On Feynman–Kac training of partial Bayesian neural networks

Zheng Zhao **Sebastian Mair** **Thomas B. Schön** **Jens Sjölund**

Uppsala University, Sweden

Abstract

Recently, partial Bayesian neural networks (pBNNs), which only consider a subset of the parameters to be stochastic, were shown to perform competitively with full Bayesian neural networks. However, pBNNs are often multi-modal in the latent variable space and thus challenging to approximate with parametric models. To address this problem, we propose an efficient sampling-based training strategy, wherein the training of a pBNN is formulated as simulating a Feynman–Kac model. We then describe variations of sequential Monte Carlo samplers that allow us to simultaneously estimate the parameters and the latent posterior distribution of this model at a tractable computational cost. Using various synthetic and real-world datasets we show that our proposed training scheme outperforms the state of the art in terms of predictive performance.

1 INTRODUCTION

Bayesian neural networks (BNNs) are an important class of machine learning models for quantifying uncertainty. However, computing BNN posterior distributions is an open challenge (Izmailov et al., 2021) since many standard statistical inference methods such as Markov chain Monte Carlo (MCMC) are poorly suited to the combination of a high-dimensional parameter space and a massive number of data points (Papamarkou et al., 2022). Many approximate solutions have been proposed to overcome this computational challenge, for example, MCMC with stochastic likelihoods (Andrieu and Roberts, 2009; Welling and Teh, 2011; Chen et al., 2014; Zhang et al., 2020), parametric approximations of the posterior distributions in the

form of variational Bayes (Blundell et al., 2015; Gal and Ghahramani, 2016) or stochastic averaging methods (Izmailov et al., 2018; Maddox et al., 2019). In a similar vein, Izmailov et al. (2020) conduct variational inference in a subspace of the parameter space. Other ways include, for instance, restricting the inference only on the last layer (Ober and Rasmussen, 2019; Kristiadi et al., 2020).

In this paper, we focus on the training of partial Bayesian neural networks (pBNNs), a family of BNNs where only part of the model parameters are random variables. This is motivated by the recent work of, for example, Daxberger et al. (2021) and Sharma et al. (2023), which show that pBNNs and standard BNNs are comparable in terms of prediction performance despite the dimensionality of the random variable in the pBNNs being much smaller than standard BNNs. While both approaches make posterior inference much easier, it comes at the cost of making the pBNN a latent variable model that requires estimation of both the deterministic parameters and the posterior.

To be more precise, let $x \mapsto f(x; \phi, \psi)$ be a neural network parametrised by a deterministic parameter $\psi \in \mathbb{R}^w$ and a random variable $\phi \in \mathbb{R}^d$ that follows a given prior distribution $\pi(\phi)$. Suppose that we have a dataset $\{(x_n, y_n)\}_{n=1}^N$ with an associated likelihood conditional density function $p(y_n \mid \phi; \psi)$[1], and that the observation random variables are independent conditioned on ϕ. The goal of training is twofold. First, to learn the deterministic parameter ψ from the dataset, and second, to compute the posterior distribution $p(\phi \mid y_{1:N}; \psi)$, where $y_{1:N} := \{y_1, y_2, \ldots, y_N\}$. This is a classical parameter estimation problem in a latent variable model (Cappé et al., 2005). The main benefit of pBNNs is that when the dimension $d \ll w$ is relatively small, the second goal of computing the posterior distribution with fixed ψ is generally tractable. However, the first goal of estimating the parameter ψ remains to be solved.

For practitioners, a popular approach is to jointly optimise ϕ and ψ according to the maximum a posteriori

---

[1]Throughout the paper we omit the data covariate x_n of y_n to make the notation cleaner.

(MAP) objective $(\phi, \psi) \mapsto \log p(y_{1:N} \mid \phi; \psi) + \log p(\phi)$, see, for example, Daxberger et al. (2021) and Sharma et al. (2023). Then, the estimated ψ is used to compute the posterior distribution over ϕ by any applicable Bayesian inference method (e.g., MCMC). The advantages of this method lie in the simplicity and fast computation. Moreover, there is a plethora of optimisers (e.g., stochastic gradient descent and its variants) specifically designed for the setting of large w and N. However, for multi-modal distributions the MAP estimated ψ and ϕ are prone to be restricted on a single mode.

Another popular approach is to use the maximum likelihood estimation (MLE) for estimating ψ. Compared to the aforementioned MAP approach, the MLE objective takes all possible outcomes of the random variable ϕ into account by integrating out ϕ, and the estimator can be consistent (Cappé et al., 2005). However, the MLE method is not directly applicable to pBNNs since the marginal likelihood $p(y_{1:N}; \psi) = \int p(y_{1:N} \mid \phi; \psi) \pi(\phi) \, d\phi$ is in general intractable. Instead, various lower bounds on the marginal likelihood are used by, for instance, brute-force Monte Carlo, expectation maximisation (EM, Bishop, 2006), and variational Bayes (VB, Blei et al., 2017; Sjölund, 2023). However, EM and VB often compromise in using parametric families of distributions, and they often incur Monte Carlo computation overheads to approximate expectations. Another common approach is to transform the (gradient of) MLE into an expectation with respect to the posterior distribution by Fisher's identity (see, e.g., Cappé et al., 2005, Chap. 10), and then use an MCMC sampler to approximate the expectation (at the cost of demanding computations).

In this paper, we study how to apply and adapt sequential Monte Carlo (SMC) methods to train pBNNs, by representing the training as a simulation of a Feynman–Kac model (Chopin and Papaspiliopoulos, 2020). More specifically, the Feynman–Kac model is composed of a sequence of potential functions given by the likelihood model, and a sequence of invariant Markov kernels that anneals to the target posterior distribution $p(\phi \mid y_{1:N}; \psi)$. Computing the target posterior distribution then amounts to a sequential simulation of the Feynman–Kac model, and SMC is the natural sampling framework for the model. The motivation for studying SMC samplers to train pBNNs is that SMC samplers are able to simultaneously produce consistent MLEs and to sample from the posterior distribution while retaining a tractable computational cost. Compared to most MCMC-based methods, SMC samplers are immediately parallelisable (Lee et al., 2010) by leveraging graphics processing units, and are easier to calibrate. On the other hand, parallelising MCMC

is still an open problem (see, e.g., Jacob et al., 2020).

Our contributions are as follows. (i) We show and discuss how to apply and adapt SMC samplers to train pBNNs (Section 2). (ii) We propose approximate SMC samplers that are scalable in the number of data points and are therefore better suited to pBNNs (Section 3). (iii) We benchmark the proposed samplers on a variety of synthetic and real-world datasets, and the results show that the proposed training scheme is state-of-the-art in terms of predictive performance (Section 4).

2 TRAINING VIA FEYNMAN–KAC

Recall that we aim to find a maximum likelihood estimate for the deterministic part of the pBNN and then to compute the posterior distribution $p(\phi \mid y_{1:N}; \psi)$ of the stochastic part of the pBNN based on the learnt parameter ψ. In this section, we recap a sequential online algorithm to recursively compute the target posterior distribution via a Feynman–Kac model and how to jointly estimate the parameter ψ.

To ease the exposition of the idea, let us for now suppose that the deterministic variable ψ in the pBNN is fixed. Due to the conditional independence of the observations, the posterior distribution admits a recursion

$$p(\phi \mid y_{1:n}; \psi) = \frac{p(y_n \mid \phi; \psi)}{z_n(\psi)} p(\phi \mid y_{1:n-1}; \psi), \quad (1)$$

for $n = 1, 2, \ldots, N$, where $z_n(\psi) := p(y_n \mid y_{1:n-1}; \psi)$ is the normalising constant, and the initial is defined by $p(\phi \mid y_{1:0}; \psi) := \pi(\phi)$. If we have computed the distribution $p(\phi \mid y_{1:n-1}; \psi)$ for some n, we can then compute the next $p(\phi \mid y_{1:n}; \psi)$ by Equation (1). Ultimately, we continue the iteration until $n = N$ to reach the target posterior distribution. This recursion is the gist of sequential online learning or annealing.

As in most Bayesian inference problems, the challenge lies in the computationally intractable normalising constant $z_n(\psi)$. Hence, in practice, we often use a tractable sequence of approximations $Q_N := \{q_n(\phi \mid y_{1:n}; \psi): n = 1, 2, \ldots, N\}$ such that $q_n(\phi \mid y_{1:n}; \psi) \propto q_{n-1}(\phi \mid y_{1:n-1}; \psi)$ is computable for all n's. A convenient choice is a Gaussian q_n and then to use, for example, Taylor expansions, variational Bayes (Opper, 1999), or Gauss quadrature methods (Golub and Meurant, 2010) to run the sequence. However, in the context of BNNs, Gaussian approximations can result in large errors (Foong et al., 2020) particularly when the true posterior distribution is non-Gaussian (e.g., multi-modal ones). This motivates us to use a Monte Carlo (MC)-based method to come up with such an approximate sequence.

