czakon, *Tops from Light Quarks: Full Mass Dependence at Two-Loops in QCD*, Phys. Lett. B **664**, 307 (2008), doi:10.1016/j.physletb.2008.05.028, arXiv:0803.1400.

[5] S. Borowka, N. Greiner, G. Heinrich, S. P. Jones, M. Kerner, J. Schlenk, U. Schubert and T. Zirke, *Higgs Boson Pair Production in Gluon Fusion at Next-to-Leading Order with Full Top-Quark Mass Dependence*, Phys. Rev. Lett. **117**(1), 012001 (2016), doi:10.1103/PhysRevLett.117.079901, [Erratum: Phys.Rev.Lett. 117, 079901 (2016)], arXiv:1604.06447.

[6] K. Danziger, T. Janßen, S. Schumann and F. Siegert, *Accelerating Monte Carlo event generation – rejection sampling using neural network event-weight estimates*, SciPost Phys. **12**(5), 164 (2022), doi:10.21468/SciPostPhys.12.5.164, arXiv:2109.11964.

[7] S. Badger and J. Bullock, *Using neural networks for efficient evaluation of high multiplicity scattering amplitudes*, JHEP **06**, 114 (2020), doi:10.1007/JHEP06(2020)114, arXiv:2002.07516.

[8] D. Maître and H. Truong, *A factorisation-aware Matrix element emulator*, JHEP **11**, 066 (2021), doi:10.1007/JHEP11(2021)066, arXiv:2107.06625.

[9] J. Aylett-Bullock, S. Badger and R. Moodie, *Optimising simulations for diphoton production at hadron colliders using amplitude neural networks*, JHEP **08**, 066 (2021), doi:10.1007/JHEP08(2021)066, arXiv:2106.09474.

[10] Y. Gal, *Uncertainty in Deep Learning*, Ph.D. thesis, Cambridge, http://mlg.eng.cam.ac.uk/yarin/thesis/thesis.pdf (2016).

[11] G. Kasieczka, M. Luchmann, F. Otterpohl and T. Plehn, *Per-Object Systematics using Deep-Learned Calibration*, SciPost Phys. **9**, 089 (2020), doi:10.21468/SciPostPhys.9.6.089, arXiv:2003.11099.

[12] S. Bollweg, M. Haußmann, G. Kasieczka, M. Luchmann, T. Plehn and J. Thompson, *Deep-Learning Jets with Uncertainties and More*, SciPost Phys. **8**(1), 006 (2020), doi:10.21468/SciPostPhys.8.1.006, arXiv:1904.10004.

[13] J. Y. Araz and M. Spannowsky, *Combine and Conquer: Event Reconstruction with Bayesian Ensemble Neural Networks* (2021), arXiv:2102.01078.

[14] M. Bellagente, M. Haußmann, M. Luchmann and T. Plehn, *Understanding Event-Generation Networks via Uncertainties* (2021), arXiv:2104.04543.

[15] A. Butter, T. Heimel, S. Hummerich, T. Krebs, T. Plehn, A. Rousselot and S. Vent, *Generative Networks for Precision Enthusiasts* (2021), arXiv:2110.13632.

[16] I. Chahrour and J. D. Wells, *Comparing Machine Learning and Interpolation Methods for Loop-Level Calculations*, SciPost Phys. **12**, 187 (2022), doi:10.21468/SciPostPhys.12.6.187, arXiv:2111.14788.

[17] E. Bothmann *et al.*, *Event Generation with Sherpa 2.2*, SciPost Phys. **7**(3), 034 (2019), doi:10.21468/SciPostPhys.7.3.034, arXiv:1905.09127.

[18] S. Badger, B. Biedermann, P. Uwer and V. Yundin, *Numerical evaluation of virtual corrections to multi-jet production in massless QCD*, Comput. Phys. Commun. **184**, 1981 (2013), doi:10.1016/j.cpc.2013.03.018, arXiv:1209.0100.

[19] A. Butter, T. Plehn and R. Winterhalder, *How to GAN LHC Events*, SciPost Phys. **7**(6), 075 (2019), doi:10.21468/SciPostPhys.7.6.075, arXiv:1907.03764.

[20] D. MacKay, *Probable Networks and Plausible Predictions – A Review of Practical Bayesian Methods for Supervised Neural Networks*, Comp. in Neural Systems **6**, 4679 (1995), http://www.inference.org.uk/mackay/network.pdf.

[21] A. Kendall and Y. Gal, *What uncertainties do we need in bayesian deep learning for computer vision?*, CoRR **abs/1703.04977** (2017), http://arxiv.org/abs/1703.04977, 1703.04977.

[22] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization* (2014), arXiv:1412.6980.

[23] S. Diefenbacher, E. Eren, G. Kasieczka, A. Korol, B. Nachman and D. Shih, *DCTRGAN: Improving the Precision of Generative Models with Reweighting*, JINST **15**(11), P11004 (2020), doi:10.1088/1748-0221/15/11/P11004, arXiv:2009.03796.

[24] R. Winterhalder, M. Bellagente and B. Nachman, *Latent Space Refinement for Deep Generative Models* (2021), arXiv:2106.00792.

[25] R. Kleiss, W. J. Stirling and S. D. Ellis, *A New Monte Carlo Treatment of Multiparticle Phase Space at High-energies*, Comput. Phys. Commun. **40**, 359 (1986), doi:10.1016/0010-4655(86)90119-0.

[26] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale and S. Schumann, *Exploring phase space with Neural Importance Sampling*, SciPost Phys. **8**(4), 069 (2020), doi:10.21468/SciPostPhys.8.4.069, arXiv:2001.05478.

[27] C. Gao, S. Höche, J. Isaacson, C. Krause and H. Schulz, *Event Generation with Normalizing Flows*, Phys. Rev. D **101**(7), 076002 (2020), doi:10.1103/PhysRevD.101.076002, arXiv:2001.10028.