Department of Physics and Astronomy

University of Heidelberg

Master thesis

in Physics

submitted by

Michel Luchmann

born in Bad Wildungen

2019

Uncertainties with Bayesian neural networks in particle physics

This Master thesis has been carried out by Michel Luchmann

at the

Institut für theoretische Physik

under the supervision of

Prof. Dr. Tilman Plehn

Kurzzusammenfassung:

Aktuelle Studien zeigen, dass Deep Learning basiertes Jet-Tagging etablierte Methoden, welche auf multivariaten Analysen von Jet-Observablen beruhen, übertrifft. Wie Unsicherheiten zu diesen Deep Learning Methoden hinzugefügt werden können und wie Stabil diese Resultate mit Hinsicht auf systematische Unsicherheiten sind, bleiben offene Fragen. Bayes'sche Neuonale Netzwerke stellen neben der Standardklassifikationsausgabe eine Unsicherheit auf Jet-Ebene zur Verfügung. Um zu überprüfen, wie diese Unsicherheiten mit Unsicherheiten korrelieren, die von limitierten Trainingsdatensätzen kommen, Pile-up, oder systematischen Unsicherheiten, wurden für diese Masterarbeit Bayes'sche Neuroanle Netze auf einem Top-Tagging Datensatz trainiert. Außerdem wird ein Vergleich mit einem frequentistischen Ansatz gegeben und die Abhängigkeit des Priors wird überprüft.

Abstract:

Recent studies showed that deep learning based jet tagging outperforms established methods based on multivariate analyses of jet observables. However, how uncertainties can be included to these deep learning approaches and how stable these results are with respect to systematic uncertainties remain open questions. Bayesian neural networks provide, in addition to the standard classification output, a jet-level/event-level uncertainty. In this thesis, to investigate how these event-level uncertainties correlate with uncertainties from finite training data, pile-up and systematic uncertainties, Bayesian neural networks were trained on a top tagging dataset. Furthermore, a comparison with a frequentist-like approach is given and the prior dependence is studied.

Contents

1 Introduction

The Large Hadron Collider (LHC) was built to study the fundamental interactions and building blocks of nature, such as the Higgs boson, whose discovery was announced in 2012 [1]. Because of the large momentum transfer provided by the colliding protons, boosted heavy excitations such as the W-boson, the Z-boson, the Higgs boson or top quarks are frequently produced at the LHC. The transverse momentum p_T is often larger than the mass threshold leading to boosted decay products. These heavy states often decay into quarks and gluons. However, quarks and gluons are not the final states, that are observed as energy depositions in the detector. Instead, collinear sprays of hadrons are the real final states and, thus, the observed particles. These collections of hadrons are referred to as jets. The boosted decay products of a heavy excitation, such as a top quark, typically lead to a jet with a large spatial width and is therefore referred to as a fat jet. These fat jets inherit substructures, which need to be studied to identify the underlying event [2, 3, 4, 5]. A boosted top jet, for instance, is expected to lead to a three-prong substructure. The top quark decays in a W-boson and a bottom quark and the W-boson itself decays in 67% of the cases into two light quarks [6], leading to a three-prong fat jet. This substructure distinguishes top jets from jets arising from hard processes involving only light quarks and gluons. Therefore, it is crucial to perform an analysis including this substructure information to be able to study processes involving hadronically decaying Higgs bosons or top quarks. Established methods mostly rely on jet observables [7], which are sensitive to the prong-structure or other kinematic observables such as the jet mass. However, the studies discussed in this thesis rely on a different approach. Instead of constructing jet observables and combining them via multivariate analyses [8], deep learning methods are applied. The high-dimensionality of jets and the Monte Carlo simulations are perfect conditions for this approach. These methods rely on constructing neural networks with thousands or millions of parameters, giving the approach the ability to learn arbitrary relationships between a high-dimensional input, such as the energy depositions or 4-momenta of jet-constituents, and an output, such as the likelihood of having a top jet. However, despite recent promising results showing increased performance [9, 10] these deep learning taggers have to be evaluated not just on their performance, but their ability of capturing uncertainties or their stability with respect to real experimental conditions, involving for instance pile-up. Bayesian neural networks provide, in addition to the standard classifier output, a jet-level uncertainty on each prediction. How this jet-level uncertainty correlates with uncertainties known in physics, such as statistical uncertainties or systematic uncertainties, will be one of the main subjects of this thesis.

Most of the studies presented in this thesis were published in Ref. [11]. The thesis is structured in the following way. Chapter 2 will give a short introduction to jets and top tagging. Chapter 3 will introduce neural networks and how they can be used for classification. Chapter 4 will explain the concept of Bayesian neural networks, the relation to standard regularization techniques and how an event/jet-level uncertainty can be constructed. Chapter 5 will show how Bayesian neural networks can be applied to top tagging. It will be shown how the output uncertainties correlate with the statistics of the training data and how Bayesian neural networks react to effects like pile-up or systematic uncertainties. Furthermore, some additional features will be presented such as the calibration of classifiers and a comparison to a frequentist-like method will be made.

2 Jets and top tagging

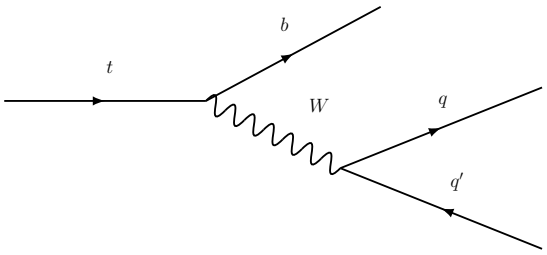

Figure 2.1: Feynman diagram of a hadronically decaying top quark.

Heavy excitations, such as the top quark, are produced frequently at the LHC, often with a transverse momentum, p_T, fulfilling the condition $\frac{p_T}{m_t} \gtrsim 1$ and, therefore, leading to boosted decay products. These boosted decay products contain quarks and gluons, which produce jets. The description of this process is usually split into three parts. The hard process, which describes the high energy process of the incoming partons interacting with each other and forming intermediate excitations such as a Higgs boson or a top quark, which themselves decay further into leptons and quarks; the showering, which refers to the collinear and soft radiation arising from quarks and gluons of the hard process; and the hadronization, in which the partons at an energy of around Λ_{QCD} hadronize to mesons and baryons. Jets arising from boosted heavy excitations are referred to as fat jets, because of their typical large geometrical size. These fat jets inherit inner structures, distinguishing them from QCD jets[1], i.e. jets arising from hard processes only involving light quarks and gluons. A boosted top quark decays into a W-boson and bottom quark. The W-boson itself can decay into light quarks (see Figure 2), leading to a three-prong structure. In contrast, a QCD jet will typically have a one-prong structure, because the jet is created by collinear and soft radiation of just one hard parton.

Before a jet-observable or a neural network can be applied to a jet, the jet has to be properly defined. There are various different jet algorithms, which cluster the observed hadrons into jets. A commonly used algorithm for LHC physics is the anti-k_t algorithm [12]. These algorithms are based on a distance measure in the detector plane. For LHC physics usually the azimuthal angle, ϕ, the (pseudo)-rapidity, η, and the transverse momentum, p_T, are used to describe observed particles and jets.

[1]QCD stands for Quantum Chromo Dynamics, which refers to the fact that these processes are completely described by QCD without any electroweak interactions.

The relevant distance measure for the anti-k_t algorithm is defined as:

$$d_{ij} = \Delta R_{ij}^2 \min\left(p_{Ti}^{-2}, p_{Tj}^{-2}\right) , \tag{2.1}$$

where R_{ij} is given by $R_{ij} = \sqrt{(\phi_i - \phi_j)^2 + (\eta_i - \eta_j)^2}$. The additional beam distance is defined as $d_{iB} = p_{ti}^{-2} R^2$ with R being the jet parameter, which determines the typical scale of the clustered jets. The algorithm clusters particles iteratively to jets by computing all possible distances d_{ij} of particles/jets and combines these with the shortest distance. If the shortest distance is, however, the beam distance d_{iB}, the algorithm stops and the jet i is considered a final jet. Because of the inverse power of the transverse momentum k_t, the anti-k_t algorithm starts with the hardest constituents, building spherical jets around it. The resulting jets can then be used for an analysis by computing jet observables or deep learning based methods as they are subject of this thesis. The individual particles of a jet will be referred to as constituents throughout this thesis.

While this thesis will present studies involving deep learning based top taggers, the methods discussed in this thesis are not in any way restricted to this case. For instance, quark gluon discrimination [13] would be an example not relying on fat jets.

3 Neural networks

This chapter will give a short overview of the topic of neural networks by introducing the most common types of layers, fully connected dense layers and convolutional layers, and the two theory-motivated layers, CoLa [14] and LoLa [14]. All four types of layers are part of the neural network architectures studied in this thesis. Furthermore, classification in the context of neural networks will be discussed. Building up on this overview, the next chapter will introduce Bayesian neural networks.

3.1 Fully connected and convolutional networks

Neural networks are defined by a set of layers which are described by specific linear and non-linear operations, involving parameters which are tuned in the so-called training process (see Section 3.2). In this context, these parameters are referred to as the weights of the network. The output of one layer is the input of the next layer which in turn provides the input of the following layer. Networks with many intermediate, or inner, layers are referred to as deep neural networks. Although it is proven that a neural network with only one hidden layer with a sufficient number of trainable parameters, can approximate arbitrary functions [15], in practice deep networks turned out to show significant improvements [16], which makes them powerful enough to handle high dimensional input and learn arbitrary relations between input and output.

The most common type of layers are fully connected dense layers, which are defined as:

$$y = f(z), \quad z = Wx + b\,, \tag{3.1}$$

where x is the input vector, y the output vector, b the bias vector and W a matrix of model parameters. The elements of W and b are the weights[1] of the layer. The individual entries of y are usually referred to as (hidden) units. The number of units, i.e. the dimension of y, is a free parameter and has to be chosen in advance, when constructing the architecture of a network; $f(.)$ is an arbitrary non-linear function applied after the linear operation. A common choice is the Rectified Linear Unit (ReLU) activation function [17], which is defined as:

$$\mathrm{ReLU}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} . \tag{3.2}$$

[1] In some notations the bias is not referred to as a weight

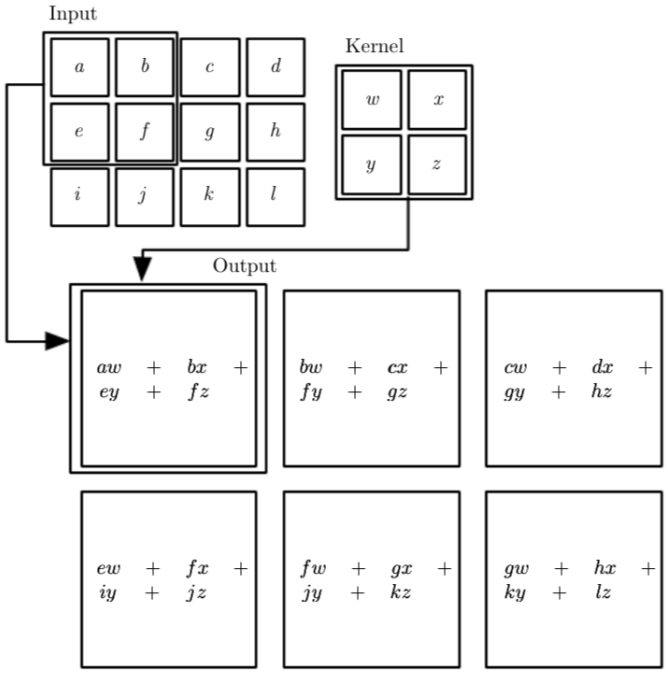

Figure 3.1: Illustration of a convolutional operation for one feature map. The figure is taken from Ref. [19, p. 330].

Fully connected dense layer appear in almost every network architecture. As the name suggests, every output unit is connected with every input unit. This gives a fully connected dense network the freedom to learn various different relations between input and output. However, connecting every input pixel with every output unit can be very inefficient for the case of image data. Therefore, another common choice of layers are so called convolutional layers. Convolutional layers exploit two common properties of images. First, features can usually be found anywhere in the image, i.e. translational symmetry, and second, features are usually local, i.e a group of surrounding pixels makes up the eye of a dog or the ears of a cat. For this kind of layer the standard linear operation of a dense layer is replaced by a linear convolutional operation, which is for 2D images defined[2] as:

$$F_{ij}^{k} = \sum_{l=0}^{n_{\text{kernel}}-1} \sum_{r,s=0}^{n_{\text{size}}-1} W_{rs}^{kl} I_{i+r,j+s}^{l} + b_k \quad \text{with } k = 0, \ldots, n_{\text{kernel}} - 1\,. \tag{3.3}$$

I_{ij}^{l} stands for the input-image, where l is the index of the different input feature maps, e.g. different colors of an image or the output of another convolutional layer; b_k is referred to as the bias vector. The expression from above is illustrated in Figure 3.1. The kernel matrix W_{rs} of size $(n_{\text{size}}, n_{\text{size}})$ is moved stepwise over the input feature map and constructs in this way one output feature map. If the 2D image is

[2]The notations are taken from [18]

flattened and represented as a vector, convolutional layer can be seen as sparse dense layer. The sparsity arises from the convolutional operation; just neighbouring pixels are connected to corresponding output units and the kernels lead to shared weights across the layer. Depending on the task convolutional layers are often followed by a series of fully connected dense layers (see architectures discussed in Section 5.7).

There are two additional types of layers used for the models studied in this thesis: CoLa and LoLa [14]. Both layers are especially designed to handle 4-momenta and are in this sense theory-motivated. Section 3.4 will explain both layers in detail.

3.2 Training

The weights, ω, of a neural network are obtained by minimizing a loss function, L, on the training dataset, D. The choice of this function depends on the problem (see Section 3.3 for more details). To find the global minimum of $L(D, \omega)$ with respect to the weights different optimization algorithms exist. They rely on calculating the gradient $\nabla_\omega L(D, \omega)$ and performing step-wise updates of the form:

$$\omega_{k+1} = \omega_k - \alpha \nabla L(D, \omega_k) \tag{3.4}$$

with α being the so called learning rate. Because of the large training datasets typically used in deep learning, the loss function is evaluated on N subsets $D_i \subset D$ with $\cup_{i=1}^{N} D_i = D$ and not on the full dataset. This is known as minibatch optimization and adds stochastic fluctuations to the gradients, which often help the training to not get stuck in local minima. Iterating over all minibatches is called an epoch and a simple update strategy as given above is known as stochastic gradient descant (SGD). More advanced update strategies are, for example, Adam [20] or Adadelta [21].

Besides the training dataset, D, typically two additional datasets are used, the validation and test dataset. All three datasets are statistically independent which means that they don't include the exact same examples. The validation dataset is used to monitor the performance during training and help to determine the optimal values of the hyperparameters, i.e. parameters which have to be tuned prior to the training such as the learning rate. The test dataset is used for the final evaluation of the network. For a binary classification task, for instance, the final evaluation would determine how well the trained network distinguishes two classes.

3.3 Classification with Neural networks

A classification task involves predicting a label c for each element x of a given dataset X. For instance, X could be a set of images of cats and dogs or, as it is considered

in this thesis, top and QCD jets. With only two classes, this corresponds to a binary classification task. Common practice is to construct the classifier in a way to predict a probability, p, rather than the label directly. In this case the label can be obtained by choosing, for instance, the class with the largest probability. The classifier can be written as:

$$f : X \to [0,1] \times \cdots \times [0,1] : f(x) = p(c|x) \tag{3.5}$$

with $p(c|x)$ being the probability vector of the image x being a member of one of the classes. If N classes are present, $p(c|x)$ is a vector with N entries. The probability of one of the classes is redundant, because the individual probabilities sum up to one. For a binary classification task, the function f can, therefore, be constructed as predicting only one of these probabilities.

A binary classification task in physics is, for instance, distinguishing between signal and background data as it will be discussed later in this thesis for top tagging. In this case, just the signal class is of importance and one would like to find an optimal trade off between rejecting as much background as possible while keeping a significant amount of signal data. For this purpose, rather than taking the class with the largest probability, usually a receiver operating characteristic curve (ROC curve) is constructed and, in the absence of systematic uncertainties, used to find an optimal working point.

