Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users

Laurent Valentin Jospin, University of Western Australia, Australia
Hamid Laga, Murdoch University, Australia
Farid Boussaid, University of Western Australia, Australia
Wray Buntine, Monash University, Australia
Mohammed Bennamoun, University of Western Australia, Australia

***Abstract*—Modern deep learning methods constitute incredibly powerful tools to tackle a myriad of challenging problems. However, since deep learning methods operate as black boxes, the uncertainty associated with their predictions is often challenging to quantify. Bayesian statistics offer a formalism to understand and quantify the uncertainty associated with deep neural network predictions. This tutorial provides deep learning practitioners with an overview of the relevant literature and a complete toolset to design, implement, train, use and evaluate Bayesian neural networks, *i.e.*, stochastic artificial neural networks trained using Bayesian methods.**

***Index Terms*—Bayesian methods, Bayesian Deep Learning, Bayesian neural networks, Approximate Bayesian methods**

I. Introduction

Deep learning has led to a revolution in machine learning, providing solutions to tackle problems that were traditionally difficult to solve. However, deep learning models are prone to overfitting, which adversely affects their generalization capabilities [1]. They also tend to be overconfident about their predictions when they provide a confidence interval. This is problematic for applications where silent failures can lead to dramatic outcomes, *e.g.*, autonomous driving [2], medical diagnosis [3] or finance [4]. Consequently, many approaches have been proposed to mitigate this risk [5]. Among them, the Bayesian paradigm provides a rigorous framework to analyze and train uncertainty-aware neural networks, and more generally, to support the development of learning algorithms.

The Bayesian paradigm in statistics contrasts with the frequentist paradigm, with a major area of distinction in hypothesis testing [6]. It is based on two simple ideas. The **first** is that probability is a measure of belief in the occurrence of events, rather than the limit in the frequency of occurrence when the number of samples goes toward infinity, as assumed in the frequentist paradigm. The **second** idea is that prior beliefs influence posterior beliefs. Bayes' theorem, which states that:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} = \frac{P(D,H)}{\int_H P(D,H')dH'}, \quad (1)$$

summarizes this interpretation. Formula (1) is still true in the frequentist interpretation, where H and D are considered as sets of outcomes. The Bayesian interpretation considers H to be a hypothesis about which one holds some prior belief, and D to be some data that will update one's belief about H. The probability distribution $P(D|H)$ is called the likelihood. It encodes the aleatoric uncertainty in the model, *i.e.*, the uncertainty due to the noise in the process. $P(H)$ is the prior and $P(D) = \int_H P(D,H')dH'$ the evidence. $P(H|D)$ is called the posterior. It encodes the epistemic uncertainty, *i.e.*, the uncertainty due to the lack of data. $P(D|H)P(H) = P(D,H)$ is the joint probability of D and H.

Corresponding author: Laurent Valentin Jospin (Email: laurent.jospin@research.uwa.edu.au)

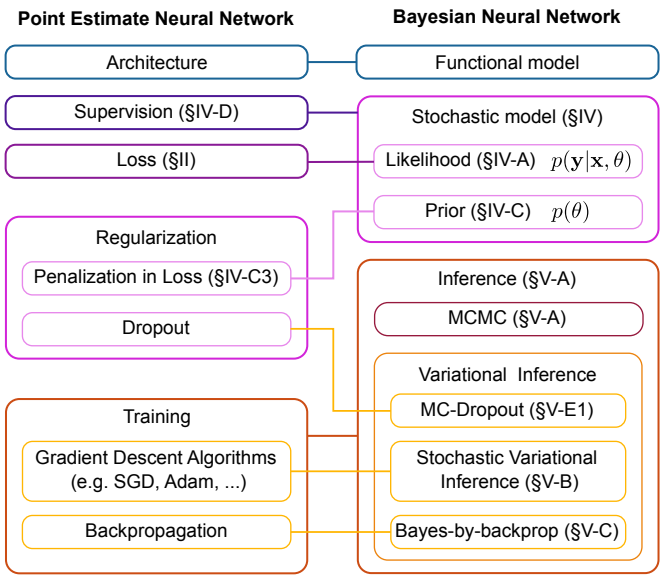


Fig. 1: Illustration of the correspondence between the concepts used in deep learning for point-estimate neural networks and their counterparts in Bayesian neural networks (BNNs).

Using Bayes' formula to train a predictor can be understood as learning from the data D. In other words, the Bayesian paradigm not only offers a solid approach for the quantification of uncertainty in deep learning models but also provides a mathematical framework to understand many regularization techniques and learning strategies that are already used in classic deep learning [7] (Section IV-C3).

Bayesian neural networks (BNNs) [8, 9, 10] are stochastic neural networks trained using a Bayesian approach. There is a rich literature about BNNs and the related field of Bayesian

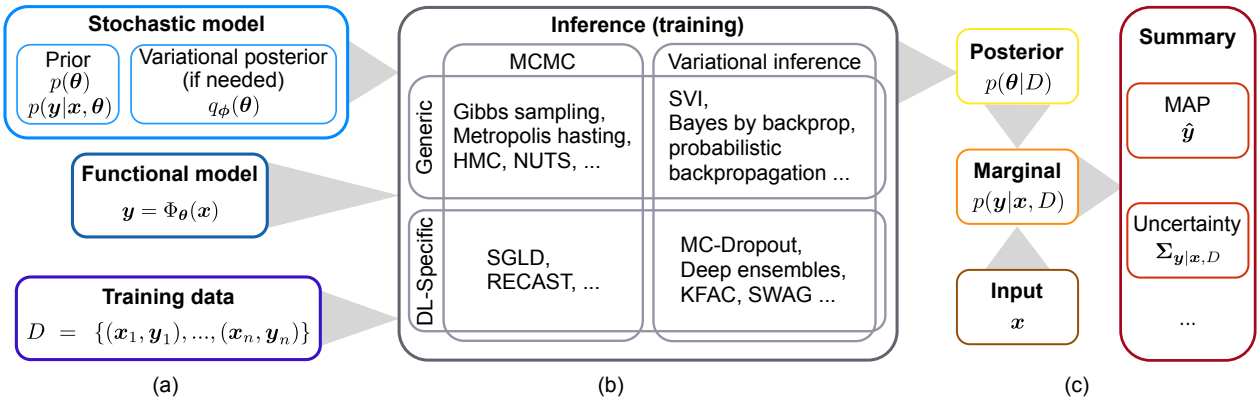


Fig. 2: Workflow to design (a), train (b) and use a BNN for predictions (c).

deep learning, which is referred to by Wang and Yeung [11] as the conjoint use of deep learning for perception and traditional Bayesian models for inference.[1] However, navigating through this literature is challenging without some prior background in Bayesian statistics. This brings an additional layer of complexity for deep learning practitioners interested in building and using BNNs.

This paper, conceived as a tutorial, presents a unified workflow to design, implement, train and evaluate a BNN (Figure 2). It also provides an overview of the relevant literature where a large number of approaches have been developed to efficiently train and use BNNs. A good knowledge of those different methods is a prerequisite for an efficient use of BNNs in big data applications of deep learning. In this tutorial, we assume that the reader is already familiar with the concepts of traditional deep learning such as artificial neural networks, training algorithms, supervision strategies, and loss functions [13]. This paper focuses on exploring the correspondences between traditional deep learning approaches and Bayesian methods (Figure 1). It is intended to motivate and help researchers and students to use BNNs in measuring uncertainty for problems in their respective fields of study and research, helping them relate their existing knowledge in deep learning to the relevant Bayesian methods.

The remaining parts of this paper are organized as follows. Section II introduces the concept of a BNN. Section III presents the motivations for BNNs as well as their applications. Section IV explains how to design the stochastic model associated with a BNN. Section V explores the most important algorithms used for Bayesian inference and how they were adapted for deep learning. Section VI reviews BNN simplification methods. Section VII presents the methods used to evaluate the performance of a BNN. Finally, Section VIII concludes the paper. The supplementary material contains a gallery of practical examples illustrating the theoretical concepts presented in Sections II, IV and V of the main paper. Each example source code is also available online on GitHub to provide implementation examples of the most important algorithms to work with BNNs.

[1]Note that some other authors use a different definition of Bayesian deep learning, which is closer to the idea of a BNN [12]).

II. What is a Bayesian Neural Network?

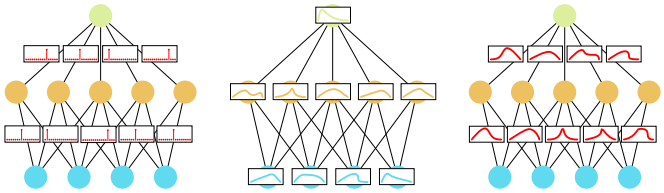
A BNN is defined slightly differently across the literature, but a commonly agreed definition is that a BNN is a stochastic artificial neural network trained using Bayesian inference.

The goal of artificial neural networks (ANNs) is to represent an arbitrary function $\boldsymbol{y} = \Phi(\boldsymbol{x})$. Traditional ANNs such as feedforward networks and recurrent networks are built using one input layer $\boldsymbol{l}_0$, a succession of hidden layers $\boldsymbol{l}_i, i = 1, \ldots, n-1$, and one output layer $\boldsymbol{l}_n$. (Here, $n+1$ is the total number of layers.) In the simplest architecture of feedforward networks, each layer $\boldsymbol{l}$ is represented as a linear transformation, followed by a nonlinear operation s, also known as an *activation function*:

$$\begin{aligned} &\boldsymbol{l}_0 = \boldsymbol{x}, \\ &\boldsymbol{l}_i = s_i(\boldsymbol{W}_i \boldsymbol{l}_{i-1} + \boldsymbol{b}_i) \quad \forall i \in [1, n], \\ &\boldsymbol{y} = \boldsymbol{l}_n. \end{aligned} \tag{2}$$

Here, $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{b})$ are the parameters of the network, where $\boldsymbol{W}$ are the weights of the network connections and $\boldsymbol{b}$ the biases. A given ANN architecture represents a set of functions isomorphic to the set of possible parameters $\boldsymbol{\theta}$. Deep learning is the process of regressing the parameters $\boldsymbol{\theta}$ from the training data D, where D is composed of a series of input $\boldsymbol{x}$ and their corresponding labels $\boldsymbol{y}$. The standard approach is to approximate a minimal cost point estimate of the network parameters $\hat{\boldsymbol{\theta}}$, *i.e.*, a single value for each parameter (Figure 3a), using the backpropagation algorithm, with all other possible parametrizations of the network discarded. The cost function is often defined as the log likelihood of the training set, sometimes with a regularization term included. From a statistician's point of view, this is a maximum likelihood estimation (MLE), or a maximum a posteriori (MAP) estimation when regularization is used.

The point estimate approach, which is the traditional approach in deep learning, is relatively easy to deploy with modern algorithms and software packages, but tends to lack explainability [14]. The final model might also generalize



(a) (b) (c)

Fig. 3: (a) Point estimate neural network, (b) stochastic neural network with a probability distribution for the activations, and (c) stochastic neural network with a probability distribution over the weights.

in unforeseen and overconfident ways on out-of-training-distribution data points [15, 16]. This property, in addition to the inability of ANNs to say *"I don't know"*, is problematic for many critical applications. Of all the techniques that exist to mitigate this [17], stochastic neural networks have proven to be one of the most generic and flexible.

Stochastic neural networks are a type of ANN built by introducing stochastic components into the network. This is performed by giving the network either a stochastic activation (Figure 3b) or stochastic weights (Figure 3c) to simulate multiple possible models $\boldsymbol{\theta}$ with their associated probability distribution $p(\boldsymbol{\theta})$. Thus, BNNs can be considered a special case of **ensemble learning** [18].

The main motivation behind ensemble learning comes from the observation that aggregating the predictions of a large set of average-performing but independent predictors can lead to better predictions than a single well-performing expert predictor [19, 20]. Stochastic neural networks might improve their performance over their point estimate counterparts in a similar fashion, but this is not their main aim. Rather, the main goal of using a stochastic neural network architecture is to obtain a better idea of the uncertainty associated with the underlying processes. This is accomplished by comparing the predictions of multiple sampled model parametrizations $\boldsymbol{\theta}$. If the different models agree, then the uncertainty is low. If they disagree, then the uncertainty is high. This process can be summarized as follows:

$$\begin{aligned} \boldsymbol{\theta} &\sim p(\boldsymbol{\theta}), \\ \boldsymbol{y} &= \Phi_{\boldsymbol{\theta}}(\boldsymbol{x}) \ + \ \boldsymbol{\epsilon}, \end{aligned} \tag{3}$$

where $\boldsymbol{\epsilon}$ represents random noise to account for the fact that the function Φ is only an approximation. A BNN can then be defined as any stochastic artificial neural network trained using Bayesian inference [21].

To design a BNN, the first step is the choice of a deep neural network architecture, *i.e.*, **a functional model**. Then, one has to choose a **stochastic model**, *i.e.*, a prior distribution over the possible model parametrization $p(\boldsymbol{\theta})$ and a prior confidence in the predictive power of the model $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (Figure 2a). The model parametrization can be considered to be the hypothesis H and the training set is the data D. The choice of a BNN's stochastic model is somehow equivalent to the choice of a loss function when training a point estimate neural network; see Section IV-C3. In the rest of this paper, we will denote the model parameters by $\boldsymbol{\theta}$, the training set by D, the training inputs by $D_{\boldsymbol{x}}$, and the training labels by $D_{\boldsymbol{y}}$. By applying Bayes' theorem, and enforcing independence between the model parameters and the input, the Bayesian posterior can be written as:

$$p(\boldsymbol{\theta}|D) = \frac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta'})p(\boldsymbol{\theta'})d\boldsymbol{\theta'}} \propto p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{4}$$

The Bayesian posterior for complex models such as artificial neural networks is a high dimensional and highly non-convex probability distribution [22]. This complexity makes computing and sampling it using standard methods an intractable problem, especially because computing the evidence $\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta'})p(\boldsymbol{\theta'})d\boldsymbol{\theta'}$ is difficult. To address this problem, two broad approaches have been introduced: **(1)** Markov chain Monte Carlo and **(2)** variational inference. These are presented in more details in Section V.

When **using a BNN for prediction**, the probability distribution $p(\boldsymbol{y}|\boldsymbol{x}, D)$ [12], called the marginal and which quantifies the model's uncertainty on its prediction, is of particular interest. Given $p(\boldsymbol{\theta}|D)$, $p(\boldsymbol{y}|\boldsymbol{x}, D)$ can be computed as:

$$p(\boldsymbol{y}|\boldsymbol{x}, D) = \int_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta'})p(\boldsymbol{\theta'}|D)d\boldsymbol{\theta'}. \tag{5}$$

In practice, $p(\boldsymbol{y}|\boldsymbol{x}, D)$ is sampled indirectly using Equation (3). The final prediction can be summarized by statistics computed using a Monte Carlo approach (Figure 2c). A large set of weights $\boldsymbol{\theta}_i$ is sampled from the posterior and used to compute a series of possible outputs $\boldsymbol{y}_i$, as shown in Algorithm 1, which corresponds to samples from the marginal.

Algorithm 1 Inference procedure for a BNN.

Define $p(\boldsymbol{\theta}|D) = \dfrac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta'})p(\boldsymbol{\theta'})d\boldsymbol{\theta'}}$;
for $i = 0$ **to** N **do**
  Draw $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|D)$;
  $\boldsymbol{y}_i = \Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x})$;
end for
return $Y = \{\boldsymbol{y}_i | i \in [0, N)\}$, $\Theta = \{\boldsymbol{\theta}_i | i \in [0, N)\}$;

In Algorithm 1, Y is a set of samples from $p(\boldsymbol{y}|\boldsymbol{x}, D)$ and Θ a collection of samples from $p(\boldsymbol{\theta}|D)$. Usually, aggregates are computed on those samples to summarize the uncertainty of the BNN and obtain an estimator for the output $\boldsymbol{y}$. This estimator is denoted by $\hat{\boldsymbol{y}}$.

When performing **regression**, the procedure that is usually used to summarize the predictions of a BNN is model averaging [23]:

$$\hat{\boldsymbol{y}} = \frac{1}{|\Theta|} \sum_{\boldsymbol{\theta}_i \in \Theta} \Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}). \tag{6}$$

This approach is so common in ensemble learning that it is sometimes called ensembling. To quantify uncertainty, the covariance matrix can be computed as follows:

$$\boldsymbol{\Sigma}_{\boldsymbol{y}|\boldsymbol{x},D} = \frac{1}{|\Theta|-1} \sum_{\boldsymbol{\theta}_i \in \Theta} \left(\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right) \left(\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right)^{\intercal}. \tag{7}$$

When performing **classification**, the average model prediction will give the relative probability of each class, which can be considered a measure of uncertainty:

$$\hat{\boldsymbol{p}} = \frac{1}{|\Theta|} \sum_{\boldsymbol{\theta}_i \in \Theta} \Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}). \tag{8}$$

The final prediction is taken as the most likely class:

$$\hat{\boldsymbol{y}} = \arg\max_i p_i \in \hat{\boldsymbol{p}}. \tag{9}$$

This definition considers BNNs as discriminative models, *i.e.*, models that aim to reconstruct a target variable $\boldsymbol{y}$ given observations $\boldsymbol{x}$. This excludes generative models, although there are examples of generative ANNs based on the Bayesian formalism, *e.g.*, Variational autoencoders [24]. Those are out of the scope of this tutorial.

III. Advantages of Bayesian methods for deep learning

One of the major critiques of Bayesian methods is that they rely on prior knowledge. This is especially true in deep learning, as deriving any insight about plausible parametrization for a given model before training is very challenging. Thus, why use Bayesian methods for deep learning? Discriminative models implicitly represent the conditional probability $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$, and Bayes' formula is an appropriate tool to invert conditional probabilities, even if one has little insight about $p(\boldsymbol{\theta})$ a priori. While there are strong theoretical principles and schema upon which this Bayes' formula can be based [25], this section focuses on some practical benefits of using BNNs.

First, Bayesian methods provide a natural approach to **quantify uncertainty** in deep learning since BNNs have better calibration than classical neural networks [26, 27, 28], *i.e.*, their uncertainty is more consistent with the observed errors. They are less often overconfident or underconfident.

Second, a BNN allows distinguishing between the **epistemic uncertainty** $p(\boldsymbol{\theta}|D)$ and the **aleatoric uncertainty** $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ [29]. This makes BNNs very data-efficient since they can learn from a small dataset without overfitting [30]. At prediction time, out-of-training distribution points will have high epistemic uncertainty instead of blindly giving a wrong prediction.

Third, the no-free-lunch theorem for machine learning [31] can be interpreted as stating that any supervised learning algorithm includes some implicit prior. Bayesian methods, when used correctly, will at least make the prior explicit. **Integrating prior knowledge** into ANNs, which work as black boxes, is difficult but not impossible. In Bayesian deep learning, priors are often considered as soft constraints, analogous to regularization, or data transformations such as data augmentation in traditional deep learning; see Section IV-C. Most regularization methods used for point estimate neural networks can be understood from a Bayesian perspective as setting a prior; see Section IV-C3.

Finally, the Bayesian paradigm enables **the analysis of learning methods**. A number of those methods initially not presented as Bayesian can be **implicitly understood** as being approximate Bayesian, *e.g.*, regularization (Section IV-C3) or ensembling (Section V-E2b). In fact, most of the BNNs used in practice rely on methods that are approximately or implicitly Bayesian (Section V-E) since the exact algorithms are computationally too expensive. The Bayesian paradigm also provides a systematic framework to design new learning and regularization strategies, even for point estimate models.

BNNs have been used in many fields to quantify uncertainty, *e.g.*, in computer vision [32], network traffic monitoring [33], aviation [34], civil engineering [35, 36], hydrology [37], astronomy [38], electronics [39], and medicine [40]. BNNs are useful in **(1)** active learning [41, 42] where an oracle (*e.g.*, a human annotator, a crowd, an expensive algorithm) can label new points from an unlabeled dataset U. The model needs to determine which points should be submitted to the oracle to maximize its performance while minimizing the calls to the oracle. BNNs are also useful in **(2)** online learning [43], where the model is retrained multiple times as new data become available. **For active learning**, data points in the training set with high epistemic uncertainty are scheduled to be labeled with higher priority; see Algorithm 2. In contrast, **in online learning**, previous posteriors can be recycled as priors when new data become available to avoid the so-called problem of catastrophic forgetting [44]; see Algorithm 3.

