Uncertainty quantification[1]

July 2, 2025
ML4Physics@Ljubljana Summer School

Luigi Favaro

1 Introduction

These notes are a first introduction to the interpretation of neural networks as Bayesian probabilistic models and the extraction of uncertainties from trained neural networks. Because of limited time we only cover the basics of methodologies which are promising for the future of learned uncertainties. This selection is based on a set of desiderata, in particular:

- the training time should not differ too much from a deterministic counterpart;
- the method has to scale well with large amount of data;
- it has to be applicable to various problems and and neural network architectures;

Because of professional bias, some of the examples are taken from particle physics, but the general theory is much more widely applicable.

1.1 Learned uncertainties

The characterization of uncertainties is problem specific and a general definition which applies to everyone is pretty much impossible. Particle physics has one of the most stringent requirements before a discovery can be claimed and proper treating of uncertainties is essential. We define, most generally, two kinds of uncertainties, *statistical uncertainties* and systematic uncertainties. The first kind is defined by the fact that they vanish for large statistics, while the second class do not vanish with large statistics, because they come from a reference measurement or calibration, because they describe detector effects, or they arise from theory predictions which do not offer a statistical interpretation. Some systematic uncertainties are Gaussian, for example when they describe a high-statistics measurement in a control region. Others just give a range and no preference for a single central value, for instance in the case of theory predictions based on perturbative QCD. Sometimes a Gaussian systematic uncertainty is a valid approximation for a measurement. For instance, different non-Gaussian noise sources can add up to a Gaussian contribution as a consequence of the central limit theorem.

[1]These notes are mostly built upon:
"Uncertainty in Deep Learning", Yarin Gal, [1]
"Modern machine learning for LHC physicists", Plehn et al., [2]

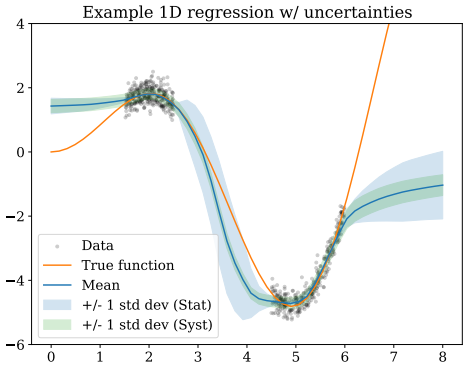

Figure 1: Example 1D regression problem with limited statistics. (blue) Expected behaviour of uncertainties related to lack of knowledge. (green) Uncertainties of systematic nature inherently included in the data.

The machine learning community also distinguishes between two kinds of uncertainties, *aleatoric uncertainties* related to the (training) data and *epistemic uncertainties* related to the model we use to describe the data. This separation is similar to statistical and systematic uncertainties: first, epistemic uncertainties vanish when we train a network on more, assumed perfect, data — in physics we would call them statistical uncertainties; aleatoric uncertainties arise from imperfect data, such that more of the same data will not help — in physics this defines systematic uncertainties. An example would be the intrinsic noise introduced by the measurement process of an observable quantity.

1D example Let us have a look at a 1D examples which we will also use during the exercise session. Fig. 1 shows a simple regression problem where, given x observed values, we have to infer the functional form of y.

We see that the data contains noise spread around the true values. We would like to capture the shape, per phase space point, of the observed noise. Then, we may also ask for predictions outside the training distribution. How can the network know if the prediction is accurate? Because of this lack of information, we would like to have a second uncertainty which covers the limited knowledge that can be extracted from data. In general, we would also like to disentangle the two uncertainties and part of the discussion will be on how to separate them.

Neural network uncertainties are typically a small component in the much larger effort of a full statistical analysis. Although, the uncertainty in defining an observable might not be part of the uncertainty treatment at the analysis level, it will affect its optimality. This means that especially for numerically defined observable we need to control underlying uncertainty like the statistics of the training data, systematic affects

related to the training, or theoretical aspects related to the definition of an observable. Historically, these uncertainties mattered less, but with the rapidly growing complexity of LHC data and analyses, they should be controlled. This motivates the study and treatment of uncertainties in ML techniques.

2 Bayesian modeling

In a Bayesian approach to the problem, we seek to find the function $f_\theta(\cdot)$ that best describes the data pairs (x, y) in a probabilistic sense. A Bayesian description requires some ingredients:

- a prior $p(\theta)$: the prior belief that the data is generated by the parameters θ;
- the likelihood $p(y|x, \theta)$: a probabilistic model which returns the prediction, given the input x and the parameters θ;
- the posterior $p(\theta|x, y)$: the distribution that describes the most probable parameters given the current observations;

Given these elements, we ask the question: which parameters have most likely generated the data? Hence, we want to have access to the posterior of the neural network. This will allow us to extract uncertainties on the predicted output $y = f_\theta(\cdot)$, or often also called *predictive uncertainty*.

2.1 Mathematical formulation

Let us now formulate the problem in a mathematically rigorous way and highlight the challenges related to uncertainty estimation and why this is an open problem in modern deep learning. We use Bayes theorem to relate the posterior distribution to the likelihood, i.e.

$$p(\theta|x, y) = \frac{p(y|x, \theta)p(\theta)}{p(y|x)} \,, \tag{1}$$

where we introduced the new term $p(y|x) = \int d\theta p(y|x, \theta)p(\theta)$, also called the *evidence* or *marginal likelihood*. The prior distribution is set by us and the likelihood can, in principle, be chosen according to the problem. For instance, we could assume a Gaussian likelihood of the form

$$p(y|x, \theta) = \mathcal{N}(y; f_\theta(x), \sigma(x)) \,, \tag{2}$$

or a binary cross-entropy likelihood for a classification problem.

