Efficient Bayes Inference in Neural Networks through Adaptive Importance Sampling

Yunshi Huang, Emilie Chouzenoux, Víctor Elvira, Jean-Christophe Pesquet

Efficient Bayes Inference in Neural Networks through Adaptive Importance Sampling

Yunshi Huang[a], Emilie Chouzenoux[a,*], Víctor Elvira[b], Jean-Christophe Pesquet[a]

[a] *CVN, Inria Saclay, CentraleSupélec, Université Paris-Saclay, France*
[b] *University of Edinburgh, UK*

Abstract

Bayesian neural networks (BNNs) have received an increased interest in the last years. In BNNs, a complete posterior distribution of the unknown weight and bias parameters of the network is produced during the training stage. This probabilistic estimation offers several advantages with respect to point-wise estimates, in particular, the ability to provide uncertainty quantification when predicting new data. This feature inherent to the Bayesian paradigm, is useful in countless machine learning applications. It is particularly appealing in areas where decision-making has a crucial impact, such as medical healthcare or autonomous driving. The main challenge of BNNs is the computational cost of the training procedure since Bayesian techniques often face a severe curse of dimensionality. Adaptive importance sampling (AIS) is one of the most prominent Monte Carlo methodologies benefiting from sounded convergence guarantees and ease for adaptation. This work aims to show that AIS constitutes a successful approach for designing BNNs. More precisely, we propose a novel algorithm PMC-net that includes an efficient adaptation mechanism, exploiting geometric information on the complex (often multimodal) posterior distribution. Numerical results illustrate the excellent performance and the improved exploration capabilities of the proposed method for both shallow and deep neural networks.

Keywords: Bayesian neural networks, adaptive importance sampling, Bayesian inference, deep learning, confidence intervals, uncertainty quantification.

*Corresponding author
Email address: emilie.chouzenoux@centralesupelec.fr (Emilie Chouzenoux)

1. Introduction

Deep neural networks (DNNs) are often the current state-of-the-art for solving a wide range of diverse tasks in machine learning. They consist in a cascade of linear and nonlinear operators that are optimized from large amounts of labeled data using back-propagation techniques. However, this optimization procedure often relies on ad-hoc machinery which may not lead to relevant local minima without good numerical recipes. Furthermore, it provides no information regarding the uncertainty of the obtained predictions. However, uncertainty is inherent in machine learning, stemming either from noise in/variability of the data values, the sample selection procedure, and the imperfect nature of any developed model. Quantifying this uncertainty is of paramount importance in a wide array of applied fields such as self-driving cars, medicine, or forecasting. Bayesian neural network (BNN) approaches offer a grounded theoretical framework to tackle model uncertainty in the context of DNNs [1].

In the Bayesian inference framework, a statistical model is assumed between the unknown parameters and the given data in order to build a posterior distribution of those unknowns conditioned to the data. However, for most practical models, the posterior distribution is not available in a closed form, mostly due to intractable integrals, and approximations must be performed via Monte Carlo (MC) methods [2]. Importance sampling (IS) is a Monte Carlo family of methods that consists in simulating random samples from a proposal distribution and weighting them properly with the aim of building consistent estimators of the moments of the posterior distribution. The performance of IS depends on the choice of the proposal distribution [3, 4, 5]. Adaptive IS (AIS) is an iterative version of IS where the proposal distributions are adapted based on their performance at previous iterations [6]. In the last decade, many AIS algorithms have been proposed in the literature [7, 8, 9, 10, 11, 12, 13]. However, two main challenges still exist and need to be tackled. First, few AIS algorithms adapt the scale parameter, which is problematic when the unknowns have different orders of magnitude. For instance, the covariance matrix is adapted via robust moment matching strategies in [13, 14]. Second, the use of the geometry of the target for adaptation rule has only been explored scarcely in the recent AIS literature [15, 16, 17]. On the

one hand, optimization-based schemes have been proposed to accelerate MCMC algorithms convergence [18, 19, 20], such as in Metropolis adjusted Langevin algorithm (MALA), which combines an unadjusted Langevin (ULA) update with an acceptance-rejection step. MALA performance can be further improved by a preconditioning strategy [21, 22]. The recent SL-PMC algorithm [23] is up to our knowledge the only AIS-based method that exploits first and second-order information on the target to adapt both the location and scale parameters of the proposals.

BNN inference is usually performed using the variational Bayesian technique [24, 25, 26], which consists in constructing a tractable approximation to the posterior distribution (e.g., based on a mean field approximation). However, the results may be sensitive to the approximation error and to initialization. Promising results have recently been reached by using MC sampling strategies instead. Again, a key ingredient for good performance lies in an efficient adaptation strategy, usually by relying on tools from optimization. The stochastic gradient Langevin dynamics method from [27], a mini-batched version of ULA, seems now to be able to reach state-of-the-art results with reasonable computational cost, as illustrated in [28, 29]. One can also mention the Hamiltonian MC sampler with local scale adaptation, proposed in [30]. In [31], the dropout in the neural network is given by an approximation to the probabilistic deep Gaussian process. In [32], the method called Sequential Anchored Ensembles, trains the ensemble sequentially starting from the previous solution to reduce the computational cost of the training process.

In this paper, we propose the first AIS algorithm for BNN inference. IS-based methods have several advantages w.r.t. MCMC, e.g., all the generated samples are employed in the estimation (i.e., there is no "burn-in" period) and the corresponding adaptive schemes are more flexible (see the theoretical issues of adaptive MCMC in [2, Section 7.6.3],[33]). In return, the challenge is to design adaptive mechanisms for the proposal densities in order to iteratively improve the performance of the IS estimators [6]. We develop a new strategy to adapt efficiently the proposal using a scaled ULA step. The scaling matrix is adapted via robust covariance estimators, using the weighted samples of AIS, thus avoiding the computation of a costly Hessian matrix. Another novelty is the joint mean and covariance adaptation, offering the advantage

of fitting the proposal distributions locally, boosting the exploration and increasing the performance. The most noteworthy feature of the proposed novel approach is its ability to provide meaningful uncertainty quantification with a reasonable computation cost.

Numerical experiments on classification and regression problems illustrate the efficiency of our method when compared to a state-of-the-art back-propagation procedure and other BNN methods. The outline of the paper is as follows. Section 2 introduces the problem and notation related to Bayesian inference in machine learning, and recall the principle of AIS with proposal adaptation. Section 3 presents the BNN inference problem and the proposed AIS algorithm. Section 4 provides numerical results and Section 5 concludes the paper.

2. Motivating framework and background

2.1. Bayesian inference in supervised machine learning

Supervised machine learning aims at estimating a vector of unknown parameters $\boldsymbol{\theta} \in \mathbb{R}^{d_\theta}$ from a training set of N_{train} input/output pairs of data $\left\{\mathbf{x}_0^{(n)}, \mathbf{y}^{(n)}\right\}_{1\leq n\leq N_{\text{train}}} \in \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$. Let us denote by $\mathbf{X}_0 \in \mathbb{R}^{d_x \times N_{\text{train}}}$, and $\mathbf{Y} \in \mathbb{R}^{d_y \times N_{\text{train}}}$ the columnwise concatenation of $\left\{\mathbf{x}_0^{(n)}\right\}_{1\leq n\leq N_{\text{train}}}$, and $\left\{\mathbf{y}^{(n)}\right\}_{1\leq n\leq N_{\text{train}}}$, respectively. The unknown $\boldsymbol{\theta}$ is related to $\mathbf{X}_0$ and $\mathbf{Y}$ through a statistical model given by the likelihood function $\ell(\mathbf{Y} \mid \boldsymbol{\theta}, \mathbf{X}_0)$. The prior probabilistic knowledge about the unknown is summarized in $p(\boldsymbol{\theta})$, $\boldsymbol{\theta}$ being assumed to be independent of $\mathbf{X}_0$. In probabilistic machine learning, the goal is then to infer the posterior distribution

$$p(\boldsymbol{\theta} \mid \mathbf{X}_0, \mathbf{Y}) = \frac{\ell(\mathbf{Y} \mid \boldsymbol{\theta}, \mathbf{X}_0)\, p(\boldsymbol{\theta})}{Z(\mathbf{X}_0, \mathbf{Y})} := \widetilde{\pi}(\boldsymbol{\theta}) \propto \pi(\boldsymbol{\theta}), \tag{1}$$

where $\pi(\boldsymbol{\theta}) := \ell(\mathbf{Y} \mid \boldsymbol{\theta}, \mathbf{X}_0)\, p(\boldsymbol{\theta})$ and $Z = \int \pi(\boldsymbol{\theta}) d\boldsymbol{\theta}$.[1]

Usually the interest is also in computing integrals of the form

$$I = \int h(\boldsymbol{\theta}) \widetilde{\pi}(\boldsymbol{\theta}) d\boldsymbol{\theta}, \tag{2}$$

[1] We now drop $\mathbf{Y}$ and $\mathbf{X}_0$ in Z, $\pi(\boldsymbol{\theta})$, and $\widetilde{\pi}(\boldsymbol{\theta})$ to alleviate the notation.

where h is any integrable function w.r.t. $\widetilde{\pi}(\boldsymbol{\theta})$. However, realistic predictive models in machine learning include non-linearities (e.g., sigmoid activation functions) and loss functions corresponding to non-Gaussian potentials (e.g., cross-entropy). Hence, neither Eq. (2) nor the normalizing constant Z can be computed easily. In this case, we resort to sampling methods to find approximations to the posterior distribution and get access to the uncertainty in the estimation.

2.2. Adaptive Importance Sampling

In the following, we briefly describe the basic importance sampling (IS) methodology and state-of-the-art adaptive IS (AIS) algorithms.

2.2.1. Importance sampling (IS)

Importance sampling (IS) is a Monte Carlo methodology to approximate intractable integrals. The standard IS implementation is composed of two steps. First, K samples are simulated from the so-called proposal distribution $q(\cdot)$, as $\boldsymbol{\theta}_k \sim q(\boldsymbol{\theta})$, $k \in \{1,\dots,K\}$. Second, each sample is assigned an importance weight computed as $w_k = \frac{\pi(\boldsymbol{\theta}_k)}{q(\boldsymbol{\theta}_k)}$, $k \in \{1,\dots,K\}$. The targeted integral given by Eq. (2) can be approximated by the self-normalized IS (SNIS) estimator given by

$$\widetilde{I} = \sum_{k=1}^{K} \overline{w}_k h(\boldsymbol{\theta}_k), \tag{3}$$

where $\overline{w}_k = w_k \Big/ \sum_{j=1}^{K} w_j$ are the normalized weights. The key lies in the selection of $q(\boldsymbol{\theta})$, which must be nonzero for every $\boldsymbol{\theta}$ such that $h(\boldsymbol{\theta})\widetilde{\pi}(\boldsymbol{\theta}) > 0$. For a generic $h(\boldsymbol{\theta})$ (or a bunch of them), a common strategy is to find the proposal $q(\boldsymbol{\theta})$ that minimizes in some sense (e.g., χ^2 divergence [34]) the mismatch with the target $\widetilde{\pi}(\boldsymbol{\theta})$. However, since it is usually impossible to know in advance the best proposal, adaptive mechanisms are employed.

