International Journal for Uncertainty Quantification, 14(1):43–60 (2024)

A BAYESIAN NEURAL NETWORK APPROACH TO MULTI-FIDELITY SURROGATE MODELING

Baptiste Kerleguer,[1,2,*] Claire Cannamela,[1] & Josselin Garnier[2]

[1]*Commissariat à l'Énergie Atomique et aus Energies Alternatives (CEA), DAM, DIF, Arpajon, France*

[2]*Centre de Mathématiques Appliquées, Ecole Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau Cedex, France*

*Address all correspondence to: Baptiste Kerleguer, CEA, DAM, DIF, F-91297, Arpajon, France, E-mail: baptiste.kerleguer@polytechnique.edu

Original Manuscript Submitted: 6/3/2022; Final Draft Received: 4/2/2023

This paper deals with surrogate modeling of a computer code output in a hierarchical multi-fidelity context, i.e., when the output can be evaluated at different levels of accuracy and computational cost. Using observations of the output at low- and high-fidelity levels, we propose a method that combines Gaussian process (GP) regression and the Bayesian neural network (BNN), called the GPBNN method. The low-fidelity output is treated as a single-fidelity code using classical GP regression. The high-fidelity output is approximated by a BNN that incorporates, in addition to the high-fidelity observations, well-chosen realizations of the low-fidelity output emulator. The predictive uncertainty of the final surrogate model is then quantified by a complete characterization of the uncertainties of the different models and their interaction. The GPBNN is compared to most of the multi-fidelity regression methods allowing one to quantify the prediction uncertainty.

KEY WORDS: *multi-fidelity, surrogate modeling, Bayesian neural network, Gaussian process regression*

1. INTRODUCTION

We consider the situation in which two levels of code that simulate the same system have different costs and accuracies. This framework is called hierarchical multi-fidelity. We want to build a surrogate model of the most accurate and most costly code level, also called the high-fidelity code. The underlying motivation is to carry out an uncertainty propagation study or a sensitivity analysis that require many calls; therefore, the substitution of the high-fidelity code by a surrogate model with quantified prediction uncertainty is necessary. To build the surrogate, a small number N_H of high-fidelity code outputs and a large number N_L of low-fidelity code outputs are given. In some applications we may have $N_L \gg N_H$, but this paper will focus on $N_L > N_H$ and the context of small data. Then, the low-fidelity surrogate model uncertainty must be taken into account.

A well-known method to build a surrogate model with uncertainty quantification is Gaussian process (GP) regression, GP 1F in this paper. This method has become popular in computer experiments [1,2] and now allows scaling up in the number of learning points [3]. The emergence of multi-fidelity codes (codes that can be run at different levels of accuracy and cost) has motivated the introduction of new GP regression approaches. The first one was the Gaussian process auto-regressive or AR(1) scheme proposed by [4]. The form of the AR(1) model expresses a simple and linear relationship between the codes and it follows from a Markov property [5]. This method is used in [6] for optimization. This approach has been improved by [7] with the decoupling of the recursive estimation of the hyper-parameters of the different code levels. In the following, the recursive AR(1) model is called the AR(1) model because the results do not change and only the computation time is reduced. The Deep GP method introduced in [8] makes it possible to adapt the approach to cases in which the relationships between the code levels are nonlinear. An improvement has been further made by adding to the covariance function of the high-fidelity GP proposed by [8] a linear kernel in [9]. Multi-fidelity GP regression has been used in several fields, as illustrated in [10,11]. Multi-fidelity polynomial

chaos expension (MF-PCE) can also be exploited as in [12] for multi-fidelity regression. MF-PCE is mainly used for sensitivity analysis (see [13]), but shows its limitations for surrogate modeling in terms of uncertainty quantification.

Recent improvements in the implementation of neural networks have motivated research on multi-fidelity neural networks [14]. In [15], Li et al. combined a fully connected neural network (NN) and a linear system for the interactions between codes. The low-fidelity surrogate model was built using a fully connected NN, see [16] for a direct application. In order to quantify the prediction deviations and to evaluate the reliability of the prediction, the NNs have been improved to become Bayesian neural networks (BNNs) [17]. The multi-fidelity model has been improved by using a BNN for high-fidelity modeling in [18].

In this paper, this method will be called Meng, Babaee, and Karniadakis (MBK). The method in [18] offers options for single-fidelity active learning and is more general for multi-fidelity modeling. The disadvantages of methods using NNs are the nonprediction of the model uncertainties and the difficult optimization of the hyperparameters in a small data context. These disadvantages can be overcome by the use of BNNs. The ability of a BNN for uncertainty quantification is explained in [19].

The purpose of this paper is to present a method competing in terms of prediction with the methods of multi-fidelity regression, AR(1) model, Deep GP, and MBK. We also aim to improve the quantification of uncertainties in the nonlinear case for all these methods. Because not all the proposed methods give Gaussian processes as outputs, the uncertainties have to be quantified in an appropriate way through prediction intervals. An approach combining two methods, GP regression and a BNN, is proposed in this paper. We use a Gauss-Hermite quadrature-based method to transfer the low-fidelity GP, a posteriori law to the high-fidelity BNN. We compared the properties and results of our strategy with the ones of these methods. For evaluation, we examine our surrogate model approach with two-level multi-fidelity benchmark functions and a simulation example. The results demonstrate that the method presented in this paper is easier to train and more accurate than any other multi-fidelity neural network-based method. Moreover, the method is more flexible compared to other GP-based methods, and it gives reliable estimations of the predictive uncertainties.

In this paper, we propose to split the multi-fidelity surrogate modeling problem into two regression problems. The first problem is the single-fidelity regression for the low-fidelity code. The second problem is the regression for the high-fidelity code knowing the predictions and the predictive uncertainties of the low-fidelity surrogate model. GP regression allows prediction with quantified uncertainty for the low-fidelity code, which is important to minimize the predictive error and to quantify the predictive uncertainty of the surrogate model of the high-fidelity code. As in [18], we want to use a BNN for the regression knowing the low-fidelity prediction. The contribution we propose is an original strategy to transfer the low-fidelity predictive uncertainties to the BNN. For that we take well-chosen realizations of the predictor of the low-fidelity code by a quasi-Monte Carlo method based on Gauss-Hermite quadrature nodes, and we give them as inputs of the BNN in addition to the high-fidelity code inputs. A predictor is obtained by a weighted average of the BNN outputs corresponding to the different realizations. The predictive variance can also be assessed with the same sample.

The paper is organized as follows. Section 2 presents different methods to build single-fidelity surrogate models. The complete multi-fidelity method is presented in Section 3. The specific interaction between GP regression and a BNN is explained in Section 4, and Section 5 shows numerical results. Based on these results, the interest of the method is discussed in Section 6.

2. BACKGROUND: REGRESSION WITH UNCERTAINTY QUANTIFICATION

In this section, classical surrogate modeling methods with uncertainty quantification are presented. The GP regression method is presented in Section 2.1. The BNN method is presented in Section 2.2. Here, we want to predict the scalar output $y = f(\boldsymbol{x})$, with $y \in \mathbb{R}$, of a computer code with input $\boldsymbol{x} \in \mathbb{R}^d$ from the data set $(\boldsymbol{x}_i, y_i)_{i=1}^N$ with $y_i = f(\boldsymbol{x}_i)$.

2.1 Gaussian Process Regression

GP regression can be used to emulate a computer code with uncertainty quantification [1]. The prior output model as a function of the input $\boldsymbol{x}$ is a Gaussian process $Y(\boldsymbol{x})$ with mean $\mu(\boldsymbol{x})$ and stationary covariance function $C(\boldsymbol{x}, \boldsymbol{x}')$.