Given a posterior distribution, we can predict the output of new datapoints as

$$p(\bar{y}|\bar{x}) = \int d\theta \, p(\bar{y}|\bar{x}, \theta) \, p(\theta|x, y) \,. \tag{3}$$

For instance this will be the process which we are going to apply for the prediction of observables on the test set (*inference*).

The true posterior $p(\theta|x, y)$ is often not analytically accessible, especially for modern neural network architectures. The core of the research in Bayesian neural networks consists of constructing approximating distributions that are as close as possible to the original (intractable) posterior distribution.

We will explore two widely used way to approximate the posterior of the network parameters θ called *variational inference* and *repulsive ensembles*. Both will lead us to the Bayesian Neural Networks. The difference between a deterministic and a Bayesian network is that the latter allows for distributions of network parameters, which then define distributions of the network output and provide central values $f_\theta(x)$ as well as uncertainties on $f_\theta(x)$ by sampling over θ-space.

Let us now further discuss the variational approximation of the true posterior. Instead of dealing with the intractable posterior, we define an approximate variational distribution $q_\phi(\theta)$, which is easy to evaluate. This function will be parametrised by a general set of parameters ϕ, for instance the mean and variance of a Gaussian distribution. This will be the tractable approximation of the true posterior $p(\theta|x_{\text{train}})$, where we use x_{train} to indicate that posterior is defined by the training dataset.

Now, we can think of $p(y|x, \theta)$ as a single model described through a set of network parameters, while $p(\theta|x_{\text{train}})$ weights this model by its level of agreement with the training data. A *variational approximation* consist of training a neural network to approximate the unknown $p(\theta|x_{\text{train}})$.

$$p(y|x) = \int d\theta\; p(y|x,\theta)\; p(\theta|x_{\text{train}}) \approx \int d\theta\; p(y|x,\theta)\; q_\phi(\theta|x) \equiv \int d\theta\; p(y|x,\theta)\; q_\phi(\theta)\;. \quad (4)$$

In the next section, we discuss one practical method for defining $q_\phi(\theta)$ which satisfies the requirements we discussed in the introduction.

The second method assumes an intractable posterior distribution $q(\theta)$ and the training procedure will provide us with only a fixed number of samples, or weight configurations, asymptotically distributed according to the posterior distribution.

3 Variational Inference

To derive a loss function for the for the approximation of the true posterior of the neural network we need to define a scalar quantity which measures the distance between the target and the learnable distribution. Importantly, it should allow to solve an optimization problem without having access to the analytic target distribution, and it should allow for an unbiased Monte Carlo estimate.

3.1 Intermezzo: Kullback-Leibler divergence

We introduce the *Kullback–Leibler divergence*, which compares two probability distributions, evaluated on a dataset corresponding to the first distribution,

$$\mathrm{KL}[p_a, p_b] = \sum p_a \ln \frac{p_a}{p_b} \neq \mathrm{KL}[p_b, p_a] \; . \tag{5}$$

For continuous distributions it reads

$$\mathrm{KL}[p_a, p_b] = \left\langle \log \frac{p_a}{p_b} \right\rangle_{p_a} \equiv \int dx \; p_a(x) \; \log \frac{p_a(x)}{p_b(x)} \; . \tag{6}$$

The KL-divergence is definite positive and is zero if and only if $p_a = p_b$. On the negative side, it is not a distance, meaning that the KL with swapped arguments will generally give different results. Because the KL-divergence is not symmetric in its two arguments we can evaluate a forward and a reverse KL-divergence,

$$\mathrm{KL}[p_a, p_b] = \left\langle \log \frac{p_a}{p_b} \right\rangle_{p_a} \qquad \text{or} \qquad \mathrm{KL}[p_b, p_a] = \left\langle \log \frac{p_b}{p_a} \right\rangle_{p_b} \; . \tag{7}$$

The difference between them is which of the two distributions we choose to sample the logarithm from, and we will always choose the version that suits the problem better.

Considering p_a as the reference distribution, minimizing the forward KL ensures that all modes of p_a are covered by p_b. Instead, the reverse KL prefers to push towards zero the regions where p_a is zero, therefore seeking the main modes of the reference distribution.

Question: assuming that $p_a = p(\theta|x\mathrm{train})$ and $p_b = q(\theta)$, Should we try to minimize the forward or the inverse KL-divergence?

3.2 Loss derivation

We now define the variational approximation using the KL-divergence introduced in Eq.(6),

$$\mathrm{KL}[q(\theta), p(\theta|x_{\mathrm{train}})] = \left\langle \log \frac{q(\theta)}{p(\theta|x_{\mathrm{train}})} \right\rangle_q = \int d\theta \; q(\theta) \; \log \frac{q(\theta)}{p(\theta|x_{\mathrm{train}})} \; . \tag{8}$$

Using Bayes' theorem we can write the KL-divergence as

$$\begin{aligned} \mathrm{KL}[q(\theta), p(\theta|x_{\mathrm{train}})] &= \int d\theta \; q(\theta) \; \log \frac{q(\theta) p(x_{\mathrm{train}})}{p(\theta) p(x_{\mathrm{train}}|\theta)} \\ &= \mathrm{KL}[q(\theta), p(\theta)] - \int d\theta \; q(\theta) \; \log p(x_{\mathrm{train}}|\theta) + \log p(x_{\mathrm{train}}) \int d\theta \; q(\theta) \; . \end{aligned} \tag{9}$$

The prior $p(\theta)$ describes the network parameters before training; since it does not really include prior physics or training information we will still refer to it as a prior, but we

think about it as a hyperparameter which can be chosen to optimize performance and stability. From a practical perspective, a good prior should be weakly informative since it will help the network converge more efficiently, but any prior should give the correct results, and we always need to test the effect of different priors.

