UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

Bayesian Statistics and Probabilistic Programming

COMPARISON AND APPLICATION OF BAYESIAN NEURAL NETWORKS AND CLASSICAL NEURAL NETWORKS

Sara Bardají Serra
Sergi Bech Sala
David Rosado Rodríguez

Barcelona, June 12, 2023

Abstract

In this work, we delve into the exploration and comparison of two powerful machine learning models: neural networks and Bayesian neural networks. We begin by providing a comprehensive overview of both approaches, elucidating their underlying principles and architectures. Neural networks, known for their ability to learn complex patterns and relationships, rely on deterministic weights and biases to make predictions. On the other hand, Bayesian neural networks embrace uncertainty by incorporating probabilistic modeling, enabling them to generate posterior distributions of model parameters and predictions. To evaluate their performance, we conduct an empirical study using the wine quality dataset. By implementing and training both models on this dataset, we analyze their respective accuracies, robustness, and ability to handle uncertainty.

Contents

1 Introduction

Neural networks have revolutionized various fields by enabling remarkable advancements in machine learning and artificial intelligence. However, a significant limitation of traditional neural networks lies in their inherent inability to understand and quantify uncertainty in their predictions. This limitation can hinder decision-making, particularly in domains where uncertainty assessment is critical, such as medical diagnosis, autonomous driving, or financial risk management.

To overcome this challenge, Bayesian neural networks (BNNs) offer a powerful approach that incorporates probabilistic modeling and inference techniques into the neural network framework. By leveraging Bayesian principles, BNNs provide a robust framework to understand and quantify uncertainty, going beyond the point estimates produced by conventional neural networks.

In general terms (see section 3.1 for detailed information), utilizing a BNN offers multiple advantages. Firstly, BNNs enable the capture of both epistemic and aleatoric uncertainties, providing insights into the confidence or uncertainty associated with the model's predictions. Secondly, BNNs promote model robustness by considering a distribution of models instead of relying on a single point estimate, mitigating the risk of overfitting. Lastly, Bayesian modeling with BNNs facilitates principled decision-making by providing not only predictions but also confidence intervals or posterior distributions. These additional measures assist in risk assessment and support more well-informed choices.

Overall, BNNs offer the benefits of uncertainty quantification, enhanced model robustness, and improved decision-making, making them a valuable tool in various domains. Nevertheless, it is important to acknowledge that there are some challenges associated with utilizing BNNs. One notable challenge is the requirement for a strong background in mathematics to fully comprehend and work with BNNs effectively. The underlying principles of Bayesian inference and the computation of posterior distributions demand a solid understanding of advanced mathematical concepts. Furthermore, computing the exact posterior distribution of network weights in BNNs is often infeasible, particularly for complex models. This limitation necessitates the use of Bayesian inference techniques such as Monte Carlo sampling or Markov chain Monte Carlo methods to approximate the posterior distribution. While these methods provide valuable approximations, they can be computationally intensive and time-consuming.

The project aims to initially present the foundational concepts of neural networks, with a primary focus on the Multi-Layer Perceptron (MLP) network. Additionally, we will introduce and define the notation to be utilized throughout this chapter. Subsequently, a detailed explanation of the structure and functionality of BNNs will be provided, along with highlighting their key distinctions, advantages, and disadvantages compared to previous models. The theoretical section will conclude by discussing several Bayesian Inference Algorithms employed for estimating the posterior distribution of the network's parameters. Lastly, we will construct a concise BNN and employ it to tackle a regression task involving the prediction of wine quality using its attributes and characteristics.

2 Background

In this work, we assume that the reader possesses a foundational understanding of Bayesian statistics, encompassing key concepts such as prior and posterior distributions as well as Bayes' theorem. For readers seeking a more comprehensive understanding of Bayesian statistics, we highly recommend readers to check [DP15]. Subsequently, we are going to introduce and explore Neural networks, studying their inner workings and mechanisms. A thorough understanding of neural networks is crucial for comprehending the subsequent discussion on Bayesian neural networks.

2.1 Neural Networks

Neural networks (NNs) are a class of machine learning models inspired by the structure and function of the human brain. They are designed to recognize patterns and make predictions based on input data. One commonly used type of NN is the Multi-Layer Perceptron (MLP) network. At its core, a MLP network consists of multiple layers of interconnected artificial neurons, known as perceptrons. Each perceptron takes in one or more inputs, applies a mathematical transformation to them, and produces an output. The output of one perceptron serves as an input to the next, forming a chain-like structure.

The MLP network typically has three types of layers: an input layer, one or more hidden layers, and an output layer. The input layer receives the initial data, which could be a set of numerical values or features extracted from raw data. The hidden layers perform computations on the inputs, transforming them in a way that allows the network to learn complex relationships. Finally, the output layer produces the network's prediction or classification based on the processed information. An example of a MLP is shown in Figure 1.

Each connection between perceptrons in adjacent layers (arrows) is associated with a weight value. The weights determine the strength and direction of the connection, effectively controlling how much influence one perceptron has on another. During the training process, the network adjusts these weights to optimize its performance.

We can express a general NN with one hidden layer in terms of the input array $\mathbf{x} = (x_1, \ldots, x_{N_1})$ as,

$$h_j = \sigma_h \left(\sum_{i=1}^{N_1} x_i w_{ij}^{(1)} + b_1 \right), \tag{2.1}$$

$$f_k = \sigma_f \left(\sum_{i=1}^{N_2} h_j w_{jk}^{(2)} + b_2 \right). \tag{2.2}$$

where the parameters ω represents the weights of each connection, σ_h and σ_f are activation functions and the values b_i represents the bias parameters of the NN [GF20].