Consequently, the posterior distribution of the output $Y(\boldsymbol{x})$ given $Y(\boldsymbol{x}_1) = y_1, \ldots, Y(\boldsymbol{x}_N) = y_N$ is Gaussian with mean, as follows:

$$\mu_\star(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \boldsymbol{r}(\boldsymbol{x})^T \mathbf{C}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}) \tag{1}$$

and covariance

$$C_\star(\boldsymbol{x}, \boldsymbol{x'}) = C(\boldsymbol{x}, \boldsymbol{x'}) - \boldsymbol{r}(\boldsymbol{x})^T \mathbf{C}^{-1} \boldsymbol{r}(\boldsymbol{x'}) \tag{2}$$

with the vector $\boldsymbol{r}(\boldsymbol{x}) = (C(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, C(\boldsymbol{x}, \boldsymbol{x}_N))^T$, the matrix $\mathbf{C}$ defined by $C_{i,j} = C(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the vector $\boldsymbol{\mu} = (\mu(\boldsymbol{x}_1), \ldots, \mu(\boldsymbol{x}_N))^T$, and the vector $\boldsymbol{y} = (y_1, \ldots, y_N)^T$. The covariance function is chosen within a parametric family of kernels, whose parameters are fitted by maximizing the log marginal likelihood of the data (see [1], Chapter 2.2). For practical applications, the implementation of Algorithm 2.1 in [1] can be used.

2.2 Bayesian Neural Network

Neural networks have been used to emulate unknown functions based on data [20], and in particular, computer codes [21]. Our goal, however, is also to quantify the uncertainty of the emulation. A BNN makes it possible to quantify predictive uncertainties. Here, we present the BNN structure and the priors for the parameters.

A BNN with one hidden layer. Let N_l be the number of neurons in the hidden layer is presented. The output of the first layer is

$$\boldsymbol{y}_1 = \Phi(\boldsymbol{w}_1 \boldsymbol{x} + \boldsymbol{b}_1) \tag{3}$$

where $\boldsymbol{x} \in I\!R^d$ is the input vector of the BNN, $\boldsymbol{b}_1 \in I\!R^{N_l}$ is the bias vector, $\boldsymbol{w}_1 \in I\!R^{N_l \times d}$ is the weight matrix, and $\boldsymbol{y}_1 \in I\!R^{N_l}$ is the output of the hidden layer. The function $\Phi : I\!R^{N_l} \to I\!R^{N_l}$ is of the form $\Phi(\boldsymbol{b}) = (\phi(b_j))_{j=1}^{N_l}$, where the activation function ϕ can be a hyperbolic tangent or ReLU, for instance. The second (and last) layer is fully linear, as follows:

$$\mathrm{BNN}(\boldsymbol{x}) = \boldsymbol{w}_2^T \boldsymbol{y}_1 + b_2 \tag{4}$$

where $\boldsymbol{w}_2 \in I\!R^{N_n}$ is the weight matrix, $b_2 \in I\!R$ is the bias vector, and $\mathrm{BNN}(\boldsymbol{x}) \in I\!R$ is the scalar output of the BNN at point $\boldsymbol{x}$.

We use a Bayesian framework similar to the one presented in [22]. Let θ denote the parameter vector of the BNN, which is $\theta = (\boldsymbol{w}_i, \boldsymbol{b}_i)_{i=1,2}$. The probability distribution function (pdf) of the output given $\boldsymbol{x}$ and θ is

$$p(y|\boldsymbol{x}, \theta, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - \mathrm{BNN}_\theta(\boldsymbol{x}))^2}{2\sigma^2} \right) \tag{5}$$

where σ^2 is the variance of the random noise added to account for the fact that the neural network is an approximation. $\mathrm{BNN}_\theta(\boldsymbol{x})$ is the output of the neural network with parameter θ at point $\boldsymbol{x}$.

Here, we choose a prior distribution for (θ, σ) that is classic in the field of BNNs ([22], Part 5). The prior laws of the parameters $(\boldsymbol{w}_i, \boldsymbol{b}_i)_{i=1,2}$ are as follows:

$$\boldsymbol{w}_i \sim \mathcal{N}\big(0, \sigma_{w_i}^2 \boldsymbol{I}\big), \quad \boldsymbol{b}_i \sim \mathcal{N}\big(0, \sigma_{b_i}^2 \boldsymbol{I}\big), \quad i = 1, 2 \tag{6}$$

where $\sigma_{w,b_{1,2}}$ are the prior standard deviations. The prior for σ is the standard Gaussian $\mathcal{N}(0, 1)$ (assuming the function f has been normalized to be of the order 1). All parameters are assumed to be independent.

Applying Bayes' theorem, the posterior pdf of (θ, σ) given the data $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^N$ is

$$p(\theta, \sigma|\mathcal{D}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \theta, \sigma) p(\theta, \sigma) \tag{7}$$

up to a multiplicative constant, where $p(\theta, \sigma)$ is the prior distribution of θ, σ described earlier. The posterior distribution of the output at $\boldsymbol{x}$ has pdf, as follows:

$$p(y|\boldsymbol{x}, \mathcal{D}) = \iint p(y|\boldsymbol{x}, \theta, \sigma) p(\theta, \sigma|\mathcal{D}) d\theta d\sigma \tag{8}$$

and the two first moments are

$$\mathbb{E}_{\text{post}}[Y] = \iint \text{BNN}_{\theta}(\boldsymbol{x}) p(\theta, \sigma|\mathcal{D}) d\theta d\sigma \tag{9}$$

$$\mathbb{E}_{\text{post}}\left[Y^2\right] = \iint \left(\text{BNN}_{\theta}(\boldsymbol{x})^2 + \sigma^2\right) p(\theta, \sigma|\mathcal{D}) d\theta d\sigma \tag{10}$$

Contrarily to GP regression, the prediction of a BNN cannot be expressed analytically as shown by Eq. (8), but there exist efficient sampling methods. To sample the posterior distribution of the BNN output, we need to sample the posterior distribution of (θ, σ). In this paper the No-U-Turn Sampler (NUTS), which is a Hamiltonian Monte Carlo (HMC) method, is used to sample the posterior distribution of (θ, σ) [23]. By Eqs. (9) and (10), the estimated mean $\tilde{f}$ and variance $\tilde{V}$ of the output at point $\boldsymbol{x}$ are

$$\tilde{f}(\boldsymbol{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} \text{BNN}_{\theta_i}(\boldsymbol{x}) \tag{11}$$

$$\tilde{V}(\boldsymbol{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} \left[\text{BNN}_{\theta_i}(\boldsymbol{x}) - \tilde{f}(\boldsymbol{x})\right]^2 + \frac{1}{N_v} \sum_{i=1}^{N_v} \sigma_i^2 \tag{12}$$

where the $(\theta_i, \sigma_i)_{i=1}^{N_v}$ is the HMC sample of (θ, σ) with its posterior distribution.

Here we study the sampling algorithm for the posterior law. The Markov Chain Monte Carlo method aims at generating the terms of an ergodic Markov chain $(X_n)_{n\in\mathbb{N}}$ whose invariant measure is the target law with density $p(\boldsymbol{x})$, which is known up to a multiplicative constant. This Markov chain is specially constructed for this purpose, in the sense that its transition kernel is defined such that p is its unique invariant probability. There are several possible variations of this principle, such as the Metropolis-Hastings algorithm (see [24], Section 6.3.2 and [25]).

The Metropolis-Hastings algorithm is as follows. We give ourselves a starting point $\boldsymbol{x}_0$ and an exploration law, i.e., a family $(q(\boldsymbol{x}', \boldsymbol{x}))_{\boldsymbol{x}\in\mathbb{R}^d}$ of probability densities on $\mathbb{R}^d$ parametrized by $\boldsymbol{x}' \in \mathbb{R}^d$ that are easily simulated. We assume that we have simulated the nth term of the chain X_n.