2.2.2. Adaptive importance sampling methods (AIS)

AIS methods are IS-based methods that improve the proposal distribution iteratively [6]. Arguably one of the main families of AIS is the population Monte Carlo (PMC) algorithms [35, 7, 36, 37]. In PMC, the classical sampling-weighting steps

are followed by an adaptation step based on resampling. This mechanism promotes the concentration of proposals in areas where the targeted distribution has significant probability mass [12, Section 4.1]. The DM-PMC in [12] enhances the estimation and adaptation capabilities of existing PMC methods. Recently, the SL-PMC [23] has improved the performance of DM-PMC by incorporating geometric information on the target.

3. Proposed method

3.1. Bayesian neural network model

We focus on the probabilistic inference of the parameters of an L-layer fully connected neural network (FCNN) with $L \geq 1$, relating an input vector of dimension d_x to an output vector of dimension d_y. We will assume that sequences of N_{train} input entries $\{\mathbf{x}_0^{(n)}\}_{1\leq n\leq N_{\text{train}}}$ in $\mathbb{R}^{d_x}$ and of N_{train} associated output values $\left\{\mathbf{y}^{(n)}\right\}_{1\leq n\leq N_{\text{train}}}$ in $\mathbb{R}^{d_y}$ are available. Each layer $\ell \in \{1,\ldots,L\}$ of the considered feedforward neural model is parametrized by a weight matrix $\mathbf{W}_\ell \in \mathbb{R}^{S_\ell \times S_{\ell-1}}$, a bias vector $\mathbf{b}_\ell \in \mathbb{R}^{S_\ell}$, and a nonlinear activation function $\mathcal{R}_\ell$ from $\mathbb{R}^{S_\ell}$ to $\mathbb{R}^{S_\ell}$. For every $n \in \{1,\ldots,N_{\text{train}}\}$ and $\ell \in \{1,\ldots,L\}$,

$$\mathbf{x}_\ell^{(n)} = \mathcal{R}_\ell\left(\mathbf{W}_\ell \mathbf{x}_{\ell-1}^{(n)} + \mathbf{b}_\ell\right). \tag{4}$$

The output vectors $\left\{\mathbf{y}^{(n)}\right\}_{1\leq n\leq N_{\text{train}}}$ are linked to $\{\mathbf{x}_L^{(n)}\}_{1\leq n\leq N_{\text{train}}}$ through the probability density function $p(\mathbf{y}^{(n)}|\mathbf{x}_L^{(n)})$ that depends on the machine learning task of interest. Conditionally to $\{\mathbf{x}_L^{(n)}\}_{1\leq n\leq N_{\text{train}}}$, $\left\{\mathbf{y}^{(n)}\right\}_{1\leq n\leq N_{\text{train}}}$ are assumed to be independent of $\{\mathbf{x}_\ell^{(n)}\}_{1\leq n\leq N_{\text{train}},1\leq \ell\leq L-1}$. The vector $\boldsymbol{\theta} = \{\mathbf{W}_1,\ldots,\mathbf{W}_L,\mathbf{b}_1,\ldots,\mathbf{b}_L\}$ contains all unknown parameters, with size $d_{\boldsymbol{\theta}} = \sum_{\ell=1}^{L} S_\ell(S_{\ell-1}+1)$, so that we can rewrite, for every $n \in \{1,\ldots,N_{\text{train}}\}$, $\mathbf{x}_L^{(n)} = \Phi(\boldsymbol{\theta},\mathbf{x}_0^{(n)})$, with Φ a suitable non-linear mapping directly deduced from Eq. (4). If the involved r.v.'s (ouput samples) are assumed to be i.i.d., the unnormalized version of the targeted distribution is given by

$$\pi(\boldsymbol{\theta}) = p(\boldsymbol{\theta})\,\ell(\mathbf{Y}|\boldsymbol{\theta},\mathbf{X}_0) \tag{5}$$

with

$$\ell(\mathbf{Y}|\boldsymbol{\theta},\mathbf{X}_0) = \prod_{n=1}^{N_{\text{train}}} p(\mathbf{y}^{(n)}|\Phi(\boldsymbol{\theta},\mathbf{x}_0^{(n)})). \tag{6}$$

Hereabove, $p(\boldsymbol{\theta})$ is the prior density on the network parameters that will be useful to limit overfitting issues. Training a BNN thus amounts to learning the distribution $\pi(\boldsymbol{\theta})$ through sampling and/or variational approximation strategies.

We now discuss three illustrations of Eq. (6) corresponding to standard machine learning scenarios.

3.1.1. Regression problem:

A simple regression problem is recovered by considering, for every $n \in \{1,\dots,N_{\text{train}}\}$, $\mathbf{y}^{(n)} \sim \mathcal{N}(\mathbf{x}_L^{(n)}, \sigma^2)$, with $\sigma > 0$, which yields

$$\ell(\mathbf{Y}|\boldsymbol{\theta},\mathbf{X}_0) \propto \exp\left(-\frac{1}{2\sigma^2}\sum_{n=1}^{N_{\text{train}}} \|\Phi(\boldsymbol{\theta},\mathbf{x}_0^{(n)}) - \mathbf{y}^{(n)}\|^2\right). \tag{7}$$

This amounts to using a mean square error (MSE) loss in the standard regression context. A more robust regression can also be performed, by using generalized Gaussian pdfs instead of the Gaussian one.

3.1.2. Binary classification problem:

Binary classification consists of setting $d_y = 1$ and using a Bernoulli model $y^{(n)} \sim \text{Ber}(x_L^{(n)})$, assuming that $x_L^{(n)} \in [0,1]$. This range condition can be easily met by a suitable choice for the output activation function $\mathcal{R}_L$, e.g. sigmoid. Hence,

$$\ell(\mathbf{Y}|\boldsymbol{\theta},\mathbf{X}_0) = \exp\left(\sum_{n=1}^{N_{\text{train}}} \left(y^{(n)} \log\left(\Phi\left(\boldsymbol{\theta},\mathbf{x}_0^{(n)}\right)\right) + \left(1-y^{(n)}\right)\log\left(1-\Phi\left(\boldsymbol{\theta},\mathbf{x}_0^{(n)}\right)\right)\right)\right). \tag{8}$$

3.1.3. General multi-class classification problem:

In classification problems with $C > 2$ classes, we have $d_y = C$ and $\mathbf{y}^{(n)} \in \{0,1\}^C$, which means that the n-th ouput is in class $c \in \{1,\dots,C\}$ if and only if $y_c^{(n)} = 1$, while $y_\ell^{(n)} = 0$ for every $\ell \neq c$ (one-hot encoding). The multinomial model $\mathbf{y}^{(n)} \sim \text{Mult}([\mathbf{x}_L^{(n)}]_1,\dots,[\mathbf{x}_L^{(n)}]_C)$ leads to the generalized cross-entropy training loss:

$$\ell(\mathbf{Y}|\boldsymbol{\theta},\mathbf{X}_0) = \exp\left(\sum_{n=1}^{N_{\text{train}}}\sum_{c=1}^{C} (y_c^{(n)} \log\left([\Phi(\boldsymbol{\theta},\mathbf{x}_0^{(n)})]_c\right)\right). \tag{9}$$

In this case, we must choose $\mathcal{R}_L$ so as to satisfy the unit simplex constraints: $\mathbf{x}_L^{(n)} \in [0,1]^C$ and $\sum_{c=1}^C [\mathbf{x}_L^{(n)}]_c = 1$. For example the soft-max activation function can be used.

3.2. Proposed AIS method

We propose an adaptive importance sampler to deal with the challenges present in the inferential process in BNNs. The (unnormalized) posterior distribution of the unknown parameters $\pi(\boldsymbol{\theta})$ is intricate due to highly non-linear relationships between the unknown parameters and the data, and also possibly to sophisticated priors. We will assume differentiability of the activation functions and of the prior terms, so we can compute and exploit the gradient of $-\log \pi(\boldsymbol{\theta})$ (via the classical back-propagation approach). Our algorithm belongs to the family of population Monte Carlo (PMC) methods, enhanced by a gradient step in the location parameters update, and a robust and efficient covariance adaptation. Moreover, we introduce a light version of our algorithm which, unlike standard PMC methods, follows a mini-batch strategy, leading to a particularly efficient algorithm, from the viewpoint of both computations and storage.

3.2.1. Description of the algorithm

Algorithm 1 shows the proposed PMC algorithm for inference in BNNs that we denote as PMCnet later. Without loss of generality and to ease the description, we initialize the algorithm with M Gaussian proposal distributions, $q_m^{(1)}(\boldsymbol{\theta}) \equiv \mathcal{N}\left(\boldsymbol{\theta}; \boldsymbol{\mu}_m^{(1)}, \boldsymbol{\Sigma}_m^{(1)}\right)$, $m \in \{1,\dots,M\}$, with location (i.e., mean) parameters $\{\boldsymbol{\mu}_m^{(1)}\}_{m=1}^M$ and scale (i.e., covariance) parameters $\{\boldsymbol{\Sigma}_m^{(1)}\}_{m=1}^M$, that will be adapted iteratively. The algorithm consists of T iterations decomposed into four steps. The sampling step is performed in step 2(a), where K samples are simulated from each proposal pdf. In the weighting step 2(b), each of the MK samples receives an IS weight using the whole mixture of proposals in the denominator (see [5] for a proof of the associated variance reduction and increased exploratory capabilities). The resampling step is performed in step 2(c), following the local resampling of [12], simulating the set of auxiliary location parameters $\{\widetilde{\boldsymbol{\mu}}_m^{(t+1)}\}_{m=1}^M$. Under this local resampling, the m-th parameter $\widetilde{\boldsymbol{\mu}}_m^{(t+1)}$ is resampled from the set of K samples generated by the proposal located at $\boldsymbol{\mu}_m^{(t)}$, i.e., from the set $\{\boldsymbol{\theta}_{m,1}^{(t)},\dots,\boldsymbol{\theta}_{m,K}^{(t)}\}$ with associated probabilities $\overline{w}_{m,k}^{(t)} = \frac{w_{m,k}^{(t)}}{\sum_{\ell=1}^K w_{m,\ell}^{(t)}}$, $k \in \{1,\dots,K\}$. This approach guarantees that exactly one sample per proposal survives from t to $t+1$, preserving both diversity and local exploration. In practice, the local resampling then consists in simulating from the categorical distribution $\widetilde{\boldsymbol{\mu}}_m^{(t+1)} \sim \mathrm{Cat}(\{\boldsymbol{\theta}_{m,k}^{(t)}\}_{k=1}^K; \{\overline{w}_{m,k}^{(t)}\}_{k=1}^K)$ for every

Algorithm 1 PMCnet for BNN learning.