The activation function introduces non-linearity terms to the network, allowing it to model complex relationships between inputs and outputs. Without activation functions, a neural network would reduce to a linear function, as the sum of linear operations remains linear. There are several commonly used activation functions in neural networks, some of

them are: the Sigmoid, Rectified Linear Unit (ReLU), Hyperbolic Tangent (Tanh) and the Gaussian Error Linear Unit (GELU). Figure 2 shows these functions. The bias of a neural network refers to a parameter that allows the network to shift the activation function's curve. It is an additional input to each neuron, typically represented as a constant value. The bias term helps the network learn and approximate complex functions more effectively by allowing the activation function to be shifted horizontally.

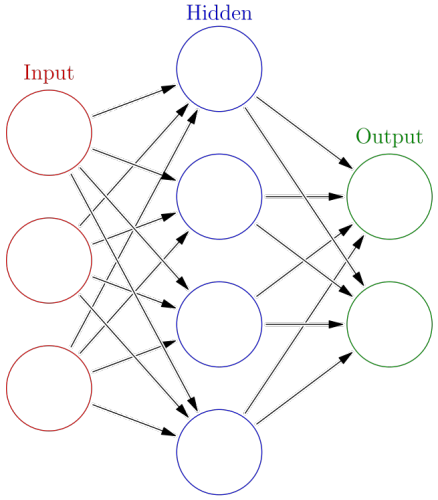


Figure 1: Example of a NN architecture with a single hidden layer. Each node represents a neuron, that processes and transmits information, while the arrows, also known as edges, represent the connections between neurons.

Equation (2.1) represents the output of the hidden layer, which will be of dimension N_2, while equation (2.2), shows the final output of the NN. The last activation function in a neural network, σ_f, often referred to as the output activation function, determines the format or interpretation of the network's final prediction or output. The choice of an activation function depends on the nature of the problem being solved and the desired properties of the output. For example, the identity activation function ($\sigma(x) = x$) is commonly used in regression problems where the network aims to predict a continuous value, while the Sigmoid activation function is often used for binary classification problems, where the network's output represents the probability of belonging to a certain class.

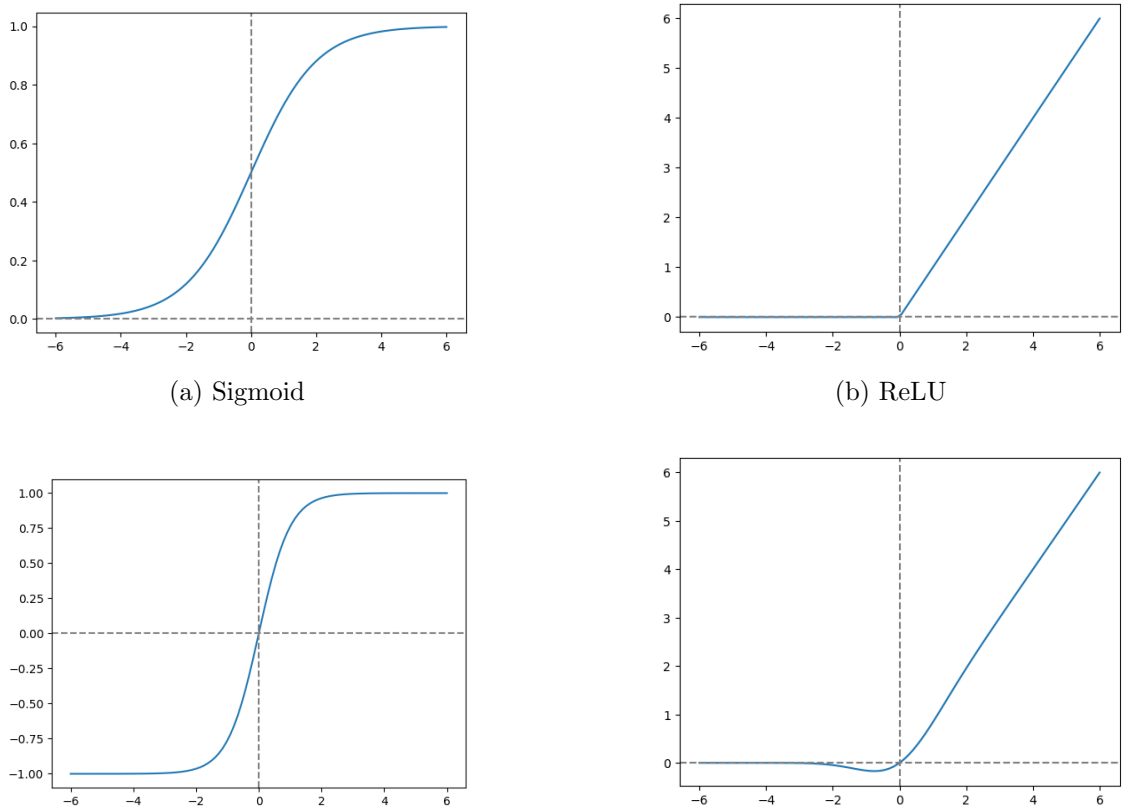
In the Machine Learning literature, equations (2.1) and (2.2) are often shown using matrix representation. This is achieved by stacking the input vector in our data set as a column in $\mathbf{X}$, i.e.,

$$H = \sigma_h\left(W^{(1)}\mathbf{X} + b_1\right),$$

$$F = \sigma_f\left(W^{(2)}\mathbf{H} + b_2\right).$$

During the learning process of a NN, a loss function plays a critical role in determining how well the network is performing. The loss function measures the discrepancy between the predicted output of the network and the actual expected output, quantifying the network's performance on a given task. By evaluating the discrepancy between predicted

and target outputs, the loss function serves as a guide for the network to adjust its internal parameters (weights and biases) during the process known as backpropagation. This optimization process aims to find the optimal set of parameters that minimizes the overall error of the network's predictions. The choice of an appropriate loss function depends on the specific problem being addressed and the desired outcome of the network.