Algorithm 2 Active learning loop with a BNN.

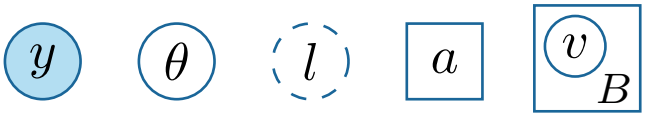
while $U \neq \varnothing$ and $\boldsymbol{\Sigma}_{\boldsymbol{y}|\boldsymbol{x}_{max},D} <$ threshold and $C <$ MaxC **do**
  Draw $\Theta = \{\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|D) | i \in [0, N)\}$;
  for $\boldsymbol{x} \in U$ **do**
    $\boldsymbol{\Sigma}_{\boldsymbol{y}|\boldsymbol{x},D} = \frac{1}{|\Theta|-1} \sum_{\boldsymbol{\theta}_i \in \Theta} (\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}) (\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}})^\intercal$;
    if $\boldsymbol{\Sigma}_{\boldsymbol{y}|\boldsymbol{x},D} > \boldsymbol{\Sigma}_{\boldsymbol{y}|\boldsymbol{x}_{max},D}$ **then**
      $\boldsymbol{x}_{max} = \boldsymbol{x}$;
    end if
  end for
  $D_{\boldsymbol{x}} = D_{\boldsymbol{x}} \cup \{\boldsymbol{x}_{\text{max}}\}$;
  $D_{\boldsymbol{y}} = D_{\boldsymbol{y}} \cup \{\text{Oracle}(\boldsymbol{x}_{\text{max}})\}$;
  $U = U \setminus \{\boldsymbol{x}_{\text{max}}\}$;
  $C = C + 1$;
end while

Algorithm 3 Online learning loop with a BNN.

Define $p(\boldsymbol{\theta}) = p(\boldsymbol{\theta})_0$;
while true do
  Define $p(\boldsymbol{\theta}|D_i) = \frac{p(D_{\boldsymbol{y},i}|D_{\boldsymbol{x},i}, \boldsymbol{\theta}) p(\boldsymbol{\theta})_i}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y},i}|D_{\boldsymbol{x},i}, \boldsymbol{\theta}') p(\boldsymbol{\theta}')_i d\boldsymbol{\theta}'}$;
  Define $p(\boldsymbol{\theta})_{i+1} = p(\boldsymbol{\theta}|D_i)$;
end while

IV. Setting the stochastic model for a Bayesian Neural Network

Designing a BNN requires choosing a **functional model** and a **stochastic model**. This tutorial will not cover the design of the functional model, **as almost any model used for point estimate networks can be used as a functional model for a BNN**. Furthermore, a rich literature on the subject exists already; see, for example, [45]. Instead, this section will



(a) (b) (c) (d) (e)

Fig. 4: The different symbols PGM, (a) observed variables are in colored circles, (b) unobserved variables are in white circles, (c) deterministic variables are in dashed circles and (d) parameters are in rectangles. Plates, represented as a rectangle around a subgraph, indicate multiple independent instances of the subgraph for a batch of variables B (e).

focus on how to design the stochastic model. Section IV-A introduces probabilistic graphical models (PGMs), a tool used to represent the relationships between the model's stochastic variables. Section IV-B details how to derive the posterior for a BNN from its PGM. Section IV-C discusses how to choose the probability laws used as priors. Finally, Section IV-D presents how the choice of a PGM can affect the degree of supervision or incorporate other forms of prior knowledge into the model.

A. Probabilistic graphical models

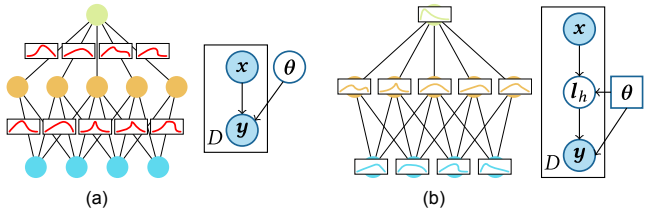
Probabilistic graphical models (PGMs) use graphs to represent the interdependence of multivariate stochastic variables and subsequently decompose their probability distributions. PGMs cover a large variety of models. The type of PGMs this tutorial focuses on are Bayesian belief networks (BBN), which are PGMs whose graphs are acyclic and directed. We refer the reader to [46] for more details on how to represent learning algorithms using general PGMs.

In a PGM, variables $\boldsymbol{v}_i$ are the nodes in the graph. Different symbols are used to distinguish the nature of the considered variables (Figure 4). A directed link, which is the only type of link allowed in a BBN, means that the probability distribution of the target variable is defined conditioned on the source variable. The fact that the BBN is acyclic allows the computation of the joint probability distribution of all the variables $\boldsymbol{v}_i$ in the graph:

$$p(\boldsymbol{v}_1, ..., \boldsymbol{v}_n) = \prod_{i=1}^{n} p(\boldsymbol{v}_i | \text{parents}(\boldsymbol{v}_i)). \tag{10}$$

The type of distribution used to define the conditional probabilities $p(\boldsymbol{v}_i|\text{parents}(\boldsymbol{v}_i))$ depends on the context. Once the conditional probabilities are defined, the BBN describes a data generation process. Parents are sampled before their children. This is always possible since the graph is acyclic. All the variables together represent a sample from the joint probability distribution $p(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)$.

Models usually learn from multiple examples sampled from the same distribution. To highlight this fact, the plate notation (Figure 4e) has been introduced. A plate indicates that the variables $(\boldsymbol{v}_1, ..., \boldsymbol{v}_n)$ in the subgraph encapsulated by the plate are copied along a given batch dimension. A plate implies independence between all the duplicated nodes. This fact can



(a) (b)

Fig. 5: BBNs with (a) coefficients as stochastic variables and (b) activations as stochastic variables.

be exploited to compute the joint probability of a batch $B = \{(\boldsymbol{v}_1, ..., \boldsymbol{v}_n)_b : b = 1, \ldots, |B|\}$ as:

$$p(B) = \prod_{(\boldsymbol{v}_1, ..., \boldsymbol{v}_n) \in B} p(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n). \tag{11}$$

In a PGM, the observed variables, depicted in Figure 4a using colored circles, are treated as the data. The unobserved, also called latent variables, represented by a white circle in Figure 4b, are treated as the hypothesis. From the joint probability derived from the PGM, defining the posterior for the latent variables given the observed variables is straightforward using Bayes' formula:

$$p(\boldsymbol{v}_{\text{latent}} | \boldsymbol{v}_{\text{obs}}) \propto p(\boldsymbol{v}_{\text{obs}}, \boldsymbol{v}_{\text{latent}}). \tag{12}$$

The joint distribution $p(\boldsymbol{v}_{\text{obs}}, \boldsymbol{v}_{\text{latent}})$ is then used by the different inference algorithms; see Section V.

B. Defining the stochastic model of a BNN from a PGM

Consider the two models presented in Figure 5, with both the BNN and the corresponding BBN depicted. The BNN with stochastic weights (Figure 5a), if meant to perform regression, could represent the following data generation process:

$$\begin{aligned} \boldsymbol{\theta} &\sim p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \\ \boldsymbol{y} &\sim p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{\Sigma}). \end{aligned} \tag{13}$$

The choice of using normal laws $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$), is arbitrary but is common in practice because of its good mathematical properties.

For classification, the model samples the prediction from a categorical law $\text{Cat}(p_i)$, *i.e.*,

$$\begin{aligned} \boldsymbol{\theta} &\sim p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \\ \boldsymbol{y} &\sim p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) = \text{Cat}(\Phi_{\boldsymbol{\theta}}(\boldsymbol{x})). \end{aligned} \tag{14}$$

Then, one can use the fact that multiple data points from the training set are independent, as indicated by the plate notation in Figure 5, to write the probability of the training set as:

$$p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta}) = \prod_{(\boldsymbol{x}, \boldsymbol{y}) \in D} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}). \tag{15}$$

In the case of stochastic activations (Figure 5b), the data generation process might become:

$$\begin{aligned} &\boldsymbol{l}_0 = \boldsymbol{x}, \\ &\boldsymbol{l}_i \sim p(\boldsymbol{l}_i|\boldsymbol{l}_{i-1}) = s_i(\mathcal{N}(\boldsymbol{W}_i \boldsymbol{l}_{i-1} + \boldsymbol{b}_i, \Sigma)) \quad \forall i \in [1, n], \\ &\boldsymbol{y} = \boldsymbol{l}_n. \end{aligned} \tag{16}$$

The formulation of the joint probability is slightly more complex as we have to account for the chain of dependencies spanned by the BBN over the multiple latent variables $\boldsymbol{l}_{[1,n-1]}$:

$$p(D_{\boldsymbol{y}}, \boldsymbol{l}_{[1,n-1]}|D_{\boldsymbol{x}}) = \prod_{(\boldsymbol{l}_0,\boldsymbol{l}_n)\in D} \left(\prod_{i=1}^{n} p(\boldsymbol{l}_i|\boldsymbol{l}_{i-1}) \right). \quad (17)$$

It is sometimes possible, and often desirable, to define $p(\boldsymbol{l}_i|\boldsymbol{l}_{i-1})$ such that the BNNs described in Figure 5a and in Figure 5b can be considered equivalent. For instance, sampling l as:

$$\begin{aligned} \boldsymbol{W} &\sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{W}}, \boldsymbol{\Sigma}_{\boldsymbol{W}}), \\ \boldsymbol{b} &\sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{b}}, \boldsymbol{\Sigma}_{\boldsymbol{b}}), \\ \boldsymbol{l} &= s(\boldsymbol{W}\boldsymbol{l}_{-1} + \boldsymbol{b}) \end{aligned} \quad (18)$$

is equivalent to sampling l as:

$$\boldsymbol{l} \sim s(\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{W}}\boldsymbol{l}_{-1}+\boldsymbol{\mu}_{\boldsymbol{b}}, (\boldsymbol{I}\otimes\boldsymbol{l}_{-1})^{\intercal}\boldsymbol{\Sigma}_{\boldsymbol{W}}(\boldsymbol{I}\otimes\boldsymbol{l}_{-1})+\boldsymbol{\Sigma}_{\boldsymbol{b}})), \quad (19)$$

where $\otimes$ denotes a Kronecker product.

The basic Bayesian regression architecture shown in Figure 5a is more common in practice. The alternative architecture shown in Figure 5b is sometimes used as it allows compressing the number of variational parameters when using variational inference [47]; see also Section V.

C. Setting the priors

Setting the prior of a deep neural network is often not an intuitive task. The main problem is that it is not truly explicit how models with a very large number of parameters and a nontrivial architecture such as an ANN will generalize for a given parametrization [48]. In this Section, we first present the common practice, discuss the issues related to the statistical unidentifiability of ANNs, and then show the link between the prior for BNNs and regularization for the point estimate algorithms. Finally, we present a method to build the prior from high level knowledge.

1) A good default prior: For basic architectures such as Bayesian regression (Figure 5a), a standard procedure is to use a normal prior with a zero mean $\mathbf{0}$ and a diagonal covariance $\sigma\boldsymbol{I}$ on the coefficients of the network:

$$p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \sigma\boldsymbol{I}). \quad (20)$$

This approach is equivalent to a weighted ℓ_2 regularization (with weights $1/\sigma$) when training a point estimate network, as will be demonstrated in Section IV-C3. The documentation of the probabilistic programming language Stan [49] provides examples on how to choose σ knowing the expected scale of the considered parameters [50].

Although such an approach is often used in practice, there is no theoretical argument that makes it better than any other formulation [51]. The normal law is preferred due to its mathematical properties and the simple formulation of its log, which is used in most of the learning algorithms.

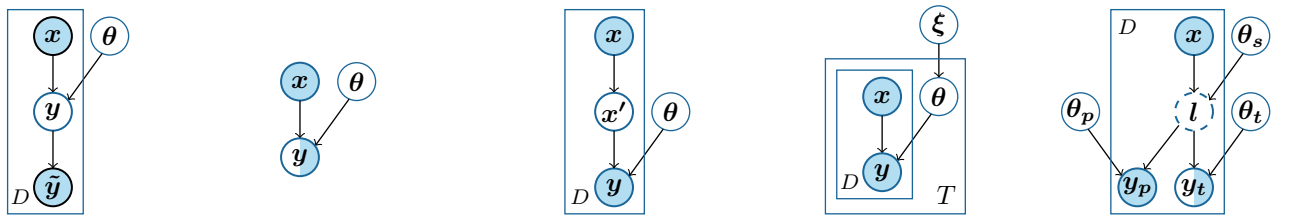
2) Addressing unidentifiability in Bayesian neural networks: One of the main problems with Bayesian deep learning is that deep neural networks are overparametrized models, *i.e.*, they have many equivalent parametrizations [52]. This is an example of statistical unidentifiability, which can lead to complex multimodal posteriors that are hard to sample and approximate when training a BNN [22]. There are two solutions to deal with this issue: **(1)** changing the functional model parametrization, or **(2)** constraining the support of the prior to remove unidentifiability.

The two most common classes of nonuniqueness in ANNs are weight-space symmetry and scaling symmetry [53]. Both are not a concern for point estimate neural networks but might be for BNNs. Weight-space symmetry implies that one can build an equivalent parametrization of an ANN with at least one hidden layer. This is achieved by permuting two rows in $(\boldsymbol{W}_i, \boldsymbol{b}_i)$, the weights and their corresponding bias $\boldsymbol{b}_i$, of one of the hidden layers as well as the corresponding columns in the following layer's weight matrix $\boldsymbol{W}_{i+1}$. This means that as the number of hidden layers and the number of units in the hidden layers grow, the number of equivalent representations, which would roughly correspond to the modes in the posterior distribution, grows factorially. A mitigation strategy is to enforce the bias vector in each layer to be sorted in an ascending or a descending order. However, the practical effects of doing so may be to degrade optimization: weight-space symmetry may implicitly support the exploration of the parameter space during the early stages of the optimization.

Scaling symmetry is an unidentifiability problem arising when using nonlinearities with the property $s(\alpha x) = \alpha s(x)$, which is the case of RELU and Leaky-RELU, two popular nonlinearities in modern machine learning. In this case, assigning the weights $\boldsymbol{W}_l, \boldsymbol{W}_{l+1}$ to two consecutive layers l and $l+1$ becomes strictly equivalent to assigning $\alpha\boldsymbol{W}_l, (1/\alpha)\boldsymbol{W}_{l+1}$. This can reduce the convergence speed for point estimate neural networks, a problem that is addressed in practice with various activation normalization techniques [54]. BNNs are slightly more complex as the scaling symmetry influences the posterior shape, making it harder to approximate. Givens transformations (also called Givens rotations) have been proposed as a mean to constrain the norm of the hidden layers [53] and address the scaling symmetry issue. In practice, using a Gaussian prior already reduces the scaling symmetry problem, as it favors weights with the same Frobenius norm on each layer. A soft version of the activation normalization can also be implemented by using a consistency condition; see Section IV-C4. The additional complexity associated with sampling the network parameters in a constrained space to perfectly remove the scaling symmetry is computationally prohibitive. We provide, in the Practical Example III of the Supplementary Material , additional discussion on this issue using the "Paperfold" practical example.

3) The link between regularization and priors: The usual learning procedure for a point estimate neural network is to find the set of parameters $\boldsymbol{\theta}$ that minimize a loss function built using the training set D:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \ \text{loss}_{D_x, D_y}(\boldsymbol{\theta}). \quad (21)$$



(a) Noisy labels (b) Semi-supervised learning (c) Data augmentation (d) Meta-learning (e) Self-supervised learning

Fig. 6: Different examples of PGMs to adapt the learning strategy for a given BNN (with stochastic weights).

Assuming that the loss is defined as minus the log-likelihood function up to an additive constant, the problem can be rewritten as:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \; p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta}), \tag{22}$$

which would be the first half of the model according to the Bayesian paradigm. Now, assume that we also have a prior for $\boldsymbol{\theta}$, and we want to find the most likely point estimate from the posterior. The problem can be reformulated as:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \; p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{23}$$

Next, one would go back to a log-likelihood formulation:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \; \text{loss}_{D_x, D_y}(\boldsymbol{\theta}) + \text{reg}(\boldsymbol{\theta}), \tag{24}$$

which is easier to optimize. Equation (24) is how regularization is usually applied in machine learning and in many other fields. Another argument, less formal, is that regularization acts as a soft constraint on the search space, in a manner similar to what a prior does for a posterior.

4) Prior with a consistency condition: Regularization can also be implemented with a consistency condition $C(\boldsymbol{\theta}, \boldsymbol{x})$, which is a function used to measure how well the model respects some hypothesis given a parametrization $\boldsymbol{\theta}$ and an input $\boldsymbol{x}$. For example, C can be set to favor sparse or regular predictions to encourage monotonicity of predictions with respect to some input variables (*e.g.*, the probability of getting the flu increases with age), or to favor decision boundaries in low density regions when using semi-supervised learning; see Section IV-D1. C can be seen as the relative log likelihood of a prediction given the input $\boldsymbol{x}$ and parameter set $\boldsymbol{\theta}$. Thus, it can be included in the prior. To this end, C should be averaged over all possible inputs:

$$C(\boldsymbol{\theta}) = \int_{\boldsymbol{x}} C(\boldsymbol{\theta}, \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}. \tag{25}$$

In practice, as $p(\boldsymbol{x})$ is unknown, $C(\boldsymbol{\theta})$ is approximated from the features in the training set:

$$C(\boldsymbol{\theta}) \approx \frac{1}{|D_{\boldsymbol{x}}|} \sum_{\boldsymbol{x} \in |D_{\boldsymbol{x}}|} C(\boldsymbol{\theta}, \boldsymbol{x}). \tag{26}$$

We can now write a function proportional to the prior with the consistency condition included:

$$p(\boldsymbol{\theta}|D_{\boldsymbol{x}}) \propto p(\boldsymbol{\theta}) \exp\left(-\frac{1}{|D_{\boldsymbol{x}}|} \sum_{\boldsymbol{x} \in |D_{\boldsymbol{x}}|} C(\boldsymbol{\theta}, \boldsymbol{x}) \right), \tag{27}$$

where $p(\boldsymbol{\theta})$ is the prior without the consistency condition.

D. Degree of supervision and alternative forms of prior knowledge

The architecture presented in Section IV-B focuses mainly on the use of BNNs in a supervised learning setting. However, in real world applications, obtaining ground-truth labels can be expensive. Thus, new learning strategies should be adopted [55]. We will now present how to adapt BNNs for different degrees of supervision. While doing so, we will also demonstrate how PGMs in general and BBNs in particular are useful in designing or interpreting learning strategies. In particular, the formulation of the Bayesian posterior, which is derived from the different PGMs presented in Figure 6, can also be used for a point estimate neural network to obtain a suitable loss function to search for an MAP estimator for the parameters (Section IV-C3). We also provide a practical example in the Supplementary Material (Practical Example II) to illustrate how such strategies can be implemented for an actual BNN.

1) Noisy labels and semi-supervised learning: The inputs $D_{\boldsymbol{x}}$ in the training sets can be uncertain, either because the labels $D_{\boldsymbol{y}}$ are corrupted by noise [56], or because labels are missing for a number of points. In the case of **noisy labels**, one should extend the BBN to add a new variable for the noisy labels $\tilde{\boldsymbol{y}}$ conditioned on $\boldsymbol{y}$ (Figure 6a). It is common, as the noise level itself is often unknown, to add a variable $\boldsymbol{\sigma}$ to characterize the noise. Frenay *et al.* [57] proposed a taxonomy of the different approaches used to integrate $\boldsymbol{\sigma}$ in a PGM (Figure 7). They distinguish three cases: noise completely at random (NCAR); noise at random (NAR); and noise not at random (NNAR) models. In the NCAR model, the noise $\boldsymbol{\sigma}$ is independent of any other variable, *i.e.*, it is homoscedastic. In the NAR model, $\boldsymbol{\sigma}$ is dependent on the true label $\boldsymbol{y}$ but remains independent of the features. NNAC models also account for the influence of the features $\boldsymbol{x}$, *e.g.*, if the level of noise in an image increases, then the probability that the image has been mislabeled also increases. Both NAR and NNAC models represent heteroscedastic, *i.e.*, the antonym of homoscedastic, noise.