2.1 Sequential Monte Carlo sampling

Sequential Monte Carlo (SMC) samplers (Del Moral et al., 2006; Chopin and Papaspiliopoulos, 2020) are natural Monte Carlo (MC) methods for approximating the target posterior distribution in the sequential learning framework. Specifically, we choose $Q_N = \{S_n^J : n = 1, 2, \ldots, N\}$, where $S_n^J := \{(w_{n,j}, \phi_{n,j}) : j = 1, 2, \ldots, J\}$ are J weighted Monte Carlo samples that approximately represent $p(\phi \mid y_{1:n}; \psi)$. Suppose that we are able to draw the initial samples $S_0^J \sim \pi(\phi)$ from the given prior, then we can recursively compute S_n^J for any $n = 1, 2, \ldots, N$ via

$$\phi_{n,j} = \phi_{n-1,j}, \qquad (2)$$
$$\overline{w}_{n,j} = w_{n-1,j}\, p(y_n \mid \phi_{n-1,j}; \psi),$$

and normalisation $w_{n,j} = \overline{w}_{n,j} / \sum_{i=1}^J \overline{w}_{n,i}$ for all samples $j = 1, 2, \ldots, J$. This approximation is consistent in the sense that the resulting Dirac measure given by S_n^J converges weakly to that of $p(\phi \mid y_{1:n}; \psi)$ as $J \to \infty$ for all n (Chopin and Papaspiliopoulos, 2020). Furthermore, the method is particularly favourable in the pBNN context, as the dimension d for the stochastic part of the pBNN is usually not large which in turn allows us to use significantly more MC samples compared to full BNNs. This method dates back to Neal (2001) who simulates static target distributions with a likelihood tempering (see also Chopin, 2002).

However, the SMC chain using Equation (2) rarely works in reality, since the samples $\{\phi_{n,j}\}_{j=1}^J$ will become less informative and the weights $\{w_{n,j}\}_{j=1}^J$ will degenerate as n increases (Del Moral et al., 2006). In practice, we additionally introduce a Markov transition kernel $h_n(\cdot \mid \phi_{n-1})$ between each step $(n-1, n)$, so that the samples $\{\phi_{n,j}\}_{j=1}^J$ are perturbed according to the kernel. Specifically, the update of samples in Equation (2) modifies to

$$\phi_{n,j} \mid \phi_{n-1,j} \sim h_n(\phi_{n,j} \mid \phi_{n,j-1}). \qquad (3)$$

Then, we arrive at the so-called (marginal) Feynman–Kac model

$$q_N(\phi_N \mid y_{1:N}; \psi)$$
$$:= \frac{1}{l_N(\psi)} \int \prod_{n=1}^N p(y_n \mid \phi_n; \psi) \prod_{n=1}^N h_n(\phi_n \mid \phi_{n-1}) \quad (4)$$
$$\times\, \pi(\phi_0)\, \mathrm{d}\phi_{0:N-1},$$

where $l_N(\psi)$ is the normalising constant (see the definition of Feynman–Kac in Chopin and Papaspiliopoulos, 2020, Chap. 5). Now, to guarantee that the terminal $q_N(\phi_N \mid y_{1:N}; \psi)$ exactly hits the target posterior distribution $p(\phi \mid y_{1:N}; \psi)$, we need to choose the Markov kernel in a way that h_n leaves the previous posterior distribution $p(\phi \mid y_{1:n-1}; \psi)$ invariant,

viz., $\int h_n(\phi \mid \phi_{n-1})\, \rho(\phi_{n-1})\, \mathrm{d}\phi_{n-1} = p(\phi \mid y_{1:n-1}; \psi)$ for any distribution ρ. Note that h_n indeed depends on $y_{1:n-1}$ and ψ, but we omit them for clean notation. This choice of Markov kernel also ensures that the marginal likelihood is given by $l_N(\psi) = \prod_{n=1}^N z_n(\psi) = p(y_{1:N}; \psi)$.

It is now clear that computing the target posterior distribution $p(\phi \mid y_{1:N}; \psi)$ amounts to simulating the Feynman–Kac model in Equation (4). The SMC sampler simulates the Feynman–Kac model with the weighted samples S_n^J for $n = 0, 1, \ldots, N$ using Equations (2) and (3). Moreover, recall that we need to estimate the parameter ψ via MLE

$$\underset{\psi \in \mathbb{R}^w}{\arg\min}\, -\log l_N(\psi).$$

It turns out that the SMC sampler produces a consistent estimate $z_n(\psi) \approx \sum_{j=1}^J \overline{w}_{n,j}$, so that we can estimate the log-likelihood via $\log l_N(\psi) \approx \sum_{n=1}^N \log\left(\sum_{j=1}^J \overline{w}_{n,j}\right)$ as a by-product of the computation of the posterior distributions. Then, we can solve the optimisation problem above by using any optimiser for training neural networks. For a detailed exposition of SMC parameter estimators, see, for example, Johansen et al. (2008), Schön et al. (2011), and Kantas et al. (2015). We summarise the SMC training of pBNNs in Algorithm 1 together with a gradient descent-based optimisation of the MLE objective. Within the algorithm, we use the shorthand $\ell_n(\cdot)$ for the approximation to the marginal log-likelihood $\log l_n(\cdot)$, and we also use $r(i)$ for the learning rate at the i-th iteration.

3 SCALABLE SEQUENTIAL MONTE CARLO SAMPLERS

We still have a few critical challenges left to solve in order to apply the SMC sampler in Algorithm 1 for training pBNNs. First, the gradient computation $\nabla \ell_N$ might be biased due to the non-differentiability of the resampling and Markov transition steps. Second, it is not easy to design the Markov kernel h_n and also to compute the Markov moves. Third, the algorithm does not scale well in the number of data points N. In particular, for each gradient descent step to compute $\nabla \ell_N$, we need a complete SMC loop over the entire dataset. In what follows, we detail these issues and show how to tackle them.

3.1 Compensating gradient biases

The non-differentiability is a notorious issue for sequential Monte Carlo samplers, since most resampling techniques (e.g., systematic and stratified) and MCMC algorithms incur discrete randomness. This

---

Algorithm 1: Sequential Monte Carlo (SMC) sampler for pBNN

Inputs: Training data $\{(x_n, y_n)\}_{n=1}^N$, number of samples J, initial parameter ψ_0, learning rate function r

Outputs: The MLE estimate ψ_i and weighted posterior samples
$\{(w_{N,j}, \phi_{N,j})\}_{j=1}^J \sim p(\phi \mid y_{1:N}; \psi_i)$

for $i = 1, 2, \ldots$ **until** *convergent* **do**

    Draw $\{\phi_{0,j}\}_{j=1}^J \sim \pi(\phi)$

    $w_{0,j} = 1 / J$ for all $j = 1, 2, \ldots, J$

    $\ell_0(\psi_{i-1}) = 0$

    for $n = 1$ **to** N **do**   // Parallelise j

        Resample $\{(w_{n-1,j}, \phi_{n-1,j})\}_{j=1}^J$ if needed

        Draw $\phi_{n,j} \mid \phi_{n-1,j} \sim h_n(\phi_{n,j} \mid \phi_{n,j-1})$

        $\overline{w}_{n,j} = w_{n-1,j}\, p(y_n \mid \phi_{n,j}; \psi_{i-1})$

        $\ell_n(\psi_{i-1}) = \ell_{n-1}(\psi_{i-1}) - \log\big(\sum_{j=1}^J \overline{w}_{n,j}\big)$

        $w_{n,j} = \overline{w}_{n,j} / \sum_{k=1}^J \overline{w}_{n,k}$

    end

    $\psi_i = \psi_{i-1} + r(i)\, \nabla \ell_N(\psi_{i-1})$

end

---

in turns induces biases in the gradient computation $\nabla \ell_N$. There are recent developments to tackle the differentiability issue by, for example, optimal transport-based smooth resampling (Corenflos et al., 2021) and coupled MCMC (Arya et al., 2022, 2023), but they come at the cost of introducing additional computation costs and calibrations. However, we can in fact avoid differentiating the SMC algorithm by invoking Fisher's identity (see, e.g., Cappé et al., 2005, Chap. 10), which states that $\nabla \log l_N(\psi) = \int \nabla \log p(y_{1:N}, \phi; \psi)\, p(\phi \mid y_{1:N}; \psi)\, \mathrm{d}\phi$. Moreover, unlike particle filtering in the system identification context (see, e.g., Poyiadjis et al., 2011, Sec. 2.2), the model design facilitates $\nabla \log p(y_{1:N}, \phi; \psi) = \nabla \log p(y_{1:N} \mid \phi; \psi)$. Hence, we can modify the gradient computation in Algorithm 1 to

$$\nabla \ell_N(\psi) = \sum_{j=1}^J w_{N,j} \nabla \log p(y_{1:N} \mid \phi_{N,j}; \psi), \qquad (5)$$

which only requires the log-likelihood to be differentiated.

3.2 Choosing the Markov kernel

Recall the definition of the Markov kernel h_n: It leaves the posterior distribution $p(\phi \mid y_{1:n-1}; \psi)$ invariant. Since the joint $p(y_{1:n-1} \mid \phi; \psi)\, \pi(\phi)$ is usually analytically available, it is common to use any MCMC chain for h_n (Del Moral et al., 2006). In particular, when the problem dimension d is low, using a standard random walk Metropolis–Rosenbluth–Teller–Hasting MCMC suffices in practice. If the dimension is relatively high, we can also leverage the gradient information of the energy to define the kernel with, for instance, a Langevin dynamic. However, it is worth remarking that choosing the number of MCMC steps is non-trivial, and this still remains an open challenge in the SMC community, see, for example, the discussion in Chopin and Papaspiliopoulos (2020, Sec. 17.2) or a recent development in Dau and Chopin (2021).