The function, f, can be constructed by using a neural network and its weights, ω, which are obtained in the training process. To get an output between $[0, 1]$ the activation function of the last layer has to be chosen accordingly. A common choice is the sigmoid function for a binary classification task. The sigmoid function is defined as:

$$\text{sigmoid}(x) = \frac{e^x}{1+e^x}, \quad \text{sigmoid}^{-1}(y) = \ln\left(\frac{y}{1-y}\right). \tag{3.6}$$

For classification, the most frequently used loss function is the cross entropy, which gives the network output its probability interpretation. Because the labels of the training dataset have to be known for the evaluation of this loss function, this approach falls under the category of supervised learning. If $f^\omega(x_i)$ is the neural network output for a given training example x_i, the cross entropy can be written as:

$$L = \sum_{i=0}^{M} \sum_{c=0}^{N} -y_i^c \log p(c|x_i), \quad p(c|x_i) = f^\omega(x_i)\,, \tag{3.7}$$

where the first sum goes over the training data $D = (y_i, x_i)$, which contains M examples, and the second sum over all possible classes c; y_i^c is a binary label, which is 1 if class c is the correct class, and 0 otherwise. N is the number of classes and

$p(c|x_i)$ is the probability vector mentioned above. For binary classification, if the probability of the class c_1 is given by $p(c_1|x_i)$, the probability of the second class is just $1 - p(c_1|x_i)$. In this case, Equation 3.7 can be written as:

$$L = \sum_{i=0}^{M} \left(-y_i \log(p_i) - (1 - y_i) \log(1 - p_i)\right), \quad p_i = f^{\omega}(x_i)\,. \tag{3.8}$$

To make the equation more easily readable, $p(c_1|x_i)$ was shortened to p_i and $y_i^{c_1}$ to y_i respectively.

The minimization of Equation 3.8 can be thought of as a maximum likelihood (ML) fit. Because, if the expression for L is seen as a negative log-likelihood, the likelihood of observing the dataset D for a given set of weights ω becomes:

$$p(D|\omega) = \prod_{i=1}^{M} p_i^{y_i} \left(1 - p_i\right)^{1-y_i}\,. \tag{3.9}$$

The total likelihood of observing the correct labels is given as a product of the individual probabilities, where for each decision of the classifier either p_i or $1 - p_i$ is the probability of observing the correct label. This likelihood interpretation is important in the context of Bayesian neural networks (see Chapter 4).

3.4 LoLa and CoLa

The two types of layers CoLa [14], which stands for Combination Layer, and LoLa [14], which stands for Lorentz Layer, are designed to help a neural network extract physical features from a given list of 4-momenta. Both layers are theory-motivated and operate on 4-momenta. In this thesis, the architectures containing LoLa and CoLa were applied to top tagging. However, they could in principle be applied to any kind of physical task involving a set of 4-momenta.

The idea behind CoLa and LoLa is that first relevant linear combinations of constituents are performed, e.g. from substructures in jets, and then physical features like the mass and the transverse momentum of these linear combinations are constructed. The several linear combinations for CoLa and the repeating aspect of LoLa gives the network enough freedom to build various kind of combinations of physically relevant features. LoLa is typically followed by a series of fully connected dense layers, which further process these features and combine them to a probability output needed for classification (see Section 3.3).

CoLa builds linear combinations of 4-momenta. The number of constituents per jet has to be fixed in advance, thus leading to the requirement of having a standardized input. This can be accomplished by adding zeros to jets with fewer constituents

and removing the softest constituents for jets with too many. The operation of the CoLa layer can be summarized as:

$$\widetilde{p}_j^{\,\mu} = p_i^{\mu} C_{ij} \tag{3.10}$$

with the trainable matrix

$$C = \begin{pmatrix} 1 & 0 & \cdots & 0 & \omega_{11} & \omega_{12} & \cdots & \omega_{1M} \\ 0 & 1 & & & \omega_{21} & \omega_{22} & \cdots & \omega_{2M} \\ \vdots & & \ddots & & \vdots & \vdots & & \vdots \\ 0 & & & 1 & \omega_{N1} & \omega_{N2} & \cdots & \omega_{NM} \end{pmatrix} . \tag{3.11}$$

The unitary matrix is included to pass the original 4-momenta of each constituent, in addition to the linear combinations, to the next layer. The number of combinations of 4-momenta, M, i.e. the number of additional columns added to C, is a free parameter of CoLa. Experiments showed that $M = 10$ is a good choice for top tagging, however small variations do not perform significantly worse [14, p. 5]. Therefore, for all architectures discussed in this thesis M was chosen to be 10.

After CoLa, a Lorentz layer is applied. If we think of the input as a matrix with the entries of the 4-momenta as rows and the constituents as columns, CoLa adds entries in the constituent direction $(4, N_{\text{constit}}) \to (4, N_{\text{constit}} + M)$, while LoLa transforms the feature space. There are many different options for LoLa, but for the architectures used in these studies the following replacements are performed:

$$\begin{pmatrix} \tilde{E}_i \\ \tilde{p}_{x,i} \\ \tilde{p}_{y,i} \\ \tilde{p}_{z,i} \end{pmatrix} \to \begin{pmatrix} m_i^2 \\ p_{T,i} \\ \sum_j \omega_{ij}^{(E)} E_j \\ \sum_j \omega_{ij}^{(m)} m_j^2 \\ \sum_j \omega_{ij}^{(p)} p_{T,j} \\ 2 \times \sum_j \omega_{ij}^{(d)} d_{ij} \\ 5 \times \min_j \omega_{ij}^{(d)} d_{ij} \end{pmatrix} , \tag{3.12}$$

where m_i and $p_{T,i}$ are the invariant mass and transverse momentum of the constituent i. Note that these operations are performed after CoLa, therefore some of the constituent entries are linear combinations of 4-momenta, leading to non-zero invariant masses. The following three entries are the weighted sum over energies, invariant masses and transverse momenta. The last two entries describe weighted Minkowski distances between two constituents i and j. The Minkowski distance d_{ij} is defined as:

$$d_{ij} = (\tilde{p}_i - \tilde{p}_j)^{\mu} \, (\tilde{p}_i - \tilde{p}_j)_{\mu} \, . \tag{3.13}$$

For the last entry of the feature vector, the sum over the constituents is replaced by the minimum. The factors 2 and 5 indicate that the entries are repeated with

different weights, resulting in a total of 12 feature entries. These replacements are performed on each individual constituent, even for the entries with a sum over all constituents. Thus, the individual entries are $N_{\text{constit}} + M$ copies of the same operation but with different weights. As for CoLa, this specific choice for LoLa was found to perform well for top tagging, but variations do not perform significantly worse [14, p. 5].

4 Bayesian neural networks

While the last chapter gave a short overview of the topic of neural networks and explained how neural networks can be used for classification, this chapter builds on this and explains the idea behind Bayesian neural networks. Bayesian neural networks produce output distributions instead of single numbers, providing an uncertainty for each prediction. Section 4.1 will present the theoretical background needed to understand the output of a Bayesian neural network. The following section will show parallels to existing regularization techniques, which will be important for some of the discussions in Chapter 5. In the Sections 4.3 and 4.4 some technical details about the training and justifications of the approach will be given, while Section 4.5 will explain in detail the shape of the output distributions arising for binary classification. The results of applying Bayesian neural networks to top tagging will be covered in Chapter 5.

4.1 Bayes' theorem and variational interference

As discussed in Chapter 3, the weights of a neural network are obtained by minimizing a loss function on a training dataset D. This can usually be seen as a maximum likelihood fit (see Section 3.3). In contrast, for a Bayesian neural network (BNN) [22, 23, 24] the weights are not fixed to specific values, but treated as random variables following a probability distribution $p(\omega|D)$ (see Figure 4.1). This distribution is obtained via Bayes' theorem:

$$p(\omega|D) = \frac{p(D|\omega)p(\omega)}{p(D)} \tag{4.1}$$

and is called the posterior distribution, where $p(D|\omega)$ is the likelihood of observing the training data D for a given network and its parameters ω. $p(\omega)$ is a the prior and can be chosen freely. A standard choice would be a Gaussian distribution with a mean of $\mu_{\text{prior}} = 0$ and the width σ_{p} treated as a hyperparameter, i.e. a free parameter which has to be chosen prior to the training. For a classification task, we are interested in predicting the likelihood $p(c|x)$ of an event x (see also Section 3.3), which is not a part of the training sample, to be a member of the class c. In the Bayesian approach, $p(c|x)$ is obtained via:

$$p(c|x) = \int d\omega \, p(\omega|D) \, p(c|x,\omega) \, , \tag{4.2}$$

where $p(c|x,\omega)$ is the neural network output. It can be interpreted as an average over an infinite number of possible weight configurations. The likelihood of each

weight configuration is determined by $p(\omega|D)$. The integral from Equation 4.2 is intractable for neural networks involving thousands or even million of parameters. One way to solve this problem is called variational interference [25]. The posterior is approximated by simple tractable distributions $q_\theta(\omega)$. In this way, the integral in Equation 4.2 can be approximated as:

$$p(c|x) \approx \int d\omega \, q_\theta(\omega) \, p(c|x, \omega) \,, \tag{4.3}$$

which can be solved by Monte Carlo integration. To obtain the parameters θ of $q_\theta(\omega)$ the Kullback-Leibler-divergence (KL-divergence) is minimized:

$$\mathrm{KL}[q(\omega), p(\omega|D)] = \int d\omega \; q(\omega) \; \log \frac{q(\omega)}{p(\omega|D)} \,. \tag{4.4}$$

The KL-divergence is 0 for identical distributions and positive otherwise. It can be seen as a measure for how similar two distributions are. Using Equation 4.2 leads to:

$$\begin{aligned} \mathrm{KL}[q_\theta(\omega), p(\omega|D)] &= \int d\omega \; q_\theta(\omega) \; \log \frac{q(\omega)p(D)}{p(D|\omega)p(\omega)} \\ &= \underbrace{\mathrm{KL}[q_\theta(\omega), p(\omega)]}_{\text{regularization}} + \underbrace{\log p(D) \int d\omega \; q(\omega)}_{\text{const.}} - \int d\omega \; q_\theta(\omega) \; \underbrace{\log p(D|\omega)}_{\text{log-likelihood}} \,. \end{aligned} \tag{4.5}$$

The second term can be omitted because it does not depend on the variational parameters θ. The first term depends on the prior, but not on the training data. A non-data dependent part in the loss function can be seen as a regularization term. The third term involves the negative log-likelihood, which would be the cross entropy for classification (see Section 3.3), with a $q_\theta(\omega)$-weighted integral over weight-space. Both terms will be discussed in more detail in the next section.

Having optimized the parameters of the variational distributions, the prediction can be computed with the help of Equation 4.2 as:

$$p(c|x) = \int d\omega \; q(\omega) p(c|x; \omega) \approx \frac{1}{N} \sum_{i=1}^{N} p(c|x, \omega_i) =: \mu_{pred} \quad \omega_i \in q(\omega) \,, \tag{4.6}$$

which we call the predictive mean. The mean can be seen as averaging over an ensemble of networks with different weight configurations. In addition, we can define the predictive standard deviation [26] as:

$$\sigma^2_{pred} = \frac{1}{N} \sum_{i=1}^{N} \left(p(c|x; \omega_i) - \mu_{pred} \right)^2 \,, \tag{4.7}$$

which can be seen as an uncertainty for each prediction. The meaning of this uncertainty and how it correlates with statistics and systematic uncertainties will be one of the main aspects of this thesis.

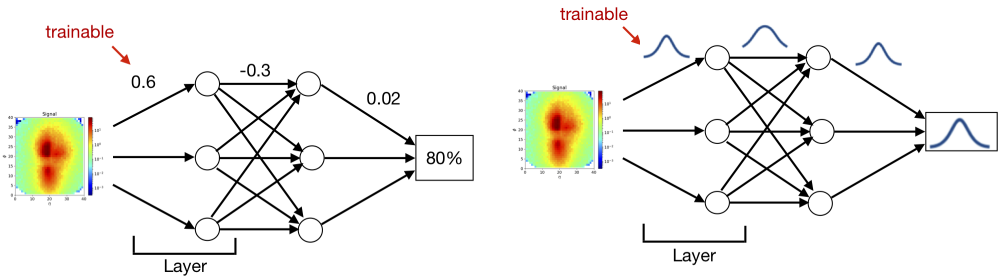

Figure 4.1: Illustration of the difference between a neural network and a Bayesian neural network. The scalar weights are replaced by distributions, leading to an output distribution, instead of a number.

4.2 Bayes, L2 regularization and dropout

Many aspects of the loss function introduced in the last section can be related to standard regularization methods such as dropout and L2 regularization. Both are techniques used in the machine learning community to prevent overfitting. While this will help to understand the loss function in more detail, it will be also important for Section 5.5, where an alternative frequentist-like method is discussed.

Deep neural networks are usually powerful enough to not just learn the general features but also the statistical fluctuations of the training dataset. This is referred to as overfitting and decreases the performance on a test sample. One way to solve this problem is by reducing the model complexity, i.e. using less expressive models [27, p. 9]. L2 regularization adds a term like:

$$L_2 = \lambda |\omega|^2 \tag{4.8}$$

to the loss function, where λ is a free parameter and $|\omega|^2 = \sum_{i=1}^{N} \omega_i^2$ the squared sum over all weights. Minimizing the loss function with this additional term will lead to smaller weights. Thus, L2 regularization restricts the parameter space of the network and reduces the complexity. This potentially reduces overfitting. The parameter λ determines how strongly the network weights should be restricted. It will be shown in this section that L2 regularization is equivalent to a Gaussian prior.

For standard dropout [28], on the other hand, for each training iteration a randomly chosen fraction of network weights are set to 0. The only parameter determining dropout is the dropout rate, which determines the fraction of weights set to 0. Dropout prevents highly tuned co-adaptation between hidden units. The idea is that these co-adaptations between hidden units, which encode learnt features, are fine tuned on the training dataset. Thus, by randomly dropping units during training co-adaptation is reduced and single hidden units are forced to learn features without relying too much on other units. This is the usual interpretation given, however this

section will show how dropout can be understood in a Bayesian context as well.

The first part of the Bayesian loss function (Equation 4.5) involves the KL-divergence between the prior $p(\omega)$ and the variational distribution $q_\theta(\omega)$. Because of simplicity both are chosen to be Gaussian distributions (see Section 4.3 for more details). To each weight ω a variational distribution $q_\theta(\omega) = G(\omega|\mu_{\text{weight}}, \sigma_{\text{weight}})$ and a prior $p(\omega) = G(\omega|\mu_{\text{prior}} = 0, \sigma_{\text{prior}})$ is assigned. The prior width μ_{prior} is set to 0, because we do not assume to have positive values preferred to negative values or vice versa. The parameters of the prior are shared across all weights. The integral factorizes into separate integrals over each weight and each term can be computed as:

$$\begin{aligned}
\text{KL}[q_\theta(\omega), p(\omega)] &= \int d\omega\, q_\theta(\omega)\, \log \frac{q(\omega)}{p(\omega)} && (4.9)\\
&\sim \log\left(\frac{\sigma_{\text{prior}}}{\sigma_{\text{weight}}}\right) + \frac{\sigma^2_{\text{weight}} + (\mu_{\text{weight}} - \mu_{\text{prior}})^2}{2\sigma^2_{\text{prior}}} - \frac{1}{2} && (4.10)\\
&\sim \log\left(\frac{\sigma_{\text{prior}}}{\sigma_{\text{weight}}}\right) + \frac{\sigma^2_{\text{weight}}}{2\sigma^2_{\text{prior}}} + \frac{\mu^2_{\text{weight}}}{2\sigma^2_{\text{prior}}} - \frac{1}{2}\,. && (4.11)
\end{aligned}$$

The first two terms are specific to the Bayesian approach, while the third term can be interpreted as L2 regularization. Identifying ω^2 with μ^2_{weight} and comparing the term to the definition of L2 regularization (see Equation 4.8) leads to:

$$\frac{\mu^2_{\text{weight}}}{2\sigma^2_{\text{prior}}} \sim \lambda\omega^2 \quad \rightarrow \quad \lambda = \frac{1}{2\sigma^2_{\text{prior}}}\,, \tag{4.12}$$

Thus, choosing a Gaussian prior is equivalent to L2 regularization. The purpose of the prior for Bayesian neural networks can, therefore, be seen as restricting the weight space and reducing the model complexity. Or turning the argument around, L2 regularization can be interpreted as assuming a priori Gaussian distributed weights.