These noise-aware PGMs are slightly more complex than a purely supervised BNN, as presented in Section IV-B. However, they can be treated in a similar fashion by deriving the formula for the posterior from the PGM (Equation (12)) and applying the chosen inference algorithm. For the NNAR model, the most generic stochastic model of the three described above (since the NCAR and NAR models are special

cases of the NNAR model), the posterior becomes:

$$p(\boldsymbol{y}, \boldsymbol{\sigma}, \boldsymbol{\theta}|D) \propto p(D_{\tilde{\boldsymbol{y}}}|\boldsymbol{y}, \boldsymbol{\sigma})p(\boldsymbol{\sigma}|D_{\boldsymbol{x}}, \boldsymbol{y})p(\boldsymbol{y}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}). \quad (28)$$

During the prediction phase, $\boldsymbol{y}$ and $\boldsymbol{\sigma}$ can simply be discarded for each tuple $(\boldsymbol{y}, \boldsymbol{\sigma}, \boldsymbol{\theta})$ sampled from the posterior.

In the case of **partially labeled data** (Figure 6b), also known as semi-supervised learning, the dataset D is split into labeled L and unlabeled U examples. In theory, this PGM can be considered equivalent to the one used in the supervised learning case depicted in Figure 5a, but in this case the unobserved data U would bring no information. The additional information of unlabeled data comes from the prior and only the prior. Similar to traditional machine learning, the most common approaches to implement semi-supervised learning in Bayesian learning are either to use some type of data-driven regularization [58] or to rely on pseudo labels [59].

Data-driven regularization implies modifying the prior assumptions, and thus the stochastic model, to be able to extract meaningful information from the unlabeled dataset U. There are two common ways to approach this process. **The first** one is to condition the prior distribution of the model parameters on the unlabeled examples to favor certain properties of the model, such as a decision boundary in a low density region, using a distribution $p(\boldsymbol{\theta}|U)$ instead of $p(\boldsymbol{\theta})$. This implies formulating the stochastic model as:

$$p(\boldsymbol{\theta}|D) \propto p(L_{\boldsymbol{y}}|L_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}|U), \quad (29)$$

where $p(\boldsymbol{\theta}|U)$ is a prior with a consistency condition, as defined in Equation (27). The consistency condition usually expresses the fact that points that are close to each other should lead to the same prediction, *e.g.*, graph Laplacian norm regularization [60].

The second way is to assume some kind of dependency across the observed and unobserved labels in the dataset. This type of semi-supervised Bayesian learning relies either on an undirected PGM [61] to build the prior or at least does not assume independence between different training pairs $(\boldsymbol{x}, \boldsymbol{y})$ [62]. To keep things simple, we represent this fact by dropping the plate around $\boldsymbol{y}$ in Figure 6b. The posterior is written in the usual way (Equation (4)). The main difference is that $p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})$ is chosen to enforce some kind of consistency across the dataset. For example, one can assume that two close points are likely to have similar labels $\boldsymbol{y}$ with a level of uncertainty that increases with the distance.

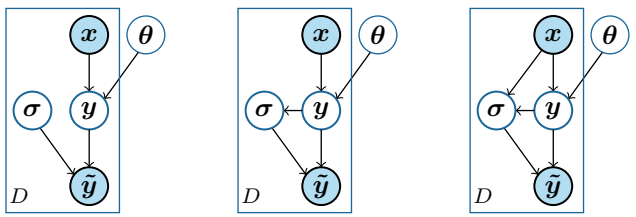


Fig. 7: BBNs corresponding to (a) the noise completely at random (NCAR), (b) noise at random (NAR) and (c) noise not at random (NNAR) models from [57].

Both approaches have a similar effect and the choice of one over the other will depend on the mathematical formulation favored to build the model.

The semi-supervised learning strategy can also be reformulated as having a weak predictor capable of generating some **pseudo labels** $\tilde{\boldsymbol{y}}$, sometimes with some confidence level. Many of the algorithms used for semi-supervised learning use an initial version of the model trained with the labeled examples [63] to generate the pseudo labels $\tilde{\boldsymbol{y}}$ and train the final model with $\tilde{\boldsymbol{y}}$. This is problematic for BNNs. When the prediction uncertainty is accounted for, reducing the uncertainty associated with the unlabeled data becomes impossible, at least not without an additional hypothesis in the prior. Even if it is less current in practice, using a simpler model [64] to obtain the pseudo labels can help mitigate that problem.

2) Data augmentation: Data augmentation in machine learning is a strategy that is used to significantly increase the diversity of the data D available to train deep models, without actually collecting new data. It relies on transformations that act on the input but have no or very low probability to change the label (or at least do so in a predictable way) to generate an augmented dataset $A(D)$. Examples of such transformations include applying rotations, flipping or adding noise in the case of images. Data augmentation is now at the forefront of state-of-the-art techniques in computer vision [59] and increasingly in natural language processing [65].

The augmented dataset $A(D)$ could contain an infinite set of possible variants of the initial dataset D, *e.g.*, when using continuous transforms such as rotations or additional noise. To achieve this in practice, $A(D)$ is sampled on the fly during training, rather than generating in advance all possible augmentations in the training set. This process is straightforward when training point estimate neural networks, but there are some subtleties when applying it in Bayesian statistics. The main concern is that the posterior of interest is $p(\boldsymbol{\phi}|D, Aug)$, where Aug represents some knowledge about augmentation, not $p(\boldsymbol{\phi}|A(D), D)$, since $A(D)$ is not observed. From a Bayesian perspective, the additional information is brought by the knowledge of the augmentation process rather than by some additional data. Stated otherwise, the data augmentation is a part of the stochastic model (Figure 6c).

The idea is that if one is given data D, then one could also have been given data D', where each element in D is replaced by an augmentation. Then, D' is a different perspective of the data D. To model this, we have the augmentation distribution $p(\boldsymbol{x}'|\boldsymbol{x}, Aug)$ that augments the observed data using the augmentation model Aug to generate (probabilistically) x', which represents data in the vicinity of x (Figure 6c). x' can then be marginalized to simplify the stochastic model. The posterior is given by:

$$p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y}, Aug) \propto \left(\int_{\boldsymbol{x}'} p(\boldsymbol{y}|\boldsymbol{x}', \boldsymbol{\theta})p(\boldsymbol{x}'|\boldsymbol{x}, Aug)d\boldsymbol{x}' \right) p(\boldsymbol{\theta}). \quad (30)$$

This is a probabilistic counterpart to vicinal risk [66].

The integral in Equation (30) can be approximated using Monte Carlo integration by sampling a small set of augmen-

tations A_x according to $p(x'|x, Aug)$ and averaging:

$$p(y|x, \boldsymbol{\theta}, Aug) \approx \frac{1}{|A_x|} \sum_{x' \in A_x} p(y|x', \boldsymbol{\theta}). \tag{31}$$

When training using a Monte-Carlo-based estimate of the loss, A_x can contain as few as a single eleme[illegible]s long as it is resampled for each optimization iteration. is greatly simplifies the evaluation of Equation (31).

An extension of this approach works in the context of semi-supervised learning. The prior can be designed to encourage consistency of predictions under augmentation [67, 59], using unlabeled data to build the samples for the consistency condition, as defined in Equation (27). Note that this does not add labeling to the unlabeled examples but only adds a term to encourage consistency between the labels for an unlabeled data point and its augmentation.

3) Meta-learning, transfer learning, and self-supervised learning: **Meta-learning** [68], in the broadest sense, is the use of machine learning algorithms to assist in the training and optimization of other machine learning models. The meta knowledge acquired by meta-learning can be distinguished from standard knowledge in the sense that it is applicable to a set of related tasks rather than a single task.

Transfer learning designates methods that reuse some intermediate knowledge acquired on a given problem to address a different problem. In deep learning, it is used mostly for domain adaptation, when labeled data are abundant in a domain that is in some way similar to the domain of interest but scarce in the domain of interest [69]. Alternatively, pre-trained models [70] could be used to study large architectures whose complete training would be very computationally expensive.

Self-supervised learning is a learning strategy where the data themselves provide the labels [71]. Since the labels directly obtainable from the data do not match the task of interest, the problem is approached by learning a pretext (or proxy) task in addition to the task of interest. The use of self-supervision is now generally regarded as an essential step in some areas. For instance, in natural language processing, most state-of-the-art methods use these pre-trained models [70]. In addition, modern deep learning-based 3D object reconstruction [72] and disparity estimation in stereo vision [73] rely on self-supervised learning to overcome the time-consuming manual annotation of training data.

A common approach for meta-learning in Bayesian statistics is to recast the problem as hierarchical Bayes [74], with the prior $p(\boldsymbol{\theta}_t|\boldsymbol{\xi})$ for each task conditioned on a new global variable $\boldsymbol{\xi}$ (Figure 6d). $\boldsymbol{\xi}$ can represent continuous metaparameters or discrete information about the structure of the BNN, *i.e.*, to learn probable functional models, or the underlying subgraph of the PGM, *i.e.*, to learn probable stochastic models. Multiple levels can be added to organize the tasks in a more complex hierarchy if needed. Here, we present only the case with one level since the generalization is straightforward. With this broad Bayesian understanding of meta-learning, both transfer learning and self-supervised learning are special cases of meta-learning. The general posterior becomes:

$$p(\boldsymbol{\theta}, \boldsymbol{\xi}|D) \propto \left(\prod_{t \in T} p(D^t_{\boldsymbol{y}}|D^t_{\boldsymbol{x}}, \boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t|\boldsymbol{\xi}) \right) p(\boldsymbol{\xi}). \tag{32}$$

In practice, the problem is often approached with empirical Bayes (Section V-D), and only a point estimate $\hat{\boldsymbol{\xi}}$ is considered for the global variable, ideally the MAP estimate obtained by marginalizing $p(\boldsymbol{\theta}, \boldsymbol{\xi}|D)$ and selecting the most likely point, but this is not always the case.

In transfer learning, the usual approach would be to set $\hat{\boldsymbol{\xi}} = \boldsymbol{\theta}_m$, with $\boldsymbol{\theta}_m$ being the coefficients of the main task. The new prior can then be obtained from $\hat{\boldsymbol{\xi}}$, for example:

$$p(\boldsymbol{\theta}|\boldsymbol{\xi}) = \mathcal{N}((\tau(\boldsymbol{\xi}), \mathbf{0}), \sigma \boldsymbol{I}), \tag{33}$$

where τ is a selection of the parameters to transfer and σ is a parameter to tune manually. Unselected parameters are assigned a new prior, with a mean of 0 by convention. If a BNN has been trained for the main task, then σ can be estimated from the previous posterior, with an increment to account for the additional uncertainty caused by the domain shift.

Self-supervised learning can be implemented in two steps. The **first** step learns the pretext task while the second one performs transfer learning. This can be considered overly complex but might be required if the pretext task has a high computational complexity (*e.g.*, BERT models in natural language processing [70]). Recent contributions [75] have shown that jointly learning the pretext task and the final task (Figure 6e) can improve the results obtained in self-supervised learning. This approach, which is closer to hierarchical Bayes, also allows setting the prior a single time while still retaining the benefits of self-supervised learning.

V. Bayesian Inference Algorithms

A priori, a BNN does not require a learning phase as one just needs to sample the posterior and do model averaging; see Algorithm 1. However, sampling the posterior is not easy in the general case. While the conditional probability $P(D|H)$ of the data and the probability $P(H)$ of the model are given by the stochastic model, the integral for the evidence term $\int_H P(D|H')P(H')dH'$ might be excessively difficult to compute. For nontrivial models, even if the evidence has been computed, directly sampling the posterior is prohibitively difficult due to the high dimensionality of the sampling space. Instead of using traditional methods, *e.g.*, inversion sampling or rejection sampling to sample the posterior, dedicated algorithms are used. The most popular ones are Markov chain Monte Carlo (MCMC) methods [76], a family of algorithms that exactly sample the posterior, or variational inference [77], a method for learning an approximation of the posterior; see Figure 2.

This section reviews these methods. First, in subsection V-A and V-B, we introduce MCMC and variational inference as they are used in traditional Bayesian statistics. Then, in subsection V-E, we review different simplifications or approximations that have been proposed for deep learning. We also provide a practical example in the Supplementary Material (Practical example III), which compares different learning strategies.

A. Markov Chain Monte Carlo (MCMC)

The idea behind MCMC methods is to construct a Markov chain, a sequence of random samples S_i, which probabilistically depend only on the previous sample S_{i-1}, such that the S_i are distributed following a desired distribution. Unlike standard sampling methods such as rejection or inversion sampling, most MCMC algorithms require an initial burn-in time before the Markov chain converges to the desired distribution. Moreover, the successive S_i's might be autocorrelated. This means that a large set of samples Θ has to be generated and subsampled to obtain approximately independent samples from the underlying distribution. The final collection of samples Θ has to be stored after training, which is expensive for most deep learning models.

Despite their inherent drawbacks, MCMC methods can be considered among the best available and the most popular solutions for sampling from exact posterior distributions in Bayesian statistics [78]. However, not all MCMC algorithms are relevant for Bayesian deep learning. Gibbs sampling [79], for example, is very popular in general statistics and unsupervised machine learning but is very ill-suited for BNNs. The most relevant MCMC method for BNNs is the Metropolis-Hastings algorithm [80]. The property that makes the Metropolis-Hasting algorithm popular is that it does not require knowledge about the exact probability distribution $P(\boldsymbol{x})$ to sample from. Instead, a function $f(\boldsymbol{x})$ that is proportional to that distribution is sufficient. This is the case of a Bayesian posterior distribution, which is usually quite easy to compute except for the evidence term.

The Metropolis-Hasting algorithm, see Algorithm 4, starts with a random initial guess, $\boldsymbol{\theta}_0$, and then samples a new candidate point θ' around the previous θ, using a proposal distribution $Q(\theta'|\theta)$. If θ' is more likely than θ according to the target distribution, it is accepted. If it is less likely, it is accepted with a certain probability or rejected otherwise.

Algorithm 4 Metropolis-Hasting algorithm.

Draw $\boldsymbol{\theta}_0 \sim Initial\ probability\ distribution$;
while $n = 0$ **to** N **do**
  Draw $\boldsymbol{\theta}' \sim Q(\boldsymbol{\theta}'|\boldsymbol{\theta}_n)$;
  $p = \min\left(1, \frac{Q(\boldsymbol{\theta}'|\boldsymbol{\theta}_n)}{Q(\boldsymbol{\theta}_n|\boldsymbol{\theta}')}\frac{f(\boldsymbol{\theta}')}{f(\boldsymbol{\theta}_n)}\right)$;
  Draw $k \sim$ Bernoulli(p);
  if k **then**
    $\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}'$;
    $n = n + 1$;
  end if
end while

The acceptance probability p can be simplified if Q is chosen to be symmetric, *i.e.*, $Q(\boldsymbol{\theta}'|\boldsymbol{\theta}_n) = Q(\boldsymbol{\theta}_n|\boldsymbol{\theta}')$. The formula for the acceptance rate then becomes:

$$p = \min\left(1, \frac{f(\boldsymbol{\theta}')}{f(\boldsymbol{\theta}_n)}\right). \tag{34}$$

In this situation, the algorithm is simply called **the Metropolis method**. Common choices for Q can be a normal distribution $Q(\boldsymbol{\theta}'|\boldsymbol{\theta}_n) = \mathcal{N}(\boldsymbol{\theta}_n, \sigma^2)$, or a uniform distribution $Q(\boldsymbol{\theta}'|\boldsymbol{\theta}_n) = \mathcal{U}(\boldsymbol{\theta}_n - \boldsymbol{\varepsilon}, \boldsymbol{\theta}_n + \boldsymbol{\varepsilon})$, centered around the previous sample. To deal with non-symmetric proposal distributions, *e.g.*, to accommodate a constraint in the model such as a bounded domain, one has to take into account the correction term imposed by the full Metropolis-Hasting algorithm.

The spread of $Q(\boldsymbol{\theta}'|\boldsymbol{\theta}_n)$ has to be tweaked. If it is too large, the rejection rate will be too high. If it is too small, the samples will be more autocorrelated. There is no general method to tweak those parameters. However, a clever strategy to obtain the new proposed sample $\boldsymbol{\theta}'$ can reduce their impact. This is why the Hamiltonian Monte-Carlo method has been proposed.

The Hamiltonian Monte Carlo algorithm (HMC) [81] is another example of Metropolis-Hasting algorithms for continuous distributions. It is designed with a clever scheme to draw a new proposal $\boldsymbol{\theta}'$ to ensure that as few samples as possible are rejected and there is as few correlation as possible between samples. In addition, the HMC's burn-in time is extremely short compared to the standard Metropolis-Hasting algorithm.

Most software packages for Bayesian statistics implement the **No-U-Turn sampler** (NUTS for short) [82], which is an improvement over the classic HMC algorithm allowing the hyperparameters of the algorithm to be automatically tweaked instead of manually setting them.

B. Variational inference

MCMC algorithms are the best tools for sampling from the exact posterior. However, their lack of scalability has made them less popular for BNNs, given the size of the models under consideration. Variational inference [77], which scales better than MCMC algorithms, gained considerable popularity. Variational inference is not an exact method. Rather than allowing sampling from the exact posterior, the idea is to have a distribution $q_\phi(H)$, called the variational distribution, parametrized by a set of parameters $\boldsymbol{\phi}$. The values of the parameters $\boldsymbol{\phi}$ are then learned such that the variational distribution $q_\phi(H)$ is as close as possible to the exact posterior $P(H|D)$. The measure of closeness that is commonly used is the Kullback-Leibler divergence (KL-divergence) [83]. It measures the differences between probability distributions based on Shannon's information theory [84]. The KL-divergence represents the average number of additional bits required to encode a sample from P using a code optimized for q. For Bayesian inference, it is computed as:

$$D_{KL}(q_\phi||P) = \int_H q_\phi(H') \log\left(\frac{q_\phi(H')}{P(H'|D)}\right) dH'. \tag{35}$$

There is an apparent problem here, which is, to compute $D_{KL}(q_\phi||P)$, one needs to compute $P(H|D)$ anyway. To overcome this, a different, easily derived formula called the *evidence lower bound*, or ELBO, serves as a loss:

$$\int_H q_\phi(H') \log\left(\frac{P(H', D)}{q_\phi(H')}\right) dH' = \log(P(D)) - D_{KL}(q_\phi||P). \tag{36}$$

Since $\log(P(D))$ only depends on the prior, minimizing $D_{KL}(q_\phi||P)$ is equivalent to maximizing the ELBO.

The most popular method to optimize the ELBO is stochastic variational inference (SVI) [85], which is in fact the stochastic gradient descent method applied to variational inference. This allows the algorithm to scale to the large datasets

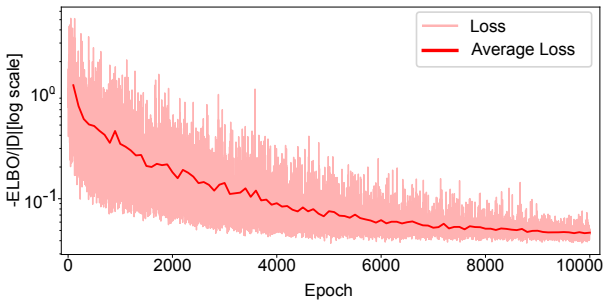


Fig. 8: Typical training curve for Bayes-by-backprop.

that are encountered in modern machine learning, since the ELBO can be computed on a single mini-batch at each iteration.

Convergence, when learning the posterior with SVI, will be slow compared to the usual gradient descent. Moreover, most implementations use a small number of samples to evaluate the ELBO, often just one, before taking a gradient step. In other words, the ELBO estimate will be noisy at each iteration.

In traditional machine learning and statistics, $q_\phi(H)$ is mostly constructed from distributions in the exponential family, *e.g.*, multivariate normal [86], Gamma and Dirichlet distributions. The ELBO can then be dramatically simplified into components [87] leading to a generalization of the well-known expectation-maximization algorithm. To account for correlations between the large number of parameters, certain approximations are made. For instance, block diagonal [88] or low rank plus diagonal [89] covariance matrices can be used to reduce the number of variational parameters $\boldsymbol{\phi}$ from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$, where n is the number of model parameters $\boldsymbol{\theta}$. Appendix A gives more details on how these simplifications are implemented in practice.