The computational problem of using MCMC for the kernels is that at each n, we need to load all the data before n to evaluate the energy function. For gradient-based MCMCs, we also need to compute $\nabla_\phi \log p(\phi \mid y_{1:n-1}; \psi)$. This computation becomes even more demanding when n approaches N (which is large), and hence the computational cost grows at least quadratically in n. A straightforward remedy to this problem is to use pseudo-marginal MCMC samplers (see, e.g., Andrieu and Roberts, 2009; Welling and Teh, 2011; Chen et al., 2014; Zhang et al., 2020). However, these mini-batching MCMC samplers usually take long mixing steps and are hard to calibrate, which contradicts the gist of using MCMC kernels in SMC: An ideal SMC sampler should need as few mixing steps as possible. Furthermore, the variance of the estimator increases in n, and it is likely that we need to adaptively increase the batch size to control the error. However, it turns out that we can rectify this problem by moving the stochastic mini-batching approximation outside of the Markov kernel to that of the gradient step as explained in the next section.

3.3 Stochastic gradient sequential Monte Carlo

To make Algorithm 1 scalable in the number of data points N, it is natural to approximate the marginal log-likelihood by a stochastic approximation, as in stochastic gradient descent. Let $1 \leq M \leq N$ denote the batch size and let $S_M := \{S_M(1), S_M(2), \ldots, S_M(M)\}$ be a sequence of independent random integers (uniformly distributed in $[1, N]$) that represent the batch indices. We may approximate the marginal log-likelihood by $\log p(y_{1:N}; \psi) \approx N / M \log p(y_{S_M}; \psi)$, where $y_{S_M} := \{y_{S_M(1)}, y_{S_M(2)}, \ldots, y_{S_M(M)}\}$ represents the corresponding subdataset. The same also goes for the gradient computation. More specifically,

$$
\begin{aligned}
&\nabla \log p(y_{1:N}; \psi) \\
&\approx \frac{N}{M} \mathbb{E}\big[\nabla \log p(y_{S_M}; \psi)\big] \\
&= \frac{N}{M} \int \mathbb{E}\big[\nabla \log p(\phi, y_{S_M}; \psi)\, p(\phi \mid y_{S_M}; \psi)\big]\, \mathrm{d}\phi,
\end{aligned}
\qquad (6)
$$

where the expectation is taken over S_M. However, due to the latent variable ϕ, the approximation $N/M \log p(y_{S_M}; \psi)$ is biased. Consequently, the gradient is also biased, and the bias scales in the difference between $p(\phi \mid y_{S_M})$ and $p(\phi \mid y_{1:N})$.

Using the approximation in Equation (6) amounts to running Algorithm 1 on the subdataset y_{S_M} and then using the posterior samples of $p(\phi \mid y_{S_M}; \psi)$ to compute the gradient $\int \nabla \log(\phi, y_{S_M}; \psi) \, p(\phi \mid y_{S_M}; \psi) \, \mathrm{d}\phi$. We then arrive at the stochastic gradient SMC sampler summarised in Algorithm 2.

Algorithm 2: Stochastic gradient sequential Monte Carlo (SGSMC) sampler for pBNN

Inputs: The same as in Algorithm 1, and batch size M
Outputs: The MLE estimate ψ_i
for $i = 1, 2, \ldots$ **until** *convergent* **do**
  Draw $\{\phi_{0,j}\}_{j=1}^J \sim \pi(\phi)$
  $w_{0,j} = 1/J$ for all $j = 1, 2, \ldots, J$
  Draw subdataset $y_{S_M^i} \subseteq y_{1:N}$
  for $n = 1$ **to** M **do**   // Parallelise j
    Resample $\{(w_{n-1,j}, \phi_{n-1,j})\}_{j=1}^J$ if needed
    Draw $\phi_{n,j} \mid \phi_{n-1,j} \sim h_n(\phi_{n,j} \mid \phi_{n,j-1})$
    $\overline{w}_{n,j} = w_{n-1,j} \, p(y_{S_M^i(n)} \mid \phi_{n,j}; \psi_{i-1})$
    $w_{n,j} = \overline{w}_{n,j} / \sum_{k=1}^J \overline{w}_{n,k}$
  end
  $g(\psi_{i-1}) = \frac{N}{M} \sum_{j=1}^J w_{M,j} \nabla \log p(y_{S_M^i} \mid \phi_{M,j}; \psi)$
  $\psi_i = \psi_{i-1} + r(i) \, g(\psi_{i-1})$
end

Compared to Algorithm 1, the stochastic gradient version in Algorithm 2 does not have to load the entire dataset to compute the gradient. More importantly, this also handles the Markov kernel design dilemma in Section 3.2 in the following way. The kernel h_n is now chosen to be invariant with respect to $p(\phi \mid y_{S_M(1)}, \ldots, y_{S_M(n-1)})$ which is far easier to compute than the original when $M \ll N$. Essentially, the algorithm is a direct application of the stochastic gradient method on a latent variable model by using SMC samplers to approximate the gradient. This optimisation scheme is akin to the variational SMC method (Naesseth et al., 2018).

3.4 Open-horizon sequential Monte Carlo

Algorithm 2 defines an approximate flow of gradient that can be computed efficiently by applying SMC samplers in a closed data horizon. However, the algorithm does not directly output the target posterior distribution $p(\phi \mid y_{1:N}; \psi)$ unlike Algorithm 1. To compute the posterior distribution, we may need to run

another Bayesian inference (e.g., by SMC or MCMC) based upon the estimated ψ. Moreover, for every optimisation step, the SMC estimators are independent and cold-start from the prior π. This causes a waste of computation, since Algorithm 2 does compute the posterior distributions $p(\phi \mid y_{S_M}; \psi_i)$ at subdatasets which are approximations to the target posterior distribution. In light of this observation, we can make Algorithm 2 even more efficient by linking the posterior distribution estimates in conjunction with the gradient. More specifically, we modify Algorithm 2 by warm-starting each SMC sampler from the previous posterior distribution estimate and perform the gradient update in conjunction with the SMC sampler. We then arrive at an SMC sampler that simultaneously estimates the posterior distribution and the parameter, summarised in Algorithm 3.

Algorithm 3: Open-horizon sequential Monte Carlo (OHSMC) sampler

---

Inputs: Same as in Algorithm 2
Outputs: Same as in Algorithm 1
Draw $\{\phi_{0,j}\}_{j=1}^J \sim \pi(\phi)$
$w_{0,j} = 1/J$ for all $j = 1, 2, \ldots, J$
for $i = 1, 2, \ldots$ **until** *convergent* **do**
 // Parallelise j
  Draw subdataset $y_{S_M^i} \subseteq y_{1:N}$
  Resample $\{(w_{i-1,j}, \phi_{i-1,j})\}_{j=1}^J$ if needed
  Draw $\phi_{i,j} \mid \phi_{i-1,j} \sim h_i(\phi_{i,j} \mid \phi_{i,j-1})$
  $\overline{w}_{i,j} = w_{i-1,j} \, p(y_{S_M^i} \mid \phi_{i,j}; \psi_{i-1})$
  $w_{i,j} = \overline{w}_{i,j} / \sum_{s=1}^J \overline{w}_{i,s}$
  $g(\psi_{i-1}) = \frac{N}{M} \sum_{j=1}^J w_{i,j} \nabla \log p(y_{S_M^i} \mid \phi_{i,j}; \psi_{i-1})$
  $\psi_i = \psi_{i-1} + r(i) \, g(\psi_{i-1})$
end

In Algorithm 3, we start from samples drawn from the given prior π. Then, at each iteration i, we randomly draw a subdataset $y_{S_M^i}$ of $y_{1:N}$ and compute the posterior distribution and gradient on the subdataset by using the previous estimate. Equivalently, the algorithm is an SMC sampler applied on a growing dataset with open horizon instead of $y_{1:N}$ which has a fixed size (cf. Kantas et al., 2015, Eq. 5.6). Compared to Algorithms 1 and 2, this open-horizon SMC sampler is computationally more efficient, especially when N is large.