The last term of the loss function (Equation 4.5) involves the negative log-likelihood, which is the standard loss used in non-Bayesian approaches (see Section 3.3). However, an additional integration of the from $\int d\omega\, q_\theta(\omega)$ is present. This can be considered as an average over weight-space. Solving the integral via Monte Carlo leads to:

$$\int d\omega\; q(\omega)\; \log p(D|\omega) \approx \frac{1}{N}\sum_{i=1}^{N} \log p(D|\omega_i) \quad \text{with} \quad \omega_i \in q(\omega)\,. \tag{4.13}$$

Choosing $q_\theta(\omega)$ to be a Bernoulli distribution and N to be 1, would lead to randomly setting weights to 0 for each training iteration, which is equivalent to standard dropout. Therefore, using dropout during training can be seen as having a Bayesian

neural network with Bernoulli distribuitions upon the weights. This was first shown in Ref.[29]. For the studies presented in this thesis, $q_\theta(\omega)$ is chosen to be a product of Gaussian distributions, which could be seen as a Gaussian dropout.

To conclude, regularization appears naturally in the Bayesian loss function through the prior and the integral over weight space, which helps to prevent overfitting. Likewise, classical regularization techniques can be interpreted from a Bayesian perspective. A network with standard dropout and L2 regularization is a Bayesian neural network, which shows that the Bayesian treatment is simply an extension of the current methods and not something completely orthogonal.

4.3 The Gaussian approximation

In addition to the network and its architecture, two choices have to be made for the Bayesian approach: the form of the prior $p(\omega)$ and the variational distribution $q_\theta(\omega)$. While both distributions can, in principle, be chosen independently, for the studies discussed in this thesis both were chosen to be Gaussian distributions. Gaussian variational distributions are a common choice and were, for example, considered by Ref. [30]. The reason for a Gaussian prior is mostly simplicity. Its impact on the Bayesian network can be easily understood in terms of L2 regularization (see Section 4.2) and a closed form for the corresponding term in the loss function can be given (see Equation 4.11). For a study about the influence of the prior see Section 5.6.

The choice of the variational distribution is crucial because it involves approximating the posterior. We assume the variational posterior to be of the form:

$$q_\theta(\omega) = \prod_{i=0}^{N} G(\omega_i|\mu_i, \sigma_i)\,, \tag{4.14}$$

where N is the number of weights in the network and μ_i and σ_i are the mean and width of a Gaussian distribution G. In principle, this involves two approximations: first, the variational distribution is assumed to factorize, i.e. correlation terms are omitted, and second, the factorised distributions are approximated as Gaussians. By having a Gaussian form, a width is assigned to each individual weight which can be seen as uncorrelated errorbars. Other than showing meaningful results, which are presented in Section 5, a justification for these approximations is not easy to give. The shape of the likelihood as a function of the weights ω, which is a crucial part in the definition of the true posterior, is usually highly non-trivial. This can be seen by looking at the loss landscape of an ordinary neural network. As discussed in Section 3.3, the loss function is just the negative log-likelihood. It usually involves many local minima and maxima and, therefore, a non-Gaussian shape for the likelihood is to be expected. However, the hope of the Gaussian approximation is that the parameters, σ_i, of the factorized variational distributions approximate the widths of

the true distribution and, thus, provide an uncertainty on each weight. In most of the following studies the discussions will focus on σ_{pred} (see Equation 4.2), i.e. the uncertainty of the prediction and not of individual weights. To give meaning to individual weights and their uncertainties is often not possible or very difficult. However, in the discussion of Section 5.8.1 it will be shown how individual weight uncertainties can be understood with respect to the training data.

4.4 Training of a BNN

The minimization of the Equation 4.5 involves calculating gradients with respect to the parameters μ and σ. For the regularization term a closed form can be given (see Equation 4.11), which makes the computation of the gradients straight forward. The gradients of the log-likelihood term are computed with the help of the reparameterization trick [30, 31]. The idea is to solve the integral via Monte Carlo, but to write the sampled weights as:

$$\omega = \mu + \epsilon\,\sigma, \quad \epsilon \in G(x|\mu = 0, \sigma = 1)\,, \tag{4.15}$$

which separates the randomness ϵ from the variational parameters. For instance, the gradient with respect to σ can be computed as:

$$\nabla_\sigma \int d\omega\, G(\omega|\mu,\sigma) \log p(D|\omega) \approx \sum_{i=1}^{N} \nabla_\omega \log p(D|\omega)\Big|_{\omega_i} \epsilon_i, \quad \omega_i = \mu + \epsilon_i\,\sigma\,, \tag{4.16}$$

where N is the number of Monte Carlo samples. The gradients of the form $\nabla_\omega \log p(D|\omega)$ are the same gradients needed for non-Bayesian networks and are evaluated via the backpropagation algorithm. For the optimization process, standard methods such as SGD or Adam [20] can be used (see Section 3.2 for details about SGD).

4.5 Predictive distribution

The predictive mean and predictive standard deviation (see Equation 4.2) are the first two moments of the output distribution (or predictive distribution[1]) given by a Bayesian neural network. While for the rest of this thesis the discussion will be reduced to these two quantities, this section will study the full distribution by deriving an analytic formula and giving a discussion about the shape as a function of the width.

To calculate the predictive mean and predictive standard deviation, N Monte Carlo samples are drawn from the variational distributions. Each sample of weights generates one prediction. By creating a histogram of N predictions, a distribution

[1]Note that in the literature, in contrast to our notations, $p(c|x)$ is often referred to as the predictive distribution, which is obtained by integrating over the weight distributions.

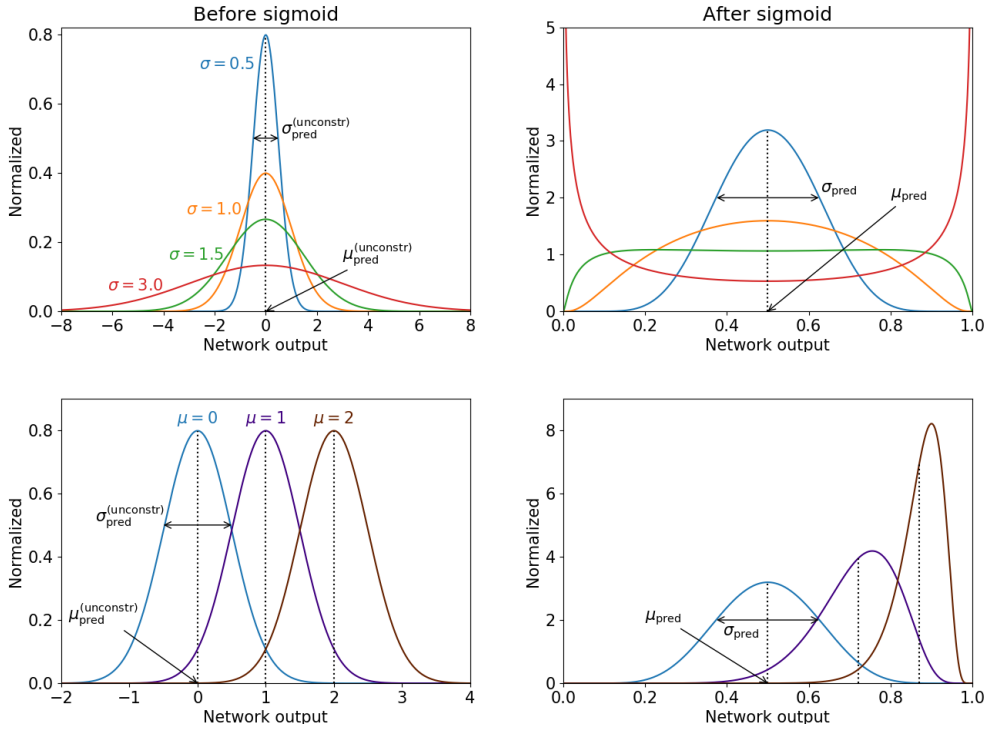

Figure 4.2: Illustration of the predictive distribution for classification. The left column shows the unconstrained distribution before applying the sigmoid activation function. The right hand side shows the transformed distributions after sigmoid. The figure is taken from Ref. [11].

over the output space of the neural network can be constructed. To get an idea about the expected shape of the predictive distribution, it is useful to look at the distribution before applying the last activation function. For binary classification, a common choice is the sigmoid function (see Equation 3.6). It maps the unconstrained output to a number between 0 and 1 or, for a Bayesian network, it maps the unconstrained output distribution to the predictive distribution. The weight distributions are chosen to be Gaussian, which led in all our experiments to approximate Gaussian shaped unconstrained output distributions[2]. Therefore, to simplify the problem the unconstrained distribution is assumed to be an exact Gaussian. This before-sigmoid distribution is then characterized by two quantities: the mean μ and the standard deviation σ. The predictive distribution is obtained by applying the sigmoid function on this Gaussian. For this simplified problem an analytical form can be derived. The mean of the after-sigmoid distribution can be written as:

$$\mu_{\text{pred}} = \frac{1}{N}\sum_{i=1}^{N} \text{sigmoid}(x_i) \quad \text{with} \quad x_i \in G(x|\mu, \sigma^2)\,. \tag{4.17}$$

[2]Even though the weight distributions are set to Gaussian distributions, the non-linearities of the neural network could in principle lead to non-Gaussian unconstrained distributions.

This can be seen as an approximation of the integral:

$$\mu_{\text{pred}} = \int_{-\infty}^{\infty} d\omega \,\text{sigmoid}(\omega)\, G(\omega|\mu,\sigma^2) \tag{4.18}$$

$$= \int_0^1 dx x\, G(\text{sigmoid}^{-1}(x)|\mu,\sigma^2)\frac{d}{dx}\,\text{sigmoid}^{-1}(x) \tag{4.19}$$

$$= \int_0^1 dx\, x \underbrace{G\left(\ln(\frac{x}{1-x})|\mu,\sigma^2\right)\frac{1}{(x+1)x}}_{=F(x)} = \int_0^1 dx\, x\, F(x)\,, \tag{4.20}$$

where a substitution of the form $x = \text{sigmoid}(\omega)$ was introduced in the second line. By using the general expression for an average, we can identify $F(x)$ as the predictive distribution. $F(x)$ is known as the logit-normal distribution and it depends on the two before-sigmoid parameters μ and σ, which we call the unconstrained mean and width. To make the following discussion easier to follow, the subscript "unconstr" is added to the parameters. There are two widths: the unconstrained width $\sigma_{\text{pred}}^{\text{unconstr}}$ and the width of the predictive distribution σ_{pred} and respectively for the mean $\mu_{\text{pred}}^{\text{unconstr}}$ and μ_{pred}. The width of the predictive distribution is defined via Equation 4.7. Using $F(x)$ the width is simply given by:

$$\sigma_{\text{pred}}^2 = \int_0^1 dx\, F(x)\, (x - \mu_{\text{pred}})^2 \tag{4.21}$$

On the right hand side of Figure 4.2, the after-sigmoid distribution $F(x)$ is illustrated for different unconstrained widths $\sigma_{\text{pred}}^{\text{unconstr}}$ and fixed mean $\mu_{\text{pred}}^{\text{unconstr}}$. Being in a regime with $\sigma_{\text{pred}}^{\text{unconstr}} \ll 1$ a Gaussian shape is to be expected. However, increasing $\sigma_{\text{pred}}^{\text{unconstr.}}$ first leads to an almost flat distribution and going even further leads to a bipolar distribution. In the limit of sending $\sigma_{\text{pred}}^{\text{unconstr}}$ to infinity, the after-sigmoid width σ_{pred} reaches 0.5. This is confirmed in Figure 4.3(a). The after-sigmoid width is shown as a function of the before-sigmoid width. The limit of 0.5 corresponds to a perfectly bipolar distribution with symmetric peaks at 0 and 1.

The lower two plots of Figure 4.2 show the dependence on the mean $\mu_{\text{pred}}^{\text{unconstr}}$ for a fixed $\sigma_{\text{pred}}^{\text{unconstr}}$. Starting with $\mu_{\text{pred}}^{\text{unconstr}} = 0$ and shifting the mean to larger values pushes the predictive distribution towards the boundary of 1. As closer the distribution gets to 1, the smaller gets the after-sigmoid width σ_{pred}. This is a result of the constrained interval. For instance, a predictive distribution with a mean of 0.9 cannot have a width larger than 0.1. This causes a correlation between σ_{pred} and μ_{pred} and will play an important role in many of the discussions of the following sections (e.g. Section 5.3 and 5.8). To get some further intuition about this correlation, we can look at the limit of small $\sigma_{\text{pred}}^{\text{uncosntr}} \ll 1$. In this limit the relation between both widths becomes:

$$\sigma_{\text{pred}} \approx \frac{d}{d\omega}\,\text{sigmoid}(\omega)|_{\omega_0}\sigma_{\text{pred}}^{\text{uncosntr}} = \mu_{\text{pred}}(1-\mu_{\text{pred}})\sigma_{\text{pred}}^{\text{unconstr}}\,, \tag{4.22}$$

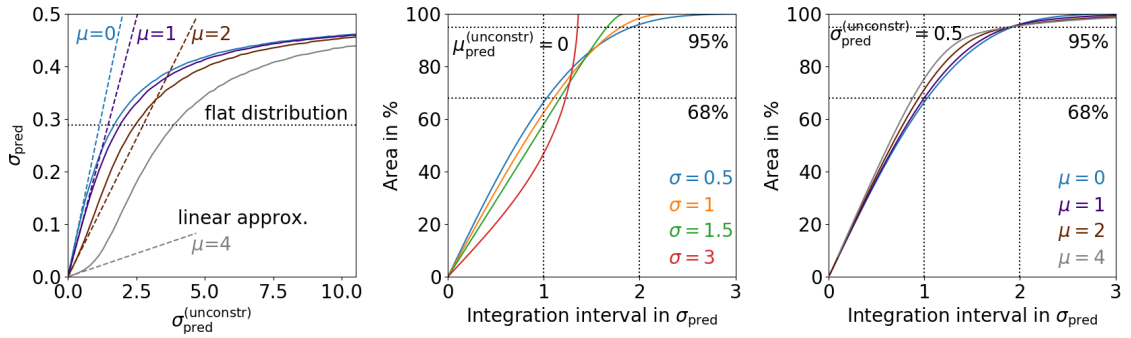

Figure 4.3: Left: predictive standard deviation as a function of the unconstrained width. The linear approximation is given by Equation 4.22. The two other plots represent the integrated area as a function of the integration interval. The values of 68% and 95% represent the Gaussian case. The figure is taken from Ref. [11].

which is, for a fixed unconstrained width, just an inverse parabola.

Figure 4.2 shows that the predictive distribution can differ significantly from a Gaussian distribution, if we go to large standard deviations. To make the difference more quantitative, the middle and right plot of Figure 4.3 illustrate the integration area as a function of the width. For a Gaussian distribution, the 1σ and 2σ-interval correspond to an area of 68% and 95%. As expected for small unconstrained widths this relation holds. The case of $\sigma_{\text{pred}}^{\text{unconstr}} = 1.5$ visualized in green in the middle plot of Figure 4.3 corresponds to the case of a flat distribution. For this case the relation between the area and the integration interval becomes almost linear. Further increasing $\sigma_{\text{pred}}^{\text{unconstr}}$ leads to a bipolar distribution with the limit of reaching 100% in a one standard deviation range.

To conclude, the predictive distribution can be seen as a logit-normal distribution, which is similar to a Gaussian for small widths. The non-Gaussian case of a flat distribution or the limit of a bipolar distribution are possible, but all the studies presented in this thesis showed that these cases do not occur in practice. Therefore, limiting the discussions to the predictive mean and predictive standard deviation should be sufficient.

4.6 BNN and regression

The last sections focused mainly on using Bayesian neural networks for classification. However, it should be mentioned that Bayesian neural networks can be used for regression as well. Depending on the problem, the likelihood has to be chosen

accordingly. A simple choice would be for example a Gaussian likelihood:

$$p(y|x,\omega) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - f^{\omega}(x))^2}{2\sigma^2}\right), \tag{4.23}$$

where $f^{\omega}(x)$ is the output of the neural network for a given data point x and σ is either a fixed width, describing the spread of the data around the "true" y-value, or itself predicted by the neural network. The negative log-likelihood in the loss function (see Equation 4.5) becomes the mean squared error. For further information, for instance, see the paper [32] by A. Kendall and Y. Gall.