C. Bayes by backpropagation

Variational inference offers a good mathematical tool for Bayesian inference, but it needs to be adapted to deep learning. The main problem is that stochasticity stops backpropagation from functioning at the internal nodes of a network [46]. Different solutions have been proposed to mitigate this problem, including probabilistic backpropagation [90] or Bayes-by-backprop [91]. The latter may appear more familiar to deep learning practitioners. We will thus focus on Bayes-by-backprop in this tutorial. Bayes-by-backprop is indeed a practical implementation of SVI combined with a reparametrization trick [92] to ensure backpropagation works as usual.

The idea is to use a random variable $\varepsilon \sim q(\varepsilon)$ as a nonvariational source of noise. $\boldsymbol{\theta}$ is not sampled directly but obtained via a deterministic transformation $t(\varepsilon, \boldsymbol{\phi})$ such that $\boldsymbol{\theta} = t(\varepsilon, \boldsymbol{\phi})$ follows $q_\phi(\boldsymbol{\theta})$. ε is sampled and thus changes at each iteration but can still be considered a constant with regard to other variables. All other transformations being non-stochastic, backpropagation works as usual for the variational parameters $\boldsymbol{\phi}$, meaning the training loop can be implemented analogous to the training loop of a non-stochastic neural network; see Algorithm 5. The general formula for the ELBO becomes:

$$\int_\varepsilon q_\phi(t(\varepsilon, \boldsymbol{\phi})) \log \left(\frac{P(t(\varepsilon, \boldsymbol{\phi}), D)}{q_\phi(t(\varepsilon, \boldsymbol{\phi}))} \right) |\text{Det}(\nabla_\varepsilon t(\varepsilon, \boldsymbol{\phi}))| \, d\varepsilon. \quad (37)$$

This is tedious to work with. Instead, to estimate the gradient of the ELBO, Blundell *et al*. [91] proposed to use the fact that if $q_\phi(\boldsymbol{\theta})d\boldsymbol{\theta} = q(\varepsilon)d\varepsilon$, then for a differentiable function $f(\boldsymbol{\theta}, \boldsymbol{\phi})$, we have:

$$\frac{\partial}{\partial \boldsymbol{\phi}} \int_{\boldsymbol{\phi}} q_\phi(\boldsymbol{\theta}') f(\boldsymbol{\theta}', \boldsymbol{\phi}) d\boldsymbol{\theta}' = \int_\varepsilon q(\varepsilon) \left(\frac{\partial f(\boldsymbol{\theta}, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{\phi}} + \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \right) d\varepsilon. \quad (38)$$

A proof is provided in [91]. We also provide in Appendix B an alternative proof to give more details on when we can assume $q_\phi(\boldsymbol{\theta})d\boldsymbol{\theta} = q(\varepsilon)d\varepsilon$. A sufficient condition is for $t(\varepsilon, \boldsymbol{\phi})$ to be invertible with respect to ε and the distributions $q(\varepsilon)$ and $q_\phi(\boldsymbol{\theta})$ to not be degenerated.

For the case where the weights are treated as stochastic variables, and thus the hypothesis H, the training loop can be implemented as described in Algorithm 5.

Algorithm 5 Bayes-by-backprop algorithm.

$\boldsymbol{\phi} = \boldsymbol{\phi}_0$;
for $i = 0$ **to** N **do**
  Draw $\varepsilon \sim q(\varepsilon)$;
  $\boldsymbol{\theta} = t(\varepsilon, \boldsymbol{\phi})$;
  $f(\boldsymbol{\theta}, \boldsymbol{\phi}) = \log(q_\phi(\boldsymbol{\theta})) - \log(p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}))$;
  $\Delta_\phi f = \text{backprop}_\phi(f)$;
  $\boldsymbol{\phi} = \boldsymbol{\phi} - \alpha \Delta_\phi f$;
end for

The objective function f corresponds to an estimate of the ELBO from a single sample. This means that the gradient estimate will be noisy. The convergence graph will also be much more noisy than in the case of classic backpropagation (Figure 8). To obtain a better estimate of the convergence, one can average the loss over multiple epochs.

Since algorithm 5 is very similar to the classical training loop for point estimate deep learning, most techniques used for optimization in deep learning are straightforward to use for Bayes-by-backprop. For example, it is perfectly fine to use the ADAM optimizer [93] instead of the stochastic gradient descent.

Note also that, if Bayes-by-backprop is presented for BNNs with stochastic weights, adapting it for BNNs with stochastic activations is straightforward. In that case, the activations $\boldsymbol{l}$ represent the hypothesis H and the weights $\boldsymbol{\theta}$ are part of the variational parameters $\boldsymbol{\phi}$.

D. Learning the prior

Learning the prior and the posterior afterwards is possible. This is meaningful if most aspects of the prior can be set using prior knowledge, and only a limited set of free parameters of the prior are learned before obtaining the posterior. In standard Bayesian statistics, this is known as **empirical Bayes**. This is usually a valid approximation when the dimensions of the prior parameters being learned are significantly smaller than the dimensions of the model parameters.

Given a parametrized prior distribution $p_\xi(H)$, maximizing the likelihood of the data is a good method to learn the parameters $\boldsymbol{\xi}$:

$$\begin{aligned}\hat{\boldsymbol{\xi}} &= \arg\max_{\xi} P(D|\boldsymbol{\xi}) \\ &= \arg\max_{\xi} \int_H p_\xi(D|H')p_\xi(H')dH'. \end{aligned} \tag{39}$$

In general, directly finding $\hat{\boldsymbol{\xi}}$ is an intractable problem. However, when using variational inference, the ELBO is the log likelihood of the data minus the KL-divergence of $q_\phi(\boldsymbol{\theta})$ and prior (Eq. 36):

$$\log(P(D|\boldsymbol{\xi})) = \text{ELBO} + D_{KL}(q_\phi||P). \tag{40}$$

This property means that maximizing the ELBO, now a function of both $\boldsymbol{\xi}$ and $\boldsymbol{\phi}$, is equivalent to maximizing a lower bound on the log likelihood of the data. This lower bound becomes tighter when q_ϕ is from a general family of probability distributions with more flexibility to fit the exact posterior $P(\boldsymbol{\theta}|D)$. The Bayes-by-backprop algorithm presented in Section V-C needs only to be slightly modified to include the additional parameters in the training loop; see Algorithm 6.

Algorithm 6 Bayes-by-backprop with parametric prior.

$\boldsymbol{\xi} = \boldsymbol{\xi}_0$;
$\boldsymbol{\phi} = \boldsymbol{\phi}_0$;
for $i = 0$ **to** N **do**
  Draw $\varepsilon \sim q(\varepsilon)$;
  $\boldsymbol{\theta} = t(\varepsilon, \boldsymbol{\phi})$;
  $f(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\xi}) = \log(q_\phi(\boldsymbol{\theta})) - \log(p_{\boldsymbol{\xi}}(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p_{\boldsymbol{\xi}}(\boldsymbol{\theta}))$;
  $\Delta_\xi f = \text{backprop}_\xi(f)$;
  $\Delta_\phi f = \text{backprop}_\phi(f)$;
  $\boldsymbol{\xi} = \boldsymbol{\xi} - \alpha_\xi \Delta_\xi f$;
  $\boldsymbol{\phi} = \boldsymbol{\phi} - \alpha_\phi \Delta_\phi f$;
end for

E. Inference algorithms adapted for deep learning

We presented thus far the fundamental theory to design and train BNNs. However, the aforementioned methods are still not easily applicable to most large scale architectures currently used in deep learning. Recent research has also shown that being only approximately Bayesian is sufficient to achieve a correctly calibrated model with uncertainty estimates [27]. This section presents how inference algorithms were adapted for deep learning, resulting in more efficient methods. Specific inference methods can still be classified as MCMC algorithms, *i.e.*, they generate a sequence of samples from the posterior, or as a form of variational inference, *i.e.*, they learn the parameters of an intermediate distribution to approximate the posterior. All methods are summarized in Figure 9.

1) Bayes via Dropout: Dropout has initially been proposed as a regularization method [94]. It works by applying multiplicative noise to the target layer. The most commonly used type of noise is Bernoulli noise, but other types such as the Gaussian noise for Gaussian Dropout [94] might be used instead.

Dropout is usually turned off at evaluation time, but leaving it on results in a distribution for the output predictions [95, 96]. It turns out that this procedure, called Monte Carlo Dropout, is in fact variational inference with a variational distribution defined for each weight matrix as:

$$\begin{aligned}\boldsymbol{z}_{i,j} &\sim \text{Bernoulli}(p_i), \\ \boldsymbol{W}_i &= \boldsymbol{M}_i \cdot \text{diag}(\boldsymbol{z}_i),\end{aligned} \tag{41}$$

with $\boldsymbol{z}_i$ being the random activation coefficients and $\boldsymbol{M}_i$ the matrix of weights before dropout is applied. p_i is the activation probability for layer i and can be learned or set manually.

When used to train a BNN, dropout should not be seen as a regularization method, as it is part of the variational posterior, not the prior. This means that it should be coupled with a different type of regularization [97], *e.g.*, ℓ^2 weight penalization. The equivalence between the objective function $\mathbf{L}_{\text{dropout}}$ used for training with dropout and ℓ^2 weight regularization, which is defined as:

$$\mathbf{L}_{\text{dropout}} = \frac{1}{N}\sum_D f(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \lambda \sum_{\boldsymbol{\theta}} \theta_i^2, \tag{42}$$

and the ELBO, assuming a normal prior on the weights and the distribution presented in Equation 41 as variational posterior, has been demonstrated in [95]. The argument is similar to the one presented in Section IV-C3.

MC-Dropout is a very convenient technique to perform Bayesian deep learning. It is straightforward to implement and requires little additional knowledge or modeling effort compared to traditional methods. It often leads to a faster training phase compared to other variational inference approaches. If a model has been trained with dropout layers, which are quite widespread in today's deep learning architectures, and an additional form of regularization acting as prior, it can be used as a BNN without any need to be retrained.

On the other hand, MC-Dropout might lack some expressiveness and may not fully capture the uncertainty associated with the model predictions [98]. It also lacks flexibility compared to other Bayesian methods for online or active learning.

2) Bayes via stochastic gradient descent: Stochastic gradient descent (SGD) and related algorithms are at the core of modern machine learning. The initial goal of SGD is to provide an algorithm that converges to an optimal point estimate solution while having only noisy estimates of the gradient of the objective function. This is especially useful when the training data has to be split into mini-batches. The parameter update rule at time t can be written as:

$$\Delta\boldsymbol{\theta}_t = \frac{\epsilon_t}{2}\left(\frac{N}{n}\nabla \log(p(D_{t,\boldsymbol{y}}|D_{t,\boldsymbol{x}}, \boldsymbol{\theta}_i)) + \nabla \log(p(\boldsymbol{\theta}_t))\right), \tag{43}$$

where D_t is a mini-batch subsampled at time t from the complete dataset D, ϵ_t is the learning rate at time t, N is the size of the whole dataset and n the size of the mini-batch.

SGD, or related optimization algorithms such as ADAM [93], can be reinterpreted as a Markov Chain algorithm [99]. Usually, the hyperparameters of the algorithm are tweaked to ensure that the chain converges to a Dirac distribution, whose position gives the final point estimate. This is done by reducing ϵ_t toward zero while ensuring that $\sum_{t=0}^{\infty} \epsilon_t = \infty$. However, if the learning rate is reduced toward a strictly positive value, the underlying Markov Chain will converge

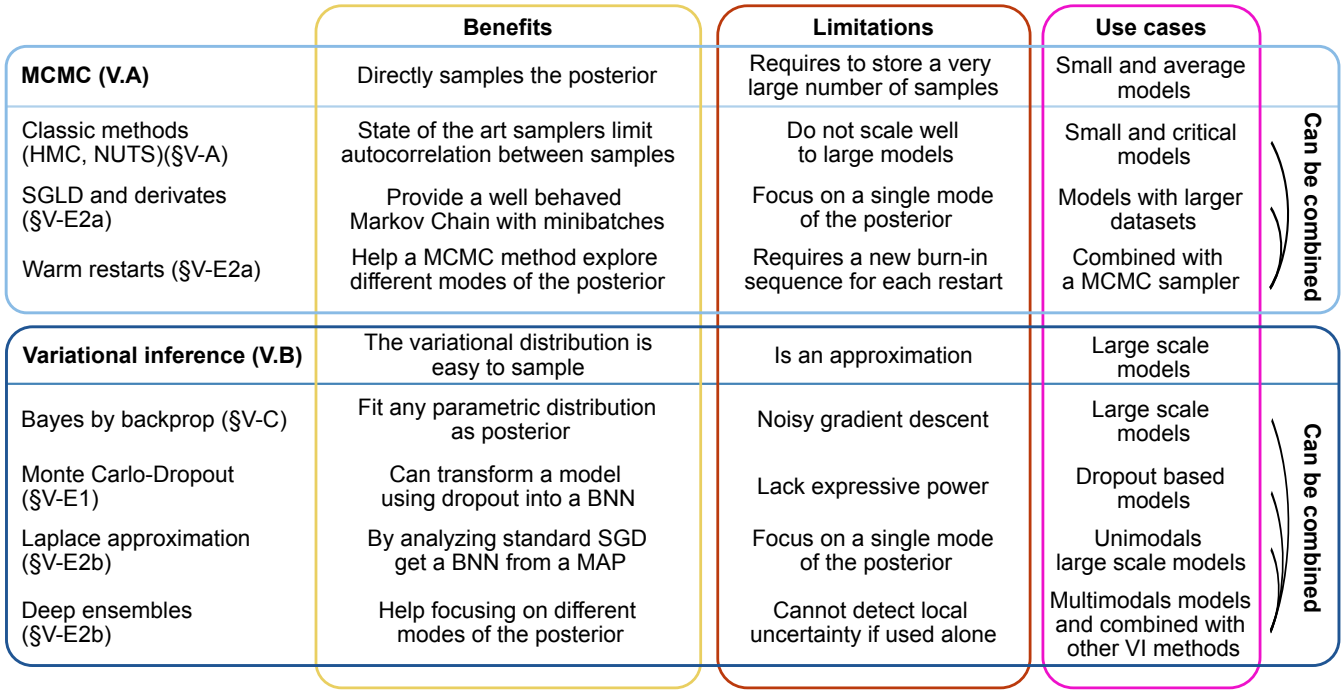


Fig. 9: Summary of the different inference approaches used to train a BNN with their benefits, limitations and use cases.

to a stationary distribution. If a Bayesian prior is accounted for in the objective function, then this stationary distribution can be an approximation of the corresponding posterior.

a) MCMC algorithms based on the SGD dynamic: To approximately sample the posterior using the SGD algorithm, a specific MCMC method, called stochastic gradient Langevin dynamic (SGLD) [100], has been developed, see Algorithm 7. Coupling SGD with Langevin dynamic leads to a slightly modified update step:

$$\Delta\boldsymbol{\theta}_t = \frac{\epsilon_t}{2}\left(\frac{N}{n}\nabla\log(p(D_t,\boldsymbol{\theta}_t)) + \nabla\log(p(\boldsymbol{\theta}_t))\right) + \eta_t,$$
$$\eta_t \sim \mathcal{N}(0,\epsilon_t). \tag{44}$$

Welling *et al.* [100] showed that this method leads to a Markov Chain that samples the posterior if ϵ_t goes toward zero. However, in that case, the successive samples become increasingly autocorrelated. To address this problem, the authors proposed to stop reducing ϵ_t at some point, thus making the samples only an approximation of the posterior. Nevertheless, SGLD offers better theoretical guarantees compared to other MCMC methods when the dataset is split into mini-batches. This makes the algorithm useful in Bayesian deep learning.

Algorithm 7 Stochastic Gradient Langevin Dynamic (SGLD).

Draw $\boldsymbol{\theta}_0 \sim$ *Initial probability distribution*;
for $t = 0$ **to** E **do**
  Select a mini-batch $D_{t,\boldsymbol{y}}, D_{t,\boldsymbol{x}} \subset D$;
  $f(\boldsymbol{\theta}_t) = \frac{N}{n}\log(p(D_{t,\boldsymbol{y}}|D_{t,\boldsymbol{x}},\boldsymbol{\theta}_t)) + \log(p(\boldsymbol{\theta}_t))$;
  $\Delta_{\boldsymbol{\theta}} f = \text{backprop}_{\boldsymbol{\theta}}(f)$;
  Draw $\eta_t \sim \mathcal{N}(0,\epsilon_t)$;
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \left(\frac{\epsilon_t}{2}\Delta_\theta f + \eta_t\right)$;
end for

To favor the exploration of the posterior, one can use warm restart of the algorithm [101], *i.e.*, restarting the algorithm at a new random position $\boldsymbol{\theta}_0$ and with a large learning rate ϵ_0. This offers multiple benefits. The main one is to avoid the mode collapse problem [102]. In the case of a BNN, the true Bayesian posterior is usually a complex multimodal distribution, as multiple and sometimes not equivalent parametrizations $\boldsymbol{\theta}$ of the network can fit the training set. Favoring exploration over precise reconstruction can help to achieve a better picture of those different modes. Then, as parameters sampled from the same mode are likely to make the model generalize in a similar manner, using warm restarts enables a much better estimate of the epistemic uncertainty when processing unseen data, even if this approach provides only a very rough approximation of the exact posterior.

Similar to other MCMC methods, this approach still suffers from a huge memory footprint. This is why a number of authors have proposed methods that are more similar to traditional variational inference than to an MCMC algorithm.

b) Variational Inference based on SGD dynamic: Instead of an MCMC algorithm, SGD dynamic can be used as a variational inference method to learn a distribution by using Laplace approximation. Laplace approximation fits a Gaussian posterior by using the maximum a posteriori estimate as the mean and the inverse of the Hessian $\boldsymbol{H}$ of the loss (assuming the loss is the log likelihood) as covariance matrix:

$$p(\boldsymbol{\theta}|D) \approx \mathcal{N}(\hat{\boldsymbol{\theta}}, \boldsymbol{H}^{-1}). \tag{45}$$

Computing $\boldsymbol{H}^{-1}$ is usually intractable for large neural network architectures. Thus, approximations are used, most of the time

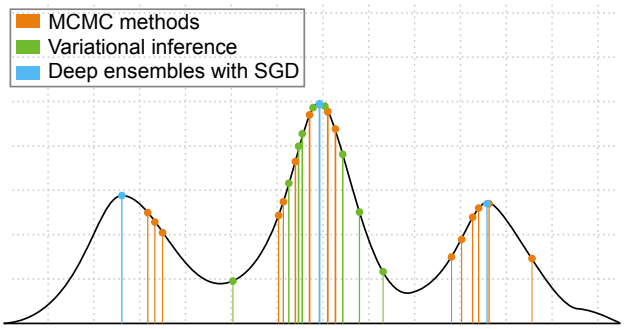


Fig. 10: Different techniques for sampling the posterior. MCMC algorithms sample the true posterior but successive samples might be correlated, Variational Inference uses a parametric distribution that can suffer from mode collapse while deep ensembles focus on the modes of the distribution.

Algorithm 8 Deep ensembles.

for $i = 0$ **to** R **do**
  Draw $\boldsymbol{\theta}_0 \sim$ *Initial probability distribution*;
  $\epsilon_t = \epsilon_0$
  for $j = 0$ **to** N **do**
    $f(\boldsymbol{\theta}_{i,j}) = \log(p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta}_{i,j})) + \log(p(\boldsymbol{\theta}_{i,j}))$;
    $\Delta_\theta f = \text{backprop}_\theta(f)$;
    $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - \alpha_{\boldsymbol{\theta}} \Delta_\theta f$;
  end for
end for

by analysing the variance of the gradient descent algorithm [88, 89, 103]. However, if those methods are able to capture the fine shape of one mode of the posterior, they cannot fit multiple modes.