1. **[Initialization]**:

   Load training set $\mathbf{X}_0 \in \mathbb{R}^{d_x \times N_{\text{train}}}$ with output values $\mathbf{Y} \in \mathbb{R}^{d_y \times N_{\text{train}}}$.

   Set $\sigma > 0$, $(M, K, T) \in (\mathbb{N}^*)^3$, $(\eta_t, \beta_t)_{1 \leq t \leq T}$.

   For $m \in \{1, \ldots, M\}$, select the initial adaptive parameters $\boldsymbol{\mu}_m^{(1)} \in \mathbb{R}^{d_\theta}$ and $\boldsymbol{\Sigma}_m^{(1)} = \sigma^2 \mathbf{I}_{d_\theta}$.

2. **[For $t = 1$ to T]**:

   (a) Draw K samples from each proposal pdf,

   $$\boldsymbol{\theta}_{m,k}^{(t)} \sim q_m^{(t)}(\boldsymbol{\theta}) \equiv \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_m^{(t)}, \boldsymbol{\Sigma}_m^{(t)}), \tag{10}$$

   with $m \in \{1, \ldots, M\}$, and $k \in \{1, \ldots, K\}$.

   (b) Compute the importance weights

   $$w_{m,k}^{(t)} = \frac{\pi(\boldsymbol{\theta}_{m,k}^{(t)})}{\frac{1}{M} \sum_{i=1}^{M} q_i^{(t)}(\boldsymbol{\theta}_{m,k}^{(t)})}, \tag{11}$$

   with π defined in (5).

   (c) Resample M location parameters $\{\widetilde{\boldsymbol{\mu}}_m^{(t+1)}\}_{m=1}^M$ from the set of MK weighted samples of iteration t using the local resampling strategy (see [12]).

   (d) Adapt the proposal parameters $\{\boldsymbol{\mu}_m^{(t+1)}, \boldsymbol{\Sigma}_m^{(t+1)}\}_{m=1}^M$ according to (12) and (13), respectively.

3. **[Output, $t = T$]**:

   Return the pairs $\{\boldsymbol{\theta}_{m,k}^{(t)}, w_{m,k}^{(t)}\}$, for $m \in \{1, \ldots, M\}$, $k \in \{1, \ldots, K\}$, and $t \in \{1, \ldots, T\}$.

$m \in \{1, \ldots, M\}$. In step 2(d), a scaled Langevin-based update (see the theoretical justification of Langevin equation in MC approaches in [36, 22]) is performed to update the mean of the proposal density at next iteration as

$$\boldsymbol{\mu}_m^{(t+1)} = \widetilde{\boldsymbol{\mu}}_m^{(t+1)} + \gamma_m^{(t+1)} \boldsymbol{\Sigma}_m^{(t+1)} \nabla \log \pi(\widetilde{\boldsymbol{\mu}}_m^{(t+1)}), \tag{12}$$

where $\boldsymbol{\Sigma}_m^{(t+1)}$ is a symmetric definite positive matrix of $\mathbb{R}^{d_\theta \times d_\theta}$, that will also be used as covariance matrix of the proposal density for next iteration, and $\gamma_m^{(t+1)} \in (0, 1]$ is a stepsize adjusted through a simple backtracking procedure. To be specific, for each iteration t and each sample m, we initialize $\gamma_m^{(t+1)}$ to one. We calculate the candidate $\boldsymbol{\mu}_m^{(t+1)}$ using (12) and the related loss value that we compare with the loss of $\widetilde{\boldsymbol{\mu}}_m^{(t+1)}$. If the loss

decreases, the stepsize is accepted, otherwise we start the backtracking process and try stepsize $\gamma_m^{(t+1)}$ reduced by a factor 1/2, until a loss decrease is observed or a maximum number of trials is achieved (typically, 20). In the SL-PMC scheme [36], the covariance matrix of the proposal density was adapted using $\mathbf{\Sigma}_m^{(t+1)} = (-\nabla^2 \pi(\widetilde{\boldsymbol{\mu}}_m^{(t+1)}))^{-1}$, assuming the inversion is well-defined. Here, we propose instead to set $\mathbf{\Sigma}_m^{(t+1)}$ as a cheaper local approximation of the target curvature, using the following robust covariance estimates, for every $m \in \{1, \ldots, M\}$:

$$\mathbf{\Sigma}_m^{(t+1)} = (1-\beta_t)\mathbf{\Sigma}_m^{(t)} + \beta_t(1-\eta_t)\widehat{\mathbf{\Sigma}}_m^{(t)} + \beta_t \eta_t \widetilde{\mathbf{\Sigma}}_m^{(t)}. \tag{13}$$

Hereabove, $\widehat{\mathbf{\Sigma}}_m^{(t)}$ and $\widetilde{\mathbf{\Sigma}}_m^{(t)}$ are two estimators for the covariance of the target. In practice, we set $\widehat{\mathbf{\Sigma}}_m^{(t)}$ as the empirical covariance computed from the K samples and K associated weights at iteration t, and $\widetilde{\mathbf{\Sigma}}_m^{(t)}$ as a biased sample covariance, computed empirically from the K samples from iteration t and a modified version of the K associated weights, where the $\sqrt{K}$ largest weights values are cropped. This cropping procedure generally reduces the variance of the importance sampling estimators [38, 39] (see [40] for a deeper review of these techniques and [41] for a similar approach). Moreover, in (13), $0 < \beta_t \leq 1$ and $(\eta_t)_{1 \leq t \leq T}$ is a decreasing sequence of constants satisfying $\eta_1 = 1$, and $\eta_T = 0$. This convex combination of estimators has shown to provide more stable updates of the covariance matrix thanks to the incremental estimate approach, governed by β_t, and also due to the introduction of lower-variance estimator, $\widetilde{\mathbf{\Sigma}}_m^{(t)}$, which is controlled by η_t. This covariance adaptation has been proposed in [14], in the context of AMIS, an AIS family of alternative algorithms to PMC.

3.2.2. Building the posterior distribution

The proposed PMCnet allows to build an approximation to the posterior distribution of the unknown parameters $\boldsymbol{\theta}$ as

$$\widetilde{\pi}(\boldsymbol{\theta}) \approx \sum_{j=1}^{J} \overline{w}_j \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_j), \tag{14}$$

with $J \geq 1$ where $\overline{w}_j$ and $\boldsymbol{\theta}_j$ are a subset of the MKT weighted samples $\{\boldsymbol{\theta}_{m,k}^{(t)}, w_{m,k}^{(t)}\}$, derived from the PMCnet, for $m \in \{1, \ldots, M\}$, $k \in \{1, \ldots, K\}$, and $t \in \{1, \ldots, T\}$. The weights $\overline{w}_j$ are normalized in such a way that $\sum_{j=1}^{J} \overline{w}_j = 1$. Larger values of J can lead

to a better approximation in (14), but at the price of a high memory burden. A typical practical choice (adopted in our experiments) is to define $\{\overline{w}_j, \boldsymbol{\theta}_j\}_{1\leq j\leq J}$ as the set of the samples produced at the last iteration, $t = T$, so that $J = MK$. The probabilistic characterization of the learned parameters $\boldsymbol{\theta}$ through (14) allows us to turn the learned BNN into a generative model, which is of clear interest in multiple data science applications [42].

In the context of machine learning, the main goal is the probabilistic prediction of the output of a trained network, given a new test input. Thanks to our approximation (14), we can perform this task also in a probabilistic manner. Recall that, in the training set, N_{train} output observations $\left\{\mathbf{y}^{(n)}\right\}_{1\leq n\leq N_{\text{train}}} \in \mathbb{R}^{d_y}$ are related to the inputs $\{\mathbf{x}_0^{(n)}\}_{1\leq n\leq N_{\text{train}}}$ through $p(\mathbf{Y}|\Phi(\boldsymbol{\theta}, \mathbf{X}_0))$. In practice, we obtain only an approximated particle-based version of the posterior of the network parameters, given in (14). It is possible to simulate from this approximation, an approximation to the distribution of the network response (i.e., output) for any given new test data among $\{\mathbf{x}_0^{(n)}\}_{N_{\text{train}}+1\leq n\leq N_{\text{train}}+N_{\text{test}}}$. In particular, the propagation of the posterior mean and some metric assessment comparing the ground truth $\left\{\mathbf{y}^{(n)}\right\}_{N_{\text{train}}+1\leq n\leq N_{\text{train}}+N_{\text{test}}} \in \mathbb{R}^{d_y}$ to the obtained outputs, can be obtained by the procedure described in Algorithm 2. Note that step (b) simply reduces to $\mathbf{y}^{(r,n)} = \Phi(\boldsymbol{\theta}^{(r)}, \mathbf{x}_0^{(n)})$ for the three examples of likelihood presented in Section 3.1.

Algorithm 2 Output computation and metric assessment of PMCnet network on a test set $\{\widetilde{\mathbf{x}}_0^{(n)}\}_{N_{\text{train}}+1\leq n\leq N_{\text{train}}+N_{\text{test}}}$.

---

1. **[Initialization]** Select J pairs $\{\boldsymbol{\theta}_j, w_j\}_{1\leq j\leq J}$ among MKT weigthed samples $\{\boldsymbol{\theta}_{m,k}^{(t)}, w_{m,k}^{(t)}\}_{m,k,t}$ derived from PMCnet. Compute normalized weights $\{\overline{w}_j\}_{1\leq j\leq J}$.
2. **[For** $r = 1, \ldots, R$**]**
   (a) Sample $\boldsymbol{\theta}^{(r)}$ from the set $\{\boldsymbol{\theta}_j\}_{j=1}^J$, with the probability $\{\overline{w}_j\}_{j=1}^J$,
   (b) Evaluate $\mathbf{y}^{(r,n)} = \mathbb{E}(\mathbf{y}|\Phi(\boldsymbol{\theta}^{(r)}, \mathbf{x}_0^{(n)})$ for $n \in \{N_{\text{train}} + 1, \ldots, N_{\text{train}} + N_{\text{test}}\}$,
   (c) Compute the metric of interest, comparing the outputs $\{\mathbf{y}^{(r,n)}\}_{N_{\text{train}}+1\leq n\leq N_{\text{train}}+N_{\text{test}}}$ to ground truth values $\{\mathbf{y}^{(n)}\}_{N_{\text{train}}+1\leq n\leq N_{\text{train}}+N_{\text{test}}}$.