1. We make a proposition X'_{n+1} drawn according to the density law $q(X_n, \cdot)$.
2. We calculate the acceptance rate $a(X_n, X'_{n+1}) = \min(1, \rho(X_n, X'_{n+1}))$ with

$$\rho(\boldsymbol{x}_n, \boldsymbol{x}'_{n+1}) = \frac{p(\boldsymbol{x}'_{n+1}) q(\boldsymbol{x}'_{n+1}, \boldsymbol{x}_n)}{p(\boldsymbol{x}_n) q(\boldsymbol{x}_n, \boldsymbol{x}'_{n+1})} \tag{13}$$

3. We draw $U_{n+1} \sim \mathcal{U}(0, 1)$.
4. We put

$$X_{n+1} = \begin{cases} X'_{n+1} & \text{if } U_{n+1} \leq a(X_n, X'_{n+1}) \\ X_n & \text{if } U_{n+1} > a(X_n, X'_{n+1}) \end{cases} \tag{14}$$

Note that we only need to know p up to a multiplicative constant to be able to implement the algorithm according to Eq. (13). In the case where q is symmetrical $q(\boldsymbol{x}', \boldsymbol{x}) = q(\boldsymbol{x}, \boldsymbol{x}')$, the acceptance rate is simply $\min[1, p(X'_{n+1}) / p(X_n)]$. We have a symmetrical q, in particular when the exploration law is Gaussian; $(q(\boldsymbol{x}', \boldsymbol{x}))_{\boldsymbol{x}\in R^d}$ is the density of the law $\mathcal{N}(\boldsymbol{x}', \sigma^2 I)$ with σ^2 to be calibrated in order to have an acceptance rate that is neither too high (which means that we do not explore enough) nor too low (which means that we reject the proposition too often because it is too far). Usually, we calibrate σ^2 during the simulation to observe a constant acceptance rate of the order of 1/4, for more details see [26].

The HMC algorithm is a special instance of the Metropolis-Hastings algorithm, because the exploration law $q(\boldsymbol{x}', \boldsymbol{x})$ is determined by a Hamiltonian dynamic, in which the potential energy depends on the target density p. This model is proposed in [27]. One algorithm that is efficient for HMC on the NUTS, which is used in this paper to sample the posterior distribution of (θ, σ), see [23].

3. COMBINING GP REGRESSION AND BNN

From now on, we consider a multi-fidelity framework with two levels of code, high f_H and low f_L fidelity, as in [4]. The input is $\boldsymbol{x} \in I\!R^d$ and the outputs of both computer code levels $f_L(\boldsymbol{x})$ and $f_H(\boldsymbol{x})$ are scalar. We have access to N_L low-fidelity points and N_H high-fidelity points, with $N_H < N_L$. In this paper, we focus on the small data framework where the low-fidelity code is not perfectly known. Under such circumstances it remains uncertainty in the low-fidelity surrogate model. If $N_H \ll N_L$, the situation would be different and we could assume that the low-fidelity emulator is perfect.

We therefore have two surrogate modeling tools: GP regression and the BNN. To do multi-fidelity with nonlinear interactions, the standard methods use combinations of regression methods. With our two methods we can make four combinations: GP-GP also called Deep GP in [8,9], the GP-BNN method proposed in our paper, BNN-GP, and BNN-BNN. The Deep GP model will be compared to the proposed method in all examples of the paper. The BNN-BNN method would be extremely expensive and very close to the full NN methods by adding the predictive uncertainty. The logic behind our choice of GP-BNN over BNN-GP is as follows: if we assume that the low-fidelity code is simpler than the high-fidelity code, then it must be approximated by a simpler model. GP regression is a surrogate model easier to obtain, and it gives a Gaussian output distribution that can be sampled easily. Whereas BNN is more complex to construct and allows for more general output distributions to be emulated.

The code output to estimate is f_H with the help of low- and high-fidelity points. As the low-fidelity code f_L is not completely known, a regression method with uncertainty quantification, GP regression, is used to emulate it, as in Section 2.1. The output of the low-fidelity GP is then integrated into the input to a BNN, described in Section 2.2, to predict f_H.

The low-fidelity surrogate model is a GP $Y_L(\boldsymbol{x})$ built from N_L low-fidelity data points $(\boldsymbol{x}_{L,i}, f_L(\boldsymbol{x}_{L,i})) \in I\!R^d \times I\!R$. The optimization of the hyper-parameters of the GP is carried out in the construction of the surrogate model. The GP is characterized by a predictive mean $\mu_L(\boldsymbol{x})$ and a predictive covariance $C_L(\boldsymbol{x}, \boldsymbol{x}')$.

To connect the GP with the BNN, the simplest way is to concatenate $\boldsymbol{x}$ and $\mu_L(\boldsymbol{x})$ (the best low-fidelity predictor) as input to the BNN. However, this does not take into account the predictive uncertainty. Consequently, we may want to add $C_L(\boldsymbol{x}, \boldsymbol{x})$ or $\sqrt{C_L(\boldsymbol{x}, \boldsymbol{x})}$ to the input vector of the BNN. The idea is that the BNN could learn from the low-fidelity GP, more than from its predictive mean only, in order to give reliable predictions of the high-fidelity code with quantified uncertainties. We show that the idea is fruitful and can be pushed even further.

We investigated three methods to combine the two surrogate models and transfer the posterior distribution of the low-fidelity emulator to the high-fidelity one. We demonstrate in Section 5 that the best solution is the so-called Gauss-Hermite method.

4. TRANSFER METHODS

The two learning sets are $\mathcal{D}^L = \{(\boldsymbol{x}_i^L, f_L(\boldsymbol{x}_i^L)), i = 1, \ldots, N_L\}$ and $\mathcal{D}^H = \{(\boldsymbol{x}_i^H, f_H(\boldsymbol{x}_i^H)), i = 1, \ldots, N_H\}$ with typically $N_H < N_L$, and we do not need to assume that the sets $\{x_i^L, i = 1, \ldots, N_L\}$ and $\{x_i^H, i = 1, \ldots, N_H\}$ are nested. The low-fidelity model is emulated using GP regression, as a consequence the result is formulated as a posterior distribution given $\mathcal{D}^L$ that has the form of a Gaussian law.

Proposition 1. *The posterior distribution of $Y_L(\boldsymbol{x})$ knowing $\mathcal{D}^L$ is the Gaussian distribution with mean $\mu_L(\boldsymbol{x})$ and variance $\sigma_L^2(\boldsymbol{x})$ of the form (1-2). We denote its pdf by $p(y_L|\mathcal{D}_L, \boldsymbol{x})$.*

Proof. The proof is given in [1] (see Chapter 2.2) (prediction with noise free observations). □

The posterior distribution of the high-fidelity code given the low-fidelity learning set $\mathcal{D}^L$ and the high-fidelity learning set $\mathcal{D}^H$ may have different forms depending on the input of the BNN.

4.1 Mean-Standard Deviation Method and Quantiles Method

In the Mean-Standard deviation method, called the Mean-Std method, we give as input to the BNN the point $\boldsymbol{x}$ and the information usually available at the output of a GP regression, i.e., the predictive mean and standard deviation of the low-fidelity emulator at the point $\boldsymbol{x}$.