The evidence $p(x_{\text{train}})$ guarantees the correct normalization of $p(\theta|x_{\text{train}})$ and is usually intractable. If we implement the normalization condition for $q(\theta)$ by construction, we find

$$\text{KL}[q(\theta), p(\theta|x_{\text{train}})] = \text{KL}[q(\theta), p(\theta)] - \int d\theta \; q(\theta) \; \log p(x_{\text{train}}|\theta) + \log p(x_{\text{train}}) \; . \tag{10}$$

The log-evidence in the last term does not depend on θ, which means that it will not be adjusted during training and we can ignore when constructing the loss. However, it ensures that $\text{KL}[q(\theta), p(\theta|x_{\text{train}})]$ can reach its minimum at zero. Alternatively, we can solve the equation for the evidence and find

$$\begin{aligned} \log p(x_{\text{train}}) &= \text{KL}[q(\theta), p(\theta|x_{\text{train}})] - \text{KL}[q(\theta), p(\theta)] + \int d\theta \; q(\theta) \; \log p(x_{\text{train}}|\theta) \\ &> \int d\theta \; q(\theta) \; \log p(x_{\text{train}}|\theta) - \text{KL}[q(\theta), p(\theta)] \end{aligned} \tag{11}$$

This condition is called the *evidence lower bound (ELBO)*, and the evidence reaches this lower bound exactly when our training condition in Eq.(6) is minimal. Combining all of this, we turn Eq.(10) or, equivalently, the ELBO into the loss function for a Bayesian network,

$$\mathcal{L}_{\text{BNN}} = - \int d\theta \; q(\theta) \; \log p(x_{\text{train}}|\theta) + \text{KL}[q(\theta), p(\theta)] \; . \tag{12}$$

The first term of the BNN loss is the likelihood sampled according to $q(\theta)$, the second enforces a given prior. Using an ELBO loss means nothing but minimizing the KL-divergence between the probability $p(\theta|x_{\text{train}})$ and its network approximation $q(\theta)$ and neglecting all terms which do not depend on θ. It results in two terms, a likelihood and a KL-divergence, which we will study in more detail next.

Clearly, we only have access to samples from the posterior $q(\theta)$ which means that we need an unbiased estimate of the loss function to perform the training,

$$\mathcal{L}_{\text{BNN}} \approx -\frac{1}{S} \sum_{i=1}^{S} \left(\frac{1}{B} \sum_{b=1}^{S} \log p(x_b|\theta^i) \right) + \text{KL}[q(\theta), p(\theta)] \tag{13}$$

where $\theta^{(i)} \sim q(\theta)$ and B is the size of the batch.

The Bayesian network output is constructed in a non-linear way with a large number of layers, to we can assume that Gaussian weight distributions do not limit us in terms of the uncertainty on the network output.

Let us now discuss one specific example of a variational inference BNN by first looking at the deterministic limit of a BNN. This means we want to look at the loss function of the BNN in the limit

$$q(\theta) = \delta(\theta - \theta_0) \; . \tag{14}$$

The easiest way to look at this limit is to first assume a Gaussian form of the network parameter distributions,

$$q_{\mu,\sigma}(\theta) = \frac{1}{\sqrt{2\pi}\sigma_q} e^{-(\theta-\mu_q)^2/(2\sigma_q^2)} \; , \tag{15}$$

and correspondingly for $p(\theta)$. The KL-divergence has a closed form,

$$\mathrm{KL}[q_{\mu,\sigma}(\theta), p_{\mu,\sigma}(\theta)] = \frac{\sigma_q^2 - \sigma_p^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} + \log \frac{\sigma_p}{\sigma_q} \; . \tag{16}$$

We can now evaluate this KL-divergence in the limit of $\sigma_q \to 0$ and finite $\mu_q(\theta) \to \theta_0$ as the one remaining θ-dependent parameter,

$$\mathrm{KL}[q_{\mu,\sigma}(\theta), p_{\mu,\sigma}(\theta)] \to \frac{(\theta_0 - \mu_p)^2}{2\sigma_p^2} + \text{const} \; . \tag{17}$$

Exercise: Using the definition of KL-divergence, derive Eq. 17. We can write down the deterministic limit of Eq.(12),

$$\mathcal{L}_{\mathrm{BNN}} \to -\log p(x_{\mathrm{train}}|\theta_0) + \frac{(\theta_0 - \mu_p)^2}{2\sigma_p^2} \; . \tag{18}$$

The first term is again the likelihood defining the correct network parameters, the second ensures that the network parameters do not become too large. Because it includes the squares of the network parameters, it is referred to as an L2-regularization. Going back to Eq.(12), an ELBO loss is a combination of a likelihood loss and a regularization. Changing the prefactor of the regularization corresponds to a change in the variance of the prior distribution.