5 Bayesian networks applied to particle physics

The following chapter will present experiments involving a Bayesian neural network applied to top tagging. Section 5.1 will give details about the top tagging samples and the preprocessing applied on jet images. Section 5.2 will explain the toy network used for many of the following studies. Section 5.3 will demonstrate how the predictive standard deviation scales with the overall training size. Sections 5.4, 5.5 and 5.6 will present the calibration of Bayesian taggers, a comparison to a frequentist-like method and the prior dependence, which will help justifying the approach and its approximations. Finally, Section 5.7 will demonstrate how Bayesian top taggers perform on data including pile-up or systematic uncertainties.

5.1 Top tagging and jet images

The signal samples [14] of $t\bar{t}$-events and the background samples of QCD dijets [14] were generated with PYTHIA8 [33] at a beam energy of 14 TeV. Pile-up was turned off. The detector simulations were done with DELPHES [34] and the standard ATLAS-card. The particle flow-entries were clustered with the anti-k_t algorithm (see Section 2) and FASTJET [35] into jets with a radius of $R = 0.8$. Jets with a p_T-range of $p_T \in [550, 650]$ GeV were selected. Thus, fulfilling the requirement of $\frac{p_T}{m_t} \gtrsim 1$ for boosted top jets. In addition, the top jets were truth matched to include the parton-level top and its parton-level decay products within the jet radius. For each jet the 4-momenta of the leading 200 constituents were saved and sorted by p_T. If less than 200 constituents were present, the missing entries were filled with zeros. If not mentioned otherwise, the training set consists of 600k top jets and 600k QCD jets, the validation and test set of 200k top jets and 200k QCD jets. While the validation set is used during training to monitor the performance of the classifier, the test set is used for the final evaluation of the classifier.

Two different approaches were used for the studies discussed in this thesis. The first approach involves applying the networks directly on the constituent data. In this case, the network architecture includes the leayers CoLa and LoLa (see Section 3.4). In the second case, the constituent data is first converted into jet-images before feeding it to a dense or convolutional neural network. The jet-images are constructed by performing several preprocessing steps, before discretizing the jets to images with 40×40 pixels. The preprocessing is based on the work of S. Macaluso and D. Shih [36], which in turn was based on the work of G. Kasieczka et al. [37]. The idea behind

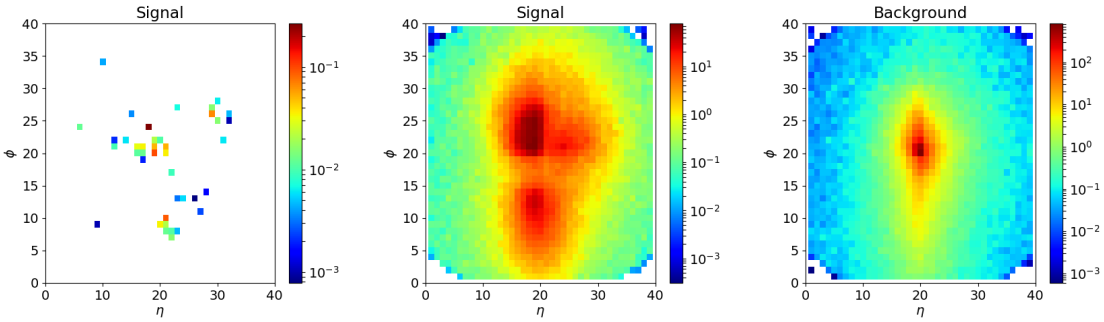

Figure 5.1: Left: single top jet. Middle and right: average top and QCD jet. The average jet images were created by piling up 100k jets. The figure is taken from Ref. [10].

it is to bring the jet images into a standard form and help the network figuring out important features. The following preprocessing steps were performed:

- calculating the azimuthal angle ϕ, the pseudo rapidity η and the transverse momentum p_T for each constituent.
- calculating the p_T-weighted centroid in the (ϕ, η)-plane and shifting the centroid to (0, 0). For the computation of the centroid and for performing the shifts, the periodicity of the angle ϕ has to be taken into account. For correctly computing the centroid, a pre-shift to the hardest constituent is performed (assuming the hottest constituent is close to the centroid) which solves the problem of the jet being to close to the boundaries $\phi \in [-\pi, \pi]$ of the parametrization.
- calculating the p_T-weighted principle axis and rotating the jet such that the principle axis is vertical.
- flipping the jet vertical or horizontal such that the maximum total p_T is in the region $\phi < 0, \eta > 0$.
- putting the jet on a grid such that each pixel entry represents the total p_T of all constituents falling into this pixel. For the top tagging samples 40×40 images with a range of $\phi \in [-0.7, 0.7]$ and $\eta \in [-0.58, 0.58]$ were created.
- normalizing the image such that the total pixel intensity (total p_T) is 1.

An example of a top jet is given in the left column of Figure 5.1. The two other plots show average jet images obtained by piling up 100k images. In these average jet images the substructure of the top jets becomes visible. As discussed in Chapter 2, the three boosted quarks produced in the hard process cause a 3-prong structure, while the light quarks or gluons simulated for the background samples lead to a 1-prong structure.

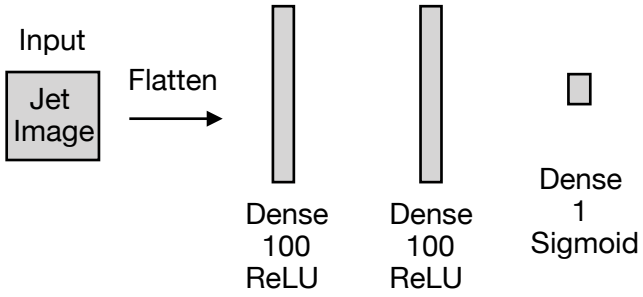

Figure 5.2: Illustration of the toy model architecture.

5.2 The toy network

The following sections will contain several experiments involving training many Bayesian and non-Bayesian networks. Training the best availble deep learning top taggers [10] is usually very time intensive. For this reason, we constructed a toy network, which consists of 2 inner fully connected layer with 100 units each. The activation function of both layers is ReLU (see Equation 3.2). The input are jet images with 40×40 pixel (see Section 5.1). The output layer consists of 1 unit with a sigmoid activation function. This leads to a network size of around 300k parameters. Thus, still having many parameters to be able to learn important features, while converging in less than an hour. Both, the variational posterior and the prior are set to be Gaussian distributions (see Section 4.3). If not specified otherwise, the mean and width of the prior are given by $\mu_{\text{prior}} = 0$ and $\sigma_{\text{prior}} = 1$. The Bayesian neural network is implemented using the Flipout Layer [38] of the tensorflow probability library [39]. The dense layers from Keras [40] were used for the implementation of the non-Bayesian network versions. The number of Monte Carlo samples for the evaluation of the predictive distributions is 100, which was found to be sufficient to get a good estimate of the predictive mean and standard deviation. Adam [20] was used as an optimizer with a learning rate of 0.001. The training was stopped, if the validation loss didn't improve for 10 epochs.

5.3 Statistics and correlation

In the machine learning literature, the predictive standard deviation is usually referred to as an epistemic uncertainty [41]. This loosely translates to an uncertainty which decreases with observing more data. In the language of physics, this would be seen as statistical uncertainties. To check if our Bayesian neural networks capture uncertainties arising from finite training data, this section will present how the

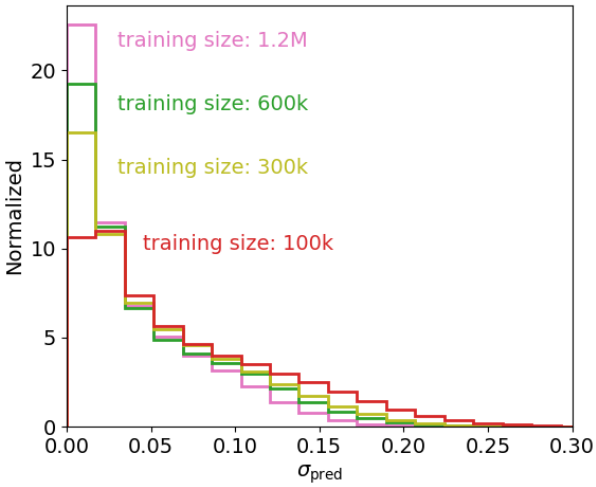

Figure 5.3: Histogram of the predictive standard deviation for 200k Top jets. The size of the training data is given in the plot. The total size is 1.2M QCD and top jets. The figure is taken from Ref. [11].

predictive standard deviation scales with the overall training size.

The toy network was trained several times on the same top tagging sample, but each time a different fraction of the total 1.2M jets was used. After the training converged, the network was evaluated on a given test sample. Figure 5.3 shows the predictive standard deviation for 200k top jets. Decreasing the training size increases the predictive standard deviation on average. This proves that the uncertainty output of the Bayesian network scales with the overall number of jets in the training data. However, as discussed in Section 4.5, the predictive mean is expected to be correlated with the predictive standard deviation. To get the full picture the prediction for each jet is, therefore, represented in the $[\mu_{\text{pred}}, \sigma_{\text{pred}}]$-plane in the lower plots of Figure 5.4. The correlation between the standard deviation and mean follows roughly an inverse parabola (see Equation 4.22). When decreasing the training size, two effects are visible. First, the predictive standard deviation increases on average, second the spread around the correlation curve increases. The correlation curve is visualized as the solid line and was evaluated as a bin-wise average of the standard deviation. To make the first effect more easily visible, the left upper plot of Figure 5.4 shows the average curves for different training sizes. To see how stable these results are with respect to different trainings and statistically independent training datasets, the toy network was trained 5 times for each curve. Each time a different set of training data was sampled from the full dataset. Because the maximum number of jets was limited to 1.2M, the network for the corresponding curve was trained 5 times on the same dataset. The errorbars represent the standard deviation of 5 curves, resulting from the 5 trained networks. The right hand side of Figure 5.4 shows the average standard deviation of all jets falling into the range

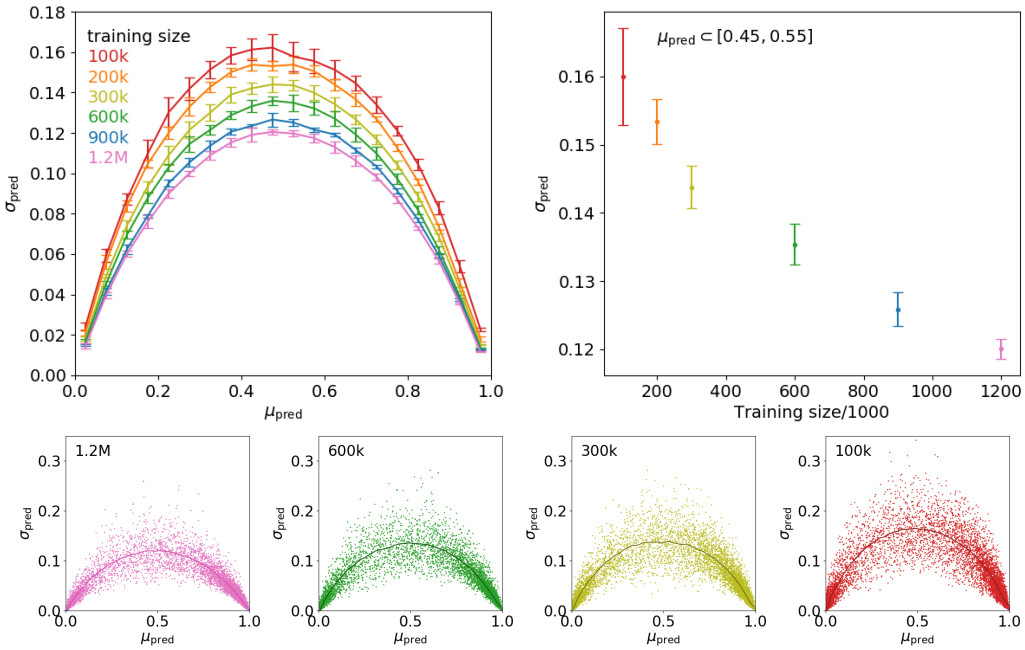

Figure 5.4: Dependence of the correlation between the two quantities μ_{pred} and σ_{pred} and the training size. Below: each point represents the prediction for one top or QCD jet. The average curve was constructed as an bin-wise average of the predictive standard deviation. The errorbars represent the standard deviation of 5 curves given by 5 different trainings. The upper right plot shows the average standard deviation for jets classified with a mean value in the range $[0.45, 0.5]$. The figure is taken from Ref. [11].

$\mu_{\text{pred}} = [0.45, 0.55]$ as a function of the training size. The errorbars are computed in the same way as for the plot on the left hand site.

From Figure 5.3, it was not clear whether the effect of increased uncertainties could be completely explained by the fact that for networks trained on less data more jets are classified with probabilities such as $\mu_{\text{pred}} = 0.4, 0.5, 0.6$. However, Figure 5.4 proves that there is a non-mean correlated effect, causing the increased uncertainties.

5.4 Calibration

If a classifier is calibrated, the output represents the probability of a data point to be a member of a specific class. For instance for jet tagging, a 0.8 output of a classifier tells us that 80% of all top jets classified with this number were correctly classified as top jets. Calibration is usually not important for constructing the ROC curve and, therefore, not important for the performance of a classifier. However, it matters if different classifiers are compared event by event to each other (see Section 5.5).

Both classifier outputs should have the same probabilistic interpretation, otherwise the comparison is meaningless.

The calibration of a classifier can be illustrated with reliability diagrams. These diagrams are constructed by discretising the probability output $p \in [0, 1]$ into several intervals I_i of equal length with $I_1 \cup I_2 \cup \ldots = [0, 1]$ and calculating the bin-wise true positive rate for each interval/bin. The bin-wise true positive rate is plotted against the average probability of all events falling into the corresponding bin. In our experiments, Bayesian neural networks turned out to be well calibrated, better than the deterministic approaches (see Figure 5.5). However, there are different post-calibration methods [42]. An example would be Platt scaling [43], which corresponds to a linear transformation applied to the before-sigmoid output:

$$z' = a\, z + b \quad p' = \text{sigmoid}(z')\,. \tag{5.1}$$

The two scalar parameters a and b are obtained by minimizing the cross entropy on the validation dataset with fixed neural network weights. The linearity ensures that the performance of the classifier is unchanged. Figure 5.5 shows the results of applying Platt scaling to the deterministic MAP approach, which will be introduced in the next section. Via the post-calibration method, the deterministic approach is as well calibrated as the Bayesian neural network.

To summarize, Bayesian networks seem to be better calibrated than deterministic networks and badly calibrated classifiers can be recalibrated by one of many different post-calibration methods.

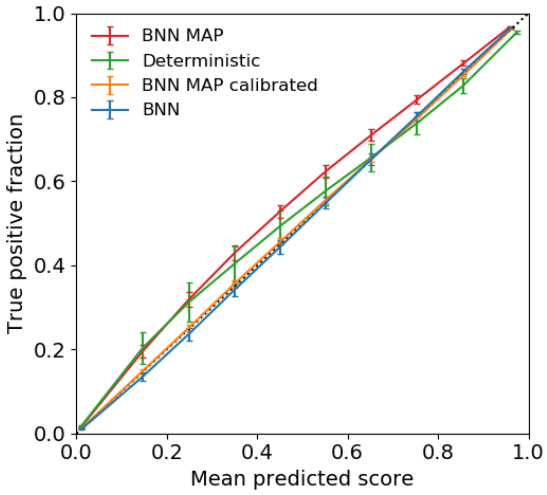

Figure 5.5: Reliability diagram of the bin-wise true positive rate over the mean prediction in one bin. The bins have equal length of 0.1. The errorbars represent the standard deviation of reliability curves for 50 trained networks. A diagonal curve indicates a well calibrated classifier. The figure is taken from Ref. [11].