In this method, the input of the BNN, whose output gives the prediction of the high-fidelity code at $\boldsymbol{x}$, is $\boldsymbol{x}^{\text{BNN}} = (\boldsymbol{x}, \mu_L, \sigma_L)$. The idea is that the BNN input consists of the input $\boldsymbol{x}$ of the code and of the mean and standard deviation of the posterior distribution of the low-fidelity emulator at $\boldsymbol{x}$. The high-fidelity emulator is modeled as follows:

$$Y_H(\boldsymbol{x}) = \text{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})) + \sigma\epsilon \tag{15}$$

with $\epsilon \sim \mathcal{N}(0,1)$. We use the learning set $\mathcal{D}^H_{MS} = \left\{\left(\boldsymbol{x}_i^H, \mu_L(\boldsymbol{x}_i^H), \sigma_L(\boldsymbol{x}_i^H), f_H(\boldsymbol{x}_i^H)\right), i = 1, \ldots, N_H\right\}$ to train the BNN and get the posterior distribution of (θ, σ). Note that $\mathcal{D}^H_{MS}$ can be deduced from $\mathcal{D}^L$ and $\mathcal{D}^H$.

Proposition 2. *The posterior distribution of $Y_H(\boldsymbol{x})$ knowing $\mathcal{D}^L$ and $\mathcal{D}^H_{MS}$ has pdf*

$$p\big(y_H|\boldsymbol{x}, \mathcal{D}^H_{MS}, \mathcal{D}^L\big) = \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y_H - \textit{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})))^2}{2\sigma^2}\right] p(\theta, \sigma|\mathcal{D}^H_{MS}) d\sigma d\theta \tag{16}$$

with $p(\theta, \sigma|\mathcal{D}^H_{MS})$ the posterior pdf of the hyper-parameters of the BNN.

Proof. Let $p(y_H|\theta, \sigma, \boldsymbol{x}, \mathcal{D}^L)$ be the probability density function (pdf) of $Y_H(\boldsymbol{x})$ given by Eq. (15). This pdf can be written as follows:

$$p(y_H|\theta, \sigma, \boldsymbol{x}, \mathcal{D}^L) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y_H - \text{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})))^2}{2\sigma^2}\right]$$

The law of total probability gives that:

$$\begin{aligned} p\big(y_H|\boldsymbol{x}, \mathcal{D}^H_{MS}, \mathcal{D}^L\big) &= \iint p(y_H|\theta, \sigma, \boldsymbol{x}, \mathcal{D}^L) p(\theta, \sigma|\mathcal{D}^H_{MS}) d\sigma d\theta \\ &= \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y_H - \text{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})))^2}{2\sigma^2}\right] p(\theta, \sigma|\mathcal{D}^H_{MS}) d\sigma d\theta \end{aligned}$$

□

Corollary 1.
The mean and variance of the posterior distribution of $Y_H(\boldsymbol{x})$ knowing $\mathcal{D}^H_{MS}$ and $\mathcal{D}^L$ is as follows:

$$\mu_H(\boldsymbol{x}) = \iint \text{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})) p(\theta, \sigma|\mathcal{D}^H_{MS}) d\sigma d\theta \tag{17}$$

and

$$C_H(\boldsymbol{x}) = \iint \big(\text{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x}))^2 + \sigma^2\big) p(\theta, \sigma|\mathcal{D}^H_{MS}) d\sigma d\theta \tag{18}$$

Proof. By the definition of the mean and variance, we get

$$\begin{aligned} \mu_H(\boldsymbol{x}) &= \int y_H p\big(y_H|\boldsymbol{x}, \mathcal{D}^H_{MS}, \mathcal{D}^L\big) dy_H \\ C_H(\boldsymbol{x}) &= \int (y_H - \mu_H(\boldsymbol{x}))^2 p\big(y_H|\boldsymbol{x}, \mathcal{D}^H_{MS}, \mathcal{D}^L\big) dy_H \end{aligned}$$

We replace by the expressions given in Proposition 2.

$$\mu_H(\boldsymbol{x}) = \int y_H \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y_H - \text{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})))^2}{2\sigma^2}\right] p(\theta, \sigma|\mathcal{D}^H_{MS}) d\sigma d\theta dy_H$$

which gives Eq. (17). For the variance,

$$C_H(\boldsymbol{x}) = \int (y_H - \mu_H(\boldsymbol{x}))^2 \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y_H - \text{BNN}_\theta(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})))^2}{2\sigma^2}\right] p(\theta, \sigma|\mathcal{D}^H_{MS}) d\sigma d\theta dy_H$$

which gives Eq. (18). □

Corollary 2.
Given $\mathcal{D}^L$ and $\mathcal{D}^H_{MS}$, given a sample $(\theta_i, \sigma_i)_{i=1}^{N_v}$ of the posterior distribution of (θ, σ)

$$\tilde{\mu}_H(\boldsymbol{x}) = \frac{1}{N_v}\sum_{i=1}^{N_v} \text{BNN}_{\theta_i}(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x})) \tag{19}$$

and

$$\tilde{C}_H(\boldsymbol{x}) = \frac{1}{N_v}\sum_{i=1}^{N_v} \big(\text{BNN}_{\theta_i}(\boldsymbol{x}, \mu_L(\boldsymbol{x}), \sigma_L(\boldsymbol{x}))^2 + \sigma_i^2\big) - \tilde{\mu}_H(\boldsymbol{x})^2 \tag{20}$$

are consistent estimators of $\mu_H(\boldsymbol{x})$ and $C_H(\boldsymbol{x})$.

The estimators need samples of the posterior distribution of (θ, σ). By the HMC method (NUTS), we get a sample of the hyperparameters (θ_j, σ_j) with the posterior distribution of the high-fidelity model. It is possible to construct a similar method, called the quantiles method, which takes as input the quantiles instead of the standard deviation. The quantiles method consists of giving the mean and two quantiles of the low-fidelity GP emulator as input to the BNN. Assuming we want to have the high-fidelity output uncertainty at level α%, we take the $\alpha/2$% and the $(1-\alpha/2)$% quantiles. The expression of the BNN input vector is $\boldsymbol{x}^{\text{BNN}} = (\boldsymbol{x}, \mu_L, Q_{L,\alpha/2}, Q_{L,(1-\alpha/2)})$ and the high-fidelity learning set is

$$\mathcal{D}^H_Q : \big\{\big((\boldsymbol{x}_i^H, \mu_L(\boldsymbol{x}_i^L), Q_{L,\alpha/2}(\boldsymbol{x}_i^H), Q_{L,(1-\alpha/2)}(\boldsymbol{x}_i^H)), f_H(\boldsymbol{x}_i^H)\big), i = 1, \cdots, N_H\big\}$$

This method is very similar to the Mean-Std method and only the BNN input changes. The estimators of the mean and variance have the same form.