We generalize this idea, choose $\mu_p = 0$, and apply the L2-regularization with a freely chosen pre-factor instead of σ_p. In that case the deterministic and L2-regularized likelihood loss reads

$$\mathcal{L}_{\mathrm{L2}} = -\log p(x_{\mathrm{train}}|\theta_0) + \lambda\theta_0^2 \; , \tag{19}$$

with a free hyperparameter λ.

Short discussion of Monte Carlo dropout We are not restricted to a Gaussian distribution for the sampling step of the neural network weights. For instance a Bernoulli distribution is also a practical, simple to implement, possibility. This especially simple sampling of weights by removing nodes is called *dropout* and is commonly used to avoid overfitting of networks. In this framework, it appears as a key component of the loss function. Therefore, the method can be used to get an approximation of the posterior distribution by simply using dropout during test time. For more details on the connections between MC dropout and approximate inference of posterior distribution, see [1].

Important! Fitting the posterior over the weights of a Bayesian NN with a unimodal approximating distribution $q_\phi(\theta)$ does not imply that the predictive distribution will be unimodal!

4 Repulsive Ensembles

The second approach to learned uncertainties is based on the training an ensemble of neural networks [3]. The update rule of a neural network uses gradient descent to minimize a loss function. similarly to the previous section, the network training should maximize the probability of the network weights given a training dataset x_{train}. The gradient descent update will then look like

$$\theta^{t+1} = \theta^t + \alpha \nabla_{\theta^t} \log p(\theta^t | x_{\text{train}}) \,, \tag{20}$$

where t is the training iteration. We can extend the update rule to an ensemble of networks by applying the update step to all networks in the ensemble. The variation between the members of the ensemble from random network initialization and training on random batches will give different predictions from the same inputs, however there is no guarantee that an ensemble trained this way will actually converge to the correct posterior.

We now study an improvement over standard ensembles that ensures the asymptotic convergence to the correct posterior distribution. Such an interaction should take into account the proximity of the ensemble member θ to all other members.

Naive approach We introduce a kernel $k(\theta, \theta_j)$, typically chosen as Gaussian, and sum the interactions with all other weight configurations

$$\theta^{t+1} = \theta^t + \alpha \nabla_{\theta^t} \left[\log p(\theta^t | x_{\text{train}}) - \frac{1}{n} \sum_{j=1}^{n} k(\theta^t, \theta_j^t) \right] \,. \tag{21}$$

The question is if this update rule, for a given kernel, leads to ensemble members sampling the weight probability,

$$\theta \sim p(\theta|x_{\text{train}}) \ . \tag{22}$$

Full derivation To answer this question we need to relate the update rule, or the discretized t-dependence of a weight vector $\theta(t)$, to a time-dependent probability density $\rho(\theta, t)$. There are two equivalent ways to describe the time evolution of a system, an ODE or a continuity equation,

$$\frac{d\theta}{dt} = v(\theta, t) \qquad \text{or} \qquad \frac{\partial \rho(\theta, t)}{\partial t} = -\nabla_\theta \left[v(\theta, t) \rho(\theta, t) \right] \ . \tag{23}$$

For a given velocity field $v(\theta, t)$ the individual paths $\theta(t)$ describe the evolving density $\rho(\theta, t)$ and the two conditions are equivalent. If we choose the velocity field as

$$v(\theta, t) = -\nabla_\theta \log \frac{\rho(\theta, t)}{\pi(\theta)} \ , \tag{24}$$

these two equivalent conditions read

$$\begin{aligned} \frac{d\theta}{dt} &= -\nabla_\theta \log \frac{\rho(\theta, t)}{\pi(\theta)} \\ \frac{\partial \rho(\theta, t)}{\partial t} &= \nabla_\theta \left[\rho(\theta, t) \nabla_\theta \log \frac{\rho(\theta, t)}{\pi(\theta)} \right] \\ &= -\nabla_\theta \left[\rho(\theta, t) \nabla_\theta \log \pi(\theta) \right] + \nabla_\theta^2 \rho(\theta, t) \ . \end{aligned} \tag{25}$$

Exercise: show that the right-hand side of Eq. 25 is zero if $\rho(\theta, t) = \pi(\theta)$

The stationary solution of the Fokker-Planck equation is unique, so we know that the evolution given by the first line of Eq.(25) is guaranteed to converge to $\rho(\theta, t) \to \pi(\theta)$. The choice of v is simply made such that we approach the stationary distribution $\pi(\theta)$ starting from the current state of the network $\rho(\theta, t)$.

Next, we relate the ODE in Eq.(25) to the update rule for repulsive ensembles, Eq.(21). The discretized version of the ODE is

$$\frac{\theta^{t+1} - \theta^t}{\alpha} = -\nabla_{\theta^t} \log \frac{\rho(\theta^t)}{\pi(\theta^t)} \ . \tag{26}$$

If we do not know the density $\rho(\theta^t)$ explicitly, we again approximate it as a superposition of kernels with the correct normalization,

$$\rho(\theta^t) \approx \frac{1}{n} \sum_{j=1}^{n} k(\theta^t, \theta_j^t) \qquad \text{with} \qquad \int d\theta^t \rho(\theta^t) = \frac{1}{n} \sum_{j=1}^{n} \int d\theta^t k(\theta^t, \theta_j^t) = \frac{1}{n} \sum_{j=1}^{n} 1 = 1 \ . \tag{27}$$