5.5 Relation to deterministic models and a frequentist approach

This section will discuss an alternative frequentist-like method and compare it to the Bayesian approach. We define the frequentist method[1] as training ordinary networks N times on statistically independent training samples, each training giving us a different set of weights and, correspondingly, a different classifier. The ordinary neural network is obtained by minimizing the negative log-likelihood plus regularization terms. As it will be shown in this section, the uncertainties on the frequentist predictions depend on the regularization used for the training. Therefore, the regularization of the deterministic models has to be chosen carefully for a proper comparison. As shown in Section 4.2, a Gaussian prior can be interpreted as introducing L2 regularization upon the weights. The prior width can be used to derive the L2-parameter[2] λ as:

$$\lambda = \frac{1}{2\sigma_{\text{prior}} N_{\text{train}}} = 2.5 \cdot 10^{-6}\,, \tag{5.2}$$

where the prior width was set to 1 (see Section 5.6) and the toy networks were trained on $N_{\text{train}} = 200\text{k}$ jets. Sampling from the variational posterior can be related to dropout (see Section 4.2). Similar to L2 regularization dropout effects the

[1]The method described here is often refereed to as an ensemble method in the literature

[2]The additional training size dependence arises from the fact that we divide the loss function by the training size (for more details see appendix A)

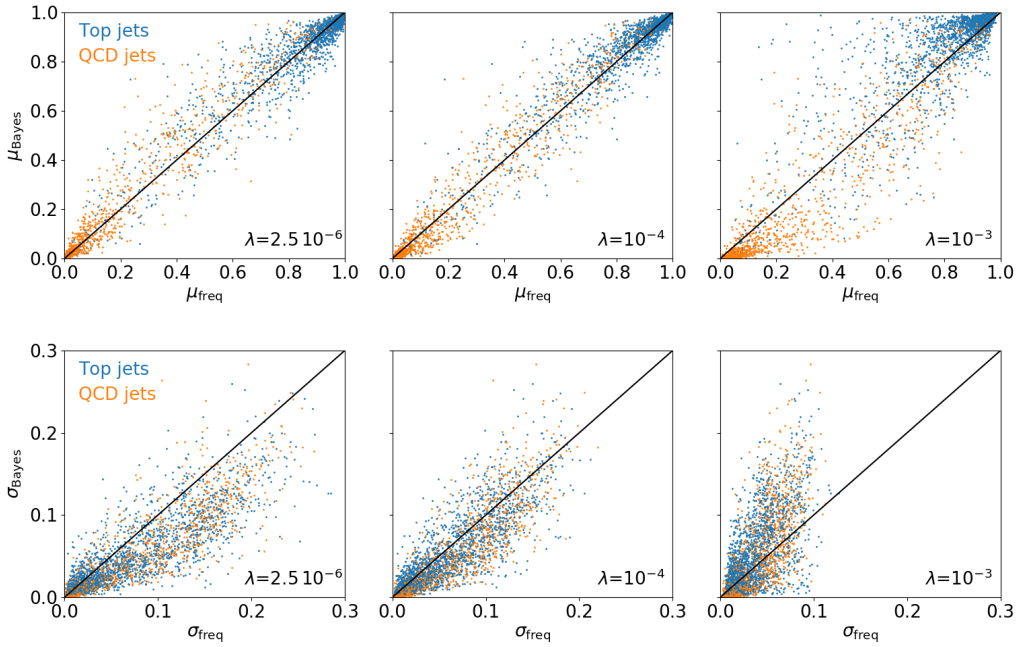

Figure 5.6: Dependence of the frequentist approach on L2-regularization. The results for the derived L2-parameter (see Equation 4.8) are shown in the left column. The upper half shows a mean comparison and the lower half a standard deviation comparison between a Bayesian network and the frequentist approach. Dropout is not used. The figure is taken from Ref. [11].

uncertainties on the frequentist predictions. However, while a value for the L2-parameter λ can be derived, there is no one to one correspondence between the Gaussian approximation, used in our set-up, and standard dropout. Therefore, the dropout rate can be chosen freely.

The training data, introduced in Section 5.1, consists of 1.2M jets, thus making it not possible to have more than 6 independent training datasets, if the training size is 200k. Therefore, about 20M new events were generated[3], using the same requirements as described in Section 5.1, resulting into 100 statistically independent training samples and 100 independently trained deterministic toy models. Figure 5.7 shows how the dropout rate effects the standard deviation and mean of the frequentist approach. The predictions of the Bayesian neural network are shown on the y-axis. While the mean-values shown in the upper part are almost unaffected by the three different rates $(0.1, 0.3, 0.5)$, the standard deviation decreases on average for larger dropout rates. The L2 parameter λ was set to the derived value given above. The best agreement between the frequentist approach and the Bayesian approach

[3]Jennifer Thompson generated the 20M new events.

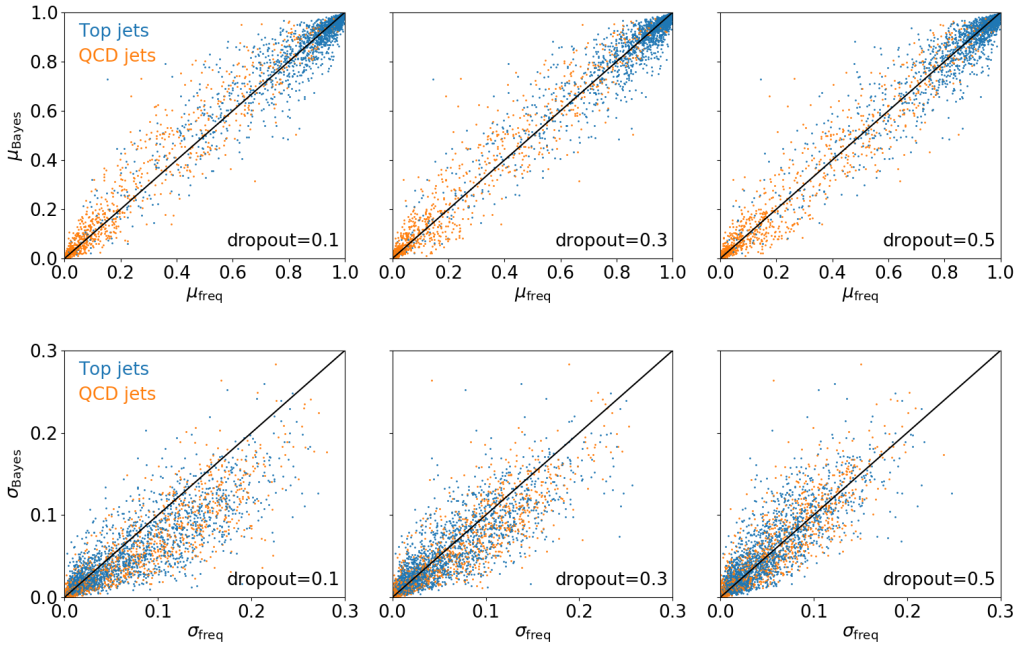

Figure 5.7: Dependence of the frequentist approach on dropout. The upper half shows a mean comparison and the lower half a standard deviation comparison between a Bayesian network and the frequentist approach. The L2-parameters is set to the derived value of $\lambda = 2.5\,10^{-6}$. The figure is taken from Ref. [11].

seems to be given by a dropout rate of 0.3. Similarly, Figure 5.6 shows the impact of the L2 parameter λ on the mean and standard deviation. For these plots the dropout rate was set 0. The derived value, which would correspond to our Bayesian set-up, is shown in the column of the left side. As both figures demonstrate, there is a dependence on the regularization of the deterministic networks.

To avoid the problem of regularization, the deterministic networks of the frequentist approach can be replaced with Bayesian neural networks. Bayesian neural networks can be easily turned into deterministic models. This gives us the benefit of having the same regularization for the "deterministic" models and the Bayesian network. Similarly to the frequentist approach discussed above, we can calculate a mean and standard deviation from 100 trained Bayesian deterministic-like models and compare it to the predictive standard deviation and predictive mean given by one Bayesian network. To turn a Bayesian network into a deterministic model, we replace the weight distributions by their maximum values. Because we use Gaussian distributions as the variational posterior $q_\theta(\omega)$, the maximum value coincides with the mean value. It is called the Maximum a Posterior (MAP) approach and can be

summarized as:

$$\omega_{\text{MAP}} = \underset{\omega}{\text{argmax}}\ q_\theta(\omega) \tag{5.3}$$

$$= \underset{\omega}{\text{argmax}}\ G(\omega_1|\mu_1, \sigma_1^2)\, G(\omega_2|\mu_2, \sigma_2^2) \ldots \tag{5.4}$$

$$= (\mu_1, \mu_2, \ldots) \tag{5.5}$$

Under the assumption that the variational distribution $q_\theta(\omega)$ is a good approximation of the true posterior $p(\omega|D)$, the MAP approach should lead to similar results as the fixed network weights obtained by training an ordinary network. Therefore, the MAP approach can be used to get a set of deterministic models without having to worry about the regularization of the deterministic networks. Figure 5.8 shows a comparison of the MAP approach and a Bayesian neural network. As discussed in Section 5.4, two networks should be both well calibrated if one is interested in an event by event comparison. If this is not the case, the predictions do not have the same probabilistic interpretation and making a comparison meaningless. The MAP approach turned out to be badly calibrated (compare Figure 5.5). Therefore, to restore the calibration of the MAP approach, Platt scaling was applied to each individual network. Figure 5.8 shows that the uncertainties of the MAP approach and the Bayesian network are compatible and in a similar range. This helps justifying the Bayesian output uncertainties. While for Bayesian neural networks the distribution over weights have to be approximated (see Section 4.3), the frequentist approach does not rely on any approximations. However, for studies involving more complex models with more parameters, the frequentist approach is very computationally expensive because it involves training many neural networks. This could be compensated by reducing the number of trained networks significantly, but for the cost of noisy uncertainty estimates.

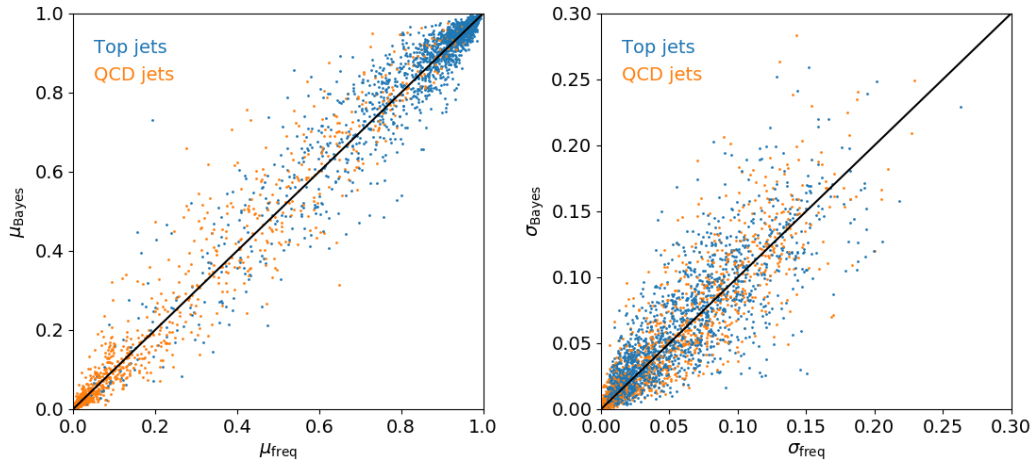

Figure 5.8: Comparing the MAP frequentist-like approach and the predictions of a Bayesian neural network. The left plot shows a jet-level mean comparison and the right plot a jet-level standard deviation comparison. The MAP approach was recalibrated to restore the probabilistic interpretation. The figure is taken from Ref. [11].

5.6 Prior dependence

The prior $p(\omega)$ is meant to encode prior knowledge about the parameters ω. However, there is no prior knowledge accessible for the weights of our Bayesian neural networks, which makes the choice for the prior $p(\omega)$ somewhat arbitrary. Therefore, it is important to check that the decisions of our Bayesian taggers are not dominated by this choice. For the studies discussed in this thesis, a factorized Gaussian prior was used, which can be written as:

$$p(\omega) = G(\omega_1|\mu_p, \sigma_p)\, G(\omega_2|\mu_p, \sigma_p) \ldots \tag{5.6}$$

where all network weights ω_i share the same mean μ_p and width σ_p. The mean μ_p is fixed to 0, because there is no reason why positive values should be preferred to negative values or vice versa. The width σ_p however is a free parameter. It was varied in a range of 6 orders of magnitude (see Table 5.1). The performance is given as area under curve (AUC), which refers to the integrated area under the ROC curve. Above a value of 1 the performance reaches a plateau value. As Section 4.2 demonstrated, the prior appears as a regularization term in the loss function. A small prior width restricts the range of possible weights strongly and does not give the network enough freedom to learn the necessary features required for top tagging. This is equivalent to the case of using L2 regularization with a very large L2 parameter λ (see Section 4.2). The given AUC values are the average of 5 AUC-values obtained by training the same Bayesian network with the same hyperparameters 5 times on different samples of 200k jets. Each sample was drawn randomly from the full training dataset, which consists of 1.2M jets. The errors represent the standard

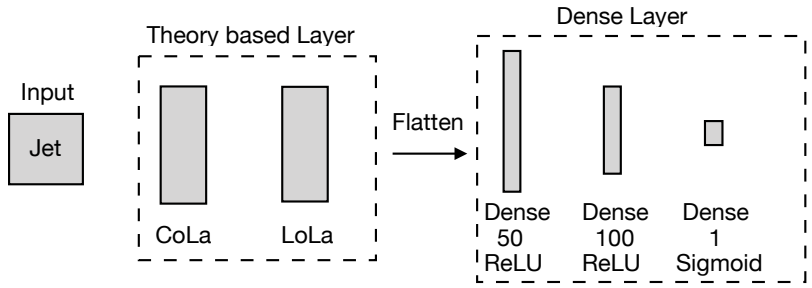

Figure 5.9: LoLa architecture. For more information about the first two layers see Section 3.4.

deviation and demonstrate that training and performance are stable and that there is no significant improvement above a prior width of $\sigma_p = 1$. The networks trained with large prior widths converge significantly slower leading to much longer training times, when using the same set of hyperparameters such as learning rate and other optimizer specific parameters. Therefore, a prior width of 1 seems to be a good choice for the toy architecture applied on top tagging.

For classification tasks, it can be argued that the network output, before applying the sigmoid activation function (see Equation 3.6 and toy network architecture in Figure 5.2), should be of the order of 1. An input value of 4 already leads to a sigmoid output of around 0.98, which shows that values of the order of 1 are sufficient to reach the full sigmoid output range of [0, 1]. In addition, the jet images are normalized to a total pixel intensity of 1 (see Section 5.1). Therefore, having an input of the order of one and an output of the order of one, makes the assumption of a prior width of 1 reasonable, because it limits the range of possible weight values to this range. Because of this argument and the results shown in Table 5.1, the prior width was set to 1 for the other architectures as well (see Section 5.7).

σ_{prior}	10^{-2}	10^{-1}	1	10	100	1000
AUC	0.5	0.9561	0.9658	0.9668	0.9669	0.9670
error	—	±0.0002	±0.0002	±0.0002	±0.0002	±0.0002

Table 5.1: Dependence of the prior width on the performance. For each column, the error represent the standard deviation of 5 different trainings. The table is taken from Ref. [11].

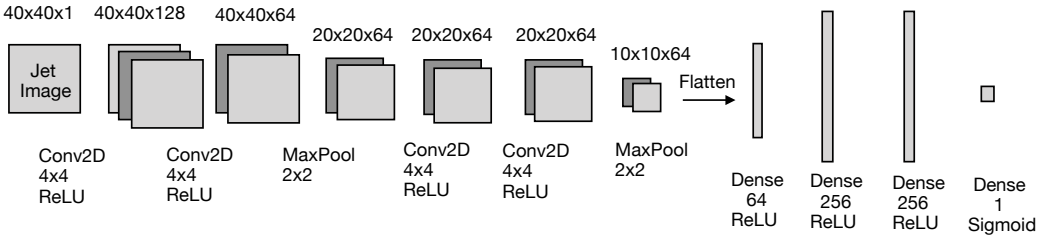

Figure 5.10: CNN architecture, based on Ref. [36].

A Gaussian prior is a common and practical choice for Bayesian networks. It is simple, because it just requires to tune one parameter, a closed form for the prior dependent part of the loss function can be given (see Equation 4.11), and it can be easily understood in terms of L2 regularization. However, in the literature other prior distributions were also studied. A Gaussian mixture for example adds additional parameters, which have to be tuned accordingly, but it may give the network more freedom and, therefore, lead to a better performance. For instance, Blundell et al. [30] introduced a Gaussian mixture with 2 components. The mean values were set to 0 with one small variance $\sigma_1 \ll 1$, causing many weights to be concentrated around 0 and the other variance large in order to allow a few large weights as well. However, the studies of this thesis showed that a Gaussian prior was sufficient to reach the same performance as the deterministic versions (see Section 5.7). Therefore, a simple Gaussian prior was chosen for all studies discussed in this thesis.