4.2 Gauss-Hermite Quadrature

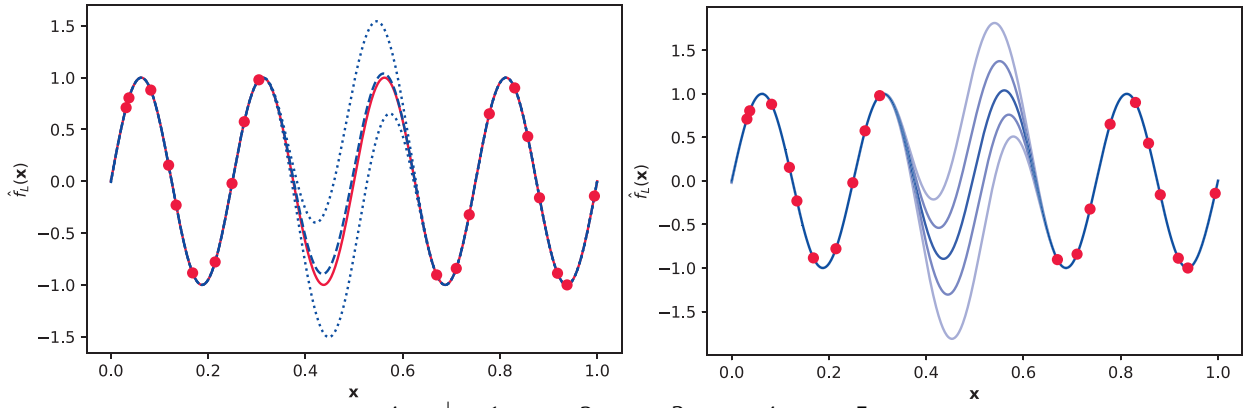
In this section, we transfer the information about the posterior distribution of the low-fidelity emulator by a sampling method. The GP posterior distribution is one-dimensional and Gaussian for each value of $\boldsymbol{x}$. Therefore, a deterministic method for sampling is preferable in order to limit the number of calls to the BNN. In the following, this method is called the GPBNN. We propose to sample the Gaussian distribution by a quasi Monte Carlo method using S Gauss-Hermite quadrature nodes ([28], Chapter 3). This method has been chosen because it gives the best interpolation of a Gaussian distribution. The samples $\tilde{f}_{L,j}(\boldsymbol{x})$, with $j = 1, \ldots, S$, of the GP posterior distribution are constructed using the roots $z_{S,j}$ of the physicists' version of the Hermite polynomials $H_S(x) = (-1)^S e^{x^2}\partial_x^S e^{-x^2}$, $S \in \mathbb{N}$. For each input $\boldsymbol{x}$ the GP posterior law has mean $\mu_L(\boldsymbol{x})$ and variance $C_L(\boldsymbol{x}, \boldsymbol{x})$. We sample S times the low-fidelity GP posterior law by using the Gauss-Hermite quadrature illustrated at Fig. 1. Therefore, the jth realization in the Gauss-Hermite quadrature formula is

$$\tilde{f}_{L,j}(\boldsymbol{x}) = \mu_L(\boldsymbol{x}) + z_{S,j}\sqrt{2}\sqrt{C_L(\boldsymbol{x}, \boldsymbol{x})} \tag{21}$$

and the associated weight is $p_{S,j} = 2^{S-1}S!/S^2 H_{S-1}^2(z_{S,j})$, for $j = 1, \ldots, S$. The learning set of the BNN is $\mathcal{D}^H_{GH} : \big[\big((\boldsymbol{x}_i^H, \mu_L(\boldsymbol{x}_i^H), \sigma_L(\boldsymbol{x}_i^H)), f_H(\boldsymbol{x}_i^H)\big), i = 1, \cdots, N_H\big]$. The GP covariance can be viewed in Fig. 1 as a difference between the S realizations of the GP posterior law. The low-fidelity uncertainty prediction is taken into account in BNN input parameters. Thus, the high-fidelity emulator is modeled as follows:

$$Y_H(\boldsymbol{x}) = \sum_{j=1}^{S} p_{S,j}\text{BNN}_{\theta}(\boldsymbol{x}, \tilde{f}_{L,j}(\boldsymbol{x})) + \sigma\epsilon \tag{22}$$

with $\epsilon \sim \mathcal{N}(0,1)$ and $\tilde{f}_{L,j}(\boldsymbol{x})$ is given by Eq. (21).



j	1	2	3	4	5
$z_{S=5,j}$	–2.02	–0.96	0	0.96	2.02
$p_{S=5},j$	0.020	0.394	0.945	0.394	0.020

FIG. 1: Illustration of a Gauss-Hermite quadrature for a GP. The left curve shows the posterior law of the GP. In solid line, we have the low-fidelity function to approach which we observe the points. The dashed line shows the mean of the GP and the dotted line shows the $\mathcal{I}_{95\%}$ the prediction interval. The right curve shows the GP with the realisations of the Gauss-Hermite quadrature for $S = 5$. The intensities of the lines are linear with the weight of the quadrature. The table gives the approximated values of the roots and weights for the Gauss-Hermite quadrature $S = 5$.

Proposition 3. *The posterior distribution of $Y_H(\boldsymbol{x})$ knowing $\mathcal{D}^H_{GH}$ and $\mathcal{D}^L$ is*

$$p\big(y_H|\boldsymbol{x},\mathcal{D}^H_{GH},\mathcal{D}^L\big) = \iint \frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{1}{2\sigma^2}\left(y_H - \sum_{j=1}^{S} p_{S,j} BNN_{\theta}(\boldsymbol{x},\tilde{f}_{L,j})\right)^2\right] p(\theta,\sigma|\mathcal{D}^H_{RS})d\sigma d\theta \tag{23}$$

with $p(\theta,\sigma|\mathcal{D}^H_{GH})$ the posterior distribution of the hyper-parameters of the BNN. $\tilde{f}_{L,j}$ is given by Eq. (21).

Corollary 3.
The posterior mean of $Y_H(\boldsymbol{x})$ is

$$\mu_H(\boldsymbol{x}) = \iint \left(\sum_{j=1}^{S} p_{S,j}\mathrm{BNN}_{\theta}(\boldsymbol{x},\tilde{f}_{L,j})\right) p(\theta,\sigma|\mathcal{D}^H_{GH})d\sigma d\theta \tag{24}$$

The posterior variance of $Y_H(\boldsymbol{x})$ is

$$C_H(\boldsymbol{x}) = \iint \left[\left(\sum_{j=1}^{S} p_{S,j}\mathrm{BNN}_{\theta}(\boldsymbol{x},\tilde{f}_{L,j})\right)^2 + \sigma^2\right] p(\theta,\sigma|\mathcal{D}^H_{GH})d\sigma d\theta - \mu^2_H(\boldsymbol{x}) \tag{25}$$

The expression of $\tilde{f}_{L,j}$ is given by Eq. (21).

Corollary 4.
Given $\mathcal{D}^L$ and $\mathcal{D}^H_{GH}$, given a sample $(\theta_i,\sigma_i)_{i=1}^{N_v}$ of the posterior distribution of (θ,σ)

$$\tilde{\mu}_H(\boldsymbol{x}) = \frac{1}{N_v}\sum_{i=1}^{N_v}\sum_{j=1}^{S} p_{S,j}\mathrm{BNN}_{\theta_i}(\boldsymbol{x},\tilde{f}_{L,j}(\boldsymbol{x})) \tag{26}$$

and

$$\tilde{C}_H(\boldsymbol{x}) = \frac{1}{N_v}\sum_{i=1}^{N_v}\left(\sum_{j=1}^{S} p_{S,j}\mathrm{BNN}_{\boldsymbol{\theta}_i}(\boldsymbol{x}, \tilde{f}_{L,j}(\boldsymbol{x}))\right)^2 + \frac{1}{N_v}\sum_{i=1}^{N_v}\sigma_i^2 - \tilde{\mu}_H^2(\boldsymbol{x}) \tag{27}$$

are consistent estimators of the mean and variance in Corollary 3.

Note that this sampling method is different from the quantiles method (even for $S = 3$ and $\alpha \approx 0.110$). Indeed, in the quantiles method the input of the BNN contains the mean and quantiles of the low-fidelity code predictor; whereas, the Gauss-Hermite method is a sampling method in which the input of the BNN contains only one sample of the low-fidelity code predictor and the predictor is a weighted average of the BNN.