However, Algorithm 3 no longer targets the posterior distribution $p(\phi \mid y_{1:N}; \psi)$. To see what the algorithm does, let us *fix* the parameter ψ, to find that the algorithm computes the following Feynman–Kac model

$$\frac{1}{\hat{l}_P(\psi)} \int \prod_{i=1}^P p(y_{S_M^i} \mid \phi_i; \psi) \prod_{i=1}^P h_i(\phi_i \mid \phi_{i-1})$$
$$\times \pi(\phi_0) \, \mathrm{d}\phi_{0:P-1}, \tag{7}$$

where P is the number of iterations for the algorithm, and $\hat{l}_P$ is the normalising constant. Evidently, Equation (7) is neither equal to the original Feynman–Kac model in Equation (4), nor to its expectation. It is, however, possible to use Poisson estimators (Beskos et al., 2006; Fearnhead et al., 2008; Jacob and Thiery, 2015) to make Equation (7) an unbiased estimator of the original Feynman–Kac model. To do so, we need to let the number of iterations $P \sim \text{Poisson}(\lambda)$ be a Poisson random variable and modify the weight update in Algorithm 3 to

$$\overline{w}_{i,j} = w_{i-1,j}\left(\frac{N}{M}\log p(y_{S_M^i} \mid \phi_{i,j}; \psi_{i-1}) + \log c\right),$$

where c is a constant that guarantees the positivity of the weights almost surely. Then, we can show by the Poisson estimator that

$$\mathbb{E}\left[\frac{\mathrm{e}^\lambda}{c}\prod_{i=1}^{P}\frac{1}{\lambda}\left(\frac{N}{M}\log p(y_{S_M^i} \mid \phi; \psi) + \log c\right)\right]$$
$$= \prod_{i=1}^{N} p(y_i \mid \phi; \psi),$$

where the expectation is taken on both P and S_M. However, finding such a positive constant c is still an open but stimulating challenge in the statistic community (see, e.g., a recent progress in Jin et al., 2022).

For training pBNNs, it may not be necessary to debias Algorithm 3. In practice, we use (p)BNNs as powerful predictive models to quantify uncertainty. Hence, computing the exact posterior distribution is often unnecessary from a practical point of view (Wilson and Izmailov, 2020; Izmailov et al., 2021), and we can control the bias via validation data. As a consequence, the prior is no longer considered as given but is a flexible model free to choose so as to aim for better predictive performance. To this end, we can further relax the invariance property of the Markov kernel h. This relaxation bridges a connection to the classical stochastic filtering approaches for optimisations (Bell, 1994; Gerber and Douc, 2021). As an example, the extended Kalman filter approximations to the algorithm are stochastic natural gradient descents with information matrices determined by the Markov kernel (Ollivier, 2018; Martens, 2020). As another example, if we choose the Markov kernel to be that of a Brownian motion, then the resulting algorithm is akin to using stochastic filters for training neural networks with uncertainties (Singhal and Wu, 1988; De Freitas et al., 2000; Chang et al., 2022). It is also possible to debias the Brownian motion kernel by that of a backward kernel as in Marques and Storvik (2013) and Del Moral et al. (2006).

It is worth remarking a minor practical advantage of Algorithm 3 over Algorithm 2. The Markov kernel in Algorithm 2 is a function that has dynamic input sizes. This means that implementing the algorithm in commonly used automatic differentiation libraries that assume static input shapes (e.g., JAX and Tensorflow) is not trivial. On the other hand, Algorithm 3 has no such an issue.

4   EXPERIMENTS

In this section, we evaluate our proposed Algorithms 2 and 3 which we call SGSMC and OHSMC, respectively, in several ways. First, we test the methods on a synthetic model to see if they can recover the true parameters and posterior distribution. Then, we train pBNNs using the methods for regression and classification tasks on synthetic, UCI, and MNIST datasets. Our implementation is publicly available at https://github.com/spdes/pbnn.

Baselines   We compare against (i) the maximum a posteriori (MAP) method for estimating the parameter and use Hamiltonian Monte Carlo (MAP-HMC) to compute the posterior distribution based on the learnt parameter (Sharma et al., 2023, Sec. 6), (ii) stochastic weight averaging Gaussian (SWAG, Maddox et al., 2019) with the MAP objective function, and (iii) stochastic mean-field Gaussian variational Bayes (VB, Hoffman et al., 2013). In addition, SGSMC-HMC refers to sampling the posterior distribution by HMC and estimating the parameters by SGSMC.

Setting   We use Adam for all methods with a learning rate of 0.01 unless stated otherwise. As for the prior, we consistently employ a standard Gaussian distribution. Whenever sampling from the posterior distribution is needed, we use $J = 1,000$ samples. We use the same amount for evaluations. For SGSMC, we use an MCMC random walk kernel, whereas for OHSMC, we use a random walk kernel (with variance 0.01). VB uses 100 MC samples to approximate the evidence lower bound. Further details (e.g., batch sizes, number of epochs, and pBNN structures) for all the experiments are found in the appendix.

4.1   Synthetic parameter estimation

Consider a model given by

$$\begin{bmatrix}\phi_0\\\phi_1\end{bmatrix} \sim \text{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}2 & 0\\0 & 1\end{bmatrix}\right),$$
$$y_n \mid \phi \sim \text{N}\left(\frac{\phi_1}{\psi} + \frac{1}{2}\left(\phi_0^2 + \psi^2\right), 1\right), \tag{8}$$

where we generate a sequence of 100 independent data points $y_{1:100}$ under the true parameter $\psi = 1$ and the realisation $\phi = \begin{bmatrix}0 & 0\end{bmatrix}$. The goal is to estimate the
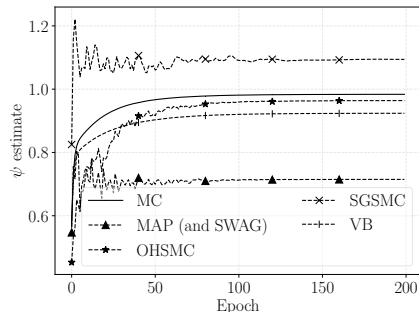
Figure 1: Traces of parameter esti-
mations. The true value is 1.
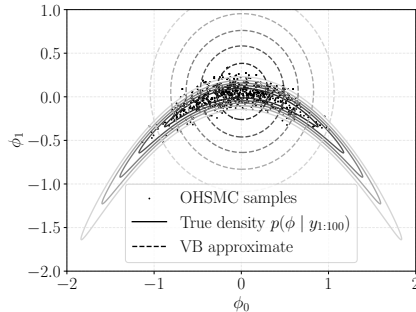
Figure 2: Posterior approximation
for the model in Equation (8).

Table 1: One-dimensional regression
results. The best result for each col-
umn is bold.

Method	NLPD (std.)	RMSE (std.)
MAP-HMC	1.53 (0.09)	0.59 (0.11)
OHSMC	**1.49** (0.07)	**0.53** (0.07)
SGSMC-HMC	1.65 (0.16)	0.75 (0.14)
SWAG	1.71 (0.14)	0.82 (0.16)
VB	2.13 (0.15)	1.46 (0.46)

parameter ψ and also to compute the posterior distribution which has a non-Gaussian crescent shape. For this model, we can conveniently compute a tight lower bound of the MLE objective by brute-force Monte Carlo with 10,000 samples, which we call MC. However, for training pBNNs, the MC approach is in general not applicable.

In Figure 1, we see that the MC estimate is closest to the truth followed by OHSMC, VB, and SGSMC. The MAP estimator on the other hand, significantly diverges from the truth. Moreover, from Figure 2 we see that the OHSMC samples are close to the true density, although the samples do not explore the tails of the true distribution. The VB estimate is correct in the mean approximation, but the Gaussian approximation does not fit the shape of the true distribution. The SWAG method, which uses the results based on the MAP estimate, produces a covariance matrix whose diagonal is numerically zero.

4.2 Synthetic regression and classification

Regression Next, we benchmark the algorithms on a synthetic regression problem where we have access to the true underlying function. The training, validation, and test data are generated as per $y_n = f(x_n) + \xi_n$, for $n = 1, 2, \ldots, 100$, where $f(x) = x \sin(x \tanh(x))$, and i.i.d. noise $\xi_n \sim N(0, 1)$. We use a three-layer pBNN (with output sizes 20, 10, and 1), where the stochastic part is on the second hidden layer. We repeat the experiment 100 times and report the averaged negative log predictive density (NLPD) on the test data, and root mean-square error (RMSE) on the true function including their standard deviations.

Table 1 shows that the proposed OHSMC method performs best in both metrics. Furthermore, Figure 3 shows that the predictive samples drawn from the pBNNs learnt by OHSMC best capture the local optima of the true function. Moreover, OHSMC extrapolates the regression problem best.

Table 2: Classification results in two-moon data.

Method	NLPD (std.)	ECE (std.)	Acc. (std.)
MAP-HMC	**0.28** (0.06)	0.07 (0.01)	0.87 (0.02)
OHSMC	**0.28** (0.07)	**0.06** (0.01)	**0.88** (0.02)
SGSMC-HMC	0.32 (0.08)	0.08 (0.01)	0.86 (0.03)
SWAG	0.31 (0.06)	0.07 (0.01)	0.86 (0.03)
VB	0.29 (0.05)	0.08 (0.01)	0.86 (0.03)

Classification As for classification, we test the methods on a synthetic two-moon dataset. Apart from NLPD, we also report the expected calibration error (ECE, Guo et al., 2017) and accuracy. The utilised neural network has four dense layers (with output sizes 100, 20, 5, and 1), where the stochastic part is the third layer.

Table 2 shows that our OHSMC either performs better than its peers or on par in terms of NLPD. According to the ECE and accuracy metrics, OHSMC is best.

4.3 UCI regression and classification

We now move from synthetic experiments to real-world UCI data (Kelly et al., Accessed 2023). Throughout, we use a neural network with four dense layers (with output sizes 50, 20, 5, and C, where C is the number of labels), and place the stochastic part on the third layer. All experiments are repeated ten times and we report averaged metrics.