(a) Sigmoid

(b) ReLU

(c) Tanh

(d) GELU

Figure 2: Example of commonly used activation functions in NNs.

Different types of loss functions are used depending on the nature of the problem. For example, the Mean Squared Error (MSE) loss, commonly used for regression tasks or the Cross-Entropy (CE) loss, frequently used for classification tasks. Mathematically,

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \qquad \text{CE} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i).$$

where y_i refers to the target (ground truth) for sample i and $\hat{y}_i$ refers to the model prediction.

In conclusion, this introductory chapter has provided a concise overview of neural networks. We have touched upon the fundamental concepts and principles behind neural networks, highlighting their ability to model complex relationships and make predictions. However, this is merely a glimpse into the vast field of neural networks, and there is much more to explore. For a deeper understanding and comprehensive insights, we encourage readers to check [H+95] for a theoretical view of neural networks and [ZLLS21] for a more practical view and to learn about modern/recent neural networks and other architectures.

3 Bayesian Neural Networks

Bayesian neural networks (BNNs) are a variant of traditional neural networks that incorporate Bayesian inference techniques. While traditional neural networks assign fixed weights and biases to the network's connections, BNNs introduce uncertainty by representing these weights and biases as probability distributions.

In a BNN, instead of having fixed values for the weights and biases, they are treated as random variables, which is reasonable since we do not know the values of the weights or biases. Actually, a common approach in Bayesian statistics is to treat unknown parameters as random variables with the goal of learning a distribution of these parameters conditional on what we can observe in the training data [GF20]. By representing the weights and biases as probability distributions, BNNs can provide not only point estimates, but also quantify the uncertainty associated with those estimates. Figure 3 portrays an illustration that highlights the graphical disparity between a NN and a BNN.

Beyond the lack of explainability that traditional neural networks give, there exists several techniques that tries to mitigate this issue, being stochastic neural networks one of the most generic ones. Stochastic neural networks are a type of NNs that incorporate randomness into their architecture or training process. This is achieved by giving the network either a stochastic activation or stochastic weights to simulate multiple possible models θ with their associated probability distribution $p(\theta)$. Thus, BNNs can be considered a special case of ensemble learning [JLB+22].

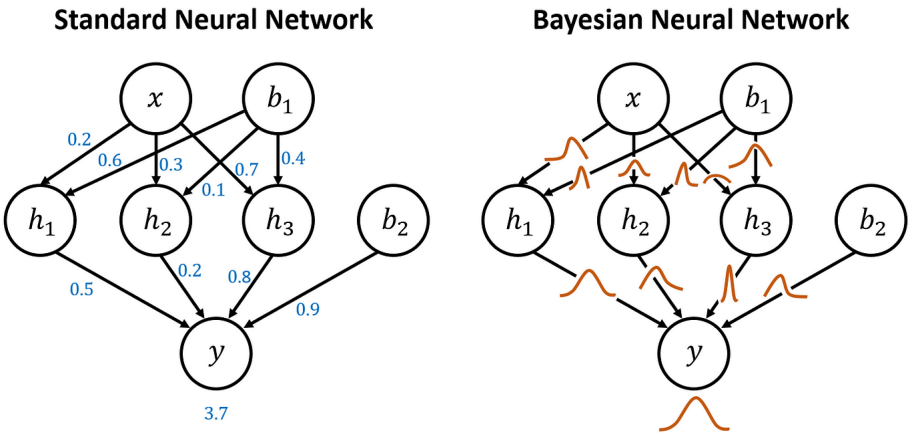


Figure 3: Comparison of a standard NN and a BNN.

Let us explain in detail how a BNN works. For this purpose, let us call $\mathcal{D}$ the set of training data. Before training, the joint distribution $p(\omega, \mathcal{D})$ captures the interrelationship between our data and the weights. This joint distribution is shaped by our prior assumptions regarding the latent variables $p(\omega)$ and our selected model/likelihood $p(\mathcal{D}|\omega)$,

$$p(\omega, \mathcal{D}) = p(\omega)p(\mathcal{D}|\omega).$$

The likelihood function strongly depends on the problem to be solved. For instance, the Gaussian likelihood is often used for regression tasks when the target variable follows a

normal distribution (this is equivalent to using the MSE) while the Bernoulli likelihood is used for binary classification tasks, where the target variable takes only two values (e.g., 0 or 1)[GF20]. At present, let us consider that the prior and the likelihood have been specified. By using Bayes theorem, the Bayesian posterior distribution can be written as:

$$p(w|\mathcal{D}) = \frac{p(\omega)p(\mathcal{D}|\omega)}{\int p(\omega)p(\mathcal{D}|\omega)d\omega} = \frac{p(\omega)p(\mathcal{D}|\omega)}{p(\mathcal{D})}. \tag{3.1}$$

The denominator $p(\mathcal{D})$ is the evidence, also known as the marginal likelihood. It acts as a normalizing constant and ensures that the posterior distribution integrates to 1.