We can insert this kernel approximation into the discretized ODE,

$$\begin{aligned}\frac{\theta^{t+1}-\theta^t}{\alpha} &= -\nabla_{\theta^t}\left[\log\rho(\theta^t)-\log\pi(\theta^t)\right] \\ &= \nabla_{\theta^t}\log\pi(\theta^t)-\nabla_{\theta^t}\log\left[\frac{1}{n}\sum_j k(\theta^t,\theta_j^t)\right] \\ &= \nabla_{\theta^t}\log\pi(\theta^t)-\nabla_{\theta^t}\log\sum_j k(\theta^t,\theta_j^t) \\ &= \nabla_{\theta^t}\log\pi(\theta^t)-\frac{\nabla_{\theta^t}\sum_j k(\theta^t,\theta_j^t)}{\sum_i k(\theta^t,\theta_i^t)} \end{aligned} \tag{28}$$

This form can be compared to the update rule in Eq.(21). To make them identical, we first identify

$$\pi(\theta) \equiv p(\theta|x_{\text{train}}) \; . \tag{29}$$

This means that our target weight density is the probability of the weights given the training data. We, therefore, derived the correct formula for the repulsive kernel to be used in the optimization process.

The derivation presented here is often referred to as *Wasserstein gradient flow*. For more details see [4, 5] and references therein.

Additional material: function space As proposed in [4] a repulsion term in weight space can be inefficient since deep neural networks have degenerate states where many different weight configurations can lead to the same result. This is why we prefer a repulsive force in the space of network outputs $f_\theta(x)$.

Symbolically, we can then write the update rule from Eq.(21) as

$$\frac{f^{t+1}-f^t}{\alpha} = \nabla_{f^t}\log p(f|x_{\text{train}}) - \frac{\sum_j \nabla_{f^t} k(f,f_j)}{\sum_j k(f,f_j)} \; . \tag{30}$$

A typical choice for the kernel in function space is still a Gaussian in the multi-dimensional function space, evaluated over a sample,

$$k(f_\theta(x), f_{\theta_j}(x)) \propto \exp\left(-\frac{|f_\theta(x)-f_{\theta_j}(x)|^2}{h}\right) \; . \tag{31}$$

The width h should be chosen such that the width of the distribution is not overestimated while still ensuring that it is sufficiently smooth.

No matter how we define the update step, the network training is always in weight space, so we have to translate the function-space update rule into weight space using the

appropriate Jacobian

$$\frac{\theta^{t+1}-\theta^t}{\alpha} = \frac{\partial f^t}{\partial \theta^t}\left[\nabla_{f^t}\log p(f_{\theta^t}|x_{\text{train}}) - \frac{\sum_j \nabla_f k(f_{\theta^t}, f_{\theta_j^t})}{\sum_j k(f_{\theta^t}, f_{\theta_j^t})}\right]$$
$$= \nabla_{\theta^t}\log p(\theta^t|x_{\text{train}}) - \frac{\sum_j \nabla_{\theta^t} k(f_{\theta^t}, f_{\theta_j^t})}{\sum_j k(f_{\theta^t}, f_{\theta_j^t})} \,. \tag{32}$$

Furthermore, we cannot evaluate the repulsive kernel in function space, so we have to evaluate the function for a finite batch of points x,

$$\frac{\theta^{t+1}-\theta^t}{\alpha} \approx \nabla_{\theta^t}\log p(\theta^t|x_{\text{train}}) - \frac{\sum_j \nabla_{\theta^t} k(f_{\theta^t}(x), f_{\theta_j^t}(x))}{\sum_j k(f_{\theta^t}(x), f_{\theta_j^t}(x))} \,. \tag{33}$$

Finally, we turn the update rule in Eq.(33) into a loss function for the repulsive ensemble training. We want to use a tractable likelihood loss, which we get from Bayes' theorem. In the loss function we neglect the evidence $p(x_{\text{data}})$, but have to include the prior $p(\theta)$, which we assume to be Gaussian,

$$\log p(\theta|x_{\text{train}}) = \log p(x_{\text{train}}|\theta) \; - \frac{\theta^2}{2\sigma^2} + \text{const} \,. \tag{34}$$

Given a training dataset of size N, we evaluate the likelihood on batches of size B, so Eq.(33) becomes

$$\frac{\theta^{t+1}-\theta^t}{\alpha} \approx \nabla_{\theta^t}\frac{N}{B}\sum_{b=1}^{B}\log p(x_b|\theta) - \frac{\sum_j \nabla_{\theta^t} k(f_{\theta^t}(x), f_{\theta_j^t}(x))}{\sum_j k(f_{\theta^t}(x), f_{\theta_j^t}(x))} - \nabla_{\theta^t}\frac{\theta^2}{2\sigma^2} \,. \tag{35}$$

Here, $f_{\theta^t}(x)$ is to be understood as evaluating the function for all samples $x_1, \ldots, x_B$ in the batch.