5.7 Deep learning top taggers

For the studies discussed in the previous sections a simple fully connected dense network was used because the studies involved training many neural networks, which is computationally very expensive for complex architectures. The following sections will show studies involving Bayesian versions of more complex architectures. It will be shown how they relate to their deterministic counterparts, whether they can reach the same performance, and what happens if systematic uncertainties or pile-up are included.

Two architectures were picked for these studies: an architecture based on convolutional neural networks [36] using jet images as an input, and a constituent based approach using LoLa (see Section 3.4). At the start of the studies presented in this thesis, these architecture showed state-of-the-art performance for top tagging. How-

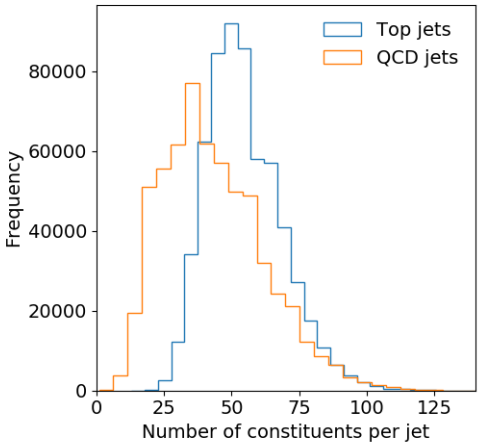

Figure 5.11: Histogram of number of constituents per jet for QCD and top jets. The total number of jets is 1.2M.

ever, because of the fast developing field of deep learning there are already networks showing improved performance with respect to the architectures discussed in this thesis. For instance, recent promising results were found by H. Qu and L. Gouskos for their ParticleNet architecture [9]. An overview of many different deep learning top taggers can be found in Ref. [10].

The last layers of the LoLa network and the CNN are fully connected dense layers leading to a single output-neuron with a sigmoid function. For the convolutional architecture, the other activation functions are ReLU. The theory based layer CoLa and LoLa do not contain activation functions. The settings for CoLa and LoLa are described in Section 3.4. A sketch of both network architectures is shown in Figures 5.9 and 5.10. For the LoLa architecture, the number of constituents per jet n_{constit} has to be set by hand (see Section 3.4). For our top tagging samples the average number of constituents per top jet is around 55 (see Figure 5.11). For QCD jets it is around 40. The maximum number of constituents per jet is around 120. Three different values for n_{constit} were tested with the largest of 200, which ensures that every constituent is fed to the network without throwing away soft constituents. As for the toy network, the convolutional and dense layers are implemented using the Flipout Layer [38] of the tensorflow probability library [39]. The implementation of the Bayesian versions of LoLa and CoLa are similar to the above mentioned dense and convolutional layers, using the tensorflow probability library as well. The dense and convolutional layers of Keras [40] were used for the non-Bayesian network versions.

Table 5.2 gives an overview of the performance of the different architectures. The entire ROC curve is of importance for the given AUC values. However, that means that the performance at very low or very large signal efficiencies could influence

this performance measure. These regions of signal efficiency wouldn't be used in practice. For that reason, a reference point at $\epsilon_t = 0.3$ (signal efficiency) was chosen to add another performance measure. The signal efficiency at $\epsilon_t = 0.3$ would be a typical working point for top tagging. Both performance measures indicate that the Bayesian approaches seems to perform similar to the deterministic versions. The AUC values of the Bayesian and deterministic taggers coincide in almost all cases within the given accuracy. Several trainings with the same hyperparameters showed that the AUC values are stable up to the forth decimal place. The small fluctuations arise, for instance, from different weight initializations. The given $1/\epsilon_{\text{QCD}}$ values are slightley more unstable. The reason for that is that the AUC values are obtained by integrating over the entire ROC curve, while for $1/\epsilon_{\text{QCD}}$ just a single reference point is chosen.

		AUC	$1/\epsilon_{\text{QCD}}$ for $\epsilon_t = 30\%$
CNN		0.982	820
B-CNN		0.982	900
LoLa	$N_{\text{const}} = 40$	0.979	630
B-Lola		0.979	600
LoLa	$N_{\text{const}} = 100$	0.981	740
B-Lola		0.980	710
LoLa	$N_{\text{const}} = 200$	0.980	600
B-Lola		0.980	710

Table 5.2: Performance of different Top taggers and their Bayesian versions. The table is taken from Ref. [11].

5.8 Systematic uncertainties and stability

If we would apply our trained top taggers to actual data, the taggers would encounter many additional effects not present in the Monte Carlo data. Pile-up or systematic uncertainties are two examples. The next sections will show how the top taggers, and in particular the Bayesian taggers, react to pile-up and some approximation of a jet energy scale (JES) uncertainty. In both studies, only the test sample will be modified. The last section will focus on the possibility of including systematic uncertainties in the training sample.

5.8.1 Pile-up

Pile-up refers to multiple proton-proton interactions per bunch crossing. The hadronic activity arising from pile-up is usually much softer than the hard process, which passed the trigger system of the detector. It is not a systematic uncertainty. However, from a machine learning perspective, it is an uncertainty in the sense of an effect which is not present in our training data. There are many existing ways to reduce the impact of pile-up such as SOFTKILLER [44] and PUPPI [45] or recent deep learining based approaches such as PUMML [46]. However, they are not considered here. The purpose of this study is not to accurately simulate experimental conditions and propose a new method to handle pile-up more efficiently, but to show how an analysis can benefit from Bayesian networks, if some inconsistencies between training and test sample are present.

In total 1M pile-up events[4] were generated with PYTHIA with the min-bias settings. DELPHES was used with the standard ATLAS-card for the detector simulation (see Section 5.1). For each pile-up event the 400 hardest constituents were saved. A variable number, n_{PU}, of these pile-up events were overlaid with the jet samples described in Section 5.1. In order to fulfil the original requirements, the jets are reclustered with the same jet radius of $R = 0.8$. If several jets were found by the algorithm, just the hardest was saved.

Figure 5.12 shows the performance of the trained top taggers as a function of the number of pile-up events. On the left hand side the results for the convolutional architecture are presented. The performance of the BNN is not any more stable than the deterministic version. On the right hand side two versions of the LoLa network and their Bayesian versions are compared to each other. While for $n_{\text{constit}} = 40$ the Bayesian version performs as well as the deterministic one, for $n_{\text{constit}} = 200$ the Bayesian network performs significantly better for large number of included pile-up events. For the $n_{\text{constit}} = 40$ network just the 40 hardest constituents are taken into account. As Table 5.2 shows, this is in principle no problem for the network performance because the discrimination power seems to come mostly

[4]The events were generated by Jennifer Thompson and included to the samples by Sven Bollweg

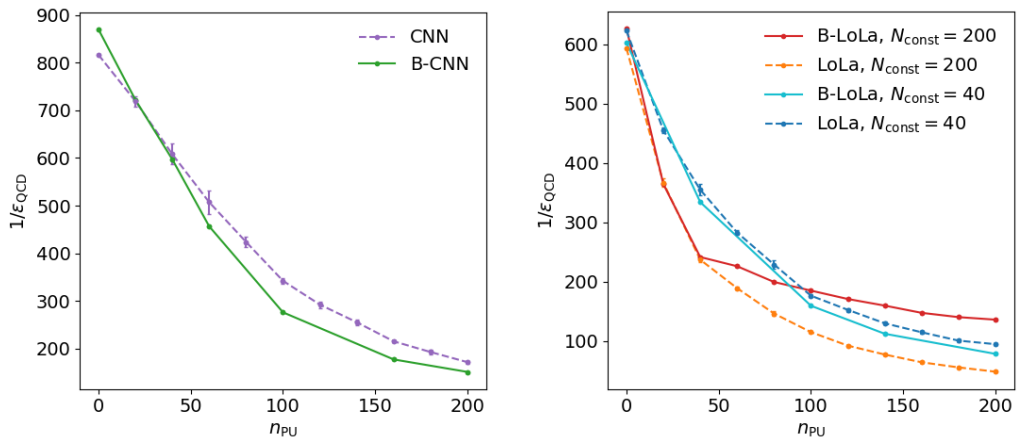

Figure 5.12: Impact of pile-up on performance. $1/\epsilon_{\text{QCD}}$ for a given signal efficiency of $\epsilon_t = 0.3$. Left: Bayesian and non-Bayesian CNN-architecture. Right: different Bayesian LoLa versions and their deterministic counterparts. The figure is taken from Ref. [11].

from the hardest constituents. Still, some information is thrown away compared to the $n_{\text{constit}} = 200$ case, causing the lowest AUC value. However, including pile-up gives the $n_{\text{constit}} = 40$ version the benefit of a soft constituent cut, which makes the network less sensitive to pile-up because pile-up adds mostly soft activity to the jets. This can explain why LoLa performs better for $n_{\text{constit}} = 40$ in the lower n_{PU} regime. However, the surprising feature of the B-LoLa architecture with $n_{\text{constit}} = 200$ is that it outperforms the other architectures in the large n_{PU} regime. To understand this further, in the following discussion will present the impact on the output uncertainties and individual weight uncertainties.

As discussed in Section 4.1, the mean-output of a Bayesian network can be seen as a average over network predictions. In that sense, one could expect the Bayesian networks to be more stable with respect to noise in the data. However, just the $n_{\text{constit}} = 200$ version shows improved performance. The Bayesian $n_{\text{constit}} = 40$ version and the Bayesian CNN do not perform better than their deterministic counterparts. To understand this behavior further, it is useful to look at the individual predictions in the $[\mu_{\text{pred}}, \sigma_{\text{pred}}]$-plane. As can be seen in Figure 5.13 for $n_{\text{constit}} = 200$, more pile-up events lead to increased output uncertainties. While the network performs better on a large amount of included pile-up, at the same moment the network also reacts with increased uncertainties on the mean values. For the CNN, we do not observe any increased uncertainties in the correlation plane (see Figure 5.14). However, for the mean-correlation curve a kink is appearing in the central region around $\mu = 0.5$. The mean-correlation curve is constructed in the same way as described in Section 5.3. The correlation is an intrinsic property of the constrained output interval for a binary classification task (see Section 4.5). For B-CNN, the

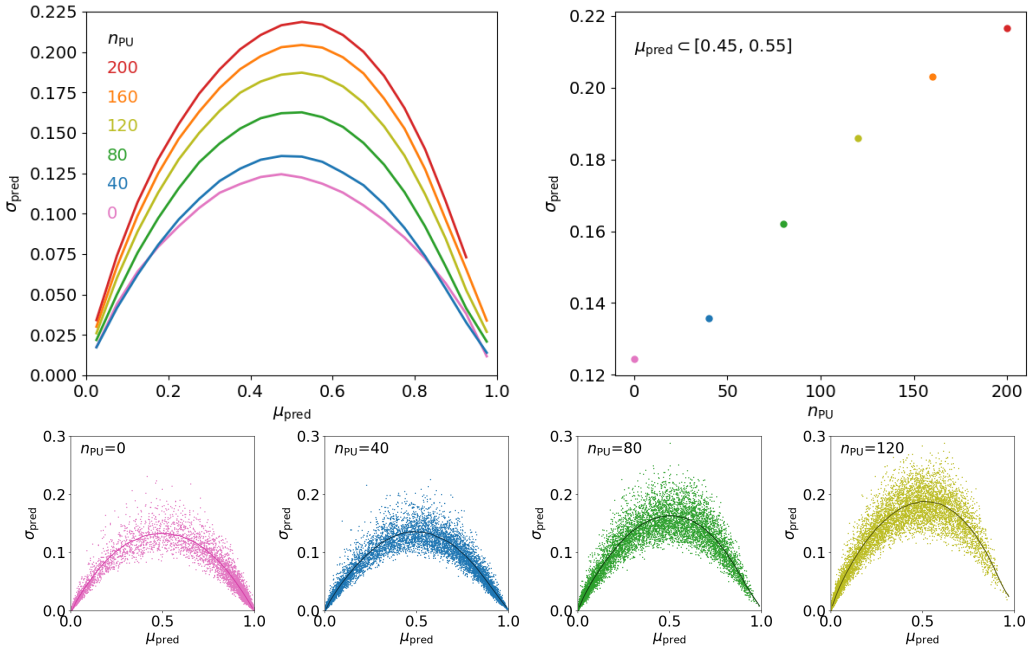

Figure 5.13: Impact of pile-up on the predictive standard deviation in the $[\mu_{\text{pred}}, \sigma_{\text{pred}}]$-plane for B-LoLa with $n_{\text{constit}} = 200$. For details about the construction of the correlation-curves, see Figure 5.4 and Section 5.3. The figure is taken from Ref. [11].

simple correlation curve does not seem to be present any longer. While the Bayesian version is not more stable with respect to pile-up, at least the analysis in the correlation plane seems to give us a hint about the fact that the network performance is somehow unstable.

While analysing the $[\mu_{\text{pred}}, \sigma_{\text{pred}}]$-plane provides a nice handle for understanding neural network based taggers in greater detail, we can even go a step further and try to understand the individual learnt weight distributions. Each weight is described by two parameters, a mean and and a width. The first layer, CoLa, of B-LoLa builds linear combinations of the form (see Section 3.4):

$$\omega_1 \, p_1^\mu + \omega_2 \, p_2^\mu + \omega_3 \, p_3^\mu + \ldots \tag{5.7}$$

Because of the p_T-ordering, weight number 1 will always be associated with the hardest constituent, while weight number 2 will be associated to the second hardest and so on. This gives us the benefit of being able to understand the individual weights of B-LoLa. In Figure 5.15, the mean and width for each weight is illustrated. The widths of the weights increase monotonically towards a threshold of around 120, which represents the largest number of constituents per jet present in the training sample. In red the number of jets which include at least a certain number of constituents is visualized. It shows that the widths of the weights loosely correspond

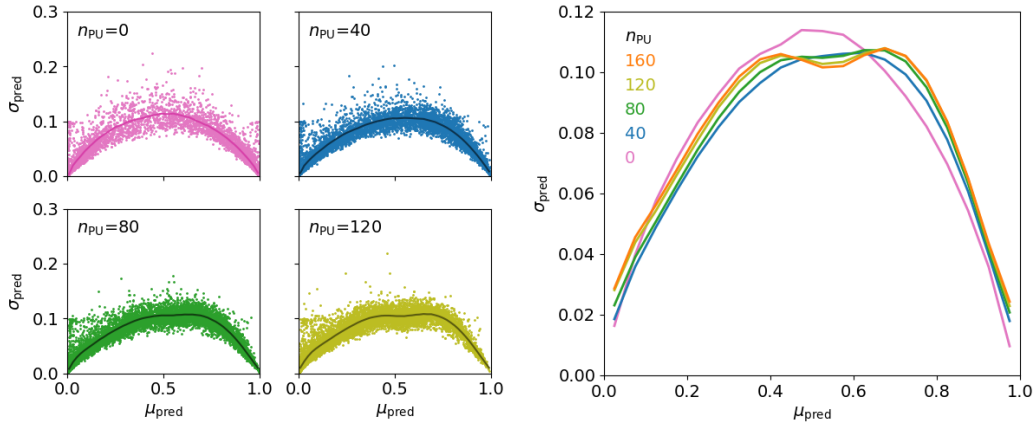

Figure 5.14: Impact of pile-up on the predictive standard deviation in the $[\mu_{\text{pred}}, \sigma_{\text{pred}}]$-plane for B-CNN. For details about the construction of the correlation-curves, see Figure 5.4 and Section 5.3. The figure is taken from Ref. [11].

to the statistics of the training data. With fewer examples present in the training data which would help fixing the value of a certain weight, the weight's uncertainty increases. This is a nice representation of the learnt weight uncertainties, but it also helps to understand the behavior of B-LoLa, when introducing pile-up. Pile-up mostly introduces soft constituents to the test samples. The corresponding weights of CoLa (and LoLa) are given large uncertainties, leading to a more careful treatment of these constituents and, therefore, a better tagging performance for large amount of pile-up. This more careful treatment for large uncertainties comes from the fact that the prediction of a Bayesian network, which is approximated via the predictive mean, is an integral over all weight distributions. The contribution from these weights eventually vanish, if integrated over. Because the included soft constituents will "activate" many of the weights with large uncertainties, the predictive standard deviation will increase, which explains the behavior visible in Figure 5.13.