The posterior distribution provides valuable information about the uncertainty associated with the model's predictions. Firstly, by sampling from the posterior distribution, we can obtain a distribution of possible predictions rather than a single point estimate. This allows us to assess the confidence or uncertainty associated with the model's predictions, which can be crucial in decision-making processes. Moreover, the posterior distribution allows the construction of prediction intervals by considering percentiles or credible intervals of the posterior distribution, leading to capture the range of plausible values with a given confidence level. Finally, we can perform predictions of any quantity of interest. These predictions take the form of an expectation derived from the posterior distribution [GF20], i.e.,

$$\mathbb{E}_p[f] = \int f(\omega)p(\omega|\mathcal{D})d\omega.$$

Let us make a theoretical example of a BNN. Consider a prior distribution for the weights of the following form:

$$p(\omega) = \frac{\exp(-\alpha E_\omega)}{\int \exp(-\alpha E_\omega)d\omega}, \qquad E_\omega = \frac{1}{2}\sum_{i=1}^{W} w_i^2,$$

where α is a hyperparameter[1]. Assume also that we are dealing with a regression task, thus, the likelihood function has the following form:

$$p(\mathcal{D}|\omega) = \frac{\exp(-\beta E_\mathcal{D})}{\int \exp(-\beta E_\mathcal{D})d\mathcal{D}},$$

where β is a hyperparameter and

$$\int d\mathcal{D} = \int dt^{(1)} \cdots dt^{(N)}, \quad t^{(i)} \in \mathcal{D} \;\; i = 1, \ldots, N.$$

Assuming that all samples from $\mathcal{D}$ are independent and identically distributed (i.i.d.), the likelihood can then be written as a product of the contribution from the N individual terms in the data set,

$$p(\mathcal{D}|\omega) = \prod_{i=1}^{N} p(t^{(i)}|\omega) = \frac{\exp(-\frac{\beta}{2}\sum_{i=1}^{N}(t^{(i)} - \mu)}{\int \exp(-\beta E_\mathcal{D})d\mathcal{D}},$$

[1]Notice that, essentially, the prior distribution is a standard normal distribution.

where μ is the mean of the normal distribution. According to equation (3.1), the posterior distribution has the following form:

$$p(\omega|\mathcal{D}) = \frac{p(\omega)p(\mathcal{D}|\omega)}{p(\mathcal{D})} = \frac{\exp(-\alpha E_\omega)\exp(-\beta E_\mathcal{D})}{p(\mathcal{D})} = \frac{\exp(-S(\omega))}{\int \exp(-S(\omega))d\omega},$$

where

$$S(\omega) = \alpha E_\omega + \beta E_\mathcal{D}.$$

Due to the complexity of the model, the high-dimensional parameter space and the complex likelihood function, computing the posterior distribution analytically is generally infeasible. For such cases, approximate methods are commonly used to estimate the posterior distribution. In section 4 we will see in detail the most popular techniques to achieve Bayesian inference.

3.1 Benefits and Drawbacks

As we have observed, the performance of BNN greatly relies on prior knowledge. This becomes especially valuable when confronted with limited data, as prior knowledge can effectively steer the model's learning process. Nevertheless, attaining any meaningful understanding regarding suitable parameterization for a given model prior to training poses a significant challenge, thus forming the primary criticism of BNN [JLB+22].

BNNs offer a structured framework for estimating prediction uncertainty, which traditional NNs lack. By incorporating Bayesian inference, BNNs can provide probabilistic predictions, allowing for better decision-making under uncertainty. In contrast, NNs have inherent limitations in addressing this aspect, leading to the well-known term "black box".

Furthermore, a BNN allows distinguishing between the epistemic uncertainty $p(w|\mathcal{D})$ and the aleatoric uncertainty $p(\mathcal{D}|\omega)$. This leads to BNN being more resistant to overfitting compared to traditional NNs. The Bayesian framework naturally includes regularization, which helps prevent overfitting and improves generalization [JLB+22].

A potential drawback for BNN is that, due to the need for approximate inference methods for computing the posterior distribution, BNNs can be computationally expensive to train and evaluate. This complexity increases with the size of the model and the amount of data. We cannot leave aside that, dealing with BNN requires a strong background in mathematics compared to traditional NNs. Understanding the underlying Bayesian framework and the mathematical concepts involved can be more challenging, especially for practitioners who are less familiar with probabilistic modeling and inference techniques.

NNs offer several notable advantages, including their remarkable flexibility, scalability, empirical track record, and ease of implementation. Firstly, NNs are highly flexible and can model complex relationships in data, making them suitable for a wide range of tasks. They can handle large-scale datasets and benefit from advancements in parallel computing, enabling efficient training on powerful hardware. Moreover, NNs have achieved remarkable success in various domains, such as computer vision, natural language processing or reinforcement learning, showing state-of-the-art performance in many applications.

Finally, NN frameworks and libraries, such as TensorFlow and PyTorch in Python, have made it easier to implement and train NN models. These frameworks provide a wide range of pre-built layers, optimizers, and other tools, reducing the implementation burden for practitioners.

In summary, BNN offer benefits such as uncertainty estimation and robustness to overfitting but come with challenges related to computational complexity and mathematical knowledge. Traditional NN provide flexibility, empirical success, and ease of implementation but lack inherent uncertainty quantification and can be prone to overfitting. The choice between BNNs and NNs depends on the specific requirements of the task at hand and the trade-offs that best align with those requirements.

4 Bayesian Inference Algorithms

In this section, we delve into the realm of Bayesian inference, aiming to provide a comprehensive overview of different Bayesian inference methods. We will explore various techniques that have emerged to address diverse challenges and scenarios encountered in real-world problems. By examining these methods, we aim to equip readers with a deeper understanding of the richness and versatility of Bayesian inference.

4.1 Monte Carlo Methods

Monte Carlo methods are computational techniques that provide approximate solutions to complex mathematical problems by utilizing random sampling. In the context of Bayesian inference, where exact solutions are often intractable, Monte Carlo methods play a crucial role in numerically approximating posterior distributions.