Lakshminarayanan *et al.* [102] proposed using warm restarts to obtain different point estimate networks instead of fitting a parametric distribution. This method, called deep ensembles; see Figure 10 and Algorithm 8, has been used in the past to perform model averaging. The main contribution of [102] was to show that it enables well-calibrated error estimates. While Lakshminarayanan *et al.* [102] claim that their method is non-Bayesian, it has been shown that their approach can still be understood from a Bayesian point of view [12, 104]. When regularization is used, the different point estimates should correspond to modes of a Bayesian posterior. This can be interpreted as approximating the posterior with a distribution parametrized as multiple Dirac deltas, *i.e.*,

$$q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) = \sum_{\boldsymbol{\theta}_i \in \boldsymbol{\phi}} \alpha_{\boldsymbol{\theta}_i} \delta_{\boldsymbol{\theta}_i}(\boldsymbol{\theta}), \tag{46}$$

with the $\alpha_{\boldsymbol{\theta}_i}$ being positive constants such that their sum is equal to one. This approach can be seen as a form of variational inference. Note however that, for a variational distribution containing Dirac deltas, computing the ELBO in a sense that is meaningful for traditional optimization is impossible.

VI. Simplifying Bayesian Neural networks

After training a BNN, one has to use Monte Carlo at evaluation time to estimate uncertainty. This is a major drawback of BNNs. For MCMC-based methods, storing a large set of parametrizations Θ is also not practical. This section presents mitigation strategies reported in the literature.

A. Bayesian inference on the (n-)last layer(s) only

The architecture of deep neural networks makes it quite redundant to account for uncertainty for a large number of successive layers. Instead, recent research aims to use only a few stochastic layers, usually positioned at the end of the networks [105, 106]; see Figure 11. With only a few stochastic layers, training and evaluation can be drastically sped up while still obtaining meaningful results from a Bayesian perspective. This approach can be seen as learning a point estimate transformation followed by a shallow BNN.

Training a BNN with some non-stochastic layers is similar to learning the parameters for the prior presented in Section V-D. The weights of the non-Bayesian layers should be considered as both prior and variational-posterior parameters.

B. Bayesian teachers

Using a BNN as a teacher is an idea derived from an approach used in Bayesian modeling [107]. The approach is to train a non-stochastic ANN to predict the marginal probability $p(\boldsymbol{y}|\boldsymbol{x}, D)$ using a BNN as a teacher [108]. This is related to the idea of knowledge distillation [109, 110] where possibly several pre-trained knowledge sources can be used to train a more functional system.

To do so, the KL-divergence between a parametric distribution $q_{\boldsymbol{\omega}}(\boldsymbol{y}|\boldsymbol{x})$, where $\boldsymbol{\omega}$ are the coefficients of the student network, and $p(\boldsymbol{y}|\boldsymbol{x}, D)$ is minimized:

$$\hat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}} D_{KL}(p(\boldsymbol{y}|\boldsymbol{x}, D)||q_{\boldsymbol{\omega}}(\boldsymbol{y}|\boldsymbol{x})). \tag{47}$$

As this is intractable, Korattikara *et al.* [108] proposed a Monte Carlo approximation:

$$\hat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}} -\frac{1}{|\Theta|} \sum_{\boldsymbol{\theta}_i \in \Theta} \mathrm{E}_{p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_i)} \left[\log\left(q_{\boldsymbol{\omega}}(\boldsymbol{y}|\boldsymbol{x})\right)\right]. \tag{48}$$

Here, $\hat{\boldsymbol{\omega}}$ can be estimated using a training dataset D' that contains only the features $\boldsymbol{x}$. During training, the probability $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ of the labels is given by the teacher BNN. Thus, D' can be much larger than D. This helps the student network retain the calibration and uncertainty from the teacher.

Menon *et al.* [110] observed that, for classification problems, simply using the class probabilities output by a BNN teacher rather than one-hot labels helps the student to retain calibration and uncertainty from the teacher.

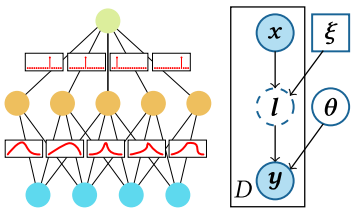


Fig. 11: PGM and BNN corresponding to a last-layer model.

A Bayesian teacher can also be used to compress a large set of samples generated using MCMC [111]. Instead of storing Θ, a generative model G (*e.g.*, a GAN in [111]) is trained against the MCMC samples to generate the coefficients θ_i at evaluation time. This approach is similar to variational inference, with G representing a parametric distribution, but the proposed algorithm allows training a much more complex model than the distributions usually considered for variational inference.

VII. Performance metrics of Bayesian Neural Networks

One big challenge with BNNs is how to evaluate their performance. They do not directly output a point estimate prediction $\hat{\boldsymbol{y}}$ but a conditional probability distribution $p(\boldsymbol{y}|\boldsymbol{x}, D)$, from which an optimal estimate $\hat{\boldsymbol{y}}$ can later be extracted. This means that both the predictive performance, *i.e.*, the ability of the model to give correct answers, and the calibration, *i.e.*, that the network is neither overconfident nor underconfident about its prediction, have to be assessed.

The predictive performance, sometimes called sharpness in statistics, of a network can be assessed by treating the estimator $\hat{\boldsymbol{y}}$ as the prediction. This procedure often depends on the type of data the network is meant to treat. Many different metrics, *e.g.*, mean square error (MSE), ℓ_n distances and cross-entropy, are used in practice. Covering these metrics is out of the scope of this tutorial. Instead, we refer the reader to [112] for more details.

The standard method to assess **the model calibration** is a calibration curve, also called a reliability diagram [32, 113]. It is defined as a function $\check{p} : [0, 1] \to [0, 1]$ that represents the observed probability $\check{p}$, or empirical frequency, as a function of the predicted probability $\hat{p}$; see Figure 12. If $\check{p} < \hat{p}$, then the model is overconfident. Otherwise, it is underconfident. A well-calibrated model should have $\check{p} \cong \hat{p}$. Using this approach requires to first choose a set of events $\mathcal{E}$ with different predicted probabilities and then to measure the empirical frequency of each event using a test set T.

For a binary classifier, the set of test events can be chosen as the set of all sets of datapoints with predicted probabilities of acceptance in interval $[p - \delta, p + \delta]$ for a chosen δ, or alternatively $[0, p]$ or $[1-p, 1]$ for small datasets. The empirical frequency is given by:

$$\check{p} = \frac{\sum_{\check{\boldsymbol{y}} \in T_{\boldsymbol{y}}} \check{\boldsymbol{y}} \cdot \mathbb{I}_{[\hat{p}-\delta, \hat{p}+\delta]}(\hat{\boldsymbol{y}})}{\sum_{\check{\boldsymbol{y}} \in T_{\boldsymbol{y}}} \mathbb{I}_{[\hat{p}-\delta, \hat{p}+\delta]}(\hat{\boldsymbol{y}})}. \tag{49}$$

For multiclass classifiers, the calibration curve can be independently checked for each class against all the other classes. In this case, the problem is reduced to a binary classifier.

Regression problems are slightly more complex since the network does not output a confidence level, as in a classifier, but a distribution of possible outputs. The solution is to use an intermediate statistic with a known probability distribution. Assuming independence between the $\hat{\boldsymbol{y}}$ for a sufficiently large set of different randomly selected inputs $\boldsymbol{x}$, one can assume that the normalized sum of squared residuals (NSSR) follows a Chi-square law:

$$\text{NSSR} = (\hat{\boldsymbol{y}} - \check{\boldsymbol{y}})^{\intercal} \boldsymbol{\Sigma}_{\hat{\boldsymbol{y}}}^{-1} (\hat{\boldsymbol{y}} - \check{\boldsymbol{y}}) \sim \chi^2_{Dim(\boldsymbol{y})}. \tag{50}$$

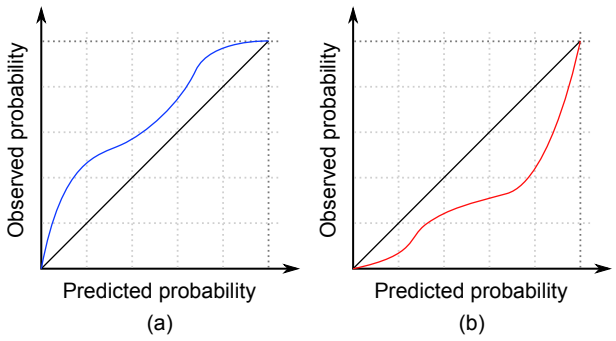


Fig. 12: Examples of calibration curves for underconfident (a) and overconfident (b) models.

This allows attributing to each data point in the test set T a predicted probability that is the probability of observing a variance-normalized distance between the prediction and the true value equal to or lower than the measured NSSR. Formally, the predicted probability is computed as:

$$\hat{p}_i = X^2_{\text{Dim}(\boldsymbol{y})}(\text{NSSR}) \quad \forall (\boldsymbol{y}_i, \boldsymbol{x}_i) \in T, \tag{51}$$

where $X^2_{\text{Dim}(\boldsymbol{y})}$ is the Chi-square cumulative distribution, with $\text{Dim}(\boldsymbol{y})$ degrees of freedom. The observed probability can be computed as:

$$\check{p}_i = \frac{1}{|T|} \sum_{j=1}^{|T|} \mathbb{I}_{[0,\infty)}(\hat{p}_j - \hat{p}_i). \tag{52}$$

We present in the Supplementary Material a practical computation of such calibration curve for the sparse measure practical example (Practical example II).

Giving the whole calibration curve for a given stochastic model allows observing where the model is likely to be overconfident or underconfident. It also allows, to a certain extent, to recalibrate the model [113]. However, providing a summary measure to ease comparison or interpretation might also be necessary. The area under the curve (AUC) is a standard metric of the form:

$$\text{AUC} = \int_0^1 \check{p} d\hat{p}. \tag{53}$$

An AUC of 0.5 indicates that the model is, on average, well calibrated.

The distance from the actual calibration curve to the ideal calibration curve is also a good indicator for the calibration of a model:

$$d(\check{p}, \hat{p}) = \sqrt{\int_0^1 (\check{p} - \hat{p})^2 d\hat{p}}. \tag{54}$$

When $d(\check{p}, \hat{p}) = 0$, then the model is perfectly calibrated.

Other measures have also been proposed. Examples include the expected calibration error and some discretized variants of the distance from the actual calibration curve to the ideal calibration curve [16].

VIII. Conclusion

This tutorial covers the design, training and evaluation of BNNs. While their underlying principle is simple, *i.e.*, just training an ANN with some probability distribution attached to its weights, designing efficient algorithms remains very challenging. Nonetheless, the potential applications of BNNs are huge. In particular, BNNs constitute a promising paradigm allowing the application of deep learning in areas where a system is not allowed to fail to generalize without emitting a warning. Finally, Bayesian methods can help design new learning and regularization strategies. Thus, their relevance extends to traditional point estimate models.

Online resources for the tutorial:
https://github.com/french-paragon/BayesianNeuralNetwork-Tutorial-Metarepos
Supplementary material, as well as additional practical examples for the covered material with the corresponding source code implementation, have been provided.

IX. Acknowledgments

This material is partially based on research sponsored by the Australian Research Council https://www.arc.gov.au/ (Grants DP150100294 and DP150104251), and Air Force Research Laboratory and DARPA https://afrl.dodlive.mil/tag/darpa/ under agreement number FA8750-19-2-0501.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[2] Q. Rao and J. Frtunikj, "Deep learning for self-driving cars: Chances and challenges," in *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, ser. SEFAIS '18, 2018, pp. 35–38.

[3] J. Ker, L. Wang, J. Rao, and T. Lim, "Deep learning applications in medical image analysis," *IEEE Access*, vol. 6, pp. 9375–9389, 2018.

[4] R. C. Cavalcante, R. C. Brasileiro, V. L. Souza, J. P. Nobrega, and A. L. Oliveira, "Computational intelligence and financial markets: A survey and future directions," *Expert Systems with Applications*, vol. 55, pp. 194–211, 2016.

[5] H. M. D. Kabir, A. Khosravi, M. A. Hosen, and S. Nahavandi, "Neural network-based uncertainty quantification: A survey of methodologies and applications," *IEEE Access*, vol. 6, pp. 36 218–36 234, 2018.

[6] A. Etz, Q. F. Gronau, F. Dablander, P. A. Edelsbrunner, and B. Baribault, "How to become a Bayesian in eight easy steps: An annotated reading list," *Psychonomic Bulletin & Review*, vol. 25, pp. 219–234, 2018.

[7] N. G. Polson, V. Sokolov *et al.*, "Deep learning: a Bayesian perspective," *Bayesian Analysis*, vol. 12, no. 4, pp. 1275–1304, 2017.

[8] J. Lampinen and A. Vehtari, "Bayesian approach for neural networks—review and case studies," *Neural Networks*, vol. 14, no. 3, pp. 257 – 274, 2001.

[9] D. M. Titterington, "Bayesian methods for neural networks and related models," *Statist. Sci.*, vol. 19, no. 1, pp. 128–139, 02 2004.

[10] E. Goan and C. Fookes, *Bayesian Neural Networks: An Introduction and Survey*. Cham: Springer International Publishing, 2020, pp. 45–87.

[11] H. Wang and D.-Y. Yeung, "A survey on bayesian deep learning," *ACM Comput. Surv.*, vol. 53, no. 5, Sep. 2020.

[12] A. G. Wilson and P. Izmailov, "Bayesian deep learning and a probabilistic perspective of generalization," *CoRR*, vol. abs/2002.08791, 2020. [Online]. Available: http://arxiv.org/abs/2002.08791

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[14] S. C.-H. Yang, W. K. Vong, R. B. Sojitra, T. Folke, and P. Shafto, "Mitigating belief projection in explainable artificial intelligence via bayesian teaching," *Scientific Reports*, vol. 11, no. 1, p. 9863, May 2021.

[15] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural network," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, 2017, pp. 1321–1330.

[16] J. Nixon, M. W. Dusenberry, L. Zhang, G. Jerfel, and D. Tran, "Measuring calibration in deep learning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[17] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *5th International Conference on Learning Representations, ICLR 2017, Conference Track Proceedings*, 2017.

[18] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, 1st ed. Chapman and Hall/CRC, 2012.

[19] F. Galton, "Vox Populi," *Nature*, vol. 75, no. 1949, pp. 450–451, Mar 1907.

[20] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug 1996.

[21] D. J. C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[22] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson, "What are Bayesian neural network posteriors really like?" *CoRR*, vol. abs/2104.14421, 2021. [Online]. Available: http://arxiv.org/abs/2104.14421

[23] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with Bernoulli approximate variational inference," in *4th International Conference on Learning Representations (ICLR) workshop track*, 2016.

[24] D. P. Kingma and M. Welling, "Stochastic gradient vb and the variational auto-encoder," in *Second International Conference on Learning Representations, ICLR*, vol. 19, 2014.

[25] C. Robert, *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

[26] J. Mitros and B. M. Namee, "On the validity of Bayesian neural networks for uncertainty estimation," in *AICS*, 2019.

[27] A. Kristiadi, M. Hein, and P. Hennig, "Being Bayesian, even just a bit, fixes overconfidence in ReLU networks," *CoRR*, vol. abs/2002.10118, 2020. [Online]. Available: http://arxiv.org/abs/2002.10118

[28] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 13 991–14 002.

[29] A. D. Kiureghian and O. Ditlevsen, "Aleatory or epistemic? does it matter?" *Structural Safety*, vol. 31, no. 2, pp. 105–112, 2009, risk Acceptance and Risk Communication.

[30] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft, "Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 1184–1193.

[31] D. H. Wolpert, "The lack of a priori distinctions between

learning algorithms," *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[32] A. Kendall and Y. Gal, "What uncertainties do we need in Bayesian deep learning for computer vision?" in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, 2017, p. 5580–5590.

[33] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for internet traffic classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, 2007.

[34] X. Zhang and S. Mahadevan, "Bayesian neural networks for flight trajectory prediction and safety assessment," *Decision Support Systems*, vol. 131, p. 113246, 2020.

[35] S. Arangio and F. Bontempi, "Structural health monitoring of a cable–stayed bridge with Bayesian neural networks," *Structure and Infrastructure Engineering*, vol. 11, no. 4, pp. 575–587, 2015.

[36] S. M. Bateni, D.-S. Jeng, and B. W. Melville, "Bayesian neural networks for prediction of equilibrium and time-dependent scour depth around bridge piers," *Advances in Engineering Software*, vol. 38, no. 2, pp. 102–111, 2007.

[37] X. Zhang, F. Liang, R. Srinivasan, and M. Van Liew, "Estimating uncertainty of streamflow simulation using bayesian neural networks," *Water Resources Research*, vol. 45, no. 2, 2009.

[38] A. D. Cobb, M. D. Himes, F. Soboczenski, S. Zorzan, M. D. O'Beirne, A. G. Baydin, Y. Gal, S. D. Domagal-Goldman, G. N. Arney, and D. A. and, "An ensemble of bayesian neural networks for exoplanetary atmospheric retrieval," *The Astronomical Journal*, vol. 158, no. 1, p. 33, jun 2019.

[39] F. Aminian and M. Aminian, "Fault diagnosis of analog circuits using Bayesian neural networks with wavelet transform as preprocessor," *Journal of Electronic Testing*, vol. 17, no. 1, pp. 29–36, Feb 2001.

[40] W. Beker, A. Wołos, S. Szymkuć, and B. A. Grzybowski, "Minimal–uncertainty prediction of general drug–likeness based on Bayesian neural networks," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 457–465, Aug 2020.

[41] Y. Gal, R. Islam, and Z. Ghahramani, "Deep Bayesian active learning with image data," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, 2017, p. 1183–1192.

[42] T. Tran, T.-T. Do, I. Reid, and G. Carneiro, "Bayesian generative active deep learning," *CoRR*, vol. abs/1904.11643, 2019. [Online]. Available: http://arxiv.org/abs/1904.11643

[43] M. Opper and O. Winther, "A Bayesian approach to on-line learning," *On-line learning in neural networks*, pp. 363–378, 1998.

[44] H. Ritter, A. Botev, and D. Barber, "Online structured Laplace approximations for overcoming catastrophic forgetting," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, 2018, pp. 3742–3752.

[45] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, vol. 51, no. 5, Sep. 2018.

[46] W. L. Buntine, "Operations for learning with graphical models," *Journal of Artificial Intelligence Research*, vol. 2, pp. 159–225, Dec 1994.

[47] Y. Wen, P. Vicol, J. Ba, D. Tran, and R. Grosse, "Flipout: Efficient pseudo-independent weight perturbations on mini-batches," in *International Conference on Learning Representations*, 2018.

[48] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *5th International Conference on Learning Representations, ICLR*, 2017.

[49] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *Journal of statistical software*, vol. 76, no. 1, 2017.

[50] A. Gelman and other Stan developers, "Prior choice recommendations," 2020, retrieved from https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations [last seen 13.07.2020].

[51] D. Silvestro and T. Andermann, "Prior choice affects ability of Bayesian neural networks to identify unknowns," *CoRR*, vol. abs/2005.04987, 2020. [Online]. Available: http://arxiv.org/abs/2005.04987

[52] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[53] A. A. Pourzanjani, R. M. Jiang, B. Mitchell, P. J. Atzberger, and L. R. Petzold, "Bayesian inference over the Stiefel manifold via the Givens representation," *CoRR*, vol. abs/1710.09443, 2017. [Online]. Available: http://arxiv.org/abs/1710.09443

[54] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. arXiv:1607.06450, 2016, in NIPS 2016 Deep Learning Symposium.

[55] G.-J. Qi and J. Luo, "Small data challenges in big data era: A survey of recent progress on unsupervised and semi-supervised methods," *CoRR*, vol. abs/1903.11260, 2019. [Online]. Available: http://arxiv.org/abs/1903.11260

[56] N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari, "Learning with noisy labels," in *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 1196–1204.

[57] B. Frenay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 5, pp. 845–869, 2014.

[58] A. C. Tommi and T. Jaakkola, "On information regularization," in *In Proceedings of the 19th UAI*, 2003.

[59] K. Sohn, D. Berthelot, C.-L. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, and C. Raffel, "FixMatch: Simplifying semi-supervised learning with consistency and confidence," *CoRR*, vol. abs/2001.07685, 2020. [Online]. Available: https://arxiv.org/abs/2001.07685

[60] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *J. Mach. Learn. Res.*, vol. 7, pp. 2399–2434, Dec. 2006.

[61] S. Yu, B. Krishnapuram, R. Rosales, and R. B. Rao, "Bayesian co-training," *Journal of Machine Learning Research*, vol. 12, no. 80, pp. 2649–2680, 2011.