The sample $(\boldsymbol{\theta}_i, \sigma_i)$ of the posterior distribution of $(\boldsymbol{\theta}, \sigma)$ can be obtained by the HMC method (NUTS). The choice of S is a trade-off between a large value that is computationally costly and a small value that does not propagate the uncertainty appropriately. $S = 2$ is the smallest admissible value regarding the information that should be transferred. At first glance, a large value of S could be expected to be the best choice in the point of view of the predictive accuracy. However, a too large value of S degrades the accuracy of the predictive mean estimation. This is due to large variations of $\tilde{f}_{L,i}(\boldsymbol{x})$ for large values of S. Interesting values turn out to be between 3 and 10, depending on the quality of the low-fidelity emulator, as discussed in Section 5. Figure 2 is a graphical representation of the GPBNN method.

The value of N_v is chosen large enough so that the estimators in Eqs. (26) and (27) have converged. As shown in the analysis of Section 5, $N_v = 500$ is sufficient.

We could expect the computational cost of the GPBNN method to be expensive. The number of operations needed to compute an iteration of the HMC optimization is proportional to $S \times N_v \times N_H$. Because S and N_H are small in our context, the computational cost of one realization of the BNN is actually low. Thus, optimization of the hyperparameters is feasible for $N_H \lesssim 100$.

5. EXPERIMENTS

In this section, we present two analytic examples and a simulated one. The first one is a one-dimensional (1D) function, and we consider that the low-fidelity function may be unknown in a certain subdomain. The second one is a two-dimensional (2D) function with noise. Finally, we test the strategy on a more complex double pendulum system. All the numerical experiments are carried out on a laptop (of 2017, Dell precision with Intel Core i7) using only a CPU and the running time never exceeds 2 h.

5.1 1D Function Approximation

The low- and high-fidelity functions are

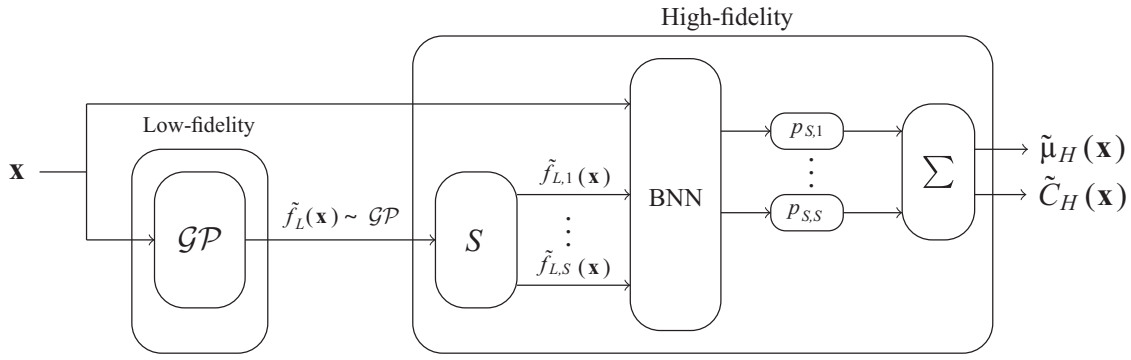


FIG. 2: Schematic of the multi-fidelity Gauss-Hermite model. The input is a point $\boldsymbol{x}$. The output consists of a predictive mean $\tilde{\mu}_H(\boldsymbol{x})$ and a predictive variance $\tilde{C}_H(\boldsymbol{x})$.

$$f_L(x) = \sin 8\pi x, \qquad f_H(x) = \left(x - \sqrt{2}\right) f_L^2(x) \tag{28}$$

with $x \in [0,1]$, where f_H is the high-fidelity function (code) and f_L the low-fidelity function (code). A graphical representation of these functions is given in Fig. 3. These functions have been introduced in [8] and are well estimated with a Deep GP and a quadratic form of the covariance. In this example, we assume that we have access to a lot of low-fidelity data, $N_L = 100$, while the high-fidelity data is small, $N_H = 20$. We also consider situations in which there is a segment $\bar{I} \subset [0,1]$, where we do not have access to $f_L(x)$. The learning set for the high-fidelity code is obtained by partitioning $[0,1]$ into N_H segments with equal lengths and then by choosing independently one point randomly on each segment with uniform distribution. The learning set for the low-fidelity code is obtained by partitioning $[0,1]\backslash\bar{I}$ into N_L segments with equal length and then by choosing independently one point randomly on each segment with uniform distribution. The test set is composed of $N_{\mathrm{T}} = 1000$ independent points following a random uniform law on $[0,1]$.

We denote by $\left(\boldsymbol{x}_{\mathrm{T}}^{(i)}, f_H(\boldsymbol{x}_{\mathrm{T}}^{(i)})\right)_{i=1,\ldots,N_{\mathrm{T}}}$ the test set. The error is evaluated by

$$Q_{\mathrm{T}}^2 = 1 - \frac{\sum_{i=1}^{N_{\mathrm{T}}} \left[\tilde{\mu}_H(\boldsymbol{x}_{\mathrm{T}}^{(i)}) - f_H(\boldsymbol{x}_{\mathrm{T}}^{(i)})\right]^2}{N_{\mathrm{T}} \mathbb{V}_{\mathrm{T}}(f_H)} \tag{29}$$

with $\mathbb{V}_{\mathrm{T}}(f_H) = (1/N_{\mathrm{T}}) \sum_{i=1}^{N_{\mathrm{T}}} \left[f_H\left(\boldsymbol{x}_{\mathrm{T}}^{(i)}\right) - (1/N_{\mathrm{T}}) \sum_{j=1}^{N_{\mathrm{T}}} f_H\left(\boldsymbol{x}_{\mathrm{T}}^{(j)}\right)\right]^2$. A highly predictive model gives a Q_{T}^2 close to 1, while a less predictive model has a smaller Q_{T}^2. The coverage probability CP_α is defined as the probability for the actual value of the function to be within the prediction interval with confidence level α of the surrogate model, as follows:

$$\mathrm{CP}_\alpha = \frac{1}{N_{\mathrm{T}}} \sum_{i=1}^{N_{\mathrm{T}}} \mathbf{1}_{f_H\left(\boldsymbol{x}_{\mathrm{T}}^{(i)}\right) \in \mathcal{I}_\alpha\left(\boldsymbol{x}_{\mathrm{T}}^{(i)}\right)} \tag{30}$$

where $\mathbf{1}$ is the indicator function and $\mathcal{I}_\alpha(\boldsymbol{x})$ the prediction interval at point $\boldsymbol{x}$ with confidence level α. The mean predictive interval width MPIW_α is the average width of the prediction intervals:

$$\mathrm{MPIW}_\alpha = \frac{1}{N_{\mathrm{T}}} \sum_{i=1}^{N_{\mathrm{T}}} \left|\mathcal{I}_\alpha\left(\boldsymbol{x}_{\mathrm{T}}^{(i)}\right)\right| \tag{31}$$

where $\mathcal{I}_\alpha(\boldsymbol{x})$ is the prediction interval at point $\boldsymbol{x}$ with confidence level α and $|\mathcal{I}_\alpha(\boldsymbol{x})|$ is the length of the prediction interval $\mathcal{I}_\alpha(\boldsymbol{x})$. The prediction uncertainty of the surrogate model is well characterized when CP_α is close to α. The prediction uncertainty of the surrogate model is small when MPIW_α is small.