One particularly important method is Markov chain Monte Carlo (MCMC). Considering that a Markov chain is a sequence of random variables or states where each state depends only on the previous state, MCMC methods construct a Markov chain such that its stationary distribution is the target distribution of interest, which is typically the posterior distribution in Bayesian inference. Most MCMC algorithms require an initial burn-in time before the Markov chain converges to the desired distribution, meaning that there is an initial phase in which the chain is still far from convergence and is not yet producing samples that accurately represent the target distribution unlike in standard sampling methods. Therefore, the samples produced during this burn-in time must be discarded from the distribution, since they could skew the approximated target distribution.

While there is a wide range of MCMC algorithms available, their suitability for Bayesian inference depends on several factors, including the characteristics of the target distribution and the specific requirements of the Bayesian model. Two of the most popular MCMC methods for BNNs are the Metropolis-Hastings algorithm and the Hamiltonian Monte Carlo.

4.1.1 Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is particularly relevant for BNNs due to its flexibility and ability to sample from distributions even when the exact probability distribution is unknown.

As shown in Algorithm 1, the algorithm has two parameters of free choice: one is the number of iterations that are needed to overcome the correlation of states in the chain and the other is the proposal distribution that determines how new candidate states are generated given the current state. Note that if the proposal distribution is chosen poorly, this can result in highly inefficient sampling [CPLL20].

The algorithm starts by initializing the starting state θ_0 by drawing a sample from the initial probability distribution. It then enters a loop that continues until the desired number of iterations, N, is reached. Within each iteration, a new candidate state θ' is proposed

by sampling from the proposal distribution $Q(\theta'|\theta_n)$. It then calculates an acceptance ratio p based on the proposal distribution and the target distribution $f(\theta)$, where $f(\theta')$ and $f(\theta_n)$ are the densities of the candidate state and the current state, respectively. A Bernoulli random variable k is generated using this acceptance ratio, determining whether the candidate state is accepted or rejected. If k is equal to 1 (i.e., the candidate state is accepted), the algorithm updates the state to θ', storing it as θ_{n+1}, and increments the iteration counter n. This process continues until the desired number of iterations is reached.

Overall, the Metropolis-Hastings algorithm iteratively proposes new states, evaluates their acceptance based on an acceptance ratio, and updates the state accordingly. Through this process, it generates a Markov chain of states that asymptotically converges to samples from the target distribution.

Algorithm 1 Metropolis-Hastings

Input: Number of iterations N, proposal distribution Q
Output: Samples $\theta_1, \theta_2, \ldots, \theta_N$ from the target distribution

Draw $\theta_0 \sim$ Initial probability distribution ▷ Initialize the starting state
while $n \leq N$ **do** ▷ Iterate until reaching the desired number of iterations
  Draw $\theta' \sim Q(\theta'|\theta_n)$ ▷ Propose a new candidate state
  $p \leftarrow \min\left(1, \frac{Q(\theta'|\theta_n)}{Q(\theta_n|\theta')} \frac{f(\theta')}{f(\theta_n)}\right)$ ▷ Calculate the acceptance ratio
  Draw $k \sim$ Bernoulli(p) ▷ Generate random number to accept or reject the candidate state
  if k **then** ▷ Accept the candidate state if k = 1
    $\theta_{n+1} = \theta'$ ▷ Update the state to the candidate state
    $n = n + 1$ ▷ Increment the iteration counter
  end if
end while

The Metropolis-Hastings algorithm offers several advantages, including its flexibility to sample from complex distributions and its ability to handle situations where computing the exact normalization constant is challenging. It is relatively easy to implement and, under appropriate conditions, guarantees convergence to the target distribution. However, the algorithm also has some limitations. The generated samples may exhibit autocorrelation, resulting in slower exploration of the parameter space. Choosing an appropriate proposal distribution can be challenging, as a small spread may lead to a high rejection rate, while a large spread can be inefficient. Additionally, for high-dimensional or complex target distributions, the computational efficiency of the Metropolis-Hastings algorithm may be a concern, and alternative methods may be more suitable.

4.1.2 Hamiltonian Monte Carlo

The Hamiltonian Monte Carlo (HMC) algorithm is a powerful and widely used method for sampling from complex probability distributions. It is particularly effective in high-dimensional spaces where other sampling techniques may struggle. The key idea behind HMC is to introduce a fictitious auxiliary "momentum" variable that allows for efficient exploration of the target distribution by simulating the dynamics of a physical system.

Unlike traditional Markov chain Monte Carlo (MCMC) algorithms, which perform random local moves in the parameter space, HMC leverages the principles of Hamiltonian mechanics to guide the exploration. By treating the target distribution as a potential energy function, and the momentum variable as kinetic energy, HMC simulates the evolution of a particle in a higher-dimensional space. Algorithm 2 presents a single iteration

of the HMC method [Nea11].

Algorithm 2 Single-Iteration of Hamiltonian Monte Carlo (HMC)

Input: U a function that returns the potential energy given a state q, $grad_U$ a function that returns the gradient of U given q, ϵ the step size, L the number of Leapfrog steps, $current_q$ the current position from which the trajectory starts.