3.2.3. Practical implementation

For reproducibility purposes, we share a repository available at `https://github.com/yunshihuang/PMCnet` with our implementation in PyTorch of the proposed algorithm. The advantage of a framework such as PyTorch is that it contains many built-in functions to deal with nonlinear feedforward neuronal structures such as as (4). These functions make an efficient use of the available GPU resources. Auto-differentiation tools are also used to compute gradient of our target function, without the need for any explicit tedious calculations. In our code, we define the parameters $\boldsymbol{\theta}$ (i.e., weights/biases of the network) as Pytorch tensors through the option `requires_grad=True`. This allows us to compute the loss and its gradient in parallel (using GPU) for a given list of weights/biases values, using `backward` PyTorch function. For instance, the output and the associated target value for M trial samples in our algorithm can be simply evaluated simultaneously, without the need for a loop. Despite these advantages, when the dimension d_θ of the unknown parameters and/or the size of the training set N_{train} increase, the memory cost of the algorithm might still be high. We thus propose a modified version of PMCnet, that avoids memory overflows without being detrimental to the numerical performance of the method (as we will illustrate in our experimental section). Two changes are done, leading to PMCnet-light. First, to cope with large values of d_θ, we propose to modify the definition of the adapted covariance matrix $\boldsymbol{\Sigma}_m^{(t+1)}$ in Eq. (13), using a diagonal scaling instead. For every iteration $t \in \{1,\dots,T\}$ and proposal $m \in \{1,\dots,M\}$, we define

$$\boldsymbol{\Delta}_m^{(t+1)} = (1-\beta_t)\boldsymbol{\Delta}_m^{(t)} + \beta_t(1-\eta_t)\widehat{\boldsymbol{\Delta}}_m^{(t)} + \beta_t\eta_t\widetilde{\boldsymbol{\Delta}}_m^{(t)}. \tag{15}$$

Hereabove, $\boldsymbol{\Delta}_m^{(t+1)}$, $\widehat{\boldsymbol{\Delta}}_m^{(t)}$, and $\widetilde{\boldsymbol{\Delta}}_m^{(t)}$ are diagonal matrices of $\mathbb{R}^{d_\theta \times d_\theta}$. Matrices $\widehat{\boldsymbol{\Delta}}_m^{(t)}$ (resp. $\widetilde{\boldsymbol{\Delta}}_m^{(t)}$) are built in such a way that its diagonal elements match with those of the previously defined $\widehat{\boldsymbol{\Sigma}}_m^{(t)}$ (resp. $\widetilde{\boldsymbol{\Sigma}}_m^{(t)}$). Parameters (β_t, η_t) play the same role as in the original version of the algorithm. Using diagonal matrices here offers the advantages of (i) a reduced memory load, and (ii) a computational complexity decrease in the sampling step in (10).

Second, to tackle large values of N_{train}, we introduce an incremental gradient strategy in our adaptation rule (12). The idea is to approximate the full batch gradient

involved in (12) by one loop (i.e., one epoch) of mini-batch gradient steps. To be specific, let us divide our training set $\{\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\}_{1 \leq n \leq N_{\text{train}}}$ into B batches of equal size N_{train}/B, and denote, for every batch index $b \in \{1, \ldots, B\}$, $\mathbf{X}_b \in \mathbb{R}^{d_x \times \frac{N_{\text{train}}}{B}}$ and $\mathbf{Y}_b \in \mathbb{R}^{d_y \times \frac{N_{\text{train}}}{B}}$, the column-wise concatenation of $\{\mathbf{x}^{(n)}\}_{N_{\text{train}}(b-1)/B+1 \leq n \leq N_{\text{train}}b/B}$ and $\{\mathbf{y}^{(n)}\}_{N_{\text{train}}(b-1)/B+1 \leq n \leq N_{\text{train}}b/B}$, respectively. Then, using (6), for every $\boldsymbol{\theta} \in \mathbb{R}^{d_\theta}$, we have: $\pi(\boldsymbol{\theta}) = \prod_{b=1}^{B} \pi_b(\boldsymbol{\theta})$, with

$$\pi_b(\boldsymbol{\theta}) = (p(\boldsymbol{\theta}))^{1/B} \ell(\mathbf{Y}_b | \boldsymbol{\theta}, \mathbf{X}_b). \tag{16}$$

For any iteration index $t \in \{1, \ldots, T\}$ and a sample index $m \in \{1, \ldots, M\}$, we set,

$$\begin{aligned}
&\widehat{\boldsymbol{\mu}}_{m,1}^{(t+1)} = \widetilde{\boldsymbol{\mu}}_m^{(t+1)} \\
&\text{For } b = 1, \ldots, B \\
&\quad \left\lfloor \quad \widehat{\boldsymbol{\mu}}_{m,b+1}^{(t+1)} = \widehat{\boldsymbol{\mu}}_{m,b}^{(t+1)} + \frac{\gamma_{m,b}^{(t+1)}}{2} \boldsymbol{\Delta}_m^{(t+1)} \nabla \log \pi_b(\widehat{\boldsymbol{\mu}}_{m,b}^{(t+1)}) \,, \right. \qquad (17) \\
&\boldsymbol{\mu}_m^{(t+1)} = \widehat{\boldsymbol{\mu}}_{m,B+1}^{(t+1)}.
\end{aligned}$$

Hereabove, $\gamma_{m,b}^{(t+1)}$ is a stepsize that is adjusted through the backtracking process on the mini-batch loss π_b, similarly to what was done for the full batch version of the algorithm. The diagonal version of the covariance matrix of the proposal $\boldsymbol{\Delta}_m^{(t+1)}$, defined in (15), is kept the same for all batches. The proposed approach can be viewed as running one epoch of an incremental (scaled) gradient strategy. Since the mean parameters are updated B times per iteration and per sample, this adaptation allows for a better exploration of the target. Moreover, this strategy is also less computationally demanding since each mini-batch data can be loaded on the fly, without being fully stored.

4. Numerical experiments

We now illustrate the good performance of PMCnet and its low complexity version PMCnet-light, on a bunch of classification and regression problems involving either shallow NNs or DNNs.

4.1. Numerical settings

4.1.1. Architecture design

For binary classification problems (i.e., $C = 2$), we consider the likelihood (8) associated with the cross-entropy loss function while, for multi-class classification (i.e., $C > 2$), we consider the one associated with the generalized cross-entropy loss in (9). For regression tasks, the Gaussian likelihood corresponding to the MSE loss (7) is used. Concerning the network structure, FCNNs make use of either hyperbolic tangent (*tanh*) or rectified linear unit (ReLU) activation functions $\mathcal{R}_\ell$ in (4), for $\ell \in \{1,\ldots,L-1\}$, as precised hereafter for each example. In binary classification problems, $\mathcal{R}_L$ is the sigmoid function whereas, for multi-class classification, we set $\mathcal{R}_L$ to the softmax function. For regression, $\mathcal{R}_L$ reduces to identity. In all experiments, for simplicity, we choose as a prior distribution of each entry of the unknown parameter $\boldsymbol{\theta}$ an i.i.d. zero-mean Gaussian distribution with constant variance manually finetuned for each network (see more details below).

4.1.2. Comparisons to other methods

Various methods are considered as comparison benchmarks. First, we provide the results obtained by a standard (i.e., non Bayesian) training procedure relying on ADAM optimizer. We use ADAM to either compute the minimizer of the neg-log-likelihood $-\log \mathcal{L}(\mathbf{Y}|\theta,\mathbf{X}_0)$, leading to the maximum likelihood solution denoted ADAM-MLE, or to compute the maximum a posteriori solution ADAM-MAP defined as the minimizer of $-\log(p(\theta)\mathcal{L}(\mathbf{Y}|\theta,\mathbf{X}_0))$. The estimated parameters obtained by ADAM-MLE are used as the initialization of our PMCnet. Then, several state-of-the-art Bayesian neural networks approaches are evaluated, namely Bayes by Backprop (BBP) [43], Stochastic Gradient Langevin Dynamics (SGLD) [27], MC dropout [31], and Sequential Anchored Ensembles (SAE) [32]. BBP uses unbiased estimates of gradients of the cost to learn a variational posterior distribution over the weights. We always choose the diagonal Gaussian distribution as the variational posterior for each weight/bias. SGLD adds noise to a standard stochastic gradient optimization algorithm, with annealing stepsize, to push the iterates towards samples from the sought posterior distribution. MC dropout drops a unit with certain probability, and thus models the variability with

dropout NN models. SAE trains an ensemble of models based on anchored losses, by using a guided walk Metropolis-Hastings procedure with Gaussian transitions, with the aim to provide an estimate of the Bayesian posterior. This method achieved the 2nd (resp. 3rd) place in the light (resp. extended) track of the NeurIPS 2021 Approximate Inference in Bayesian Deep Learning competition. To finetune the hyperparameters for these competitors, we always pick the optimal values based on their respective performance on the validation set of each example. For most competitors, we rely on the implementations available at `https://github.com/JavierAntoran/Bayesian-Neural-Networks`. We thank the authors of SAE for sharing their code.

4.1.3. Evaluation metrics

The performance of each compared methods is quantified using several standard machine learning metrics, which are evaluated on the test set. For classification tasks, we provide accuracy and confusion matrix, defining the predicted class as the one maximizing the model output. For binary classification problems, we are also able to compute precision, recall, specificity, and F1 score, using a threshold of 0.5 on each network output to determine the classification decision. For methods capable of computing probabilistic estimates, we evaluate the performance in terms of mean and standard deviation denoted as std. Precisely, for all the benchmark methods providing probabilistic estimates (namely, SGLD, MCDropout, and SAE), we rely on the strategies described in their seminal papers, to compute the mean, standard deviation and confidence intervals, for the metrics computed on the test set. For our method PMCnet (and its variant PMCnet-light), we rely on the procedure proposed in Table 2. For the sake of readibility, only the mean values are provided in the confusion matrices. As for multi-class classification problems, we compute F1 score averaged over all the classes. For regression tasks, we provide the mean (std, if available) of the mean squared error (MSE) for each method. To further assess the performance of different methods in the classification examples, we also display ROC plots (i.e., false vs true positive rate, for varying threshold values from 0 to 1 defining the predicted class) and compute the associated area-under-curve value (AUC). In multi-class case, a one-versus-all approach

is used to define the ROC and AUC per class. For methods that are able to provide probabilistic estimations, we additionally display some ROC envelopes associated to specific credible intervals (CI) in percentage (95% CI, if not specified otherwise) of the false vs true positive rate.

4.1.4. Training specifications

In all the experiments, we split the dataset into three different parts: (1) training set used for running the inference algorithms, (2) validation set used for choosing the optimal regularization weight related to the parameter prior and the optimal iterations T, and (3) test set used for evaluating quantitatively the performance of the trained models. Unless otherwise specified, the proportion of the train/validation/test is set as 6:2:2, with N_{train} (resp. N_{test}) denoting the number of examples in the training (resp. validation or test) sets. For all the methods, the train/validation/test phases are implemented in Pytorch (version 1.7.0) under Python (version 3.6.10) environment, and run on an Nvidia DGX workstation using one Tesla V100 SXM2 GPU (1290 MHz frequency, 32GB RAM). The code of our method is made available at `https://github.com/yunshihuang/PMCnet` for reproducibility purposes.