[62] R. Kunwar, U. Pal, and M. Blumenstein, "Semi-supervised online Bayesian network learner for handwritten characters recognition," in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 3104–3109.

[63] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on challenges in representation learning, ICML*, vol. 3, 2013.

[64] Z. Li, B. Ko, and H.-J. Choi, "Naive semi-supervised deep learning using pseudo-label," *Peer-to-Peer Networking and Applications*, vol. 12, no. 5, pp. 1358–1368, 2019.

[65] M. S. Bari, M. T. Mohiuddin, and S. Joty, "MultiMix: A robust data augmentation strategy for cross-lingual nlp," in *ICML*, 2020.

[66] O. Chapelle, J. Weston, L. Bottou, and V. Vapnik, "Vicinal risk minimization," in *Advances in Neural Information Processing Systems 13*. MIT Press, 2001, pp. 416–422.

[67] Q. Xie, Z. Dai, E. H. Hovy, M. Luong, and Q. V. Le, "Unsupervised data augmentation," *CoRR*, vol. abs/1904.12848, 2019. [Online]. Available: http://arxiv.org/abs/1904.12848

[68] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *CoRR*, vol. abs/2004.05439, 2020. [Online]. Available: http://arxiv.org/abs/2004.05439

[69] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[70] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *CoRR*, vol. abs/2003.08271, 2020.

[71] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[72] X.-F. Han, H. Laga, and M. Bennamoun, "Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 5, pp. 1578–1604, 2021.

[73] H. Laga, L. V. Jospin, F. Boussaid, and M. Bennamoun, "A survey on deep learning techniques for stereo-based depth estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[74] E. Grant, C. Finn, S. Levine, T. Darrell, and T. L. Griffiths, "Recasting gradient-based meta-learning as hierarchical Bayes," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[75] L. Beyer, X. Zhai, A. Oliver, and A. Kolesnikov, "S4L: Self-supervised semi-supervised learning," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1476–1485.

[76] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 04 1970.

[77] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[78] R. Bardenet, A. Doucet, and C. Holmes, "On Markov Chain Monte Carlo methods for tall data," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 1515–1557, Jan. 2017.

[79] E. I. George, G. Casella, and E. I. George, "Explaining the Gibbs sampler," *The American Statistician*, 1992.

[80] S. Chib and E. Greenberg, "Understanding the Metropolis-Hastings algorithm," *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[81] R. M. Neal *et al.*, "MCMC using Hamiltonian dynamics," *Handbook of Markov Chain Monte Carlo*, vol. 2, no. 11, p. 2, 2011.

[82] M. D. Hoffman and A. Gelman, "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

[83] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79 – 86, 1951.

[84] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[85] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 1303–1347, May 2013.

[86] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., 2011, pp. 2348–2356.

[87] Z. Ghahramani and M. J. Beal, "Propagation algorithms for variational Bayesian learning," in *Advances in Neural Information Processing Systems 13*. MIT Press, 2001, pp. 507–513.

[88] H. Ritter, A. Botev, and D. Barber, "A scalable laplace approximation for neural networks," in *International Conference on Learning Representations*, 2018.

[89] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson, "A simple baseline for Bayesian uncertainty in deep learning," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 13 153–13 164.

[90] J. M. Hernández-Lobato and R. P. Adams, "Probabilistic backpropagation for scalable learning of Bayesian neural networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, 2015, p. 1861–1869.

[91] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 37, 2015, pp. 1613–1622.

[92] D. P. Kingma, M. Welling *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[93] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[94] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[95] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on Machine Learning - Volume 48*, ser. ICML'16, 2016, p. 1050–1059.

[96] Y. Li and Y. Gal, "Dropout inference in Bayesian neural networks with alpha-divergences," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, 2017, pp. 2052–2061.

[97] J. Hron, A. Matthews, and Z. Ghahramani, "Variational Bayesian dropout: pitfalls and fixes," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 2019–2028.

[98] A. Chan, A. Alaa, Z. Qian, and M. Van Der Schaar, "Unlabelled data improves Bayesian uncertainty calibration under covariate shift," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 119. Virtual: PMLR, 13–18 Jul 2020, pp. 1392–1402.

[99] S. Mandt, M. D. Hoffman, and D. M. Blei, "Stochastic gradient descent as approximate Bayesian inference," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 4873–4907, 2017.

[100] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient Langevin dynamics," in *Proceedings of the 28th international conference on machine learning*, ser. ICML '11, 2011, pp. 681–688.

[101] N. Seedat and C. Kanan, "Towards calibrated and scalable uncertainty representations for neural networks," *CoRR*, vol. abs/1911.00104, 2019. [Online]. Available: http://arxiv.org/abs/1911.00104

[102] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 6402–6413.

[103] M. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava, "Fast and scalable Bayesian deep learning by weight-perturbation in Adam," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 2611–2620.

[104] T. Pearce, F. Leibfried, A. Brintrup, M. Zaki, and A. Neely, "Uncertainty in neural networks: Approximately Bayesian ensembling," in *AISTATS 2020*, 2020.

[105] J. Zeng, A. Lesnikowski, and J. M. Alvarez, "The relevance of Bayesian layer positioning to model uncertainty in deep Bayesian active learning," *CoRR*, vol. abs/1811.12535, 2018. [Online]. Available: http://arxiv.org/abs/1811.12535

[106] N. Brosse, C. Riquelme, A. Martin, S. Gelly, and Éric

Moulines, "On last-layer algorithms for classification: Decoupling representation from uncertainty estimation," *CoRR*, vol. abs/2001.08049, 2020. [Online]. Available: http://arxiv.org/abs/2001.08049

[107] E. Snelson and Z. Ghahramani, "Compact approximations to Bayesian predictive distributions," in *Proceedings of the 22nd International Conference on Machine Learning*, ser. ICML '05, 2005, p. 840–847.

[108] A. Korattikara, V. Rathod, K. Murphy, and M. Welling, "Bayesian dark knowledge," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, 2015, pp. 3438–3446.

[109] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015, in NIPS 2014 Deep Learning Workshop.

[110] A. K. Menon, A. S. Rawat, S. J. Reddi, S. Kim, and S. Kumar, "Why distillation helps: a statistical perspective," *CoRR*, vol. abs/2005.10419, 2020. [Online]. Available: https://arxiv.org/abs/2005.10419

[111] K.-C. Wang, P. Vicol, J. Lucas, L. Gu, R. Grosse, and R. Zemel, "Adversarial distillation of Bayesian neural network posteriors," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 5190–5199.

[112] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *Schedae Informaticae*, vol. 1/2016, 2017. [Online]. Available: http://dx.doi.org/10.4467/20838476SI.16.004.6185

[113] V. Kuleshov, N. Fenner, and S. Ermon, "Accurate uncertainties for deep learning using calibrated regression," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 2796–2804.

APPENDIX A
IMPLEMENTING PARAMETER-EFFICIENT NORMAL DISTRIBUTIONS FOR VARIATIONAL INFERENCE

Given a random vector $\boldsymbol{\varepsilon}$ in which each independent and identically distributed component follows a standard normal distribution $\mathcal{N}(0,1)$, one can obtain a sample $\boldsymbol{\theta}$ from a normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ using the following formulas:

$$\boldsymbol{\theta} = \sqrt{\boldsymbol{\Sigma}}\boldsymbol{\varepsilon} + \boldsymbol{\mu}, \tag{55}$$

where $\sqrt{\boldsymbol{\Sigma}}$ is a matrix such that $\sqrt{\boldsymbol{\Sigma}}\sqrt{\boldsymbol{\Sigma}}^{\top} = \boldsymbol{\Sigma}$. When a variational inference algorithm needs to learn the covariance matrix of $\boldsymbol{\theta}$, it is often more convenient to learn $\sqrt{\boldsymbol{\Sigma}}$. Assuming $\boldsymbol{\theta}$ has n entries, $\sqrt{\boldsymbol{\Sigma}}\sqrt{\boldsymbol{\Sigma}}^{\top}$ is the Cholesky decomposition of $\boldsymbol{\Sigma}$. As such, $\sqrt{\boldsymbol{\Sigma}}$ is a lower triangular matrix and $\mathcal{O}(n^2)$ variational parameters are required to learn the exact covariance matrix. This becomes exceedingly computationally expensive rather quickly when n becomes large.

A straightforward simplification is to only consider a diagonal approximation of $\boldsymbol{\Sigma}$. This can be enforced by learning only the diagonal coefficients of $\sqrt{\boldsymbol{\Sigma}}$, meaning only $\mathcal{O}(n)$ variational parameters are required (Figure 13a). This approach can be extended to learn more correlation coefficients by learning a block diagonal [88] covariance matrix, which can be done by learning the corresponding lower triangular entries in $\sqrt{\boldsymbol{\Sigma}}$ (Figure 13b). If the maximal size of the nonzero blocks is fixed to be w, $\mathcal{O}(w \cdot n)$ variational parameters are required. The major drawback of this model is that the index of two given parameters determines whether their covariance can be learned by the variational distribution. This is not always ideal, as it is hard to predict which parameters will be the most correlated and need to be positioned close to one another. An alternative is to learn a diagonal plus low rank approximation of $\boldsymbol{\Sigma}$ [89]. This is done by sampling a vector $\boldsymbol{\varepsilon}$ with $n+r$ (instead of n) components. $\sqrt{\boldsymbol{\Sigma}}$ is then defined as:

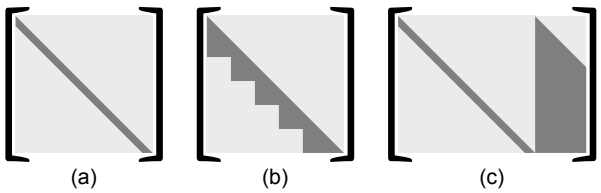


Fig. 13: Nonzero entries in $\sqrt{\boldsymbol{\Sigma}}$ when learning a diagonal (a), block diagonal (b) or diagonal plus low rank (c) approximation of $\boldsymbol{\Sigma}$

$$\sqrt{\boldsymbol{\Sigma}} = [D,\ L]\,, \tag{56}$$

where D is a diagonal matrix of size $n \times n$ and L is a lower triangular matrix of size $n \times r$ (Figure 13c). This means that the model has more flexibility to learn the correlation between all the components of $\boldsymbol{\theta}$ while only requiring $\mathcal{O}(n \cdot r)$ variational parameters.

APPENDIX B
A PROOF OF EQUATION 38

Let us assume that we have a probability space $(\Omega, \mathcal{F}, \mathrm{P})$, where Ω is a set of outcomes, $\mathcal{F}$ is a σ-algebra of Ω representing possible events and P is a measure defined on $\mathcal{F}$ and which assigns a value of 1 to Ω, representing the probability of an event. In addition, assume that we have a probability distribution $q_\phi(\boldsymbol{\theta})$ for a given random variable $\boldsymbol{\theta}$, a probability distribution $q(\varepsilon)$ for a given random variable ε and a functional relation $t(\varepsilon, \boldsymbol{\phi})$ such that $t(\varepsilon, \boldsymbol{\phi})$ is distributed according to $q_\phi(\boldsymbol{\theta})$ and $t(\varepsilon, \boldsymbol{\phi})$ is a bijection with respect to ε. Thus, we have:

$$\mathrm{P}(\boldsymbol{\theta}^{-1}(t(E, \boldsymbol{\phi}))) = \mathrm{P}(\varepsilon^{-1}(E)) \quad \forall E \in \varepsilon(\mathcal{F}), \tag{57}$$

with

$$\varepsilon(\mathcal{F}) = \{\{\varepsilon(\omega) : \omega \in e\} : e \in \mathcal{F}\}\,,$$

$$t(E, \boldsymbol{\phi}) = \{t(\varepsilon, \boldsymbol{\phi}) : \varepsilon \in E\},$$

$$\varepsilon^{-1}(E) = \textstyle\bigcup_{e \in \mathcal{F} \wedge \varepsilon(e) \subseteq E} e,$$

$$\boldsymbol{\theta}^{-1}(t(E, \boldsymbol{\phi})) = \textstyle\bigcup_{e \in \mathcal{F} \wedge \boldsymbol{\theta}(e) \subseteq t(E, \boldsymbol{\phi})} e.$$

Since $t(\varepsilon, \boldsymbol{\phi})$ is a bijection with respect to ε, we have $\varepsilon^{-1}(E) = \boldsymbol{\theta}^{-1}(t(E, \boldsymbol{\phi}))$. This implies:

$$\int_{\boldsymbol{\theta} \in t(E, \boldsymbol{\phi})} q_\phi(\boldsymbol{\theta}) d\boldsymbol{\theta} = \int_{\varepsilon \in E} q(\varepsilon) d\varepsilon \quad \forall E \in \varepsilon(\mathcal{F}). \tag{58}$$

which in turn implies:

$$q_\phi(\boldsymbol{\theta}) d\boldsymbol{\theta} = q(\varepsilon) d\varepsilon \tag{59}$$

for non-degenerated probability distributions $q_\phi(\boldsymbol{\theta})$ and $q(\varepsilon)$.

Now, given a differentiable function $f(\boldsymbol{\phi}, \boldsymbol{\theta})$, we have:

$$\int_{\boldsymbol{\theta} \in t(\varepsilon(\Omega), \boldsymbol{\phi})} f(\boldsymbol{\phi}, \boldsymbol{\theta}) q_{\phi}(\boldsymbol{\theta}) d\boldsymbol{\theta} = \int_{\varepsilon \in \varepsilon(\Omega)} f(\boldsymbol{\phi}, t(\varepsilon, \boldsymbol{\phi})) q(\varepsilon) d\varepsilon \quad (60)$$

which implies Equation (38).

Hands-on Bayesian Neural Networks – Supplementary material

I. Practical example – Bayesian MNIST

Over the years, MNIST [1] has become the most renown toy dataset in deep learning. Coding a handwritten digit classifier based on this dataset is now the Hello World of deep neural network programming. The first practical example we introduce for this tutorial is thus just a plain old classifier for MNIST implemented as a BNN. The code is available on github [https://github.com/french-paragon/BayesianMnist]. The setup is as follows;

The purpose is to show a Bayesian Neural Network (BNN) hello world project.

The problem is to train a BNN to perform hand-written digit recognition.

The dataset we are going to use is MNIST. However, to evaluate how the proposed BNN reacts to unseen data, we will remove one of the classes (we used the digit 5 for this experiment) from the training set so that the network never sees it during training.

The stochastic model is the plain Bayesian regression presented in Section IV-B of the main paper. We use a normal distribution as a prior for the network parameters $\boldsymbol{\theta}$, with a standard deviation of 5 for the weights and 25 for the bias as we expect the scale of the bias to have a slightly more variability than the weights. This is because, in a RELU or leaky-RELU network, the weights influence the local change in slope of a layer function while the bias indicates the position of those inflection points.

The functional model is a standard convolutional neural network with two convolutional layers followed by two fully connected layers; see Figure 1.

A. Training

We used Variational Inference to train this BNN. We use a Gaussian distribution, with diagonal covariance, as a variational posterior (see Section V-C of the main paper). Since we expect the posterior to be multi-modal, we train an ensemble of BNNs instead of a single one (see Section V-E2b of the main paper). Note that the top performing method on the MNIST leaderboard is actually an ensemble-based method [2]. (The authors of [2] designate their model as a multi-column neural network, but the idea is equivalent to an ensemble.)

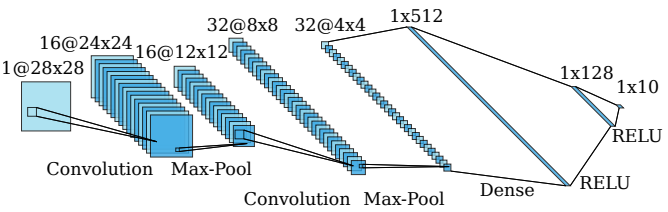


Fig. 1: The neural network architecture used for the Bayesian MNIST practical example.

B. Results

We tested the final BNN against **(1)** the test set restricted to the classes the network has been trained on, **(2)** the test set restricted to the class the network has not been trained on, and **(3)** pure white noise. For each case, we report:

- The average probabilities predicted by the BNN for all input images in the considered class (Fig. 2),
- The average standard deviation of the network on a single sample, and
- The standard deviation of the prediction for all samples.

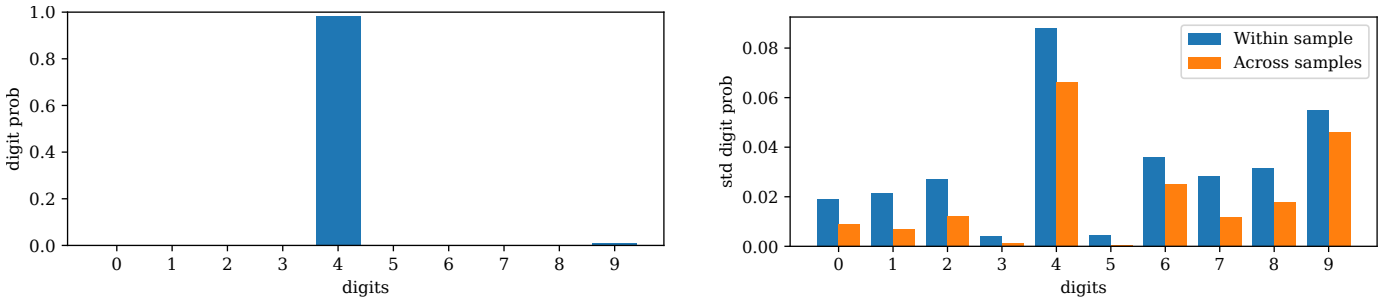
While not as informative and rigorous as a collection of calibration curves, these measures do indicate whether the BNN uncertainty matches the natural variability of the dataset. The reported results show that for a class seen during training, the network gives the correct prediction and is confident about its results. The variability per sample and the variability across samples are coherent with one another and quite small. When presented with a digit from a class unseen during training, the network attempts to match it to digits that share similarities. However, the predicted low probabilities along with the high per-sample and across-sample standard deviation show that the network is aware that it does not know what these unseen digits are. As for the white noise, the average output is constant, meaning the networks is clear about the fact this is not a character.

II. Practical example – Sparse measure

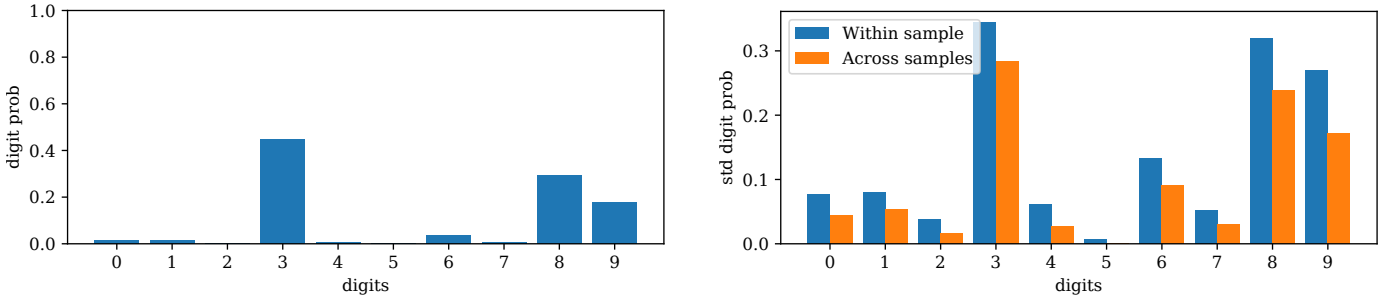
The second practical example we introduce is entitled "Sparse measure". Below, we describe in detail the definition of the stochastic model for this example. We also provide a Python implementation of this example in order to show how the different hypotheses we present in this Appendix translate to actual code [https://github.com/french-paragon/sparse_measure_bnn].

The purpose is to present a small model illustrating how different training strategies can be integrated in a BNN.