Output: Next state of the chain.

function HMC(U, $grad_U$, ϵ, L, $current_q$)
  $q \leftarrow current_q$
  $p \leftarrow \text{rnorm}(\text{length}(q), 0, 1)$ ▷ Independent standard normal variates
  $current_p \leftarrow p$
  $p \leftarrow p - \frac{\epsilon}{2} \cdot grad_U(q)$ ▷ Half step for momentum
  for $i \leftarrow 1$ to L **do**
    $q \leftarrow q + \epsilon \cdot p$ ▷ Full step for position
    if $i \neq L$ **then**
      $p \leftarrow p - \epsilon \cdot grad_U(q)$ ▷ Full step for momentum (except at end)
    end if
  end for
  $p \leftarrow p - \frac{\epsilon}{2} \cdot \text{grad\_U}(q)$ ▷ Half step for momentum at the end
  $p \leftarrow -p$ ▷ Negate momentum to make proposal symmetric
  $current_U \leftarrow U(current_q)$
  $current_K \leftarrow \sum p^2/2$
  $proposed_U \leftarrow U(q)$
  $proposed_K \leftarrow \sum p^2/2$
  if $runif(1) < \exp(current_U - proposed_U + current_K - proposed_K)$ **then**
    return q ▷ Accept
  else
    return $current_q$ ▷ Reject
  end if
end function

However, the HMC method also has some limitations. Firstly, it requires the evaluation of gradients, which can be computationally expensive, especially for complex models with a large number of parameters. The efficiency of HMC is highly dependent on the availability of accurate and efficient gradient computations. Additionally, HMC may face challenges when dealing with multimodal distributions or regions of high curvature, as the algorithm's dynamics can become sensitive to the shape of the target distribution. Careful tuning of the algorithm's parameters is crucial to ensure optimal performance, and inadequate parameter choices can lead to poor exploration and low acceptance rates. Lastly, the performance of HMC can be affected by the presence of strong correlations between variables, which can lead to slow exploration and inefficient sampling.

4.2 Variational Inference

Variational Inference is a family of techniques used in Bayesian statistics and machine learning to approximate complex posterior distributions. It provides a computationally efficient alternative to traditional Markov Chain Monte Carlo (MCMC) methods for inference in probabilistic models [CPLL20].

The main idea behind variational inference is to formulate the problem of approximating the posterior distribution as an optimization problem. It involves introducing a family of simpler distributions, called variational distributions, and finding the member of that family that is closest to the true posterior distribution. This closeness is measured using a divergence measure, such as the Kullback-Leibler (KL) divergence [JLB+22]. The Kullback-Leibler (KL) divergence between two probability distributions P and Q is typically defined as follows:

$$D_{KL}(P\|Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right),$$

or in the case of continuous distributions:

$$D_{KL}(P\|Q) = \int P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx.$$

In the formulas above, $P(x)$ and $Q(x)$ represent the probability densities (or probability mass functions) of the distributions P and Q, respectively. The KL divergence measures the dissimilarity between these two distributions and in our context it is employed to quantify the difference between the approximate posterior distribution and the true posterior distribution. It serves as a measure of how well the approximation captures the true distribution.

The optimization problem is framed as minimizing the KL divergence between the variational distribution and the true posterior. However, since this quantity can't actually be minimized, but we can optimize a function that is equal to it up to a constant [TL17]. This function is known as the evidence lower bound (ELBO) and finding an approximation that maximizes this function is equivalent to finding an approximation that minimizes the KL-divergence.

Variational inference allows for efficient computation of the posterior distribution by transforming the complex inference problem into an optimization problem. It provides fast and scalable approximations of posterior distributions, making it suitable for large datasets and complex models. However, variational inference relies on assumptions about the form of the variational family, and the quality of the approximation depends on the chosen family and the optimization algorithm. It may introduce biases in the estimates and can struggle with capturing complex dependencies and multimodal posteriors.

There exist a large variety of Variational Inference method, however in this project we will only comment on a few of them.

4.2.1 Mean-Field variation

Mean Field variation (MFV) assumes that the posterior distribution can be factorized into independent components, and each component is approximated by a simple distribution (e.g., Gaussian) as shown in equation 4.1. It minimizes the KL divergence between the approximate distribution and the true posterior. MFV is computationally efficient and widely used in practice.

$$q(z_1, \ldots, z_m) = \prod_{j=1}^{m} q(z_j). \tag{4.1}$$

Consider a Bayesian model with latent variables z and observed data x. The true posterior distribution is denoted as $p(z|x)$ and we want to approximate it with a variational distribution $q(z)$.

The mean-field assumption implies that each latent variable z_i has its own independent distribution $q(z_i)$, which is typically chosen from a tractable family of distributions, such as Gaussian, exponential, or gamma distributions. The parameters of these distributions are optimized to minimize the Kullback-Leibler (KL) divergence between the true posterior $p(z|x)$ and the approximate posterior $q(z)$. This provides a mean-field approximation of the target distribution [CPLL20].

In summary, mean-field variational inference is a technique that simplifies the approximation of the true posterior distribution by assuming independence between latent variables. While it has its limitations, mean-field variational inference provides a computationally efficient approach for approximating complex posterior distributions in many practical scenarios.

4.2.2 Bayes by Backprop

The key idea behind Bayes by Backprop is to approximate the true posterior distribution over the weights using a variational distribution. This approximation is achieved by introducing variational parameters that define the shape of the approximate distribution. The goal is to find the optimal values for these variational parameters that minimize the Kullback-Leibler (KL) divergence between the approximate posterior and the true posterior. Bayes by Backprop (Algorithm 3) provides a method for approximating a target distribution, using differentiable functions such as neural networks [CPLL20].

During the training phase, Bayes by Backprop utilizes the reparameterization trick, allowing for backpropagation through stochastic nodes. This involves reparameterizing the weights as a deterministic function of independent random variables and sampling from these variables to introduce stochasticity.

The training process of Bayes by Backprop involves two steps: the forward pass and the backward pass. In the forward pass, the network computes predictions based on sampled weights. In the backward pass, the gradients of the variational parameters are computed using the sampled weights and used to update the variational parameters through backpropagation. This process is repeated iteratively to optimize the variational parameters and approximate the posterior distribution, maximizing the ELBO.