Table 3 shows the results on two regression and two classification UCI datasets. As before, our proposed OHSMC method either outperforms or performs on par with other methods in terms of NLPD for all datasets. As for regression, the RMSE is significantly lower for OHSMC. Moreover, our methods are marginally better than the others when evaluating ECE for the classification tasks. Results on additional UCI datasets are found in Appendix C.
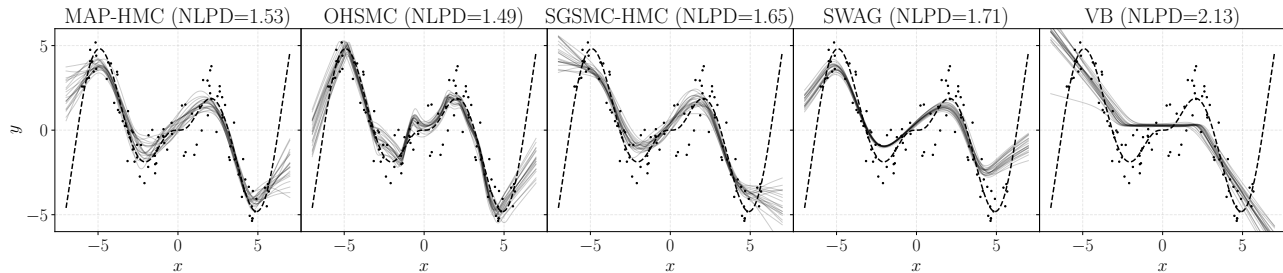
Figure 3: Visualisation of the synthetic regression. The scatter points represent the test data and the dashed line depicts the true function. The grey lines are predictive samples from their learnt pBNNs.
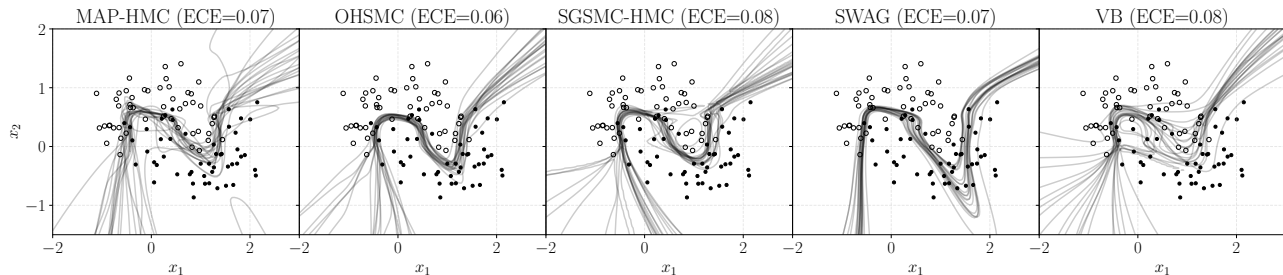

Figure 4: Visualisation of the two-moons classifications. The scatter points are the test data with hollow/solid representing the label. The grey lines represent the classification hyperplanes sampled from the trained pBNNs.

Table 3: Results on two regression and two classification UCI datasets. Results on more UCI datasets are provided in Table 8 in Appendix C.

Method	yacht		energy		satellite		ionosphere	
	NLPD	RMSE	NLPD	RMSE	NLPD	ECE	NLPD	ECE
MAP-HMC	0.95 (0.01)	0.27 (0.04)	0.94 (0.00)	0.24 (0.02)	0.28 (0.03)	0.04 (0.00)	**0.21** (0.09)	0.09 (0.03)
OHSMC	**0.92** (0.00)	**0.11** (0.04)	**0.92** (0.00)	**0.09** (0.00)	**0.27** (0.03)	**0.03** (0.00)	**0.21** (0.13)	**0.08** (0.03)
SGSMC-HMC	0.93 (0.00)	0.17 (0.02)	**0.92** (0.00)	0.14 (0.02)	0.84 (0.70)	0.10 (0.10)	0.24 (0.19)	**0.08** (0.04)
SWAG	0.95 (0.02)	0.25 (0.08)	0.95 (0.00)	0.25 (0.02)	0.35 (0.07)	0.07 (0.03)	0.36 (0.12)	0.14 (0.05)
VB	0.94 (0.00)	0.20 (0.02)	0.93 (0.00)	0.19 (0.03)	0.28 (0.04)	**0.03** (0.00)	0.28 (0.16)	0.10 (0.05)

4.4 MNIST classification

We also consider classification on MNIST data (LeCun et al., Accessed 2023) by using a neural network with two convolutional layers followed by two dense layers. The stochastic part is on the first convolution layer. Due to their high computational cost, HMC methods do not apply here. Table 4 shows that our OHSMC method significantly outperforms other methods according to all the three metrics.

In this MNIST task, OHSMC (with 100 samples), SWAG, and VB (with 100 samples) take around 0.80, 0.26, and 0.33 seconds, respectively, for running 1,000 iterations with batch size 20. Note that the time for OHSMC includes *both*, the parameter estimation and posterior sampling, while for SWAG the time is only for parameter estimation. The times are profiled on an NVIDIA A100 40GB GPU.

Table 4: Classification results on MNIST. Within the table, NLPD and ECE are scaled by 10^{-2}.

Method	NLPD (std.)	ECE (std.)	Acc. (std.)
OHSMC	**2.87** (0.24)	**0.35** (0.06)	**99.02%** (0.07)
SWAG	4.27 (0.83)	1.17 (0.43)	97.76% (0.71)
VB	4.53 (0.28)	0.44 (0.08)	98.51% (0.09)

5 CONCLUSIONS

In this paper, we have shown a sequential Monte Carlo (SMC) routine to efficiently train partial Bayesian neural networks (pBNNs). Specifically, we have proposed two approximate SMC samplers that are suitable for estimating the parameters and posterior distributions of pBNNs. The proposed training scheme either out-

performs, or performs on par with state-of-the-art methods on a variety of datasets.

Limitations and future work Algorithm 3 does not necessarily converge to the target distribution (under *specified* prior and likelihood models) due to its bias. However, the bias in the pBNN context is not an issue when we opt for the predictive performance as we have empirically demonstrated in our experiments. As future work, it is interesting to study how the algorithm performs for inference in more general latent variable models, and to analyse its theoretical convergence properties.

Acknowledgments

References

Christophe Andrieu and Gareth O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2):697–725, 2009.

Gaurav Arya, Moritz Schauer, Frank Schäfer, and Christopher Rackauckas. Automatic differentiation of programs with discrete randomness. In *Proceedings of Advances in Neural Information Processing Systems*, volume 35, pages 10435–10447. Curran Associates, Inc., 2022.

Gaurav Arya, Ruben Seyer, Frank Schäfer, Kartik Chandra, Alexander K. Lew, Mathieu Huot, Vikash K. Mansinghka, Jonathan Ragan-Kelley, Christopher Rackauckas, and Moritz Schauer. Differentiating Metropolis-Hastings to optimize intractable densities. In *The 40th International Conference on Machine Learning Workshop: Differentiable Almost Everything*, 2023.

Bradley M. Bell. The iterated Kalman smoother as a Gauss–Newton method. *SIAM Journal on Optimization*, 4(3):626–636, 1994.

Alexandros Beskos, Omiros Papaspiliopoulos, and Gareth O. Roberts. Retrospective exact simulation of diffusion sample paths with applications. *Bernoulli*, 12(6):1077–1098, 2006.

Christopher M. Bishop. *Pattern recognition and machine learning.* Springer, 2006.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112 (518):859–877, 2017.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1613–1622. PMLR, 2015.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Alberto Cabezas, Junpeng Lao, and Rémi Louf. Blackjax: a sampling library for JAX, 2023. URL `http://github.com/blackjax-devs/blackjax`.

Olivier Cappé, Eric Moulines, and Tobias Rydén. *Inference in hidden Markov models.* Springer Series in Statistics. Springer-Verlag, 2005.

Peter G. Chang, Matt Jone, and Keven Murphy. On diagonal approximations to the extended Kalman filter for online training of Bayesian neural networks. In *Proceedings of the 14th Asian Conference on Machine Learning Workshop*. OpenReview, 2022.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 1683–1691. PMLR, 2014.

Nicolas Chopin. A sequential particle filter method for static models. *Biometrika*, 89(3):539–551, 2002.

Nicolas Chopin and Omiros Papaspiliopoulos. *An introduction to sequential Monte Carlo.* Springer Series in Statistics. Springer Nature Switzerland, 2020.

Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 2100–2111. PMLR, 2021.

Hai-Dang Dau and Nicolas Chopin. Waste-free sequential Monte Carlo. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(1):114–148, 2021.

Erik Daxberger, Eric Nalisnick, James U. Allingham, Javier Antorán, and José Miguel Hernández-Lobato.

Bayesian deep learning via subnetwork inference. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 2510–2521. PMLR, 2021.

João De Freitas, Mahesan Niranjan, Andrew H. Gee, and Arnaud Doucet. Sequential Monte Carlo methods to train neural network models. *Neural Computation*, 12(4):955–993, 2000.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(3):411–436, 2006.

Paul Fearnhead, Omiros Papaspiliopoulos, and Gareth O. Roberts. Particle filters for partially observed diffusions. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 70(4):755–777, 2008.