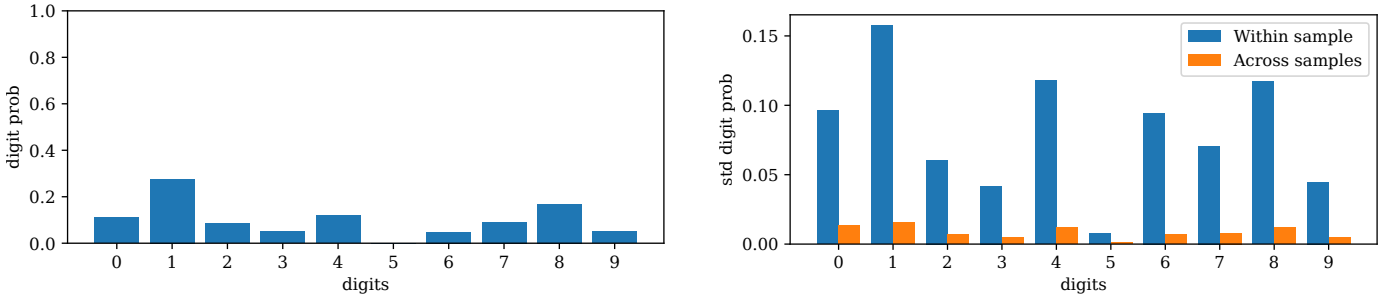
The problem is to learn a function $f : \mathbb{R}^n \to \mathbb{R}^m$ based on a dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) | i \in [1, N], \boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{y}_i \in \mathbb{R}^m\}$ where for each tuple $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, only certain elements of $\boldsymbol{y}_i$ are actually measured. This scenario corresponds to any application of machine learning where measuring a raw signal, represented by the $\boldsymbol{x}_i$'s, is easy but one wants to train an algorithm to reconstruct a derived signal, represented by the $\boldsymbol{y}_i$'s, which is much harder to acquire. On top of that, the elements of $\boldsymbol{y}_i$ are measured with a certain level of uncertainty. Also, one



(a) Results on a seen class



(b) Results on the unseen class



(c) Results on white noise

Fig. 2: Average prediction of the Bayesian MNIST practical example for (a) one of the seen class, (b) the unseen class and (c) white noise input.

element of $\boldsymbol{y}_i$, which is constant but unknown, has a much higher level of uncertainty (*e.g.*, it has been found that one of the instruments used for data acquisition is defective, but no one remembers for which experiment it has been used). It is also known that the function f is continuous and will, with very high probability, map any given input to a point near a known hyperplane of $\mathbb{R}^m$.

The dataset is generated at random for each experiment using python. Inputs are sampled from a zero-mean multivariate normal distribution with a random correlation matrix. The input is then multiplied by a random projection matrix to $\mathbb{R}^3$ before a non-linear function is applied. A random set of three orthonormal vectors in $\mathbb{R}^m$ is then generated and used to reproject the values in the final output space, constrained on a known hyperplane. A final non-linear function is applied on the input and its results are added to the previously generated outputs with a very low gain factor to ensure that the actual output is still likely to be near the known hyperplane. Finally, noise is added to the training data, with one channel receiving much more noise than the other channels. We set $n = m = 9$. The measured values for the training set are then selected at random, the other values being set to 0. A set of binary masks are returned with the data to indicate which values have actually been measured.

The stochastic model is the most interesting part of this case study. Note that learning one function $f : \mathbb{R}^n \to \mathbb{R}^m$ is equivalent to learning m functions $f : \mathbb{R}^n \to \mathbb{R}$. This means that a standard Bayesian regression would be sufficient to address this learning problem. However, one can do much more since the functions under consideration can be correlated with one another.

First, the model can be extended to account for the **noisy labels**. To learn which output channel is actually more noisy than the others, a **meta-learning** approach can be used with **hierarchical Bayes** applied not on the model parameters but on the noise model. If applied to standard point estimate networks, this approach would be seen as learning a loss. The prior can be extended with a **consistency condition** to account for the fact that the unknown function projects points near a known hyperplane. Last, but not least, a **semi-supervised**

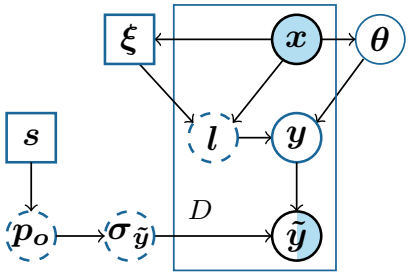


Fig. 3: BBN corresponding to the stochastic model used for the sparse measures example.

learning model can be used to share information across the training samples. The PGM corresponding to this approach is depicted in Fig. 3. We kept the plate for the dataset D as we will use a second consistency condition to implement the semi-supervised learning strategy. We also chose to use a last layer-only BNN, mostly to illustrate how it can be done. For the hierarchy of noise quantification variables, $\boldsymbol{p_o}$ and $\boldsymbol{\sigma_{\tilde{y}}}$, only point estimates will be considered to simplify the model.

The functional model $\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}$ is a feedforward artificial neural network with five hidden layers (Fig. 4). The first four layers are point-estimate layers, with parameters $\boldsymbol{\xi}$, while the last hidden layer and the output layer are stochastic, with parameters $\boldsymbol{\theta}$. We also have three Monte-Carlo dropout layers at the end of the network. The hourglass shape of the network is supposed to match the expected behaviour of the studied function, where the information is approximately located on a lower dimensional space.

A. Base stochastic model

The base stochastic model is built as a Bayesian regression with a last-layer BNN architecture. The base probability of $\boldsymbol{\theta}$, without the consistency conditions that will be applied later on, has been set to a normal distribution with mean 0 and a standard deviation of 1 for the weights and 10 for the bias. The prior distribution of the output $\boldsymbol{y}$ knowing the input $\boldsymbol{x}$ and the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\xi}$ is then given by:

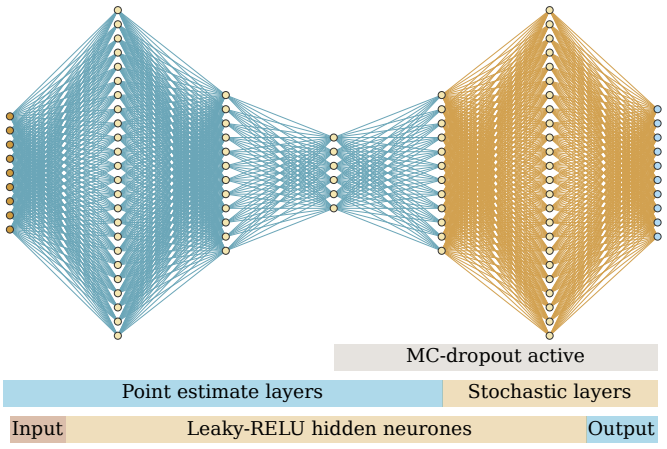


Fig. 4: The neural network architecture used for the sparse measure example.

$$p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\xi}) = \mathcal{N}(\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x}), \sigma_{\boldsymbol{y}}). \tag{1}$$

Here, $\sigma_{\boldsymbol{y}}$, which we set to 0.1, represent the small uncertainty due to the fact that the functional model will never be able to perfectly fit the actual function f.

B. Noisy labels model

For this regression problem, we assume that the measurements have been corrupted by a zero-mean Gaussian noise. This is not only a convenient formulation but also a reasonable approximation of the true noise model for continuous measurements (due to the central limit theorem). The mean of the noisy measurements $\tilde{\boldsymbol{y}}$ is thus the unobserved, true function value y. For the standard deviation $\boldsymbol{\sigma_{\tilde{y}}}$, we assume that it is $\sigma_{in} = 0.1$ for inliers and $\sigma_{out} = 5$ for outliers. The problem here is that we do not know which channel has been corrupted by the additional noise. We could just set for all channels a constant standard deviation that is slightly above σ_{in}, but instead we added an additional variable, $\boldsymbol{p_o}$, a vector representing the probability that a given channel is an outlier. Since a single channel is corrupted by noise, we could constrain $\boldsymbol{p_o}$ to lie on the probability simplex. Instead, to let the model generalize to cases with more than a single noisy channel, we will impose that $\boldsymbol{p_o} \in [0,1]^m$. For convenience, we will consider an unconstrained variable $\boldsymbol{s} \in \mathbb{R}^m$ and define $\boldsymbol{p_o}$ as a function of $\boldsymbol{s}$ to simplify the optimisation:

$$\boldsymbol{p_o} = \frac{1}{1+e^{-\boldsymbol{s}}}. \tag{2}$$

The value of σ is then determined by a Bernoulli distribution with parameter $\boldsymbol{p_o}$:

$$\begin{aligned} \boldsymbol{b} &\sim \text{Bernoulli}(\boldsymbol{p_o}), \\ \boldsymbol{\sigma_{\tilde{y}}} &= \boldsymbol{b}\cdot\sigma_{out} + (1-\boldsymbol{b})\cdot\sigma_{in}. \end{aligned} \tag{3}$$

The only thing left is to determine the prior for $\boldsymbol{s}$. Assuming we have no idea which channel is noisy and which is not, a reasonable hypothesis is a Gaussian with mean 0 and large covariance matrix $\boldsymbol{\Sigma_s}$ (we choose $\boldsymbol{\Sigma_s} = 2500\boldsymbol{I}$)). Yet, since we know that only one channel is noisy, we can add a consistency condition to enforce that $\|\boldsymbol{p_o}\|_1 \approx 1$, or any expected number of noisy channels:

$$p(\boldsymbol{s}) \propto \mathcal{N}(0,\boldsymbol{\Sigma_s}) exp\left[-\gamma_s\left(\left\|\frac{1}{1+e^{-\boldsymbol{s}}}\right\|_1 - 1\right)^2\right]. \tag{4}$$

Here, γ_s is a scaling factor representing the confidence one has in the model capacity to fit the correct number of noisy channels. We set $\gamma_s = 3000$ as we are almost certain the model should be able to fit the single noisy channel. The probability of $\tilde{\boldsymbol{y}}$ given $\boldsymbol{y}$ and $\boldsymbol{\sigma_{\tilde{y}}}$ is then given by:

$$p(\tilde{\boldsymbol{y}}|\boldsymbol{y},\boldsymbol{\sigma_{\tilde{y}}}) = \mathcal{N}(\boldsymbol{y},\boldsymbol{\sigma_{\tilde{y}}}). \tag{5}$$

Since we are not so interested in the complete posterior distribution for $\boldsymbol{s}$ or even $\boldsymbol{\sigma_{\tilde{y}}}$, we will learn point estimates of those variables instead and then derive a regularization term from the log of Equation (4). The final contribution for the

learnable parameters s to the total loss is $\log(p(\boldsymbol{s}))$ plus an unknown constant that one can just ignore:

$$\frac{1}{2}\left(\boldsymbol{s}^\top \boldsymbol{\Sigma}_{\boldsymbol{s}}^{-1}\boldsymbol{s}\right) + \gamma_s\left(\left\|\frac{1}{1+e^{-\boldsymbol{s}}}\right\|_1 - 1\right)^2. \tag{6}$$

This is implemented in the —outlierAwareSumSquareError— learnable loss in the included Python implementation.

C. Consistency condition

The function f is likely to map a given input near a known subspace of $\mathbb{R}^m$, which can be represented as span($\boldsymbol{S}$) where $\boldsymbol{S} \in \mathbb{R}^{m\times 3}$ is a given orthonormal matrix. The distance $d_{\boldsymbol{y},\boldsymbol{S}}$ between a prediction $\boldsymbol{y}$ and span($\boldsymbol{S}$) is then given by:

$$d_{\boldsymbol{y},\boldsymbol{S}} = \left\|\boldsymbol{y} - \boldsymbol{S}\boldsymbol{S}^\top \boldsymbol{y}\right\|_2. \tag{7}$$

We assume a priori that $d_{\boldsymbol{y},\boldsymbol{S}}$ knowing $\boldsymbol{x}$ follows a normal distribution. We then use this to derive a consistency condition for $p(\boldsymbol{\theta},\boldsymbol{\xi}|D_{\boldsymbol{x}})$:

$$p(\boldsymbol{\theta},\boldsymbol{\xi}|D_{\boldsymbol{x}}) \propto p(\boldsymbol{\theta})p(\boldsymbol{\xi}) e^{-\sum_{\boldsymbol{x}\in D_{\boldsymbol{x}}} \frac{1}{\sigma_d^2}\left\|\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{S}\boldsymbol{S}^\top\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x})\right\|_2^2} \tag{8}$$

where σ_d is the prior standard deviation of $d_{\boldsymbol{y},\boldsymbol{S}}$, which is set to 1.

The additional contribution to the loss by the consistency condition is thus:

$$\sum_{\boldsymbol{x}\in D_{\boldsymbol{x}}} \frac{1}{\sigma_d^2}\left\|\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{S}\boldsymbol{S}^\top\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x})\right\|_2^2. \tag{9}$$

D. Semisupervised learning strategy

As stated in Section IV-D1 of the main paper, there are two ways one can implement a semi-supervised learning strategy for a BNN. The first one is to use a consistency condition (*i.e.*, a data-driven regularization). The second one is to assume some kind of dependence across the samples. We use the former for this practical example as data-driven regularization is generally easier to implement.

The intuition behind this example is that if two points from the input space are close to one another then their image by f should also be close to one another. The simplest approach to formally measure this is to assume that the norm of the difference in the output space normalized by the distance in the input space is small. This sets the consistency condition function $C(\boldsymbol{\theta},\boldsymbol{\xi},\boldsymbol{x})$ to:

$$C(\boldsymbol{\theta},\boldsymbol{\xi},\boldsymbol{x}) = \frac{1}{2}\sum_{\boldsymbol{x}'\in D_{\boldsymbol{x}}\setminus\{\boldsymbol{x}\}} \frac{\|\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x}) - \Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x}')\|_2^2}{\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2}. \tag{10}$$

Averaging this consistency condition over the whole training set $D_{\boldsymbol{x}}$ is equivalent to computing the sum over all output channels of $\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}$ of the quadratic form of the Laplacian matrix of the dense graph spanned by the training set $D_{\boldsymbol{x}}$, where the weights of the edges are given by the inverse Euclidean distances of the points. Graph Laplacian regularization is itself a common method to implement semi-supervised learning [3].

This approach can be further improved. Assume that the differences between the components of two random input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$, as well as the differences between the components of their images by f, are normally distributed, then:

$$\frac{\sigma_{\Delta x}^2\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|_2^2}{\sigma_{\Delta f}^2\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2} \sim F(n,m), \tag{11}$$

where $F(n,m)$ is a Fisher–Snedecor distribution with parameters n and m, $\sigma_{\Delta x}$ is the prior standard deviation of the difference of two random inputs, and $\sigma_{\Delta f}$ is the prior standard deviation of the difference of the corresponding outputs. As shown in Fig. 5, such assumptions are not perfect but still match the data reasonably well, especially since those assumptions are just a prior and not an exact model. The most important point to notice is that both the observed and prior distributions have a heavy tail. This would not be the case with the consistency condition proposed in Equation (10). This naive approach would penalize outliers too heavily. We thus implemented the following, corrected, consistency condition function in the sparse measures example:

$$\sum_{\boldsymbol{x}'\in D_{\boldsymbol{x}}\setminus\{\boldsymbol{x}\}} \left(\frac{n+m}{2} - 1\right) log(1+F) - \left(\frac{n}{2}-1\right)\log(F), \tag{12}$$

where $F = \lambda_\Delta \dfrac{\|\Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x}) - \Phi_{\boldsymbol{\xi},\boldsymbol{\theta}}(\boldsymbol{x}')\|_2^2}{\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2}$ is the Fisher statistic.

The only parameter one has to define for this prior is λ_Δ, which is the ratio of $\sigma_{\Delta x}^2$ and $\sigma_{\Delta f}^2$. We set $\lambda_\Delta = 1$.

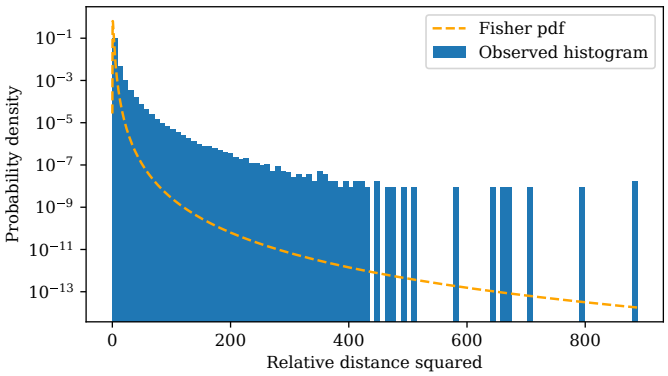


Fig. 5: Comparison between the observed distribution of relative distances squared in the test set and the Fisher–Snedecor distribution of parameters 9 and 9.

E. Results

The estimator for $\boldsymbol{y}$ reaches a RMSE of around 0.4 with an error distribution approximating a normal distribution. Variations can be observed over mutliple runs, as a different function f is generated at random each times (Fig. 6). The Python code also computes the calibration curve using the hypothesis for regression models presented in Section VII of the main paper. The results (Fig. 7) show that the model tends to be slightly underconfident, except for large outliers where the model is overconfident. This is probably due to the fact that our simple variational inference model cannot fit the exact posterior and thus tends to be slightly too pessimistic about

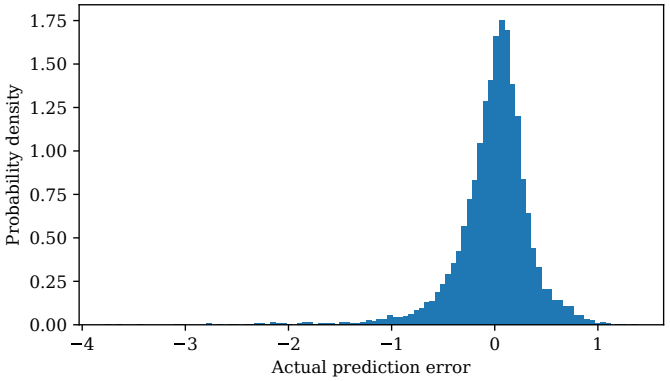


Fig. 6: Prediction error distribution for the sparse measure example.

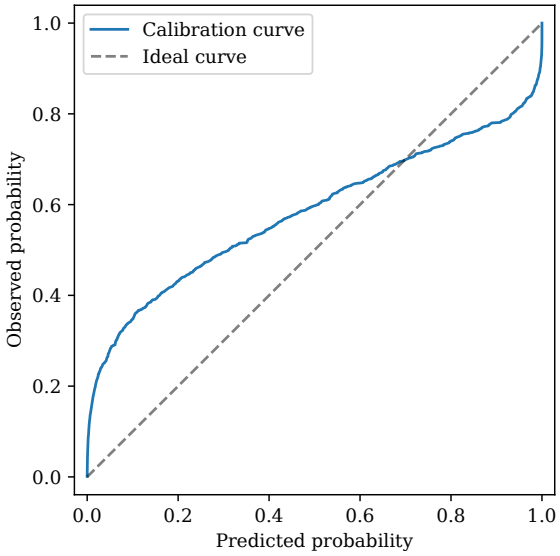


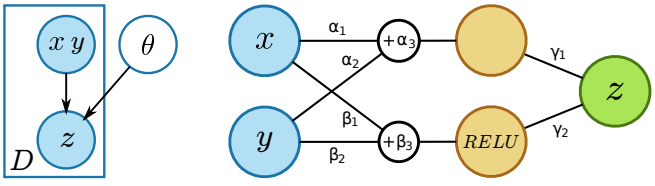
Fig. 7: Calibration curve for the sparse measure example.

its predictions to compensate for the outliers it is unable to efficiently fit.

III. Practical example – Paperfold

The last practical example we introduce is entitled "Paperfold". The code is available on github [https://github.com/french-paragon/paperfold-bnn].

The purpose of this case study is to compare different inference approaches for BNNs. We introduce a model, and a corresponding dataset, which is small enough to make all existing training methods for BNNs tractable when used for the corresponding architecture. This makes the number of parameters small enough such that the posterior and the corresponding approximations are low dimensional. Thus, they can be easily visualized and sampled without issues using exact MCMC methods. On the other hand, the model is complex enough to display most of the issues one would encounter when studying a BNN posterior.



(a) (b)

Fig. 8: Stochastic (a) and functional (b) model for the paperfold case study.

The problem is to fit a one dimensional function of two parameters of the form $z = f(x, y)$.