The resulting posterior distribution obtained through Bayes by Backprop provides uncertainty estimates for predictions. By sampling multiple sets of weights from the posterior distribution, it is possible to obtain a distribution of predictions rather than a single point estimate. This distribution can be used to quantify uncertainty, make more robust decisions, and capture epistemic uncertainty, which represents uncertainty due to limited data.

Algorithm 3 Bayes by Backprop

$\phi = \phi_0$ ▷ Initialize variational parameters
for $i = 0$ to N **do** ▷ Iterate for N steps
  Draw $\varepsilon \sim q(\varepsilon)$ ▷ Sample from variational distribution
  $\theta = t(\varepsilon, \phi)$ ▷ Transform sample to obtain BNN weights
  $f(\theta, \phi) = \log(q_\phi(\theta)) - \log(p(D_y|D_x, \theta)p(\theta))$ ▷ Compute objective function
  $\Delta\phi_f = \text{backprop}_\phi(f)$ ▷ Compute gradients using backpropagation
  $\phi = \phi - \alpha\Delta\phi_f$ ▷ Update variational parameters using gradient descent
end for

5 Experiments

In this particular section, our focus will be on the wine quality dataset. We aim to construct both a BNN and a traditional NN to assess their respective performances and draw a comparative analysis of the outcomes. Let us start by explaining the dataset.

5.1 Dataset

The wine quality dataset from TensorFlow Datasets (TFDS) comprises data on white variants of Portuguese "Vinho Verde" wine. It contains $4,898$ instances of white wine. The dataset includes several physicochemical properties of the wines, such as acidity levels, pH, alcohol content, and more. The target variable is the quality rating of the wine, ranging from 0 to 10, obtained through sensory evaluation.

Prior to starting our study, we conducted an exploratory data analysis (EDA) to enhance our comprehension of the variables within our dataset. As all the variables are numerical, no encoding is required to prepare them for input into the models. We discovered that the dataset does not contain any null values. However, we did observe the presence of outliers. While it is possible to remove these outliers, our primary objective is not to achieve an exceptionally high score. Therefore, we made the decision to retain the outliers and proceed with the entire dataset. In Figure 4, we present the correlation matrix, which has provided us with significant insights:

- The quality of wine is positively correlated with its alcohol content.
- The pH value of wine shows a negative correlation with fixed acidity.
- Fixed acidity exhibit a weak correlation with citric acid in wine.
- Density is negative correlated to alcohol.

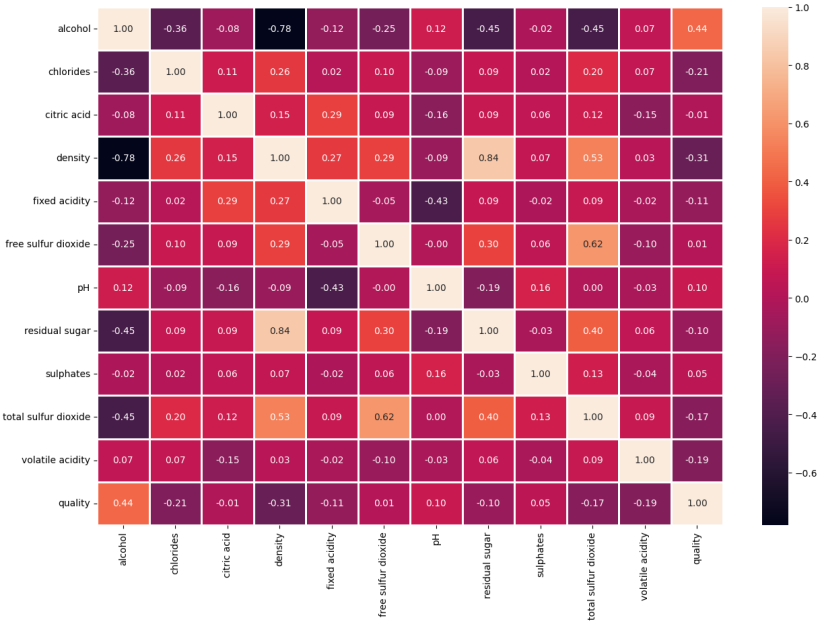


Figure 4: Correlation matrix of all variables from wine quality dataset.

The complete exploratory data analysis can be observed by reviewing the accompanying code for this project.

5.2 Methodology

The goal is to construct regression models that can predict wine quality using its physicochemical attributes. We will compare two approaches: standard neural networks with fully connected layers (such as MLP) and BNN. To ensure a fair comparison between the two methods, we split the dataset into training and testing sets and employ the same sets for both approaches.

For the standard neural networks approach, we construct a MLP model with multiple layers of densely connected neurons. The model it is trained using the training data and evaluated using MSE which is a common metric for regression tasks[2].

The BNN approach incorporates uncertainty estimation into predictions by modeling the weights of the neural network as probability distributions. We employ variational inference techniques, more specifically Bayes by backprop (see section 4.2.2), to train a Bayesian neural network on the wine quality dataset. In this approach, we define the prior weight distribution as a Normal distribution with a mean of 0 and a standard deviation of 1. It is worth noting that the prior distribution is not trainable in this particular example, as we fix its parameters without updating them during training.

To capture the variability in the weights, we specify the variational posterior weight distribution as a multivariate Gaussian distribution. The means, variances, and covariances of this distribution serve as learnable parameters. During the training process, these parameters are adjusted to optimize the model's performance.