Andrew Foong, David Burt, Yingzhen Li, and Richard Turner. On the expressiveness of approximate inference in Bayesian neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, volume 33, pages 15897–15908, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1050–1059. PMLR, 2016.

Mathieu Gerber and Randal Douc. A global stochastic optimization particle filter algorithm. *Biometrika*, 109(4):937–955, 2021.

Gene H. Golub and Géard Meurant. *Matrices, moments and quadrature with applications*. Princeton series in applied mathematics. Princeton University Press, 2010.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1321–1330. PMLR, 2017.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: a neural network library and ecosystem for JAX, 2023. URL `http://github.com/google/flax`.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14 (40):1303–1347, 2013.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2018.

Pavel Izmailov, Wesley J. Maddox, Polina Kirichenko, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Subspace inference for Bayesian deep learning. In *Proceedings of the 35th Uncertainty in Artificial Intelligence Conference*, pages 1169–1179. PMLR, 2020.

Pavel Izmailov, Sharad Vikram, Matthew D. Hoffman, and Andrew Gordon Wilson. What are Bayesian neural network posteriors really like? In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 4629–4640. PMLR, 2021.

Pierre E. Jacob and Alexandre H. Thiery. On nonnegative unbiased estimators. *The Annals of Statistics*, 43(2):769–784, 2015.

Pierre E. Jacob, John O'Leary, and Yves F. Atchadé. Unbiased Markov chain Monte Carlo methods with couplings. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 82(3):543–600, 2020.

Ruiyang Jin, Sumeetpal S. Singh, and Nicolas Chopin. De-biasing particle filtering for a continuous time hidden Markov model with a Cox process observation model. *arXiv preprint arXiv:2206.10478*, 2022.

Adam M. Johansen, Arnaud Doucet, and Manuel Davy. Particle methods for maximum likelihood estimation in latent variable models. *Statistics and Computing*, 18(1):47–57, 2008.

Nikolas Kantas, Arnaud Doucet, Sumeetpal S. Singh, Jan Maciejowski, and Nicolas Chopin. On particle methods for parameter estimation in state-space models. *Statistical Science*, 30(3):328–351, 2015.

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. UCI machine learning repository, Accessed 2023. URL `http://archive.ics.uci.edu/ml`.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 5436–5446. PMLR, 2020.

Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The MNIST database of handwritten digits, Accessed 2023. URL `http://yann.lecun.com/exdb/mnist/`.

Anthony Lee, Christopher Yau, Michael B. Giles, Arnaud Doucet, and Christopher C. Holmes. On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *Journal of Computational and Graphical Statistics*, 19(4):769–789, 2010.

Wesley J. Maddox, Pavel Izmailov, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. A

simple baseline for Bayesian uncertainty in deep learning. In *Proceedings of Advances in Neural Information Processing Systems*, volume 32, 2019.

Reinaldo A. Gomes Marques and Geir Storvik. Particle move-reweighting strategies for online inference. Technical report, University of Oslo and Statistics for Innovation Centre, 2013.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. Variational sequential Monte Carlo. In *Proceedings of the 21th International Conference on Artificial Intelligence and Statistics*, volume 84, pages 968–977. PMLR, 2018.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

Sebastian W. Ober and Carl Edward Rasmussen. Benchmarking the neural linear model for regression. In *Proceedings of the 2nd Symposium on Advances in Approximate Bayesian Inference*, pages 1–25, 2019.

Yann Ollivier. Online natural gradient as a Kalman filter. *Electronic Journal of Statistics*, 12(2):2930–2961, 2018.

Manfred Opper. A Bayesian approach to on-line learning. In *On-line learning in neural networks*. Cambridge University Press, 1999.

Theodore Papamarkou, Jacob Hinkle, M. Todd Young, and David Womble. Challenges in Markov chain Monte Carlo for Bayesian neural networks. *Statistical Science*, 37(3):425–442, 2022.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

George Poyiadjis, Arnaud Doucet, and Sumeetpal S. Singh. Particle approximations of the score and observed information matrix in state space models with application to parameter estimation. *Biometrika*, 98(1):65–80, 2011.

Thomas B. Schön, Adrian Wills, and Brett Ninness. System identification of nonlinear state-space models. *Automatica*, 47(1):39–49, 2011.

Mrinank Sharma, Sebastian Farquhar, Eric Nalisnick, and Tom Rainforth. Do Bayesian neural networks need to be fully stochastic? In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, volume 206, pages 7694–7722. PMLR, 2023.

Sharad Singhal and Lance Wu. Training multilayer perceptrons with the extended Kalman algorithm. In *Proceedings of Advances in Neural Information Processing Systems*, volume 1, pages 133–140. Morgan-Kaufmann, 1988.

Jens Sjölund. A tutorial on parametric variational inference. *arXiv preprint arXiv:2301.01236*, 2023.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, pages 681–688. ACM, 2011.

Andrew G. Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *Proceedings of Advances in Neural Information Processing Systems*, volume 33, pages 4697–4708. Curran Associates, Inc., 2020.

Ruqi Zhang, A. Feder Cooper, and Christopher De Sa. AMAGOLD: amortized Metropolis adjustment for efficient stochastic gradient MCMC. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, volume 108, pages 2142–2152. PMLR, 2020.

Checklist

1. For all models and algorithms presented, check if you include:

   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes] Please see Sections 2 and 3.

   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes] We have discussed the properties and complexities of the algorithms in Sections 2 and 3. We have also shown the computational time at the end of Section 4.4.

   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes] Please see `https://github.com/spdes/pbnn`.

2. For any theoretical claim, check if you include:

   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable] The paper is not theory focused.

   (b) Complete proofs of all theoretical results. [Not Applicable] The paper is not theory focused.

    (c) Clear explanations of any assumptions. [Not Applicable] The paper is not theory focused.

3. For all figures and tables that present empirical results, check if you include:

    (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes] We have released the code at `https://github.com/spdes/pbnn`.

    (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] This information can be found in the appendix.

    (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]

    (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] This information can be found in the appendix.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

    (a) Citations of the creator If your work uses existing assets. [Yes]

    (b) The license information of the assets, if applicable. [Not Applicable]

    (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]

    (d) Information about consent from data providers/curators. [Not Applicable]

    (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

    (a) The full text of instructions given to participants and screenshots. [Not Applicable]

    (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

    (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A   Synthetic parameter estimation in Section 4.1

The aim of this experiment is to test whether the proposed methods can estimate the model parameters and posterior distributions. Details of the experimental setup are given as follows.

Optimisation   We use the Adam optimiser with a learning rate $r(i)$ that is exponentially decaying from 0.1 with a speed of 0.96 for 200 epochs, that is, $r(i) = 0.1 \times 0.96^{i/10}$. As a batch size we use 10. The initial values for ϕ and ψ are $\begin{bmatrix} 0 & 0 \end{bmatrix}$ and 0.1, respectively.

OHSMC setting   We use a random walk kernel with a step size of 0.001. We also apply the stratified resampling at every iteration.

SGSMC setting   We apply a Metropolis–Rosenbluth–Teller–Hasting MCMC kernel with a random walk proposal with 10 steps and a step size of 0.001. Other settings are the same as for OHSMC.

Brute-force Monte Carlo setting   By brute-force Monte Carlo, we mean that we estimate the MLE log-likelihood by a Monte Carlo estimation with 10,000 samples. More specifically, we use Monte Carlo to compute the lower bound $\int \sum_{n=1}^{100} \log p(y_n \mid \phi; \psi)\, \pi(\phi)\, \mathrm{d}\phi$.

Hamiltonian Monte Carlo setting   We use 100 leapfrog steps with a step size of 0.01. The mass is an identity matrix. We draw 3,000 samples and discard the first 2,000 burn-in samples.

Variational Bayes setting   The approximate variational family is a mean-field Gaussian initialized with a zero mean and an identity matrix as the covariance. We jointly optimise the variational parameters and model parameters with respect to the evidence lower bound (ELBO), by using 100 Monte Carlo samples to approximate the expectation in the ELBO.

SWAG setting   The SWAG method uses the pre-trained results from the MAP estimator, and then performs 100 epochs for approximating the posterior mean and covariance. For the covariance, we use the low-rank approximation with rank $K = 20$.
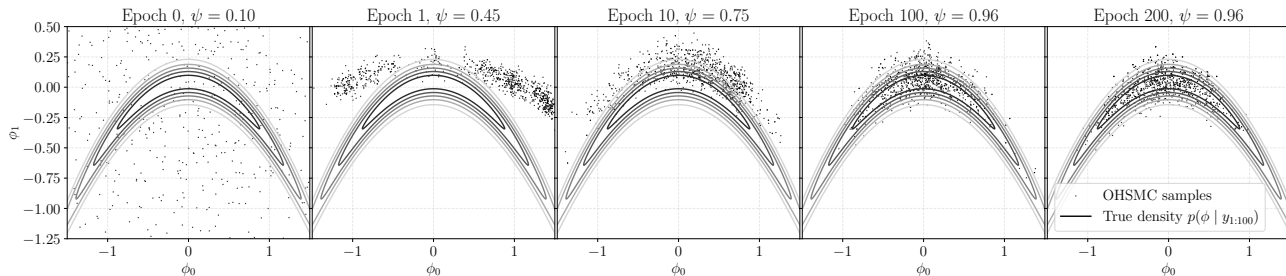

Figure 5: Visualising the flow of the posterior and parameter estimates by OHSMC. In the first epoch, the samples are drawn from the prior. We see that as the epoch increases, the estimates for both ψ and ϕ converge.