To turn the update rule into a loss function, we flip the sign of term in the gradient, divide it by N to remove the scaling with the size of the training dataset, and sum over all members of the ensemble. Since the gradients of the loss function are computed with respect to the parameters of all networks in the ensemble, we need to ensure the correct gradients of the repulsive term using a stop-gradient operation, denoted with an overline $\overline{f_{\theta_j}(x)}$. The loss function for repulsive ensembles then reads

$$\mathcal{L} = \sum_{i=1}^{n}\left[-\frac{1}{B}\sum_{b=1}^{B}\log p(x_b|\theta_i) + \frac{1}{N}\frac{\sum_{j=1}^{n} k(f_{\theta_i}(x), \overline{f_{\theta_j}(x)})}{\sum_{j=1}^{n} k(\overline{f_{\theta_i}(x)}, \overline{f_{\theta_j}(x)})} + \frac{\theta_i^2}{2N\sigma^2}\right] \,. \tag{36}$$

The prior has just become an L2-regularization with prefactor $1/(2N\sigma^2)$.

5 Regression

We now apply Bayesian neural networks to regression problems. We will derive equations for uncertainties on the network prediction and we will interpret these quantities as different contributions to the total predictive uncertainty.

5.1 Learned uncertainties

As a benchmark problem we will consider the regression of transition scattering amplitudes in QFT. The reason for this choice will be clear during the exercise session but, in general, predicting scattering amplitudes is a real research problem, defined in an extremely controlled environment, which allows for easier understanding of the predicted uncertainties.

Derivation We start with the assumption that the amplitude for a given phase space point is given by a probability distribution $p(A|x)$ with mean

$$\langle A\rangle(x) = \int dA\ A\ p(A|x)\ . \tag{37}$$

We already know how to approximate $p(A|x)$ using variational inference, therefore we can write

$$p(A|x) = \int d\theta\ p(A|x,\theta)\ p(\theta|x_{\text{train}}) \approx \int d\theta\ p(A|x,\theta)\ q(\theta|x_{\text{train}})\ . \tag{38}$$

Then, we extract the mean and the uncertainty for the amplitude A, a scalar quantity, over phase space. In this case, the phase space is a set of four-momenta equal to the number of particles of the considered scattering process. We have to do some algebra to rewrite the mean prediction $\langle A\rangle$ as a θ-sampled quantity. To extract the uncertainty for $A(x)$, we rewrite Eq.(37) such that we sample over θ and define an expectation value and the corresponding variance

$$\begin{aligned}
\langle A\rangle(x) &= \int d\theta \int dA\ q(\theta)\ A\ p(A|x,\theta) \qquad (39)\\
&= \int d\theta\ q(\theta)\ \overline{A}(x,\theta) \qquad \text{with} \qquad \overline{A}(x,\theta) = \int dA\ A\ p(A|x,\theta)\\
\sigma^2_{\text{tot}}(x) &= \langle (A - \langle A\rangle(x))^2\rangle\\
&= \int d\theta\ q(\theta) \left[\overline{A^2}(x,\theta) - \langle A\rangle\right]^2\\
&= \int d\theta\ q(\theta) \left[\overline{A^2}(x,\theta) - \overline{A}(x,\theta)^2 + \left(\overline{A}(x,\theta) - \langle A\rangle(x)\right)^2\right]\\
&\equiv \sigma^2_{\text{syst}}(x) + \sigma^2_{\text{stat}}(x)\ , \qquad (40)
\end{aligned}$$

where $\overline{A^2}(x,\theta)$ is defined in analogy to $\overline{A}(x,\theta)$. The total uncertainty factorizes into two terms. The first,

$$\sigma^2_{\text{syst}}(x) \equiv \int d\theta\; q(\theta)\; \sigma(x,\theta)^2 = \int d\theta\; q(\theta)\; \left[\overline{A^2}(x,\theta) - \overline{A}(x,\theta)^2\right] , \tag{41}$$

corresponds to a learned phase space-dependent error. Given exact $A_{\text{true}}(x)$, it vanishes in the limit of arbitrarily well-known data and perfect network training

$$p(A|x,\theta) = \delta(A - A_{\text{true}}(x)) \qquad \Leftrightarrow \qquad \overline{A^2}(x,\theta) = A_{\text{true}}(x)^2 = \overline{A}(x,\theta)^2 . \tag{42}$$

This uncertainties approaches a plateau for large training data, so we refer to it as a systematic uncertainty—accounting for a noisy data or labels, limited expressivity of the network (structure uncertainty), non-optimal network architectures in the presence of symmetries, non-smart choices of hyperparameters or any other sources of systematic uncertainty.

The second error is the θ-sampled variance

$$\sigma^2_{\text{stat}}(x) = \int d\theta\; q(\theta)\; \left[\overline{A}(x,\theta) - \langle A\rangle(x)\right]^2 , \tag{43}$$

It vanishes in the limit of perfect training leading to uniquely defined network weights θ_0,

$$q(\theta) = \delta(\theta - \theta_0) \qquad \Leftrightarrow \qquad \langle A\rangle(x) = \overline{A}(x,\theta_0) . \tag{44}$$

It represents a statistical uncertainty in that it vanishes in the limit of infinite training data.

For small training datasets, these two uncertainties cannot be easily separated, but we can separate them clearly for sufficiently large training datasets, where σ_{stat} approaches zero, while the systematic error σ_{syst} reaches a finite plateau. Usually, in LHC physics, we can make sure to use enough training data, so

$$\sigma_{\text{tot}}(x) \approx \sigma_{\text{syst}}(x) \gg \sigma_{\text{stat}}(x) . \tag{45}$$

While this systematic uncertainty might also be affected by too little training data, it approaches a plateau for perfect training. If large datasets are available, the systematic uncertainty can be extractred as

$$\sigma_{\text{syst}}(x) = \lim_{\text{large } x_{\text{train}}} \sigma_{\text{tot}}(x) . \tag{46}$$

For LHC applications, where networks are usually trained on simulations, this limit often represents reality.