The dataset is made of eight (x, y, z) samples grouped into the vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ with:

$$\boldsymbol{x} = \begin{pmatrix} \sqrt{2} \\ \frac{1}{2}\sqrt{2} \\ 0 \\ -\frac{1}{2}\sqrt{2} \\ -\sqrt{2} \\ -\frac{1}{2}\sqrt{2} \\ 0 \\ \frac{1}{2}\sqrt{2} \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} 0 \\ \frac{1}{2}\sqrt{2} \\ \sqrt{2} \\ \frac{1}{2}\sqrt{2} \\ 0 \\ -\frac{1}{2}\sqrt{2} \\ -\sqrt{2} \\ -\frac{1}{2}\sqrt{2} \end{pmatrix}, \quad \boldsymbol{z} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 1 \\ \frac{1}{2} \\ 0 \\ \frac{1}{2} \end{pmatrix}. \tag{13}$$

The stochastic model (Fig. 8a) is the usual fully supervised Baysian regression. We assume each measure in $\boldsymbol{z}$ has a small uncertainty $\varepsilon_i \sim \mathcal{N}(0, \sigma_{\boldsymbol{z}}^2)$, with $\sigma_{\boldsymbol{z}}$ being a small positive standard deviation that one can set depending on the experiment. By default, we assume that $\sigma_{\boldsymbol{z}} = 0.1$.

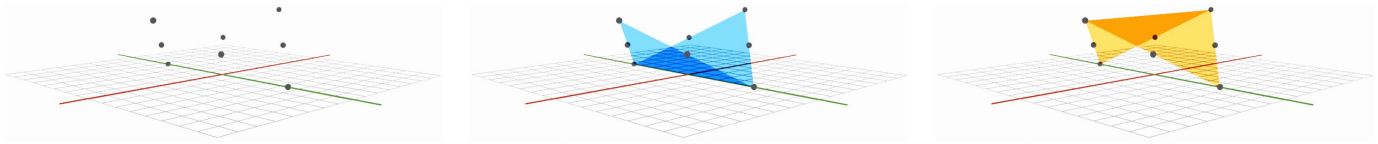
The functional model (Fig. 8b) is a two-branch and two-layer feedforward neural network. The first layer branch has no non-linearities afterwards, while the second branch is followed by a $RELU$ non-linearity. The second layer simply takes the weighed sum of both branches and thus has no bias. This model reduces to a function of the form:

$$z = \gamma_1(\alpha_1 x + \alpha_2 y + \alpha_3) + \gamma_2 \text{RELU}(\beta_1 x + \beta_2 y + \beta_3). \tag{14}$$

Here, $\boldsymbol{\alpha}$ represents the parameters of the first branch of the first layer, $\boldsymbol{\beta}$ the parameters of the second branch of the first layer, and $\boldsymbol{\gamma}$ the parameters of the second layer. This simple model represents a planar function with a single fold along the line of equation $\beta_1 x + \beta_2 y + \beta_3 = 0$.

This model has always two and only two non-equivalent ways of fitting the proposed dataset (see Fig. 9). The model also exhibits some weight-space symmetry. In other words, the half planes fitted by the first and second branches of the first layer can be swapped, resulting in a different parametrization but equivalent fitting of the data. It also exhibits a scaling symmetry between $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ against $\boldsymbol{\gamma}$.

To sample the posterior, we first have to choose a prior for the parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ of the BNN. The following procedure works for any choice of prior $p(\boldsymbol{\alpha}, \boldsymbol{\beta}), p(\boldsymbol{\gamma})$ where the probability distribution of the parameters of the first layer is independent of the probability distribution of the parameters of the second layer. For simplicity and if not



(a) Raw data (b) First possible fold (c) Second possible fold

Fig. 9: The dataset and two possible folding behaviors of the model for the paperfold case study.

specified otherwise, we assume a normal prior with diagonal constant covariance:

$$\begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \\ \boldsymbol{\gamma} \end{pmatrix} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I}), \tag{15}$$

with σ a positive standard deviation. By default, we assume that we do not know much about the model and set $\sigma = 5$ for this case study.

A. Comparison of training methods

We implemented four different samplers for the paperfold example (Fig. 10):

- The NUTS sampler [4], whose implementation is provided by the Pyro Python library. This is a state-of-the-art general purpose MCMC sampler, which was used mainly as a way of generating samples from the true posterior. It should serve as a base of comparison with other methods, because this MCMC sampler is much slower than the other methods we consider (Fig. 13).
- A variational inference model based on a Gaussian approximation of the posterior.
- An ensemble based approach, and
- An ensemble of variational inference based models.

1) MCMC: The MCMC sampler provides a very good approximation of samples from the exact posterior. In Fig. 10, one can get an appreciation of the complexity of the exact posterior even for this small and simple model. Each row and column represent one variable with the graph at the intersection showing the samples projected in the 2-dimensional plane spanned by those variables. The samples form perpendicular and diagonal structures, which probably correspond to the two non-equivalent ways of fitting the data. The symmetric lines are caused by the weight-space symmetry. The hyperbolas correspond to equivalent parametrizations of the network under scaling symmetry. Finally, the few outliers that are visible correspond to a series of small modes in

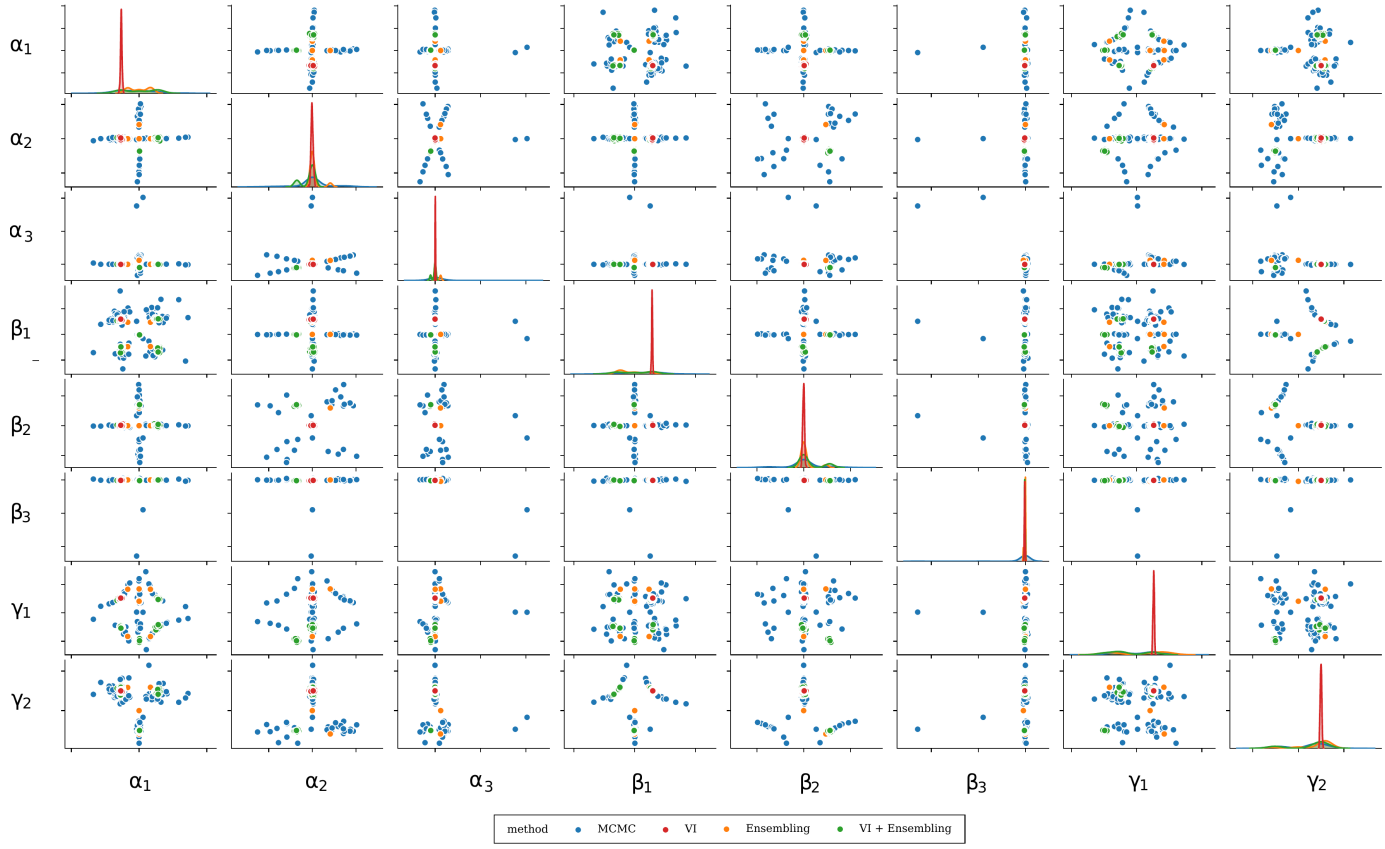


Fig. 10: Pairplot of the samples from the posterior using four different approaches.

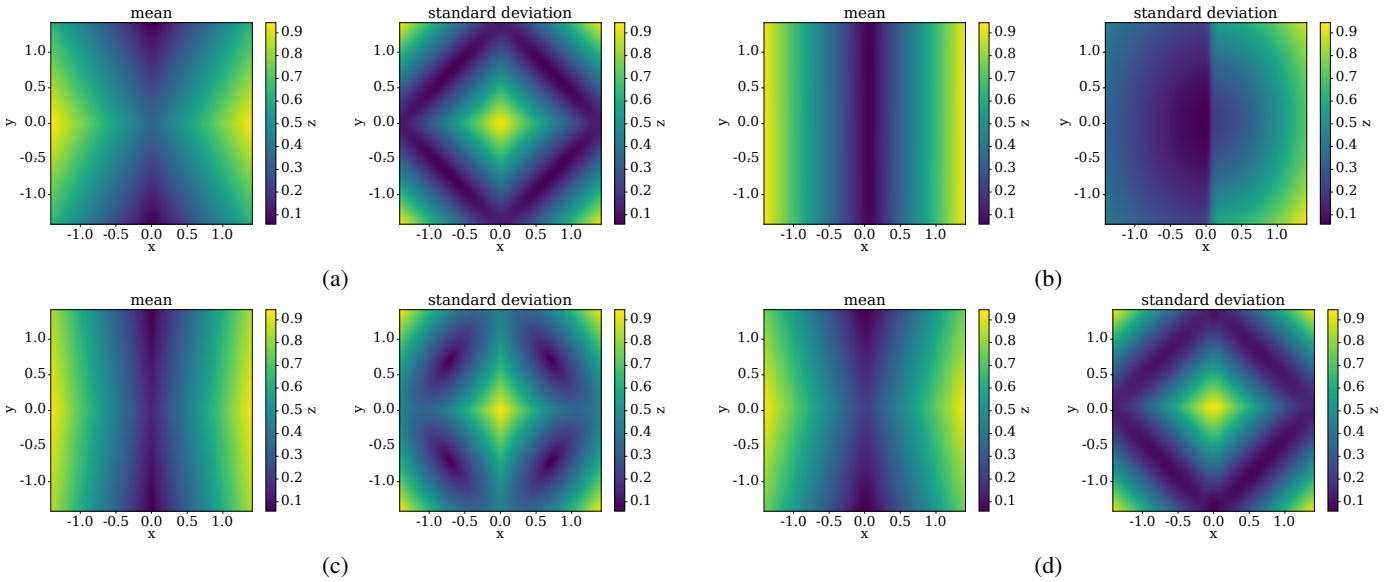


Fig. 11: Mean and standard deviation of the marginal distribution of z knowing D, x and y for the paperfold model using (a) MCMC, (b) variational inference, (c) ensembling and (d) variational inference + ensembling.

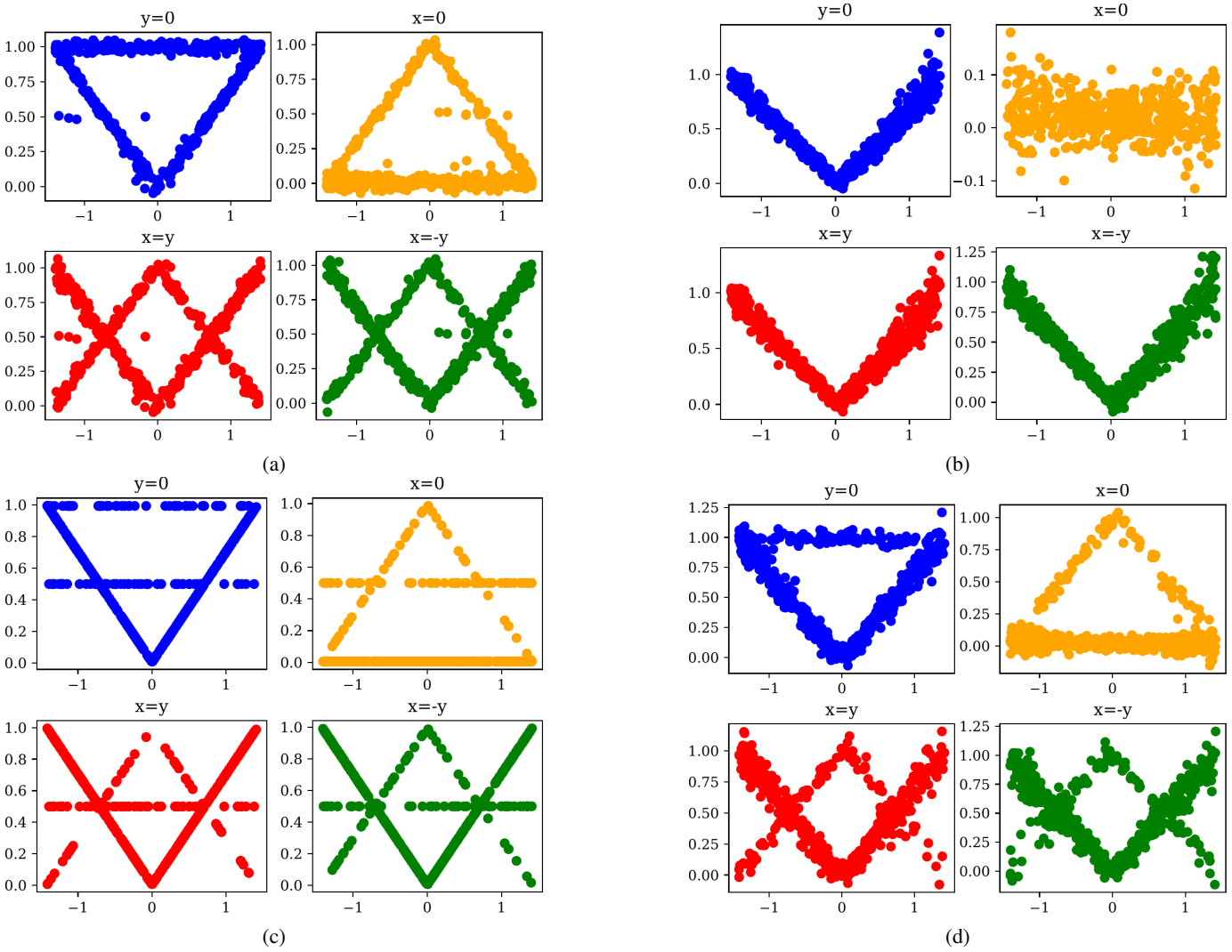


Fig. 12: Predicted z values across four different lines for the (a) MCMC, (b) variational inference, (c) ensembling and (d) variational inference + ensembling version of the paperfold BNN.

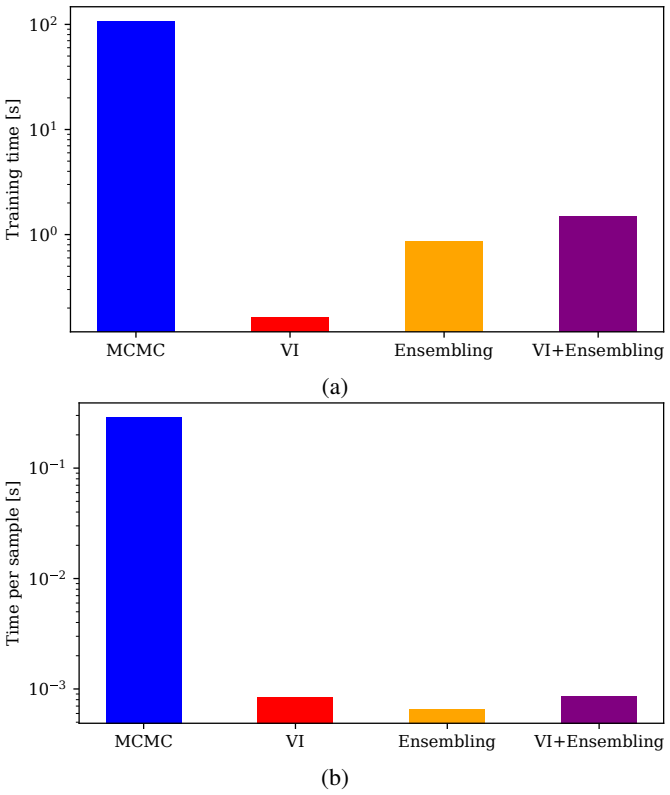


Fig. 13: Comparison between (a) training time and (b) inference time for the paperfold example with standard algorithms for the different training approaches.

the posterior around the parametrizations fitting the constant function $f(x, y) = 0.5$.

When looking at the actual aggregate results (Fig. 11a), one can see that the average prediction of the BNN fits the data with a regular function similar to a second degree polynomial. The uncertainty, as measured by the standard deviation a posteriori, is low along the lines of the square where the data points lie and increases linearly with distance, creating a diamond shape. Sampling the predicted value for z along a series of lines in the (x, y) plane of coordinates (Fig. 12a) shows that both folds appear as clearly distinct from one another, but a bit of uncertainty remains around the exact position of these folds.

2) Variational inference: As a variational inference approximation to the posterior, we used a normal distribution with a learnable mean and a diagonal covariance matrix. It is pretty clear that for this specific example, which has been designed to generate such a specific problem, this method leads to a very poor approximation of the actual posterior since it can only fit an unimodal distribution (Fig. 10). In terms of the marginal, only one of the two possible folds (Fig. 12b) has been fitted. This translates to a mean estimate for z as well as an uncertainty level and distribution well off the actual posterior (Fig. 11b). The actual side of the fold fitted by the RELU branch is also visible as its uncertainty is higher than on the other side of the fold. This highlights the major limitations of simple variational inference methods for BNNs. Since the underlying models are complex and the data can be fitted in many non-equivalent manners, simple variational posteriors lack some expressive power to provide a good estimate of the actual BNN uncertainty. It is still important to keep in mind that this example has been specifically designed to make those problems apparent and that, in practice, mean field Gaussian and similar variational inference approximations can still lead to good estimations of the actual uncertainty. An example of this is the sparse measure example provided in Appendix II. Variational inference is also several orders of magnitude faster than MCMC (Fig. 13).

3) Ensembling: The most straightforward way to perform ensemble learning with an artificial neural network architecture is just to restart the learning procedure multiple times and each time add the final model to the ensemble. Using this strategy to learn the posterior for the paperfold example already gives quite good results, even better than naive variational inference as the samples can belong to different modes of the posterior (Fig. 10). The marginal mean and standard deviation a posteriori are clearly different from the ones obtained via MCMC, but the diamond shape can be recongnized in both predictions (Fig. 11c). The training is slightly longer than with basic variational inference as it has to be run multiple times. However, the time per sample when computing the marginal is extremely low (Fig. 13). The main drawback of this approach is that it provides no estimation of the local uncertainty around the different modes of the posterior as shown by Fig. 12c.

4) Variational Inference + Ensembling: Combining the benefits of a simple variational inference and ensembling can be done in a straightforward manner by learning an ensemble of variational approximations. The resulting distribution is a Gaussian mixture, which can approximate the shape of multiple modes of the posterior. In this example, it cannot match the complex shapes of the exact posterior (Fig. 10). Despite those limitations, the resulting marginal for z is a very good match with the one generated via a MCMC sampler (Fig. 11 and 12). In addition, the time per sample when computing the marginal is still very similar to the single Gaussian variational approximation (Fig. 13). This shows the huge benefits of ensembling as a tool for approximate Bayesian methods in deep learning.

REFERENCES

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[3] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *J. Mach. Learn. Res.*, vol. 7, pp. 2399–2434, Dec. 2006.

[4] M. D. Hoffman and A. Gelman, "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.