Another effect illustrated in the plot on the right hand site of Figure 5.15 is that the mean-values of the weights above 120 are all trained to zero. This is the regime of zero training statistics. It acts like a cut off for the softest pile-up constituents. The network decisions benefit from this behavior in the same way as LoLa $n_{\text{constit}} = 40$ benefits for a small number of included pile-up events. Sending the weight values to 0 for unused/irrelevant weights could be also accomplished by introducing L2 regularization for deterministic networks. However, the deterministic case does not have an equivalent to the widths of each weight, which tell the network to treat soft constituents more carefully.

The discussed benefits of B-LoLa for $n_{\text{constit}} = 200$ are not present for the B-CNN nor the B-LoLa network with $n_{\text{constit}} = 40$. The operations of the layers CoLa and

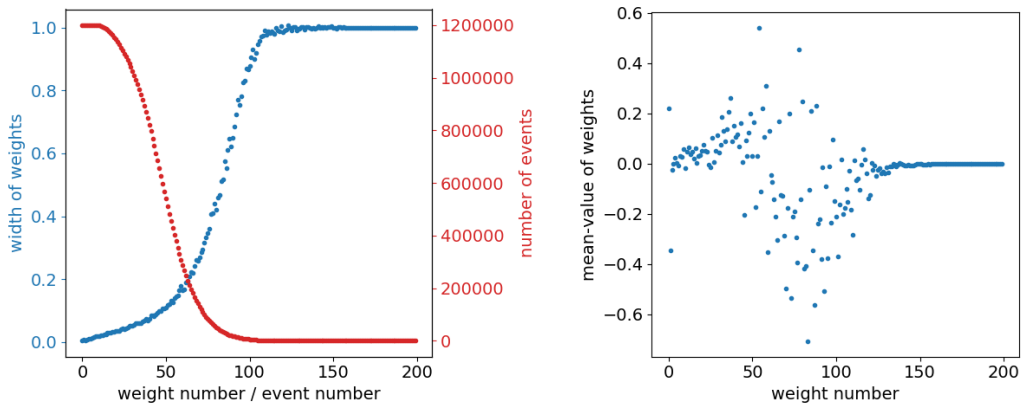

Figure 5.15: lIllustration of the learnt parameters for each weight of the first layer (CoLa). The red curve in the plot on the left side gives the number of jets with at least this amount of constituents. The value of 1 for σ_{weight} and 0 for μ_{weight} are the values of the prior.

LoLa are applied to individual constituents, making it very easy for the network to encode the training statistics related uncertainties into the uncertainties of the weights. Conversely, the B-CNN is based on convolutional operations. The kernels of the convolutions are moved over the entire image, making it difficult for the network to learn the above discussed uncertainties. This is a possible explanation for B-CNN not being any more stable than the deterministic version. For the LoLa architecture with $n_{\text{constit}} = 40$ the softest constituents are thrown away before even applying the first layer, giving the network no chance to learn these uncertainties.

To conclude, the large uncertainties given to weights which are responsible for soft constituents lead to a more careful treatment of jets with a large amount of pile-up and, therefore, a better tagging performance. In the same moment, the increased output uncertainties are a clear sign of having inconsistencies between training and test data, leading to the natural next step of including pile-up in the training sample (see Section 5.8.3 for the case of the JES uncertainty). In addition, this section demonstrates how the additional weight uncertainties can be used to better understand the behavior of a neural network based tagger, which are often seen as black box algorithm.

5.8.2 JES uncertainty

To study how our Bayesian top taggers react to systematic uncertainties, we approximated a jet energy scale (JES) uncertainty. The various different JES uncertainties arise in the calibration steps, which are required for matching jets reconstructed from the energy depositions in the detector to the truth particle-level jets. The ATLAS colaboration lists, in total, 80 different systematic uncertainties [47]. These un-

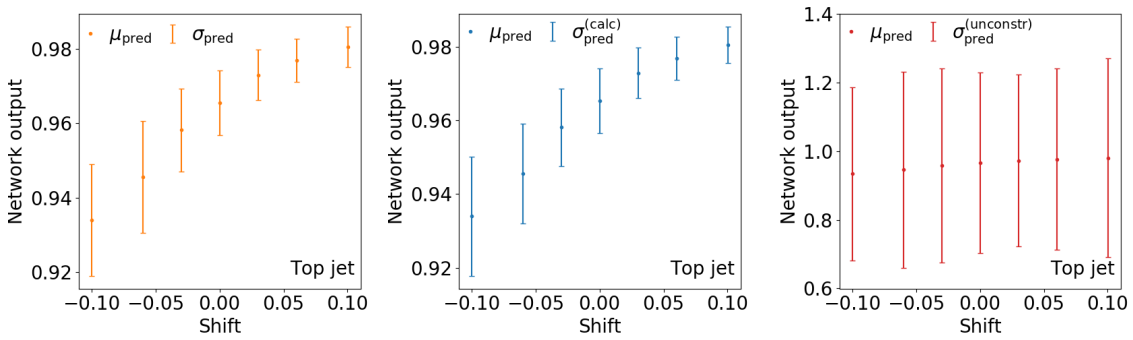

Figure 5.16: Impact of jet energy smearing. The shift factor describes the energy shift of the hardest subjet. The error bar in the middle plot was calculated via Equation 4.22. The error bar in the plot on the right hand side visualize the unconstrained predictive standard deviation, which was introduced in Section 4.5. The figure is taken from Ref. [11].

certainties arise from assumptions about event topologies, propagated uncertainties from photon, muon and electron energy scales, imperfect Monte Carlo simulations, pile-up and other sources. The total uncertainty is estimates as being in the range of $1\% \dots 4\%$, depending on the total jet p_T [47, 48]. In our studies, to approximate the total JES uncertainty the following steps[5] were performed on each fat jet:

- clustering subjets with the anti-k_t algorithm and a jet radius of $R = 0.4$.
- sampling a different random number for each subjet from a Gaussian distribution $\epsilon \in G(\mu = 0, \sigma_{\text{JES}})$ and multiplying the 4-momenta of each subjet with $1 + \epsilon$.

The JES shifts turned out to impact the performance of our top taggers only marginally. Therefore, the Gaussian width σ_{JES} was raised up to unphysically large values. Similar to the pile-up study, this study is meant to give some intuition about how Bayesian neural networks react to systematic uncertainties and is not meant to be experimentally accurate. For that purpose, this section will focus on understanding the impact on the Bayesian output uncertainty of B-LoLa. A study about the performance will be given in the next section.

In addition to the random Gaussian shifts described above, we also performed shifts on the 4-momenta of only the hardest subjet. While the procedure described above is experimentally a little bit more accurate, the random energy shifts make it difficult to understand the impact on the predictions for individual jets. The plot on the left hand side of Figure 5.16 shows the effect of smearing the 4-momenta of only the hardest subjet. The error bar represents the predictive standard deviation of a single top jet, while the mean value is represented as the central point.

[5]The smeared samples were provided by Sven Bollweg.

Shifting the 4-momenta of the hardest subjet in the negative direction, e.g. lowering p_T and energy, decreases the predictive mean. That means, by lowering the energy of the hardest subjet, the jet appears more QCD-like to the tagger. On the other hand, increasing the energy of the hardest jet, makes the jet look more top-like. Two possible explanations can be given. Lowering the energy of the hardest subjet washes out some of the substructure, because the difference between the subjets is less significant, leading to more QCD-like jets. Another explanation could be that the total jet mass is lowered by decreasing the 4-momenta of the hardest subjet, leading to the same conclusion. Further investigations are beyond the scope of this thesis. However, the important observation is that by smearing energies the mean prediction is raised or lowered which in turn impacts the output uncertainty. For top jets, decreasing the energy and, therefore, the mean value increases the uncertainty, while increasing the energy decreases the uncertainty. This behavior is caused by the correlation between the mean and the standard deviation (see Section 4.5 for details about the correlation). Going from these fixed energy shifts to the random Gaussian energy shifts introduced above will lead to both, jets with increased and decreased uncertainties. The correlation completely dominates the impact on the output uncertainties.

To further illustrate the impact of the correlation, the plot on the right hand side of Figure 5.16 shows the unconstrained predictive standard deviation. It is the standard deviation of the unconstrained neural network output before the sigmoid function is applied. By removing the step of applying the sigmoid function, most of the correlation between the mean and the standard deviation can be removed. The width seems to be almost constant as a function of the shift. For the plot in the center σ_{pred} was replaced with $\sigma_{\text{pred}}^{\text{calc}}$, which was computed from $\sigma_{\text{pred}}^{\text{unconstr}}$ for zero shift with the help of Equation 4.22. Both plots illustrate that the impact on the uncertainty is mostly caused by the correlation and that the correlation can be reduced by going into the before-sigmoid space.

As for the pile-up study, a desired effect would be to see larger output uncertainties at least on average. However, these effects seem to be much smaller for the JES uncertainty than for the pile-up study. Any possible effects are completely dominated by the correlation. To see if there are at least small effects present, we tried to decorrelate the uncertainty output from the mean value. Figure 5.17 shows a histogram of a jet by jet comparison of $\sigma_{\text{pred}}^{\text{unconstr}}$. As argued above, the width in the before-sigmoid space is mostly decorrelated from the mean value. In addition, the jets were normalized to remove the dependence on the original jet p_T. The left hand side shows the effects on QCD jets, while the right hand side shows the same for top jets. The normalized difference, which is represented on the x-axis, is defined as:

$$\Delta\sigma = \frac{\sigma_{\text{smear}}^{\text{unconstr}} - \sigma^{\text{unconstr}}}{\sigma_{\text{smear}}^{\text{unconstr}} + \sigma^{\text{unconstr}}} \,. \tag{5.8}$$

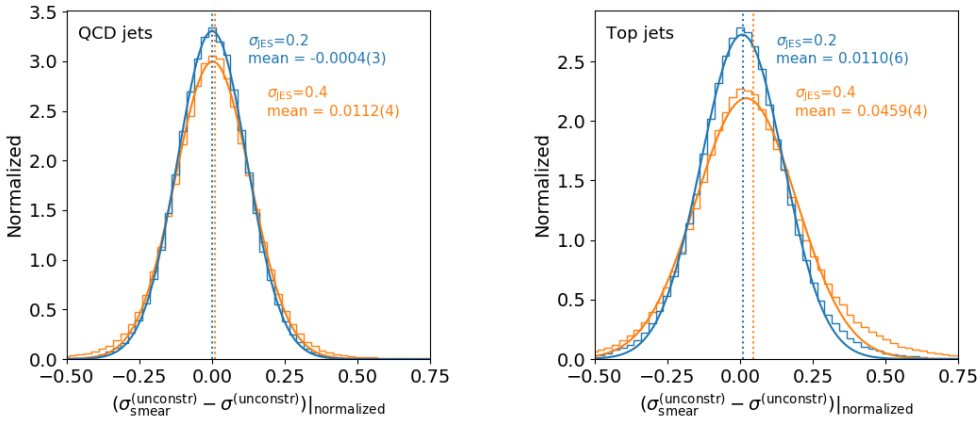

Figure 5.17: Histogram of a jet by jet comparison of the unconstrained width for smeared and nominal data. The Bayesian network was trained on nominal data. The normalized difference is defined in Equation 5.8. The figure is taken from Ref. [11].

For QCD jets the average difference is consistent with zero. However, for top jets, the tagger assigns on average a larger unconstrained uncertainty to the smeared jets. The solid line indicates a Gaussian fit. It makes a tail with increased uncertainties in the upper part visible. The given mean values and the tail in the upper part both demonstrate that there are more jets present with increased uncertainties than with decreased uncertainties.

To conclude, the correlation between the mean and the standard deviation strongly dictates how the Bayesian top tagger reacts to the JES uncertainty. However, as this section demonstrates, this correlation can be easily understood and attempts can be made to decorrelate both quantities. Decorrelating the uncertainty output shows that there are more jets present with increased output uncertainties for samples with the JES uncertainty included compared to samples without the JES uncertainty. Therefore, this section shows that the output uncertainty of Bayesian networks can, in principle, capture systematic uncertainties. The requirement is that the systematic uncertainty is not included in the training sample. However, how these effects can be made more easily visible such that the decorrelation step can be removed, requires additional investigations.

5.8.3 Data augmentation

While the last two sections discussed the behavior of Bayesian neural networks tested on samples including systematic uncertainties or pile-up, this section will demonstrate the impact on Bayesian and non-Bayesian taggers, when the JES uncertainty is included in the training data as well. In the machine learning community this

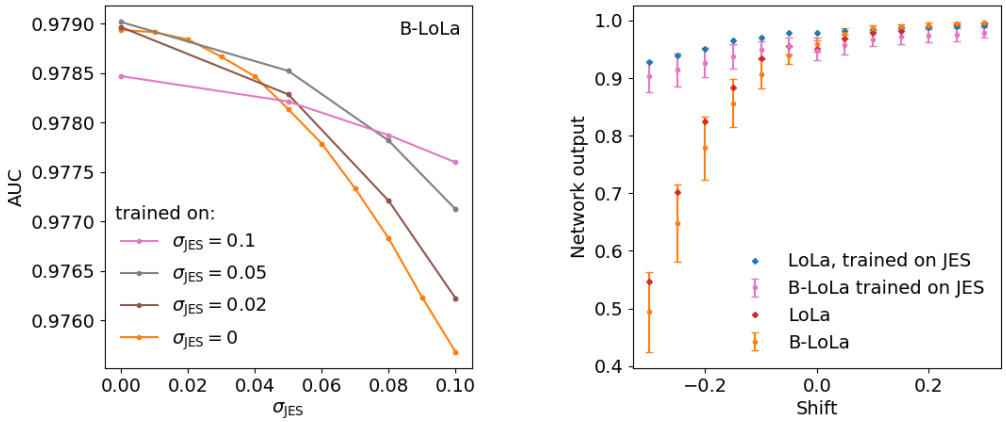

Figure 5.18: Impact of including the JES uncertainty in the training sample. The left hand side shows the impact on the performance, while the right hand side illustrates the prediction for a single top jet. In the right hand side, the network was trained on $\sigma_{\text{JES}} = 0.1$. The predictions without error bars are predictions arising from deterministic networks. The figure is taken from Ref. [11].

would fall under the category of data augmentation [49].

Data augmentation is often used for artificially increasing training statistics by augmenting the dataset with slightly modified copies of existing data. E.g. for the often used MNIST [50] dataset, which contains images of hand drawn digits, copies of images are taken and slightly rotated to generate "new" examples. However, limited training statistics is in principle no problem for our Monte Carlo based approach, because we can easily generate more training data. Instead, we will focus on the effect of including systematic uncertainties into the training data to increase the stability of the taggers. This data augmentation could be applied to any kind of systematic uncertainties, pile-up or theoretical uncertainties. The only requirement is that the uncertainty is understood well and can be simulated for the training data.

In our experiments, the JES uncertainty is included in the training set via random Gaussian shifts applied to each jet (see previous section for details). Just one smeared version of each jet is included in the training set to keep the total train size constant. The left plot in Figure 5.18 shows the performance in terms of AUC as a function of the amount of uncertainty included in the test sample (σ_{JES}). As the pink curve indicates, introducing energy smearing in the training data can decrease the performance on unsmeared data. The trained network performs the worst on data without any smearing. However, the network shows also the best performance on the dataset with largest smearing included. Thus, by simply including the systematic uncertainty in the training set the taggers show improved performance on

samples with systematic uncertainties.

The right hand side of Figure 5.18 illustrates the prediction for a single top jet and how smearing the hardest subjet does not effect the prediction for a classifier trained on smeared data. The points without error bars indicate the predictions of a non-Bayesian version of the top tagger. Both network versions react in the same way to the data augmentation. They become stable with respect to the random energy shifts. The networks learnt to not trust the absolute values of the 4-momenta any longer and, therefore, rely more on features which are unaffected by small random energy shifts. In this sense, depending on the strength of the included uncertainty the network output decorrelates from the systematic uncertainty.

For even further decorrelation of the uncertainty from the tagger output, an adversarial network could be included to the training. An adversarial network is an additional neural network working against the classifier and trying to guess in which direction the uncertainty shift was performed by only relying on the output of the classifier. This was for example considered by Ref. [51] and applied on a theoretical uncertainty.

To conclude, the simple idea of including systematic uncertainties in the training sample shows promising results. An improved performance on samples with uncertainties included is observed. Depending on the strength of the included energy shifts the tagger output starts decorrelating from the uncertainty. A full decorrelation could be accomplished by adding an adversarial network. However, fully decorrelating the uncertainty from the tagger output doesn't necessary have to lead to the best performance. The network could possibly learn the uncertainty distribution and the performance could benefit from a remaining slight dependence on the strength of the uncertainty. Further investigations have to be made to answer what the best strategy would be.