4.1.5. PMCnet tuning

Regarding the settings of PMCnet (or PMCnet-light), for $t = 1, \ldots, T$, the parameters of the robust covariance adaptation are set to $\beta_t \equiv 0.5$, $\eta_t \equiv \frac{1}{t}$. Both $\gamma_m^{(t+1)}$ and $\gamma_{m,b}^{(t+1)}$ are initialized as 1 and multiplied by 0.5, sequentially, during the backtracking trial procedure. Unless otherwise stated, we set $(M, K) = (50, 100)$. Since the regularization weight and the iteration number T are essential parameters that can affect the performance of our method, we adopt the following strategy to finetune these hyperparameters. We first run our method on the training set for a relatively large number of iterations (typically, $T = 70$), and run Algorithm 2 on the validation set, using the $J = MK$ samples of the last iteration T and $R = 100$. This allows to calculate the averaged accuracy (or averaged MSE for regression task) on validation set. The variance parameter involved in the prior distribution is set by golden search so as to maximize the averaged accuracy for classification task (resp. minimize the averaged MSE for

regression task). A similar strategy is adopted to set up the optimal iteration number T. Namely, once the regularization weight has been set, we try different values for T, and define it as the smallest value (thus, smallest complexity) allowing to reach stable performance (averaged accuracy or MSE) on the validation set.

Once hyperparameters are set, we run the procedure of section 3.2.2 on the test set, again using $R = 100$, which provides us the mean, variance, and CIs for metrics of interest (using Algorithm 2) as well as distribution plots for the predicted output values on test set examples.

4.2. Experimental results

4.2.1. Performance assessment on a control scenario

We first evaluate our approach PMCnet and its variant PMCnet-light on a control dataset mimicking a binary classification problem. No validation set is used in these simple examples, and we always set $(M,K,T) = (50,100,20)$.

The considered shallow FCNN is described in Table 1. The ground truth labels for the control dataset are generated by setting a given (shallow) FCNN architecture for which the groundtruth vector $\bar{\boldsymbol{\theta}}$ is known. Specifically, we design a synthetic FCNN with $L = 2$ layers, with *tanh* as the activation function for the hidden layer, and random weight and biases entries independently drawn from $\mathcal{N}(0,2^2)$ for every layer. We then feed the network with random N_{train} (resp. N_{test}) inputs entries $\{\mathbf{x}_0^{(n)}\}_{1\leq n\leq N_{\text{train}}}$ (resp. $\{\widetilde{\mathbf{x}}_0^{(n)}\}_{N_{\text{train}}+1\leq n\leq N_{\text{train}}+N_{\text{test}}}$) independently drawn from $\mathcal{N}(0,1)$. This yields N_{train} (resp. N_{test}) associated ground truth output values $\{\mathbf{y}^{(n)}\}_{1\leq n\leq N_{\text{train}}}$ (resp. $\{\widetilde{\mathbf{y}}^{(n)}\}_{N_{\text{train}}+1\leq n\leq N_{\text{train}}+N_{\text{test}}}$) used for loss evaluation at training (resp. metrics evaluation at testing) phases.

We first investigate the influence of the train/test split on PMCnet results. We set $N_{\text{test}} = 400$, and run experiments with either $N_{\text{train}} = 50, 400$, or 1600. We report the resulting classification metrics on the test set in Table 2, for these different values of N_{train}. The performance is good, showing that the method is learning the model in a suitable way, and as expected, the larger training set, the better the classification metrics. Interestingly, the variability of the output metrics obtained by our method is growing as the train/test split is less favorable. Similar conclusions are reached by

inspecting the ROC curves and their CI envelopes in Fig. 1. The envelopes are wider (i.e., exhibit more variability) for smaller train sets, showing the interest and relevance of the provided posterior estimation.

Number of layers L	Number of classes C	Input size S_0	Output size d_y	Number of hidden layers S_1	Number of parameters d_θ
2	2	3	1	3	16

Table 1: Settings of the FCNN architectures for the control dataset.

N_{train}	**AUC**	**Precision**	**Recall**	**Specificity**	**Accuracy**	**F1 score**	**Confusion matrix**
50	0.9183 (0.0191)	0.9347 (0.0216)	0.7087 (0.1294)	**0.9493** (0.0264)	0.8308 (0.0525)	0.7984 (0.0767)	$\begin{bmatrix} 140 & 57 \\ 10 & 193 \end{bmatrix}$
400	0.9234 (0.0070)	0.8946 (0.0364)	**0.8412** (0.0233)	0.9017 (0.0394)	0.8719 (0.0161)	0.8663 (0.0148)	$\begin{bmatrix} 166 & 31 \\ 20 & 183 \end{bmatrix}$
1600	**0.9304** (0.0051)	**0.9410** (0.0196)	0.8396 (0.0202)	0.9483 (0.0193)	**0.8948** (0.0089)	**0.8871** (0.0098)	$\begin{bmatrix} 165 & 32 \\ 10 & 193 \end{bmatrix}$

Table 2: Results of PMCnet on test set, for binary classification on control dataset using different values for N_{train}.

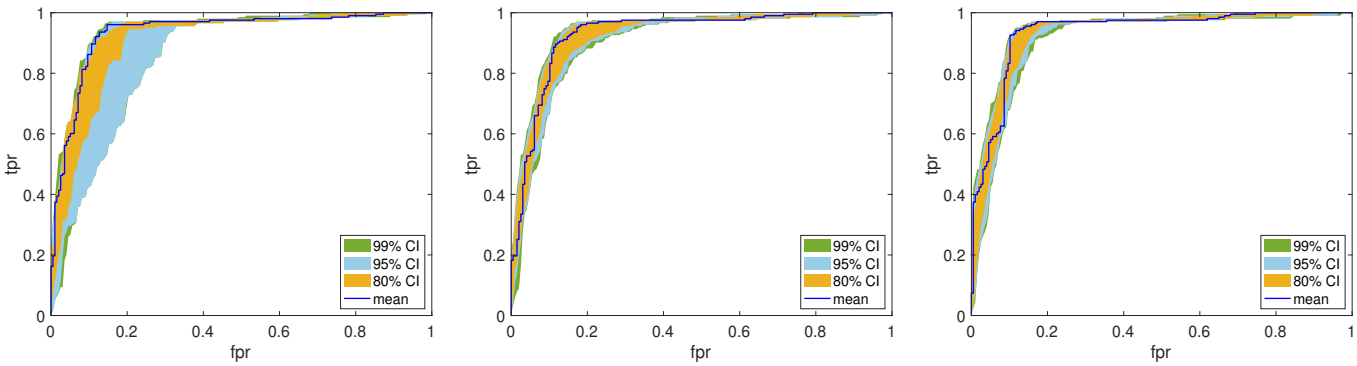

Figure 1: ROC curves of PMCnet on test set, on control dataset using (a) $N_{\text{train}} = 50$, (b) $N_{\text{train}} = 400$ and (c) $N_{\text{train}} = 1600$ with 80% CI, 95% CI and 99% CI respectively.

We now illustrate the performance of PMCnet when compared to its low complexity variant PMCnet-light, on the same control dataset. We now set $N_{\text{train}} = 1600$. The results on the test set for PMCnet and PMCnet-light are summarized in Table 3. The (probabilistic) ROC curves are provided in Fig. 2, and the metrics evolution (on test set) along training are displayed in Fig. 3. Both methods reach very similar metrics. Interestingly, both AUC and accuracy metrics converge much faster using PMCnet-light. Moreover, the latter method displays much less variability, as it can be seen in the std values and the ROC envelopes. This might be an effect of the mini-batch strategy during training phase that allows to update the unknowns after each mini-batch of samples instead of just once for each iteration. It is worth noting that the variability information provided in the ROC envelopes is a mix of the variability inherent to the dataset (which decreases with more training samples) and the variability due to the sampling strategy itself (which decreases with more particles/iterations).

Method	**AUC**	**Precision**	**Recall**	**Specificity**	**Accuracy**	**F1 score**	**Confusion matrix**
PMCnet	0.9304 (0.0051)	0.9410 (0.0196)	0.8396 (0.0202)	0.9483 (0.0193)	0.8948 (0.0089)	0.8871 (0.0098)	$\begin{bmatrix} 165 & 32 \\ 10 & 193 \end{bmatrix}$
PMCnet-light	**0.9336** (0.0009)	**0.9499** (0.0095)	**0.8552** (0.0060)	**0.9561** (0.0093)	**0.9064** (0.0021)	**0.9000** (0.0016)	$\begin{bmatrix} 168 & 29 \\ 9 & 194 \end{bmatrix}$

Table 3: Results on test set for binary classification task of the control dataset with $N_{\text{train}} = 1600$, using PMCnet and PMCnet-light, respectively.

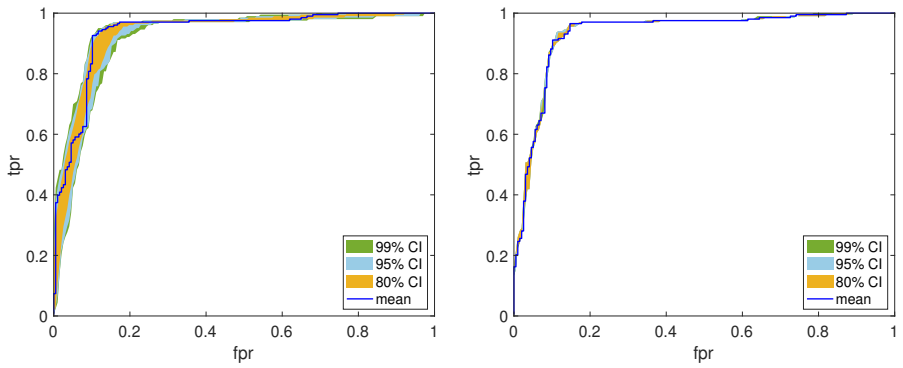

Figure 2: ROC curves (mean and CIs) on test set, using control dataset with $N_{\text{train}} = 1600$, with (a) PMCnet and (b) PMCnet-light.

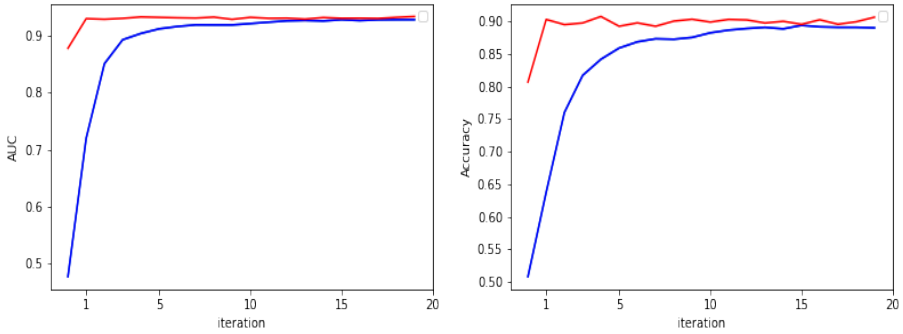

Figure 3: Averaged AUC (left) and accuracy (right) on test set along iterations $t \in \{1,\ldots,20\}$ on the control dataset, using PMCnet (blue) and PMCnet-light (red).