In addition to the results presented in Section 4.1, we plot the evolution of the posterior estimation of OHSMC in Figure 5. There, we can see that for this problem the proposed method converges to the true posterior distribution and parameters with a small bias.

B   Synthetic regression and classification in Section 4.2

The aim of this section is to profile the methods for training a partial Bayesian neural network (pBNN) for regression and classification. The details of the experiments are given as follows.

Regression pBNN   The pBNN configuration is shown in Table 5.

Classification pBNN   The pBNN configuration is shown in Table 6.

Table 5: The pBNN configuration for the synthetic regression in Section 4.2. The second layer is stochastic.

Layer	Input	First	Second	Output
Type (dimension) activation	(1)	Dense (20) GeLU	Dense (10) GeLU	Dense (1) None

Table 6: The pBNN configuration for the synthetic classification in Section 4.2. The third layer is stochastic.

Layer	Input	First	Second	Third	Output
Type (dimension) activation	(1)	Dense (100) GeLU	Dense (20) GeLU	Dense (5) GeLU	Dense (1) Sigmoid

Moon data   We create the two-moon data with a function from scikit-learn (Pedregosa et al., 2011) where we set the noise parameter to 0.3.

Optimisation   We use the Adam optimiser with a constant learning rate of 0.01. In total, we use 100 data points each for training, validation, and testing, respectively. As a batch size we use 20. At each batch computation we compute the validation loss and save the best parameters, until we reach a maximum of 200 epochs. For classification the maximum number of epochs is 100.

OHSMC setting   The same as in Section A, except that we choose the random walk variance to be 0.01.

SGSMC setting   The same as in Section A, except that we choose the MCMC random walk variance to be 0.05.

Hamiltonian Monte Carlo setting   The same as in Section A.

Variational Bayes setting   The same as in Section A.

SWAG setting   The same as in Section A, but we perform 200 SWAG iterations for approximating the posterior mean and covariance, and we use $K = 100$.

Recall that we repeat 100 independent experiments. In Figures 6 and 7, we additionally plot five regression and classification experiments, respectively.

C   UCI regression and classification in Section 4.3

The aim of this experiment is to test whether the proposed methods can train pBNNs for real-world regression and classification tasks. In addition to what we have described in Section 4.3, we detail the experimental settings and results as follows.

Datasets description   In Table 8, we show the details (i.e., number of data points and dimensions) of the UCI datasets (Kelly et al., Accessed 2023) that we use.

pBNN   We use the pBNN defined in Table 7 for UCI regression and classification.

Optimisation   The same as in Section B, except that we apply different batch sizes for each UCI dataset as detailed in Table 8.

OHSMC setting   We use two Markov kernels. One is the same random walk kernel as in Section B. The other is that of an Ornstein–Uhlenbeck process with the prior as the stationary distribution, and a terminal time of 0.1. We report the best results under the two Markov kernel choices.

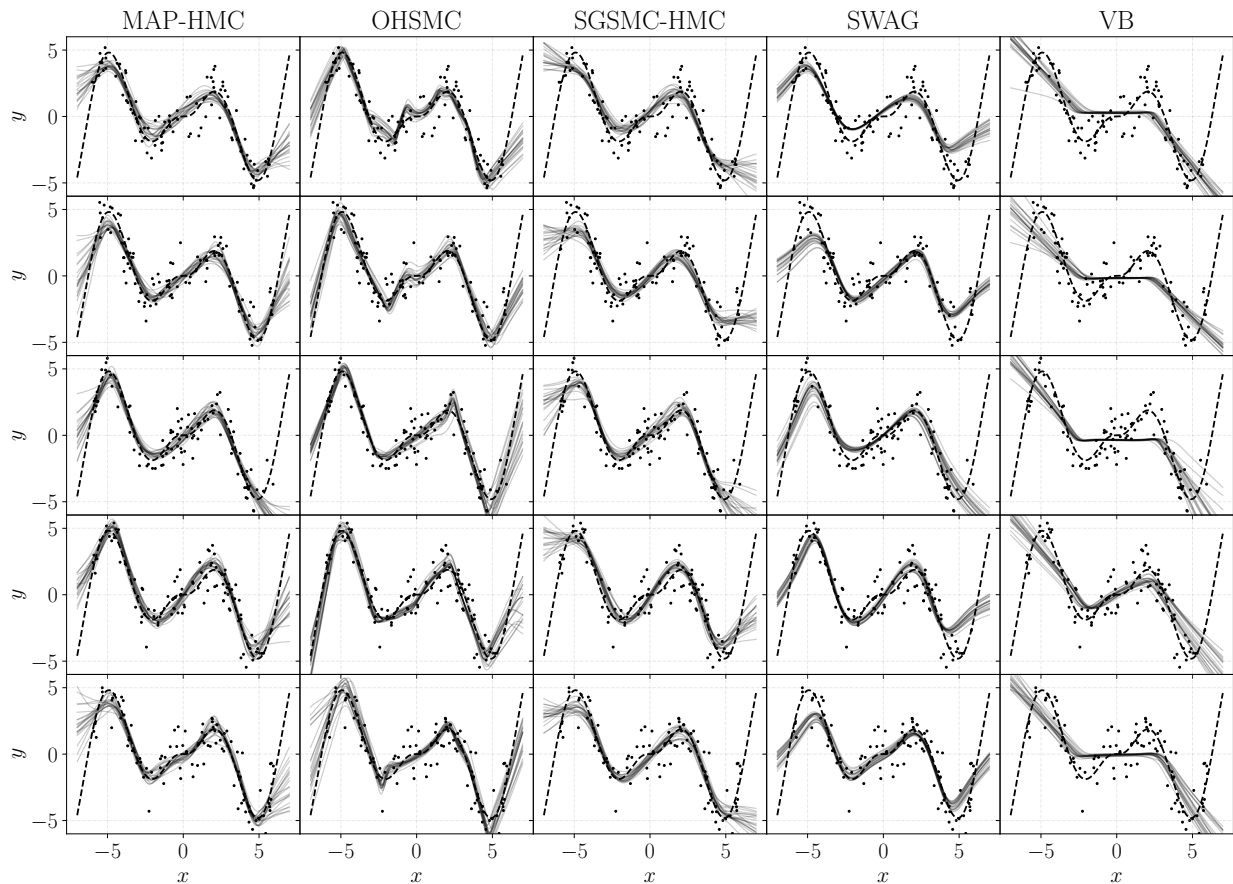Zheng Zhao, Sebastian Mair, Thomas B. Schön, Jens Sjölund

Figure 6: Regression results under five different random seeds. Each row corresponds to another random seed. We see that for all the experiments, OHSMC consistently has the best performance for extrapolation.

Table 7: The pBNN configuration for UCI regression and classification in Section 4.3. The third layer is stochastic. In the table, d_x and $d_y > 1$ stand for the input dimension and number of labels, respectively, depending on the dataset used.

Layer	Input	First	Second	Third	Output
Type (dimension) activation	(d_x)	Dense (50) GeLU	Dense (20) GeLU	Dense (5) GeLU	Dense (d_y) Softmax if $d_y > 1$ else None

SGSMC setting   The same as in Section B.

Hamiltonian Monte Carlo setting   The same as in Section B.

Variational Bayes setting   The same as in Section B.

SWAG setting   The same as in Section B, but we use $K = 50$.