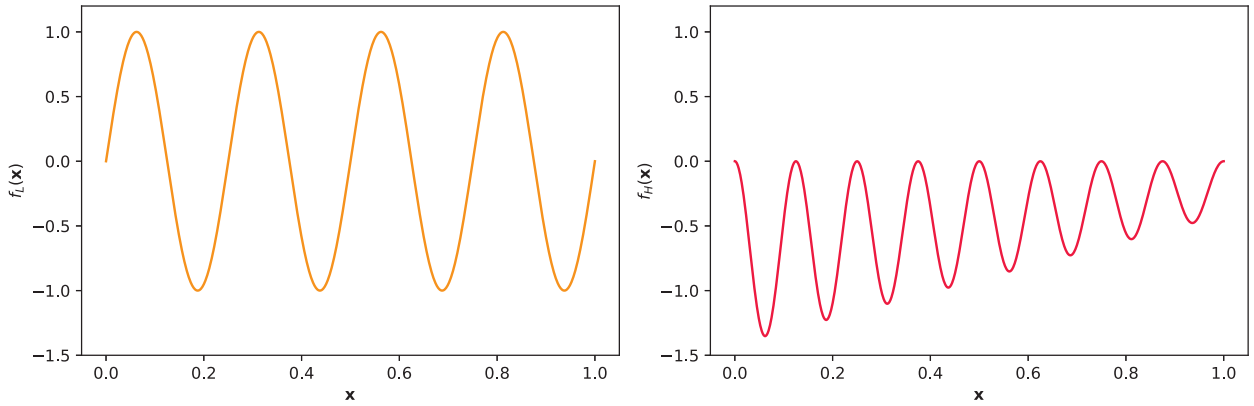


FIG. 3: Low-fidelity function on the left and high-fidelity function on the right. We can see that the range of variation is smaller for the high-fidelity function and its frequency is two times higher than low-fidelity function.

For the GPBNN method, we obtain the interval $\mathcal{I}_\alpha(\boldsymbol{x})$ by sampling N_v realizations of the posterior distribution of the random process $Y_H(\boldsymbol{x})$ described in Section 4. The interval $\mathcal{I}_\alpha(\boldsymbol{x})$ is the smallest interval that contains the fraction α of the realizations $(y_j)_{j=1}^{N_v}$ of $Y_H(\boldsymbol{x})$. For the GP 1F model and the AR(1) model, the prediction interval is centered at the predictive mean and its half-length is $q_{1-\alpha/2}$ times the predictive standard deviation, where $q_{1-\alpha/2}$ is the $1-\alpha/2$-quantile of the standard Gaussian law, because the posterior distributions are Gaussian. For the Deep GP model, the prediction interval is obtained by Monte Carlo sampling of the posterior distribution (with 1000 samples).

We use GP regression with zero mean function and tensorized Matérn 5/2 covariance function for the low-fidelity GP regression. The implementation we use is from [29]. The optimization for GP regression gives a correlation length of 0.108. For this example, we choose $N_n = 30$ neurons, and we use the ReLU function as activation function and the BNN implementation proposed in [30]. The sample size of the posterior distribution of the BNN parameter (θ, σ) is $N_v = 500$ (see Fig. 4).

Low-fidelity surrogate models of different accuracies are considered to understand how our strategy behaves under low-fidelity uncertainty. This is done by considering that low-fidelity data points are only accessible in $[0,1]\backslash\bar{I}$. We have thus chosen to study three cases: (i) a very good low-fidelity emulator with $\bar{I} = \emptyset$ (for which the $Q^2_{l\to l}$ of the low-fidelity emulator is 0.99, with $Q^2_{l\to l}$ the Q^2 of the low-fidelity model for low-fidelity prediction), (ii) a good emulator with $\bar{I} = [1/3, 2/3]$ ($Q^2_{l\to l} = 0.98$), and (iii) a poor emulator with $\bar{I} = [3/4, 1]$ ($Q^2_{l\to l} = 0.84$). The objective of these different sets is to have more or less accurate low-fidelity models to cover different learning configurations in order to test the model in different configurations.

Table 1 compares for these examples the different techniques, proposed in Section 4, for $\bar{I} = \emptyset$. All methods have the same efficiency in terms of Q^2_{T}. The uncertainties of the predictions are overestimated for all methods. However, the Gauss-Hermite method has the best CP_α and the best uncertainty interval (i.e., the smallest mean predictive interval width MPIW_α). The quantiles and mean-Std methods also have reasonable CP_α, but their uncertainty intervals are larger. This leads us to use the Gauss-Hermite method. However, we note that all methods overestimated the prediction interval. We believe this is due to the high regularity of the function to be predicted.

In Fig. 5, we report the performances of the GPBNN method as functions of S between 1 and 12 for different $\bar{I}$. For $S = 1$, the uncertainty is underestimated and the accuracy of the prediction is not optimal, which shows that it is important to exploit the uncertainty predicted by the low-fidelity model. For $2 \le S \le 5$, the prediction is good, the error is constant, and the Q^2_{T} is maximal, as seen in Fig. 5(a). Figure 5(b) shows that the coverage probability is acceptable for $2 \le S \le 7$. Finally, the $\mathrm{MPIW}_{80\%}$ on Fig. 5(c) is minimal for $3 \le S \le 7$. The best value of S is in [2, 5], depending on the accuracy of the low-fidelity emulator. Increasing S should increase the size of the training set.

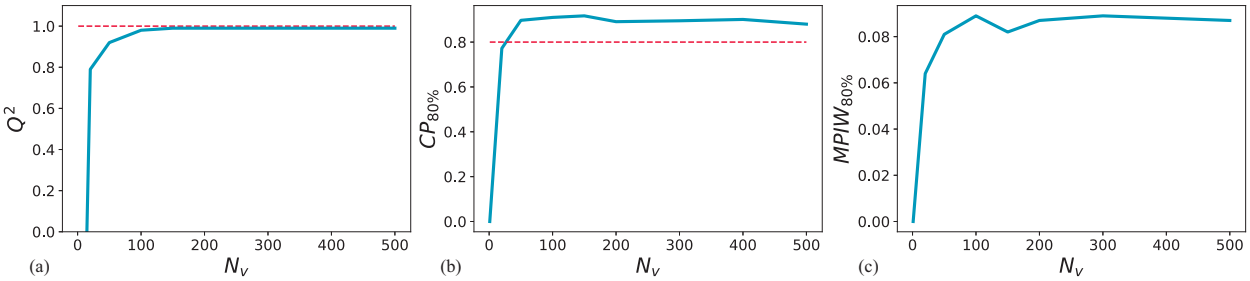


FIG. 4: Error Q^2_{T}, coverage probability at 80% and $\mathrm{MPIW}_{80\%}$ as functions of N_v for $\bar{I} = \emptyset$. (a) Q^2_T, (b) $\mathrm{CP}_{80\%}$, and (c) $\mathrm{MPIW}_{80\%}$.

TABLE 1: Error Q^2_T, coverage probability CP_α and mean predictive interval width MPIW_α for $\alpha = 80\%$ and for different methods of sampling. Here $\bar{I} = \emptyset$

	Q^2_T	CP_α	MPIW_α
Gauss-Hermite $S = 5$	0.99	0.88	0.083
Mean-Std	0.99	0.97	0.095
Quantiles	0.99	0.90	0.105