Because σ_{syst} is θ-dependent, we can read the definitions of $\overline{A}(\theta)$ and $\sigma_{\text{syst}}(\theta)^2$ as sampled over a network parameter distribution $q(\theta)$ for each phase space point x

$$\text{BNN} : x, \theta \to \begin{pmatrix} \overline{A}(\theta) \\ \sigma_{\text{syst}}(\theta) \end{pmatrix} . \tag{47}$$

While we usually do not assume a Gaussian uncertainty on the Bayesian network output, this might be a good approximation for $\sigma_{\text{syst}}(\theta)$. Using this approximation we can write the likelihood $p(y|x,\theta)$ in Eq.(12) as a Gaussian and use the closed form for the KL-divergence in Eq.(16), so the BNN loss function turns into

$$\mathcal{L}_{\text{BNN}} = \int d\theta\, q(\theta) \sum_{\text{points } j} \left[\frac{\left| \overline{A}_j(\theta) - A_j^{\text{truth}} \right|^2}{2\sigma_{\text{syst},j}(\theta)^2} + \log \sigma_{\text{syst},j}(\theta) \right] + \frac{\sigma_q^2 - \sigma_p^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} + \log \frac{\sigma_p}{\sigma_q} \,. \tag{48}$$

As always, the amplitudes and σ_{syst} are functions of phase space x and the sum is done over points in the training dataset.

For a given phase space point x we can then compute the three global network predictions $\langle A \rangle$, σ_{syst}, and eventually σ_{stat}. Unlike the distribution of the individual network weights $q(\theta)$, the amplitude output is not Gaussian.

The loss is minimized with respect to the means and standard deviations of the network weights describing $q(\theta)$. This interesting aspect of this loss function is that, while the training data just consists of amplitudes at phase space points and does not include the uncertainty estimate, the network constructs a point-wise uncertainty from the variation of the network parameters. This means we can rely on a well-defined *likelihood loss* and perform a *maximum likelihood estimate*.

We can again look at the deterministic limit of the neural network, described by $q(\theta) = \delta(\theta - \theta_0)$, with $\mu_p = 0$. We then assume a Gaussian likelihood loss including the proper normalization and learn the uncertainty using

$$\mathcal{L}_{\text{heteroscedastic}} = \sum_{\text{points } j} \left[\frac{\left| \overline{A}_j(\theta_0) - A_j^{\text{truth}} \right|^2}{2\sigma_{\text{syst},j}(\theta_0)^2} + \log \sigma_{\text{syst},j}(\theta_0) \right] + \frac{\theta_0^2}{2\sigma_p^2} \,. \tag{49}$$

The interplay between the first two terms works in a way that the first term can be minimized either by reproducing the data and minimizing the numerator, or by maximizing the denominator. The second term penalizes the second strategy, defining a correlated limit of $\overline{A}$ and σ_{syst} over phase space. Compared to the full Bayesian network this simplified approach has the disadvantages that it only includes the systematic uncertainties.

5.2 Evaluating learned uncertainties

To determine if the learned uncertainty is correctly calibrated, we can look at pull distributions. Let us start with a deterministic network learning $A_{\text{NN}}(x) \approx A_{\text{true}}(x)$ using a heteroscedastic loss. Given the learned local uncertainty $\sigma(x)$, we can evaluate the pull

of the learned combination as

$$t_{\text{het}}(x) = \frac{A_{\text{NN}}(x) - A_{\text{true}}(x)}{\sigma(x)} \; , \tag{50}$$

If $\sigma(x)$ captures the absolute value of the deviation of the learned function from the truth exactly and for any x-value the pull would be

$$A_{\text{NN}}(x) = A_{\text{true}}(x) \pm \sigma(x) \qquad \Leftrightarrow \qquad t(x) = \pm 1 \; . \tag{51}$$

Realistically, an uncertainty estimate will only capture the maximum deviation, so the per-amplitude deviation of $|A_{\text{NN}}(x) - A_{\text{true}}(x)|$ might be smaller. For a stochastic source of uncertainties, the learned values $A_{\text{NN}}(x)$ will follow a Gaussian distribution around the mean $\langle A_{\text{NN}}(x) \rangle$. The width of this Gaussian should be given by the learned uncertainty, which means the pull will follow a unit Gaussian.

As an example, let us assume that we learn $A_{\text{true}}(x)$ from noisy training data,

$$A_{\text{true}}(x) \; \to \; A_{\text{train}}(x) \qquad \text{with} \qquad p(A_{\text{train}}(x)) = \mathcal{N}(A_{\text{true}}(x), \sigma^2_{\text{train}}(x)) \; . \tag{52}$$

If the network were allowed to overfit perfectly, the outcome would be $A_{\text{NN}}(x) = A_{\text{train}}(x)$. In that case, $\sigma(x)$ is not needed for the loss minimization and hence not learned at all. If we keep the network from overtraining, the best the network can learn from unbiased data is

$$A_{\text{NN}}(x) \approx A_{\text{true}}(x) \qquad \text{and} \qquad \sigma(x) \approx \sigma_{\text{train}}(x) \; . \tag{53}$$

In that case, we can define a pull function over phase space as

$$t_{\text{het}}(x) = \frac{A_{\text{NN}}(x) - A_{\text{train}}(x)}{\sigma(x)} \; . \tag{54}$$

It follows a unit Gaussian when the input noise is learned as the network uncertainty. Note that the pull compares the learned function with the training data, not with the idealized data in the limit of zero noise.