4.2.2. Comparison with benchmarks on shallow networks

We now compare our method with the benchmarks introduced in Sec. 4.1.2. We first consider a shallow FCNN with only one hidden layer of few units, i.e., $L = 2$ with small S_1, and three small size classification datasets of LIBSVM library available at `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`. On all examples, the dimension of the unknown parameters d_θ is low (few hundreds). The architectures of

the chosen shallow FCNNs, as well as the full dataset size (train + validation + test) given by $N_{\text{train}} + 2N_{\text{test}}$, are summarized in Table 4 for each dataset. We make use of *tanh* as the activation function of the hidden layer. We set $(M, K) = (50, 100)$ for our PMCnet and finetune T for each dataset. We provide the number of samples per class for the train/validation/test splits of each classification datasets, in Tables 5, 6 and 7. We can observe that the dataset *Glass* has a major class imbalance, while the other datasets are relatively balanced.

Dataset	**Size** $N_{\text{train}} + 2N_{\text{test}}$	**Number of layers** L	**Number of classes** C	**Input size** S_0	**Output size** d_y	**Number of hidden layers** S_1	**Number of parameters** d_θ
Ionosphere	351	2	2	33	1	5	176
Wine	178	2	3	13	3	3	54
Glass	214	2	6	9	6	10	166

Table 4: Settings of the shallow FCNN architectures for each dataset.

Set name	**Class 0**	**Class 1**
Training set	81	129
Validation set	23	47
Test set	22	49

Table 5: Labels distribution on training, validation and test sets on dataset *Ionosphere*.

Set name	**Class 1**	**Class 2**	**Class 3**
Training set	31	44	31
Validation set	14	13	9
Test set	14	14	8

Table 6: Labels distribution on training, validation and test sets on dataset *Wine*.

Set name	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Training set	45	42	11	8	7	15
Validation set	12	14	4	4	1	8
Test set	13	20	2	1	1	6

Table 7: Labels distribution on training, validation and test sets on dataset *Glass*.

Let us start with the results obtained for the binary classification problem with dataset *Ionosphere*. We summarize the metrics obtained on test sets in Table 8. Our finetuning led to $T = 50$. We can observe that our proposed method reaches best precision, specificity, accuracy and F1 among all the methods, with a small variability. ADAM-MAP reaches good performance too on this experiment. Among Bayesian-based competitors, MCDropout, and SAE reach the best results, however with lower accuracy than our method. We also display ROC curves (and CIs, when available) in Fig. 4. This illustrates again that our proposed algorithm provides a good predictive performance with small variability. In Fig. 5, we pick some examples from the test sets of the dataset and display the histograms (with 10 bins) of $\{\mathbf{y}^{(r,n)}\}_{1\leq r\leq R}$ obtained by Algorithm 2. For better readability, we also superimpose a red curve obtained by simply fitting a beta distribution on the obtained histograms. One can see that our proposed algorithm has the ability to provide a meaningful probabilistic information about its binary classification decision. For instance, for the example displayed in top right of Fig. 5, our method gives the classification decision with high confidence, leading to peaky histogram. In contrast, on the other examples, the method shows more variability (i.e., less confidence) in its decision, leading to a more spread histogram for the estimated mean. Such information can be of high interest for practitioners, in sensitive fields such as healthcare.

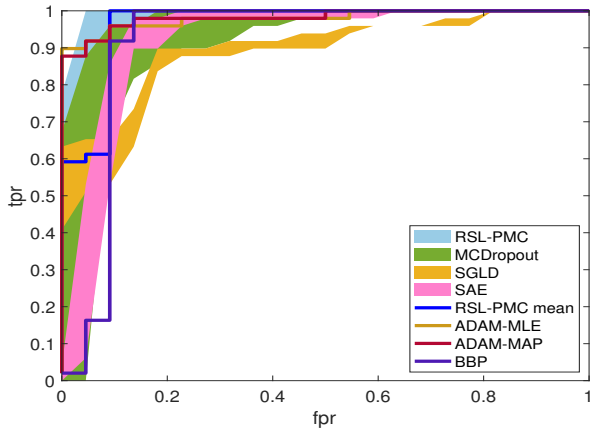

Figure 4: ROC curves for different estimators, on test set, for dataset *Ionosphere*.

Method	AUC	Precision	Recall	Specificity	Accuracy	F1 score	Confusion matrix
ADAM-MLE	0.9796	0.8077	**0.9545**	0.8980	0.9155	0.8750	$\begin{bmatrix} 21 & 1 \\ 5 & 44 \end{bmatrix}$
ADAM-MAP	**0.9814**	0.8333	0.9091	0.9184	0.9155	0.8696	$\begin{bmatrix} 20 & 2 \\ 4 & 45 \end{bmatrix}$
BBP	0.9137	0.8333	0.9091	0.9184	0.9155	0.8696	$\begin{bmatrix} 20 & 2 \\ 4 & 45 \end{bmatrix}$
SGLD	0.8824 (0.0110)	0.6959 (0.0132)	0.8100 (0.0175)	0.8410 (0.0083)	0.8314 (0.0088)	0.7486 (0.0134)	$\begin{bmatrix} 18 & 4 \\ 8 & 41 \end{bmatrix}$
MCDropout	0.9159 (0.0186)	0.8678 (0.0349)	0.7736 (0.0514)	**0.9465** (0.0160)	0.8930 (0.0177)	0.8169 (0.0335)	$\begin{bmatrix} 17 & 5 \\ 3 & 46 \end{bmatrix}$
SAE	0.9064 (0.0036)	0.8268 (0.0272)	0.8609 (0.0108)	0.9186 (0.0151)	0.9007 (0.0120)	0.8433 (0.0170)	$\begin{bmatrix} 19 & 3 \\ 4 & 45 \end{bmatrix}$
PMCnet	0.9642 (0.0146)	**0.8543** (0.0684)	0.9145 (0.0216)	0.9269 (0.0391)	**0.9231** (0.0283)	**0.8819** (0.0393)	$\begin{bmatrix} 20 & 2 \\ 4 & 45 \end{bmatrix}$

Table 8: Results for binary classification, computed on test set, on dataset *Ionosphere*.

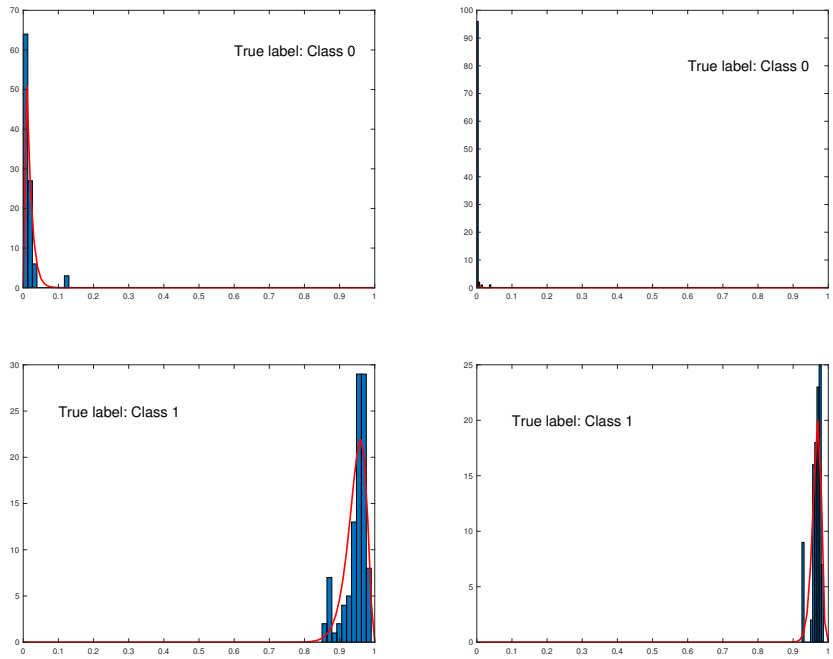

Figure 5: Distributions of the PMCnet network classification decision on four examples extracted from test sets of dataset *Ionosphere*.

Next, we focus on the multi-class classification problems related to datasets *Wine* and *Glass*. The numerical results are summarized in Table 9. We can see again that our methods gives the best accuracy for both datasets. Even more, for the dataset *Wine*, it can perfectly predict the class of wine on the considered test set. As for the dataset *Glass*, the performance of every method is not as good as those for the dataset *Wine*, probably due to the high class imbalance in this dataset. Our method still outperforms others in this dataset with acceptable variability. We also draw the heat map of the confusion matrix associated to these results in Fig. 6. Clearly, the PMCnet method leads to confusion matrices with the most zeros on the off-diagonal axis. For dataset *Wine*, we plot the mean of AUC and accuracy of test set along iterations during the training phase of PMCnet in Fig. 7. We can see that, as the training goes on, both AUC and accuracy tend to stabilize, which shows the validity of our method.

	Method	AUC	F1 score	Accuracy
Wine	ADAM-MLE	0.9950	0.9028	0.8889
	ADAM-MAP	**0.9999**	0.9521	0.9286
	BBP	0.9912	0.9761	0.9444
	SGLD	0.9776 (0.0046)	0.9396 (0.0202)	0.9297 (0.0234)
	MCDropout	0.9857 (0.0087)	0.8929 (0.0415)	0.8867 (0.0402)
	SAE	0.9938 (0.0151)	0.9669 (0.0609)	0.9836 (0.0290)
	PMCnet	**0.9974** (0.0048)	**0.9951** (0.0191)	**0.9944** (0.0215)
Glass	ADAM-MLE	0.8361	0.7240	0.6742
	ADAM-MAP	0.8347	0.7247	0.6744
	BBP	0.8360	0.6187	0.6512
	SGLD	0.8498 (0.0029)	0.7259 (0.0105)	0.7026 (0.0095)
	MCDropout	0.7877 (0.0297)	0.4045 (0.0504)	0.5502 (0.0486)
	SAE	0.7680 (0.0053)	0.7087 (0.1499)	0.7151 (0.0128)
	PMCnet	**0.8510** (0.0059)	**0.7715** (0.0580)	**0.7414** (0.0264)

Table 9: Results for multi-class classification on dataset *Wine* and dataset *Glass* respectively.

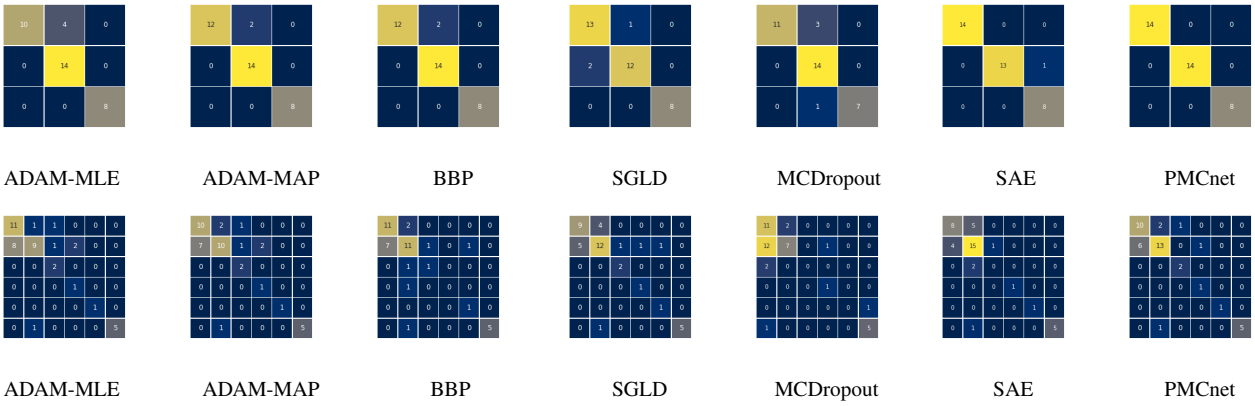

Figure 6: Confusion matrices on test set, for the different benchmarks on dataset *Wine* (top) and dataset *Glass* (bottom).