The loss function used for our BNN is the negative log-likelihood. Minimizing this function encourages the model to generate predictions that are more likely to be observed in the training data, effectively capturing the aleatoric uncertainty. Additionally, by considering the posterior distribution over the model weights, the BNN can also capture epistemic uncertainty. So, by optimizing the negative log-likelihood, the network can effectively learn to generate predictions that not only fit the training data but also quantify the uncertainties associated with those predictions. This makes the model more robust and provides a more comprehensive understanding of the inherent uncertainties in the data.

5.3 Results

In order to assess the performance of both approaches, we utilize suitable metrics to evaluate the models on the test set. The results of this evaluation are presented in Table 1, where we observe that the MSE for both models is quite similar. It is important to note that for the BNN, the MSE is computed using the mean value for each sample prediction.

Furthermore, Figure 5 compares selected test predictions from both models and show-

[2]Check section 2.1.

cases the uncertainty modeling of the BNN. The actual predictions from both models are fairly similar, with the MLP model appearing slightly more accurate. However, the Bayesian predictions not only provide point estimates but also offer a confidence interval, allowing us to comprehend the uncertainty associated with the model. In this particular case, the uncertainty is relatively high, indicating that the model is not highly confident in its predictions.

Model	**MSE**	**Negative log-likelihood**
MLP	0.596	-
BNN	0.628	1.215

Table 1: A comparative analysis of the precision exhibited by the models.

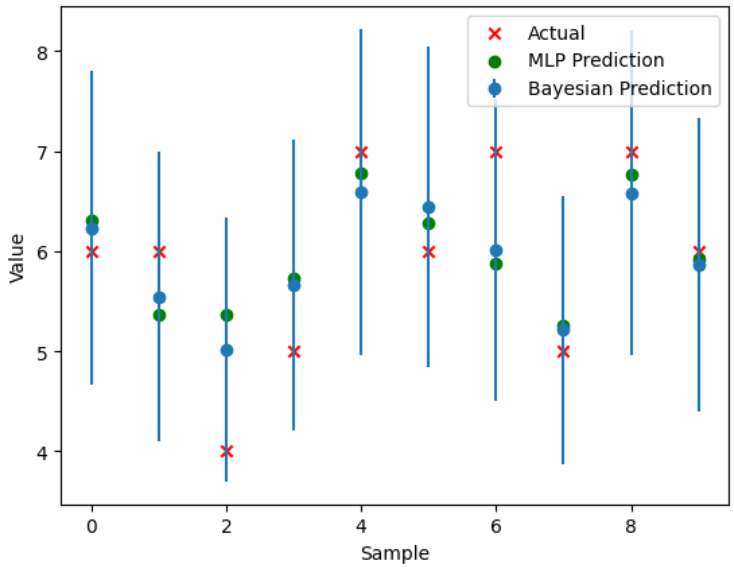


Figure 5: A comparison of the test predictions showing the uncertainty of the BNN.

6 Conclusions

In conclusion, this work has provided a detailed explanation BNNs and compared them with traditional NNs. We have explored the advantages and disadvantages of BNNs and examined various techniques to obtain the posterior distribution of BNNs using well-known inference methods.

By incorporating Bayesian principles into neural networks, BNNs offer several notable advantages. Firstly, BNNs provide a more robust estimation of uncertainty by considering the entire posterior distribution over weights, enabling better decision-making in uncertain scenarios. This uncertainty estimation is lacking in traditional NNs, which only provide point estimates. Secondly, BNNs offer automatic regularization, preventing overfitting and improving generalization by integrating prior knowledge about the model parameters. This regularization is particularly valuable in situations with limited data.

However, BNNs also come with certain challenges and drawbacks. The main disadvantage is the computational complexity involved in approximating the posterior distribution. The process of sampling or optimizing the posterior distribution can be time-consuming and resource-intensive, making BNNs less efficient compared to traditional NNs. Additionally, the requirement of a strong mathematical background can present a significant barrier for practitioners who are not familiar with advanced statistical concepts. It may limit the accessibility and adoption of BNNs, particularly in domains where practitioners are more focused on the application rather than the underlying mathematical foundations.

To address the challenge of obtaining the posterior distribution in BNNs, several inference techniques were discussed in this work. These techniques included variational inference and Markov chain Monte Carlo methods. Each method offers a trade-off between accuracy and computational efficiency, allowing practitioners to choose the most suitable technique based on the specific requirements of their problem.

Finally, to demonstrate the practical implementation and performance comparison, we coded both a traditional NN and a BNN. Through experimental evaluation, we observed that the BNN provided not only predictions but also associated uncertainties, allowing us to make more informed decisions.

In summary, Bayesian neural networks offer significant benefits over traditional neural networks by providing uncertainty estimation and automatic regularization. Although they come with computational challenges, various inference techniques can be employed to obtain the posterior distribution. The field of Bayesian neural networks continues to evolve, and further research in this area will likely yield more efficient and effective methods for uncertainty estimation and model optimization.

References

[CPLL20] Tom Charnock, Laurence Perreault-Levasseur, and François Lanusse. Bayesian neural networks, 2020.

[DP15] Cameron Davidson-Pilon. *Bayesian methods for hackers: probabilistic programming and Bayesian inference*. Addison-Wesley Professional, 2015.

[GF20] Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey. *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair, Fall 2018*, pages 45–87, 2020.

[H+95] Mohamad H Hassoun et al. *Fundamentals of artificial neural networks*. MIT press, 1995.

[JLB+22] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022.

[Nea11] Radford M. Neal and et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.

[TL17] Xupeng Tong and Minxing Liu. Lecture 13: Variational inference: Mean field approximation, 2017. Lecture notes.

[ZLLS21] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.