6 Conclusion and outlook

The studies discussed in this thesis showed that Bayesian neural networks provide many features, which could potentially be useful for LHC physics. The Bayesian neural networks were trained on distinguishing top and QCD jets. It was demonstrated that they reach the same performance as their deterministic counterparts (see Section 5.7), thus, proving that they are as good as deterministic models, while providing additional information in terms of an output uncertainty.

Is was shown that Bayesian neural networks are better calibrated as the deterministic versions and that badly calibrated taggers can always be recalibrated via the help of post-calibration methods (see Section 5.4). To justify the Bayesian output uncertainty and the approximations needed for the approach, several experiments were performed. The comparison to a frequentist-like method demonstrated that both approaches lead to similar results (see Section 5.5). However, when considering complex architectures the Bayesian networks are less computational expensive. Furthermore, the dependence on the prior was investigated and it has been shown that the prior doesn't dominate the Bayesian output distributions.

Several sections of this thesis investigated if the Bayesian output uncertainty correlates with uncertainties known in physics. Section 5.3 showed that the output uncertainties of Bayesian classifiers correlate with the overall training size and, thus, proving that the jet-level uncertainty correlates with uncertainties which arise from finite training data. These effects were found to be orthogonal to the correlation between the predictive mean and standard deviation. Furthermore, Section 5.8 showed that the output uncertainties correlate with systematic uncertainties and pile-up, if these effects are not present in the training data. For pile-up, increased uncertainties were found for the Bayesian LoLa version. For the JES uncertainty, it was demonstrated how the correlation between mean and standard deviation usually dominates the jet-level uncertainty. However, after going into before-sigmoid space, small uncorrelated effects were visible, proving that the Bayesian output uncertainty provides additional information, not contained in the mean output. These additional decorrelation steps were not required for pile-up nor the study about training statistics. Whether these steps which are needed for uncertainties with small impact on the tagger can be removed in the future, requires additional investigations.

Furthermore, an increased stability with respect to pile-up was proven for B-Lola (see Section 5.8.1). This improved stability can be easily understood by studying the individual weight uncertainties of the first two layers, demonstrating that the

Bayesian approach provides an additional handle, which can be used to better understand the decision of neural networks, which are often seen as black box algorithm. These improvements were found for Bayesian neural networks not trained on pile-up. However, large Bayesian output uncertainties, as they were present for pile-up, can be understood as a warning about inconsistencies between training and test sample, leading to the natural next step of including these inconsistencies in the training data. This was demonstrated for the JES uncertainty (see Section 5.8.3). By including the uncertainty in the training sample increased performance was found on datasets with included uncertainty. For further decorrelation of the tagger output from uncertainties adversarial networks could be introduced. However, whether a full decorrelation is really the best strategy, requires additional investigations.

Acknowledgments

I would like to thank my two supervisors Prof. Dr. Tilman Plehn and Dr. Jennifer Thompson. Without their help I could have never accomplished any of the work presented in this thesis. I would also like to thank Jun.-Prof. Dr. Gregor Kasieczka for all the helpful discussions about deep learning and uncertainties, and Sascha Diefenbacher for all the coding advices. Furthermore, I would like to thank all other members of the particle physics phenomenology research group of the ITP for the friendly environment. I am looking forward continuing my work with all of you.

7 Bibliography

[1] ATLAS Collaboration, Phys. Lett. B 716 (2012) 1-29 doi:10.1016/j.physletb.2012.08.020 [arXiv:1207.7214 [hep-ex]]; CMS Collaboration, Phys. Lett. B 716 (2012) 30 doi:10.1016/j.physletb.2012.08.021 [arXiv:1207.7235 [hep-ex]].

[2] D. E. Kaplan, K. Rehermann, M. D. Schwartz, B. Tweedie, Phys. Rev. Lett. 101:142001,2008 doi:10.1103/PhysRevLett.101.142001 [arXiv:0806.0848 [hep-ph]]; L. G. Almeida, S. J. Lee, G. Perez, I. Sung, J. Virzi, Phys. Rev. D79:074012,2009 doi:10.1103/PhysRevD.79.074012 [arXiv:0810.0934 [hep-ph]].

[3] L. G. Almeida, S. J. Lee, G. Perez, G. Sterman, I. Sung, J. Virzi, Phys. Rev. D79:074017,2009 doi:10.1103/PhysRevD.79.074017 [arXiv:0807.0234 [hep-ph]].

[4] T. Plehn, G. P. Salam, M. Spannowsky Phys. Rev. Lett.104:111801,2010 doi:10.1103/PhysRevLett.104.111801 [arXiv:0910.5472 [hep-ph]].

[5] Y. Cui, Z. Han, M. D. Schwartz, Phys. Rev. D83:074023,2011 doi:10.1103/PhysRevD.83.074023 [arXiv:10.1103/PhysRevD.83.074023].

[6] C. Patrignani *et al.* (Particle Data Group), Chin. Phys. C, **40**, 100001 (2016) and 2017 update.

[7] S. Marzani, G. Soyez, M. Spannowsky Lecture Notes in Physics, volume 958 (2019) doi:10.1007/978-3-030-15709-8 [arXiv:1901.10342 [hep-ph]].

[8] T. Plehn, M. Spannowsky, M. Takeuchi, Phys. Rev. D 85, 034029 (2012) doi:10.1103/PhysRevD.85.034029 [arXiv:1111.5034 [hep-ph]]; C. Anders, C. Bernaciak, G. Kasieczka, T. Plehn, T. Schell, Phys. Rev. D 89, 074047 (2014) doi:10.1103/PhysRevD.89.074047 [arXiv:312.1504 [hep-ph]].

[9] H. Qu, L.Gouskos (2019) [arXiv:1902.08570 [hep-ph]].

[10] G. Kasieczka *et al.* SciPost Phys. **7**, 014 (2019) doi:0.21468/SciPostPhys.7.1.014 [arXiv:1902.09914 [hep-ph]].

[11] S. Bollweg, M. Haussmann, G. Kasieczka, M. Luchmann, T. Plehn, Jennifer Thompson (2019), [arXiv:1904.10004 [hep-ph]].

[12] M. Cacciari, G. P. Salam and G. Soyez, JHEP **0804**, 063 (2008) doi:10.1088/1126-6708/2008/04/063 [arXiv:0802.1189 [hep-ph]].

[13] P. T. Komiske, E. M. Metodiev, M. D. Schwartz, JHEP 01 (2017) 110 doi:10.1007/JHEP01(2017)110 [arXiv:arXiv:1612.01551 [hep-ph]]; T. Cheng, Comput Softw Big Sci (2018) 2: 3, doi:10.1007/s41781-018-0007-y [arXiv:1711.02633 [hep-ph]]; G. Kasieczka, N. Kiefer, T. Plehn, J. M. Thompson, SciPost Phys. 6, 069 (2019) doi:10.21468/SciPostPhys.6.6.069 [arXiv:1812.09223 [hep-ph]].

[14] A. Butter, G. Kasieczka, T. Plehn and M. Russell, SciPost Phys. **5**, 028 (2018) doi:10.21468/SciPostPhys.5.3.028 [arXiv:1707.08966 [hep-ph]]. For top tagging dataset: `https://docs.google.com/document/d/1Hcuc6LBxZNX16zjEGeq16DAzspkDC4nDTyjMp1bWHRo/edit`.

[15] K. Hornik, M.Stinchcombe, H. White (1989) doi:10.1016/0893-6080(89)90020-8.

[16] D. Cireşan, U. Meier, J. Schmidhuber CVPR 2012, p. 3642-3649 doi:10.1109/CVPR.2012.6248110 [arXiv:1202.2745 [cs.CV]].

[17] P. Ramachandran, B. Zoph, Q. V. Le (2017) [arXiv:1710.05941 [cs.NE]].

[18] G. Kasieczka, T. Plehn, M. Russell, T. Schell JHEP **05** (2017) 006 doi:10.1007/JHEP05(2017)006 [arXiv:1701.08784 [hep-ph]].

[19] I. Goodfellow, Y. Bengio, A.Courville (2016) "Deep Learning", MIT Press, `http://www.deeplearningbook.org`.

[20] D. P. Kingma, J. Ba (2014) [arXiv:1412.6980 [cs.LG]].

[21] M. D. Zeiler (2012) [arXiv:arXiv:1212.5701 [cs.LG]].

[22] D. MacKay Comp. in Neural Systems, **6**, 4679 (1995); R. Neal, doctoral thesis, Toronto 1995; Y. Gal, doctoral thesis, Cambridge 2016.

[23] P. C. Bhat and H. B. Prosper, Conf. Proc. C **050912**, 151 (2005); S. R. Saucedo, Florida State University, master thesis (2007).

[24] Y. Xu, W. w. Xu, Y. x. Meng, K. Zhu and W. Xu, Nucl. Instrum. Meth. A **592**, 451 (2008) doi:10.1016/j.nima.2008.04.006 [arXiv:0712.4042 [physics.data-an]].

[25] D. M. Blei, JASA **112**, 859 (2017) doi:10.1080/01621459.2017.1285773 [arXiv:1601.00670 [stat.CO]].

[26] a similar study in a different field: C Leibig, V Allken, M. S. Ayhan1, P. Berens, and S. Wahl, Scientific Reports **7**, 17816 (2017) doi:10.1038/s41598-017-17876-z.

[27] P. Mehta, M. Bukov, C. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, D. J. Schwab, Phyics Reports 810 (2019) 1-124, doi:10.1016/j.physrep.2019.03.001 [arXiv:1803.08823 [physics.comp-ph]].

[28] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012), [arXiv:1207.0580 [cs.NE]].

[29] Y. Gal and Z. Ghahramani, Proc. ICML (2016) [arXiv:1506.02142 [stat.ML]].

[30] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra (2015), [arXiv:1505.05424 [stat.ML]].

[31] D. P. Kingma, M. Welling (2013) [arXiv:1312.6114 [stat.ML]].

[32] A. Kendall, Y.- Gal (2017) [arXiv:1703.04977 [cs.CV]].

[33] T. Sjöstrand *et al.*, Comput. Phys. Commun. **191** (2015) 159 doi:10.1016/j.cpc.2015.01.024 [arXiv:1410.3012 [hep-ph]].

[34] J. de Favereau *et al.* [DELPHES 3 Collaboration], JHEP **1402**, 057 (2014) doi:10.1007/JHEP02(2014)057 [arXiv:1307.6346 [hep-ex]].

[35] M. Cacciari and G. P. Salam, Phys. Lett. B **641**, 57 (2006) doi:10.1016/j.physletb.2006.08.037 [arXiv:hep-ph/0512210]; M. Cacciari, G. P. Salam and G. Soyez, Eur. Phys. J. C **72**, 1896 (2012) doi:10.1140/epjc/s10052-012-1896-2 [arXiv:1111.6097 [hep-ph]].

[36] S. Macaluso and D. Shih, JHEP **1810**, 121 (2018) doi:10.1007/JHEP10(2018)121 [arXiv:1803.00107 [hep-ph]].

[37] Gregor Kasieczka, Tilman Plehn, Michael Russell, Torben Schell, JHEP 05 (2017) 006 doi:10.1007/JHEP05(2017)006 [arXiv:1701.08784 [hep-ph]].

[38] Y. Wen, P. Vicol, J. Ba, D. Tran, and R. Grosse (2018) [arXiv:1803.04386 [cs.LG]].

[39] M. Abadi *et al.*, OSDI **16**, 265 (2016); Tensorflow probability.

[40] F. Chollet *et al.*, Deep learning library Keras, `https://github.com/keras-team/keras`.

[41] A. Kendall and Y. Gal, Proc. NIPS (2017) [arXiv:1703.04977 [cs.CV]].

[42] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, Proc. ICML (2017) [arXiv:1706.04599 [cs.LG]].

[43] J. Platt, Adv. Large Margin Classifiers **10(3)**, 61 (1999).

[44] M. Cacciari, G. P. Salam and G. Soyez, Eur. Phys. J. C **75**, no. 2, 59 (2015) doi:10.1140/epjc/s10052-015-3267-2 [arXiv:1407.0408 [hep-ph]].

[45] D. Bertolini, P. Harris, M. Low and N. Tran, JHEP **1410**, 059 (2014) doi:10.1007/JHEP10(2014)059 [arXiv:1407.6013 [hep-ph]].

[46] J. A. Martinez, O. Cerri, M. Pierini, M. Spiropulu, J.-R. Vlimant (2018) [arXiv:1810.07988 [hep-ph]].

[47] ATLAS Collaboration, Phys. Rev. D 96, 072002 (2017) doi:10.1103/PhysRevD.96.072002 [arXiv:1703.09665 [hep-ex]].

[48] CMS Collaboration, JINST 12 (2017) P02014 doi:10.1088/1748-0221/12/02/P02014 [arXiv:1607.03663 [hep-ex]].

[49] D. Cireşan, U. Meier, J. Schmidhuber CVPR 2012, p. 3642-3649 [arXiv:1202.2745 [cs.CV]].

[50] C. Burges and C. Cortes, the MNIST dataset (1998) `http://yann.lecun.com/exdb/mnist/`.

[51] C. Englert, P. Galler, P. Harris, M. Spannowsky (2018) doi:10.1140/epjc/s10052-018-6511-8 [arXiv:1807.08763 [hep-ph]].

A Implementation of the loss function

In Section 4.1 the KL-divergence was introduced as the loss function for Bayesian neural networks. For the implementation one has to be careful about the correct pre-factors. In Section 4.1 the loss function was derived as:

$$L = \mathrm{KL}[q_\theta(\omega), p(\omega)] - \int d\omega \; q_\theta(\omega) \; \log p(D|\omega) \tag{A.1}$$

The second term involves the negative log-likelihood, which can be written as (see Equation 3.7):

$$\log p(D|\omega) = \sum_{i=1}^{N} \sum_{c=0}^{M} y_i^c \; \log p(c|x_i, \omega) \tag{A.2}$$

where N is the number of samples in the training dataset and $p(c_i|x_i, \omega)$ is the network output and likelihood of observing the class c for a given input x_i and network parameters ω. y_i^c is a binary label which is 1 for the correct class and 0 otherwise. However, usually the loss function is not evaluated on the entire training dataset, but on minibatches of size L. Minibatch optimization approximates the negative log-likelihood as:

$$\log p(D|\omega) \approx \frac{N}{L} \sum_{i=1}^{L} \sum_{c=0}^{M} y_i^c \; \log p(c|x_i, \omega) \tag{A.3}$$

where the prefactor comes from the fact that we assume the average over one minibatch approximates the average over the entire training dataset. It is common practice to divide the loss function by he number of samples N of the training dataset. In this way the sum turns into an average. The loss function becomes:

$$L = \frac{1}{N} \mathrm{KL}[q_\theta(\omega), p(\omega)] - \frac{1}{L} \sum_{i=1}^{L} \sum_{c=0}^{M} \int d\omega \; q_\theta(\omega) \; y_i^c \; \log p(c|x_i, \omega) \tag{A.4}$$

It is important to notice the different prefactors in front of the first and second term. As mentioned in Section 4.4 the integral over the log-likelihood can be computed via Monte Carlo integration and the gradients can be computed with the help of the reparameterization trick. The term in the loss function becomes:

$$L = \frac{1}{N} \mathrm{KL}[q_\theta(\omega), p(\omega)] - \frac{1}{L} \sum_{i=1}^{L} \sum_{c=0}^{M} \frac{1}{K} \sum_{k=1}^{K} y_i^c \; \log p(c|x_i, \omega_k) \tag{A.5}$$

Where K is the number of Monte Carlo samples and ω_k are samples from the variational distribution $\omega_k \in q_\theta(\omega)$. Section 4.2 showed that it is possible to give a closed form for the first term, if the variational distribution and the prior are factorized Gaussian distributions. In this context a parameter for L2 regularization was derived

$$\lambda = \frac{1}{2\sigma_{prior}^2} \tag{A.6}$$

However, because of the additional prefactor from above the factor has to be corrected by the training size:

$$\lambda = \frac{1}{2\sigma_{prior}^2 N} \tag{A.7}$$

B Lists

B.1 List of Figures

B.2 List of Tables

Erklärung:

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

Heidelberg, den (Datum) .....