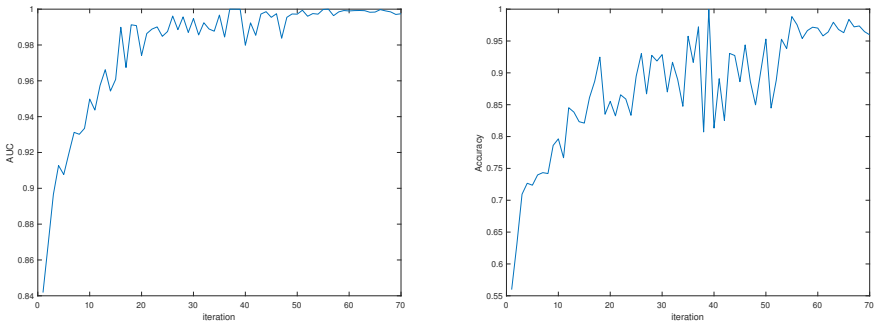

Figure 7: Averaged AUC and accuracy curves, on test set, as a function of iterations during training using PMCnet, on dataset *Wine*.

As for dataset *Glass*, we plot the mean of AUC and accuracy of test set along iterations in Fig. 8. Again, we can see the proposed method converges gradually.

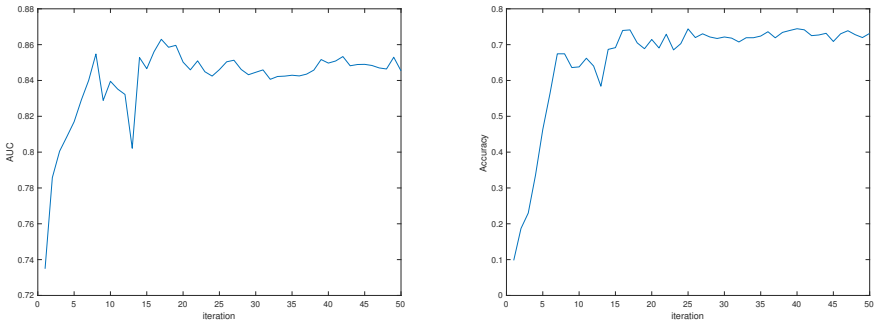

Figure 8: Averaged AUC and accuracy curves, on test set, as a function of iterations during training using PMCnet, on dataset *Glass*.

4.2.3. Deep networks

We now move to comparisons of the methods for learning the parameters of DNNs, from larger scale datasets. Namely, we focus on *MNIST-C* [44] dataset for multi-class classification task and *Protein* (available at `https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure`) for regression task. Note that in the original dataset *MNIST-C*, there are

several different variants of the input images, here we are only using hand-written digits possibly corrupted by rotation. The sample size for training set, validation set, and test set is 45000, 15000 and 10000 respectively, involving grayscale digit images of size 28×28 as input. There are 10 classes to distinguish. The architectures for the DNNs employed for both datasets are described in Table 10. For dataset *MNIST-C*, we retain LeNet-5 as the DNN for the classifier as it was used in 2021 NeurIPS competition [45]. As for dataset *Protein*, we adopt an FCNN with two wide hidden layers of 100 units (i.e., $L = 3$, $S_1 = S_2 = 100$) and we choose ReLU as the activation function for hidden layers. On such large datasets, only PMCnet-light version can be used, and we set $(M, K) = (20, 20)$ with T finetuned for each dataset.

Dataset	**Sample size**	**Networks**	**Number of classes** C	**Input size** S_0	**Output size** d_y	**Number of parameters** d_θ
MNIST-C	70000	LeNet-5	10	784	10	61706
Protein	45730	FCNN	$\times$	9	1	11201

Table 10: Settings of the DNN architectures retained for classification and regression datasets.

First we provide the results obtained for different methods on dataset *MNIST-C* in Table 11, we choose $T = 20$ in this case. Hereagain, the classifier derived by our method is the best among all the competitors in terms of both AUC and accuracy. Note that, on this dataset, almost all the probabilistic methods make predictions with small variability. Even so, our proposed method still performs best with a mean AUC close to 1 and very high accuracy, nearly one percent higher than benchmarks. Considering the sample size of this large dataset, this is already a great improvement. Our methods also reaches the best averaged F1 score. The heat map for confusion matrix of each method is provided in Fig. 9, we can see that for each class, PMCnet-light is able to predict most digits correctly.

Method	AUC	F1 score	Accuracy
ADAM-MLE	0.9998	0.9833	0.9834
ADAM-MAP	0.9988	0.9547	0.9550
BBP	0.9998	0.9855	0.9856
SGLD	0.9949 (0.0008)	0.9115 (0.0135)	0.9126 (0.0132)
MCDropout	0.9998 (0)	0.9837 (0.0008)	0.9838 (0.0008)
SAE	0.9998 (0)	0.9824 (0.0009)	**0.9998** (0.0003)
PMCnet-light	**1.0000** (0)	**0.9913** (0.0002)	0.9914 (0.0002)

Table 11: Results for multi-class classification on dataset *MNIST-C*, computed on test set.

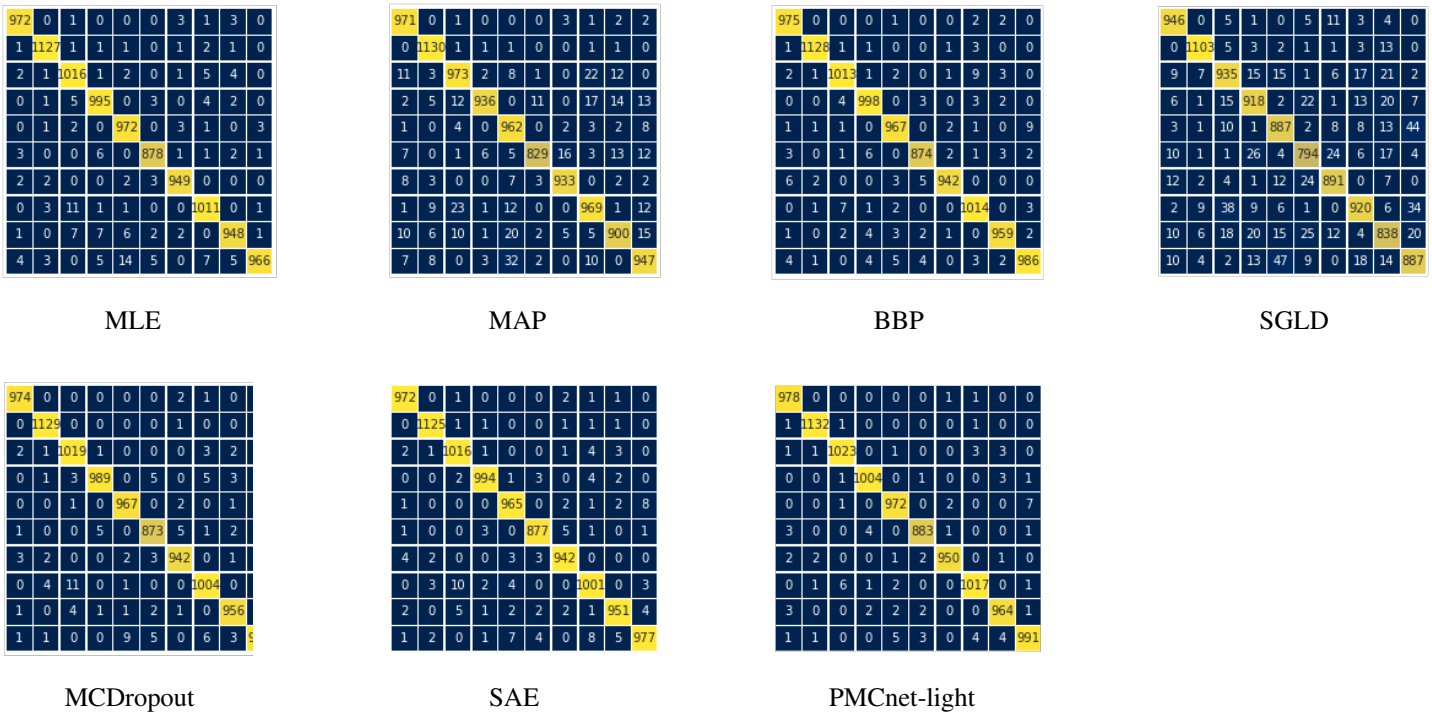

Figure 9: Confusion matrices for the different benchmarks on test set of dataset *MNIST-C*.

We finally discuss the performance of the compared methods in the dataset *Protein*. The numerical results are summarized in Table 12. For this dataset, we choose $T = 5$. The estimated output from our method has the smallest averaged MSE with rather

small variability. We also display the box and violin plot of the square errors for each test examples, in Fig. 10. Among all methods, PMCnet-light not only has the smallest MSE in average, but also shows minimal spreading of outliers, which shows again the good performance of our method.

Method	**MSE**
ADAM-MLE	17.4054
ADAM-MAP	18.6500
BBP	15.9396
SGLD	17.4039 (0.2598)
MCDropout	17.3996 (0.1430)
SAE	20.2291 (0.0297)
PMCnet-light	**15.2049** (0.0496)

Table 12: Results for regression problem on test set of dataset *Protein*.

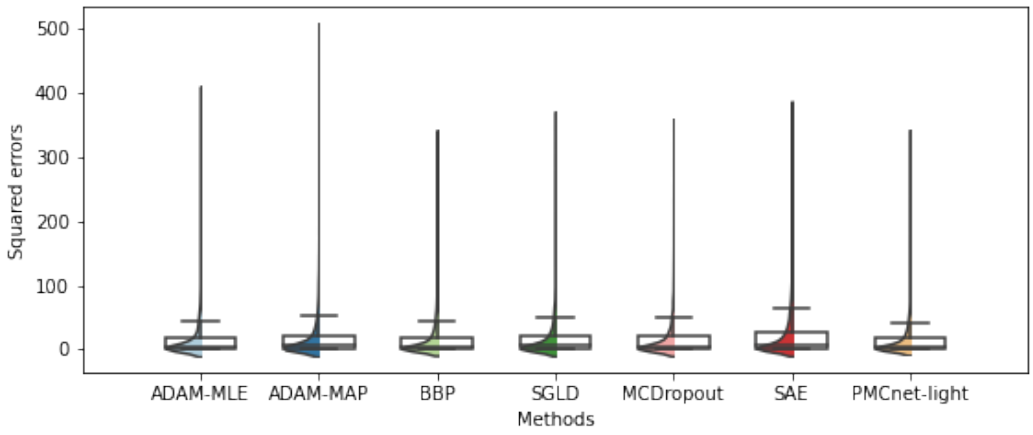

Figure 10: Box and violin plot of the squared errors of each sample in test set of dataset *Protein*.