Systematic pull

We can translate this result for a deterministic network with a heteroscedastic loss to the BNN and variational inference in the limit $q(\theta) = \delta(\theta - \theta_0)$,

$$\langle A \rangle(x) = \int d\theta dA \; A \; p(A|x,\theta) \; q(\theta) = \int dA \; A \; p(A|x,\theta_0) \; . \tag{55}$$

The network is trained well if

$$p(A|x,\theta_0) \approx p(A|x) \; . \tag{56}$$

In the Gaussian case, this is equivalent to learning the mean and the width

$$\langle A\rangle(x) \approx A_{\text{true}}(x) \qquad \text{and} \qquad \sigma_{\text{syst}}(x) \approx \sigma_{\text{train}}(x)\,. \tag{57}$$

As argued above, the systematic pull relates the learned $A(x)$ to the actual and potentially noisy training data $A_{\text{train}}(x)$,

$$\begin{aligned} t_{\text{syst}}(x) &= \frac{1}{\sigma_{\text{syst}}(x)} \int dA\ \left[A - A_{\text{train}}(x)\right]\ p(A|x,\theta_0) \\ &= \frac{1}{\sigma_{\text{syst}}(x)} \left[\int dA\ A\ p(A|x,\theta_0) - A_{\text{train}}(x) \int dA\ p(A|x,\theta_0)\right] \\ &= \frac{\langle A\rangle(x) - A_{\text{train}}(x)}{\sigma_{\text{syst}}(x)}\,. \end{aligned} \tag{58}$$

Notably, we do not have to approximate the integral in the definition of $t_{\text{syst}}(x)$ since the network directly predicts the parameters of $p(A|x,\theta_0)$.

Statistical pull

To define a pull to test the calibration of the statistical uncertainty, we remind ourselves that we need to extract the statistical uncertainties defined in Eq.(43) using the unbiased sample variance from N amplitudes sampled from the posterior weight distributions, $\theta_i \sim q(\theta)$,

$$\begin{aligned} \langle A\rangle(x) &\approx \frac{1}{N} \sum_i\ \overline{A}(x,\theta_i) \\ \sigma^2_{\text{stat}}(x) &\approx \frac{1}{N-1} \sum_i\ \left[\overline{A}(x,\theta_i) - \langle A\rangle(x)\right]^2\,. \end{aligned} \tag{59}$$

This definition provides us with the pull distribution

$$\hat{t}_{\text{stat}}(x,\theta) = \frac{\overline{A}(x,\theta) - \langle A\rangle(x)}{\sigma_{\text{stat}}(x)}\,. \tag{60}$$

It samples amplitudes from the posterior, and it calculates the deviation from the expectation value $\langle A\rangle$ and the uncertainty $\sigma_{\text{stat}}(x)$, both computed by sampling the θ. This pull is guaranteed to follow a standard Gaussian for large N, as $\sigma_{\text{stat}}(x)$ is calculated from the variance of $\overline{A}(x,\theta)$. Under the assumption of perfect training, we can replace the expectation value with its learning target, $\langle A\rangle(x) \approx A_{\text{true}}(x)$, and define the pull

$$t_{\text{stat}}(x,\theta) = \frac{\overline{A}(x,\theta) - A_{\text{true}}(x)}{\sigma_{\text{stat}}(x)}\,. \tag{61}$$

This pull and its averaged scaled counterpart should converge to a standard Gaussian distribution in the limit of a large number of samples. However, the rate of convergence to this distribution depends on the structure of the approximate posterior $q(\theta)$ and, e.g. in cases of multi-modal posterior, it may require an unfeasible large number of samples to converge.

Predictions

To estimate the performance of amplitude networks we can use the fact that regression is a supervised task, which means we know the true amplitudes. For the training or test data we can compute the relative accuracy

$$\Delta_j = \frac{\langle A \rangle_j - A_j}{A_j} \tag{62}$$

A single number which can tell us how well the model is fitting the data is the *negative log-likelihood*. Given samples from the posterior, we calculate the best mean value using the MC mean estimate.

A lower NLL indicates better agreement between the (training) data and the model fitted with the neural network.

$$\text{NLL} = \frac{\left(\langle A \rangle - A^{\text{truth}}\right)^2}{2\sigma^2_{\text{syst}}} + \log \sigma_{\text{syst}} \ . \tag{63}$$

References

[1] Y. Gal, *Uncertainty in Deep Learning*, Ph.D. thesis, Cambridge (2016).

[2] T. Plehn, A. Butter, B. Dillon, T. Heimel, C. Krause and R. Winterhalder, *Modern machine learning for lhc physicists* (2025), arXiv:2211.01421.

[3] B. Lakshminarayanan, A. Pritzel and C. Blundell, *Simple and scalable predictive uncertainty estimation using deep ensembles* (2017), arXiv:1612.01474.

[4] F. D'Angelo and V. Fortuin, *Repulsive deep ensembles are bayesian* (2023), arXiv:2106.11642.

[5] V. D. Wild, S. Ghalebikesabi, D. Sejdinovic and J. Knoblauch, *A rigorous link between deep ensembles and (variational) bayesian methods* (2023), arXiv:2305.15027.