Detailed results   In Section 4.3, we have only shown limited results for two regression and two classification datasets. Table 8 depicts more detailed results for seven regression and five classification tasks. From the table we find that the proposed methods (OHSMC and SGSMC) outperform the other methods on most of the datasets.
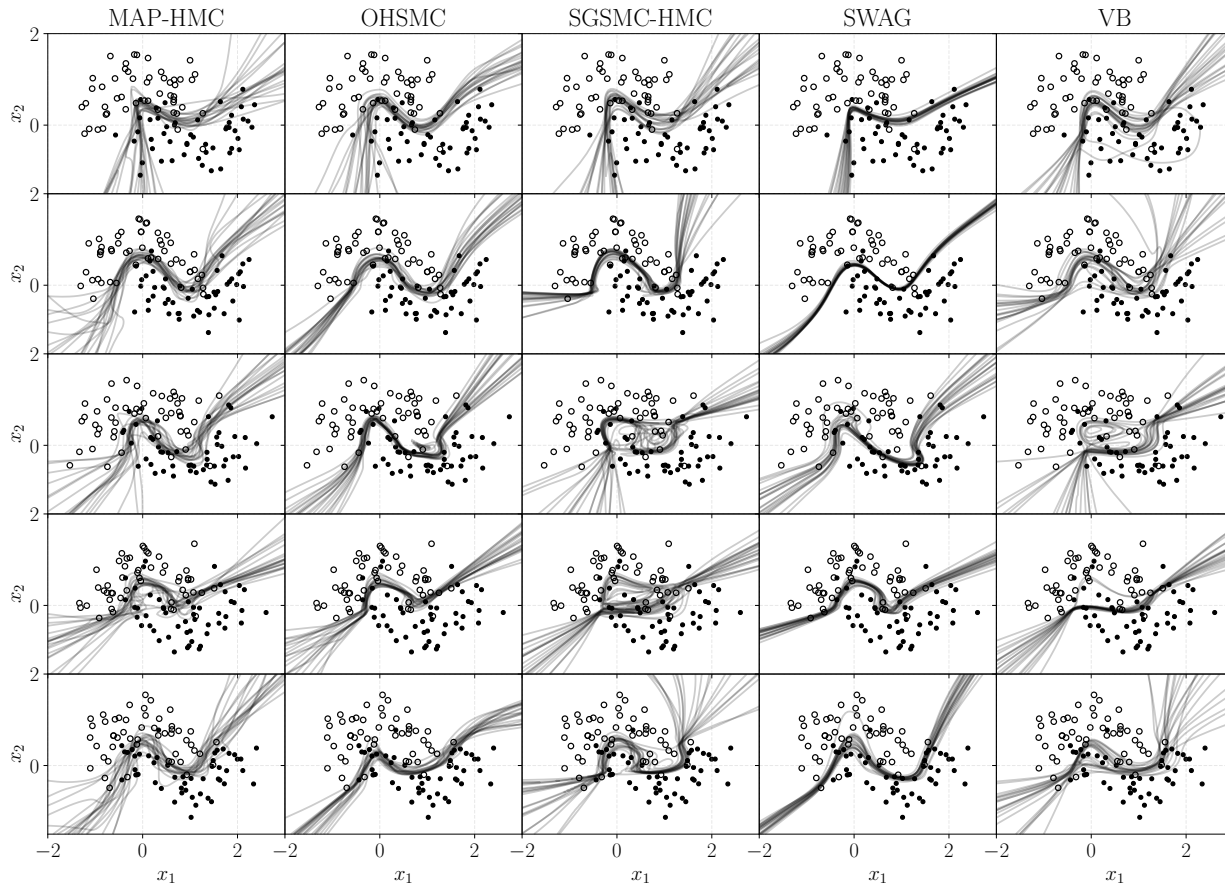
Figure 7: Classification results under five different random seeds. Each row corresponds to another random seed.

D   MNIST classification in Section 4.4

The aim of this experiment is to test whether the proposed methods can be applied on the commonly used MNIST dataset (LeCun et al., Accessed 2023), and to see whether the proposed methods outperform the baselines. For this experiment, the MCMC-based methods no longer apply due to their demanding computation. Detailed experiment settings are given in the following.

pBNN setting   The pBNN configuration is given in Table 9.

Optimisation   We use an an Adam optimiser with a constant learning rate of 0.002. In total, we use 50,000 training, 10,000 validation, and 10,000 test data points. As a batch size we use 100. At each batch computation we compute the validation loss and save the best parameters, until we reach a maximum of 10 epochs. It is worth remarking that the VB method failed to converge for learning rate greater than 0.002.

OHSMC setting   The same as in Section B.

SGSMC setting   The same as in Section B.

Hamiltonian Monte Carlo setting   The same as in Section B.

Variational Bayes setting   The same as in Section B.

SWAG setting   The same as in Section B, but we use $K = 100$ and perform 50 SWAG iterations for the Gaussian approximation.

Table 8: UCI regression and classification results. We uniformly use 60%, 30%, and 10% of the data as training, validation, and test, respectively. Within the tables, N, d_x, and d_y represent the total data points, input dimensions, and number of classes, respectively. For the regression of naval with SGSMC, the method diverged. Acc. stands for accuracy.

Method	boston		concrete		energy		kin8		naval		yacht		power	
	NLPD	RMSE	NLPD	RMSE	NLPD	RMSE	NLPD	RMSE	NLPD	RMSE	NLPD	RMSE	NLPD	RMSE
MAP-HMC	1.04 (0.02)	0.54 (0.09)	1.00 (0.00)	0.42 (0.02)	0.94 (0.00)	0.24 (0.02)	0.96 (0.00)	0.29 (0.00)	0.94 (0.06)	0.10 (0.18)	0.95 (0.01)	0.27 (0.04)	0.95 (0.00)	0.25 (0.01)
OHSMC	**1.00** (0.02)	**0.40** (0.08)	**0.97** (0.01)	**0.33** (0.03)	**0.92** (0.00)	**0.09** (0.00)	**0.95** (0.00)	**0.27** (0.00)	0.92 (0.00)	0.05 (0.00)	**0.92** (0.00)	**0.11** (0.04)	**0.94** (0.00)	**0.24** (0.01)
SGSMC-HMC	1.01 (0.05)	0.42 (0.11)	0.98 (0.00)	0.35 (0.02)	**0.92** (0.00)	0.14 (0.02)	**0.95** (0.00)	0.28 (0.00)	N/A	N/A	0.93 (0.00)	0.18 (0.01)	**0.94** (0.00)	**0.24** (0.01)
SWAG	1.02 (0.02)	0.45 (0.05)	0.99 (0.01)	0.39 (0.04)	0.95 (0.00)	0.25 (0.02)	1.10 (0.20)	0.54 (0.27)	1.05 (0.20)	0.46 (0.36)	0.95 (0.02)	0.25 (0.08)	0.96 (0.01)	0.29 (0.04)
VB	1.01 (0.03)	0.43 (0.07)	**0.97** (0.00)	0.34 (0.01)	**0.93** (0.00)	0.19 (0.03)	**0.95** (0.00)	**0.27** (0.00)	**0.91** (0.00)	**0.04** (0.00)	0.94 (0.00)	0.20 (0.02)	**0.94** (0.00)	**0.24** (0.01)
(N, d_x)	(506, 13)		(1030, 8)		(768, 8)		(8192, 8)		(11934, 14)		(308, 6)		(9568, 4)	
Batch size	50		50		50		50		50		20		50	

Method	australian			cancer			ionosphere			glass			satellite		
	NLPD	ECE	Acc.	NLPD	ECE	Acc.	NLPD	ECE	Acc.	NLPD	ECE	Acc.	NLPD	ECE	Acc.
MAP-HMC	0.36 (0.10)	**0.09** (0.02)	0.85 (0.03)	0.12 (0.06)	0.04 (0.01)	0.95 (0.02)	**0.21** (0.09)	0.09 (0.03)	0.90 (0.04)	1.14 (0.16)	0.27 (0.06)	0.58 (0.08)	0.28 (0.03)	0.04 (0.01)	**0.90** (0.00)
OHSMC	0.37 (0.09)	0.09 (0.02)	0.85 (0.03)	0.14 (0.08)	**0.03** (0.01)	**0.96** (0.02)	**0.21** (0.13)	0.08 (0.04)	0.90 (0.04)	**1.13** (0.32)	0.29 (0.06)	**0.62** (0.06)	**0.27** (0.03)	**0.03** (0.01)	0.89 (0.01)
SGSMC-HMC	0.36 (0.11)	**0.09** (0.03)	0.85 (0.04)	0.12 (0.09)	**0.03** (0.02)	**0.96** (0.02)	0.24 (0.19)	**0.08** (0.04)	**0.92** (0.05)	1.25 (0.32)	0.26 (0.07)	0.56 (0.10)	0.84 (0.74)	0.10 (0.10)	0.79 (0.14)
SWAG	0.36 (0.08)	**0.09** (0.02)	**0.86** (0.03)	0.14 (0.04)	0.06 (0.03)	0.95 (0.02)	0.36 (0.12)	0.14 (0.05)	0.84 (0.08)	1.36 (0.41)	0.26 (0.06)	0.58 (0.10)	0.35 (0.07)	0.07 (0.04)	0.85 (0.03)
VB	**0.34** (0.07)	0.10 (0.04)	0.83 (0.03)	**0.11** (0.08)	0.04 (0.03)	**0.96** (0.02)	0.28 (0.16)	0.10 (0.05)	0.89 (0.05)	1.79 (1.89)	**0.22** (0.09)	0.51 (0.11)	0.28 (0.04)	**0.03** (0.00)	**0.90** (0.00)
(N, d_x, d_y)	(690, 14, 2)			(569, 30, 2)			(351, 33, 2)			(214, 9, 6)			(6435, 36, 6)		
Batch size	50			50			20			20			50		

Table 9: The pBNN for MNIST. The stochastic layer is the first convolution layer.

Layer	Input	First	Second	Third	Output
Type (dimension)	(28, 28, 1)	Conv	Conv	Dense (256)	Dense (10)
Convolution (size)	None	32 filters (3, 3)	64 filters (3, 3)	None	None
Activation	None	ReLU	ReLU	ReLU	Softmax
Pooling (size), (stride)	None	Average pooling (2, 2), (2, 2)	Average pooling (2, 2), (2, 2)	None	None

E   Implementation and computing infrastructure

All experiments are implemented in Python using JAX, Optax (Bradbury et al., 2018), BlackJAX (Cabezas et al., 2023), and Flax (Heek et al., 2023). We run the experiments in the computational cluster Berzelius provided by National Supercomputer Centre at Linköping University, and we uniformly use one NVIDIA A100 40 GB GPU for all the experiments.