4.3. Computational complexity

We finalize our experimental section by providing the computational time of different methods. We display the (total) training and test (averaged per sample) times

in Table 13 for dataset *Ionosphere* and in Table 14 for dataset *MNIST-C*. Note that the training times do not include the hyperparameter finetuning, necessary for all approaches. In a nutshell, PMCnet approach is competitive in terms of computational time, both for training and test phases. In particular, the PMCnet-light variant has fast training, compared to the competitors achieving great performance.

Phase	**ADAM-MLE**	**ADAM-MAP**	**BBP**	**SGLD**	**MCDropout**	**SAE**	**PMCnet**	**PMCnet-light**
Training (s.)	108	152	195	17	10	392	450	250
Test (s.)	1	1	1	1	1	2	2	2

Table 13: Computational time in seconds for each method during training phase (full time) and test phase (averaged time per sample) for dataset *Ionosphere*, using the same experimental scenario as in Table 8.

Phase	**ADAM-MLE**	**ADAM-MAP**	**BBP**	**SGLD**	**MCDropout**	**SAE**	**PMCnet-light**
Training (s.)	1760	1800	1035	560	680	2100	1240
Test (s.)	7	7	7	7	7	10	20

Table 14: Computational time in seconds for each method during training phase (full time) and test phase (averaged time per sample) for dataset *MNIST-C*, using same experimental scenario as in Table 11.

5. Conclusion

This paper proposes a novel method for the Bayesian inference in neural networks that relies on adaptive importance sampling. The method is able to characterize the posterior distribution of the estimated outputs and approximate the moments of such distribution. We also propose a light version of the method for handling large datasets and deep networks. This version avoids an excess of the required memory during the training process without impacting the performance of the algorithm. The numerical experiments illustrate the good performance of this new method on severam datasets

of classification and regression, using either shallow or deep networks. Our method is also competitive in terms of computational time on both training and test stages.

References

[1] C. M. Bishop, Bayesian methods for neural networks, Tech. rep., https://www.microsoft.com/en-us/research/wp-content/uploads/1995/01/NCRG_95_009.pdf (1997).

[2] C. P. Robert, G. Casella, Monte Carlo Statistical Methods, Springer, 2004.

[3] V. Elvira, L. Martino, D. Luengo, M. F. Bugallo, Efficient multiple importance sampling estimators, IEEE Sig. Process. Lett. 22 (10) (2015) 1757–1761.

[4] V. Elvira, L. Martino, D. Luengo, M. F. Bugallo, Heretical multiple importance sampling, IEEE Signal Process. Lett. 23 (10) (2016) 1474–1478.

[5] V. Elvira, L. Martino, D. Luengo, M. F. Bugallo, Generalized multiple importance sampling, Stat. Sci. 34 (1) (2019) 129–155.

[6] M. F. Bugallo, V. Elvira, L. Martino, D. Luengo, J. Míguez, P. M. Djuric, Adaptive importance sampling: The past, the present, and the future, IEEE Signal Process. Mag. 34 (4) (2017) 60–79.

[7] O. Cappé, R. Douc, A. Guillin, J. M. Marin, C. P. Robert, Adaptive importance sampling in general mixture classes, Stat. Comput. 18 (2008) 447–459.

[8] J. M. Cornuet, J. M. Marin, A. Mira, C. P. Robert, Adaptive multiple importance sampling, Scand. Stat. Theory Appl. 39 (4) (2012) 798–812.

[9] L. Martino, V. Elvira, D. Luengo, J. Corander, An adaptive population importance sampler, in: Proc. of ICASSP 2014, Florence, Italy, 2014, pp. 8088–8092.

[10] L. Martino, V. Elvira, D. Luengo, J. Corander, An adaptive population importance sampler: Learning from the uncertanity, IEEE Trans. Sig. Process. 63 (16) (2015) 4422–4437.

[11] L. Martino, V. Elvira, D. Luengo, J. Corander, Layered adaptive importance sampling, Stat. Comput. 27 (2017) 599–623.

[12] V. Elvira, L. Martino, D. Luengo, M. F. Bugallo, Improving Population Monte Carlo: Alternative weighting and resampling schemes, Sig. Process. 131 (12) (2017) 77–91.

[13] Y. El-Laham, V. Elvira, M. F. Bugallo, Robust covariance adaptation in adaptive importance sampling, IEEE Sig. Process. Lett. 25 (9) (2018) 1049 – 1053.

[14] Y. El-Laham, M. L., V. Elvira, M. F. Bugallo, Recursive shrinkage covariance learning in adaptive importance sampling, in: Proc. of CAMSAP 2019, Le Gosier, French Indies, 2019, pp. 1–5.

[15] I. Schuster, Gradient importance sampling, Tech. rep. (2015). arXiv:1507.05781.

[16] M. Fasiolo, F. E. de Melo, S. Maskell, Langevin incremental mixture importance sampling, Stat. Comput. 28 (3) (2018) 549–561.

[17] V. Elvira, L. Martino, L. Luengo, J. Corander, A gradient adaptive population importance sampler, in: Proc. of ICASSP 2015, Brisbane, Australia, 2015, pp. 4075–4079.

[18] G. O. Roberts, O. Stramer, Langevin diffusions and Metropolis-Hastings algorithms, Methodol. Comput. Appl. Probab. 4 (4) (2002) 337–357.

[19] A. Durmus, E. Moulines, M. Pereyra, Efficient Bayesian computation by proximal Markov chain Monte Carlo: when Langevin meets Moreau, SIAM J. Imaging Sci. 11 (1) (2018) 473–506.

[20] A. Schreck, G. Fort, S. L. Corff, E. Moulines, A shrinkage-thresholding Metropolis adjusted Langevin algorithm for Bayesian variable selection, IEEE J. Sel. Top. Signal Process. 10 (2) (2016) 366–375.

[21] M. Girolami, B. Calderhead, Riemann manifold Langevin and Hamiltonian Monte Carlo methods, J. R. Stat. Soc. Series B Stat. Methodol. 73 (91) (2011) 123–214.

[22] Y. Marnissi, E. Chouzenoux, A. Benazza-Benyahia, J. Pesquet, Majorize-Minimize adapted Metropolis-Hastings algorithm, IEEE Trans. Sig. Process. 68 (2018) 2356–2369.

[23] V. Elvira, É. Chouzenoux, Langevin-based strategy for efficient proposal adaptation in population monte carlo, in: Proc. of ICASSP 2019, Brighton, UK, 2019, pp. 5077–5081.

[24] G. E. Hinton, D. van Camp, Keeping the neural networks simple by minimizing the description length of the weights, in: Proc. of COLT 1993, 1993, p. 5–13.

[25] D. Barber, C. M. Bishop, Ensemble learning in Bayesian neural networks, Tech. rep., https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bishop-ensemble-nato-98.pdf (1998).

[26] S. Sun, G. Zhang, J. Shi, R. Grosse, Functional variational Bayesian neural networks, in: Proc. of ICLR 2019, New Orleans, USA, 2019.

[27] M. Welling, Y. W. Teh, Bayesian learning via stochastic gradient Langevin dynamics, in: Proc. of ICML 2011, Bellevue, Washington, USA, 2011, p. 681–688.

[28] K.-C. Wang, P. Vicol, J. Lucas, L. Gu, R. Grosse, R. Zemel, Adversarial distillation of Bayesian neural network posteriors, in: Proc. of PMLR 2018, Vol. 80, Stockholm, Sweden, 2018.

[29] Z. Nado, J. Snoek, R. Grosse, D. Duvenaud, X. B., J. Martens, Stochastic gradient langevin dynamics that exploit neural network structure, in: Proc. of ICLR 2018, Vancouver, Canada, 2018.

[30] J. Springenberg, K. A., S. Falkner, F. Hutter, Bayesian optimization with robust Bayesian neural networks, in: Proc. of NIPS 2016, Vol. 29, Barcelona, Spain, 2016.

[31] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, in: Proc. of ICML 2016, New York City, NY, USA, 2016.

[32] A. Delaunoy, G. Louppe, Sae: Sequential anchored ensembles (2022). arXiv:2201.00649.

[33] C. Andrieu, J. Thoms, A tutorial on adaptive MCMC, Stat. Comput. 18 (4) (2008) 343–373.

[34] Ö. D. Akyildiz, J. Míguez, Convergence rates for optimised adaptive importance samplers, Statistics and Computing 31 (2) (2021) 1–17.

[35] O. Cappé, A. Guillin, J. M. Marin, C. P. Robert, Population Monte Carlo, Journal of Comp. Graph. Stat. 13 (4) (2004) 907–929.

[36] V. Elvira, E. Chouzenoux, Optimized population Monte Carlo, IEEE Trans. Sig. Process. 70 (2022) 2489–2501.

[37] C. Miller, J. N. Corcoran, M. D. Schneider, Rare events via cross-entropy population monte carlo, IEEE Signal Processing Letters 29 (2021) 439–443.

[38] E. L. Ionides, Truncated importance sampling, J. Comput. Graph. Stat. 17 (2) (2008) 295–311.

[39] E. Koblents, J. Miguez, Robust mixture population Monte Carlo scheme with adaptation of the number of components, in: Proc. EUSIPCO 2013, Marrakech, Morocco, 2013, pp. 1–5.

[40] L. Martino, V. Elvira, J. Míguez, A. Artés-Rodríguez, P. Djurić, A comparison of clipping strategies for importance sampling, in: Proc. of SSP 2018, 2018, pp. 558–562.

[41] A. Vehtari, D. Simpson, A. Gelman, Y. Yao, J. Gabry, Pareto smoothed importance sampling (2021). arXiv:1507.02646.

[42] K. P. Murphy, Machine learning: a probabilistic perspective, MIT press, 2012.

[43] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, in: Proc. of ICML 2015, Vol. 37, 2015, pp. 1613–1622.

[44] N. Mu, J. Gilmer, Mnist-c: A robustness benchmark for computer vision, Tech. rep. (2019). arXiv:1906.02337.

[45] A. G. Wilson, S. Lotfi, S. Vikram, M. D. Hoffman, Y. Gal, Y. Li, M. F. Pradier, A. Foong, S. Farquhar, P. Izmailov, Evaluating approximate inference in Bayesian deep learning, in: Proc. of NEURIPS 2021, Vol. 176, 2021, pp. 113–124.