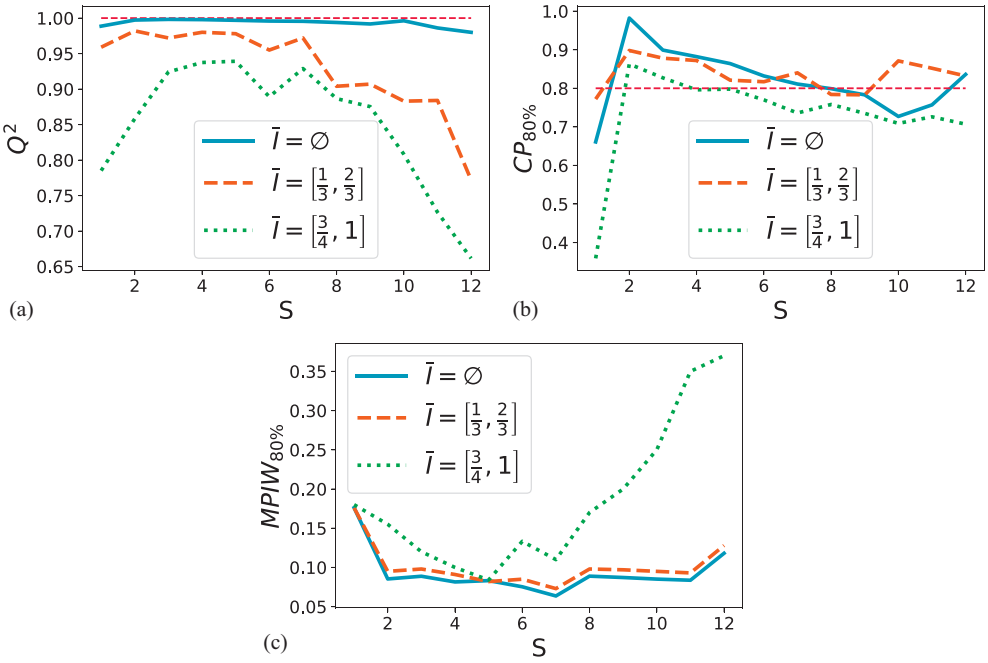


FIG. 5: Error Q^2_{T}, coverage probability at 80% and $\mathrm{MPIW}_{80\%}$ as functions of S

It is therefore expected that the larger S is, the more accurate the model will be. But for $S > 7$, we have realizations out of the 95% prediction interval. The linkage between internal BNN nonlinearities and realizations outside the 95% prediction interval tends to degrade the prediction of the high-fidelity model. In this case, we hypothesize that the BNN poorly learns the low probability values.

We carried out a study on the best value of N_v. We thought that the best value would be the largest possible. Therefore, we tested for values of N_v ranging from 1 to 1000 for $\bar{I} = \emptyset$. For values of $N_v > 200$, the performance is identical as a function of N_v. For values < 200, a greater variability was found. We choose to use $N_v = 500$ to have a sufficient margin. In Fig. 4, we averaged the estimators for five independent training sets.

We now compare our GPBNN method with other state-of-the-art methods. The single-fidelity GP method used to emulate the high-fidelity code from the N_H high-fidelity points is called GP 1F. We use the implementation in [29]. The multi-fidelity GP regression with autoregressive form introduced by [4] and improved by [7] is called autoregressive model AR(1). The method proposed in both [8,9] is called Deep GP, we use the implementation from [8] and the covariance given in [9], Eq. (11). The method from [18] is called the MBK method, which is the combination of a fully connected NN for low-fidelity regression and BNN for high fidelity. We implemented it using [30]. The methods that require the minimal assumptions on the functions f_L and f_H are the GPBNN and MBK methods. This is the reason why they are the two methods that are compared in Fig. 6.

First, the output laws for the MBK method and for the GPBNN, presented in Fig. 6, are different. For the MBK method, the posterior law is associated to the high-fidelity BNN knowing both the high-fidelity data and the low-fidelity model. Unlike the MBK method, the GBBNN method's output law represents the posterior law knowing high-fidelity points and the posterior distribution of the low-fidelity model built from the N_L low-fidelity points. In Fig. 6, $S = 5$ because, as previously discussed, this value seems to be the best value when the low-fidelity code is not so accurate.

The presented techniques for multi-fidelity regression are compared in Tables 2–4. The GP 1F model, and the multi-fidelity AR(1) model do not have good predictive properties. For the GP 1F model it is due to the lack of high-fidelity data, and for the AR(1) model, it is due to the strongly nonlinear relationship between the code levels. The three other methods perform almost perfectly when $\bar{I} = \emptyset$. The Deep GP is outstanding when $\bar{I} = \emptyset$, but the interaction between the codes (a quadratic form) is given exactly in the covariance structure, which is a strong

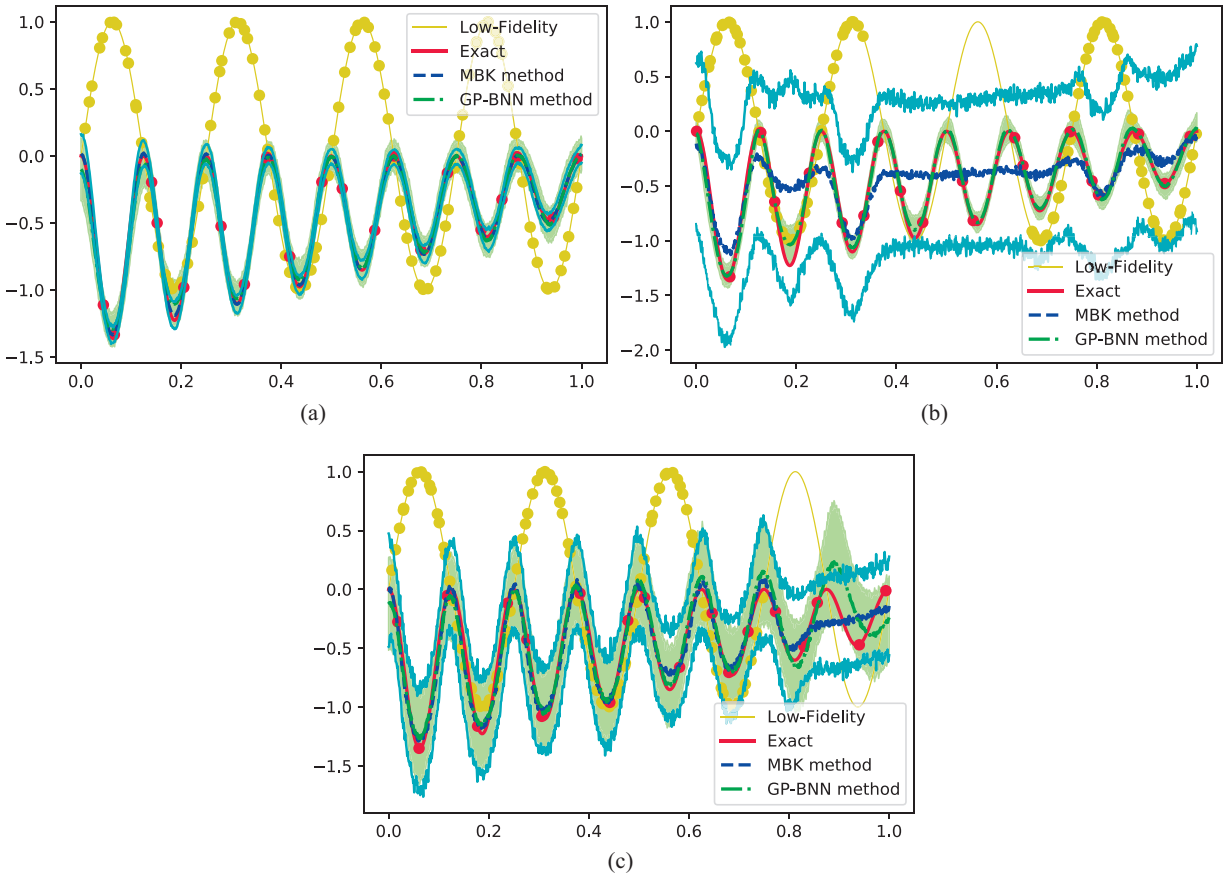


FIG. 6: Comparison between the MBK method (in blue dashed line) and the GPBNN method with $S = 5$ (in green dashed-dotted line) for the estimation of the function f_H Eq. (28) (in red solid line). The high-fidelity data points are in red. The light colored lines (blue and green) plot the predictive intervals. The uncertainty interval is given by the $\mathcal{I}_{80\%}(\boldsymbol{x})$. (a) $\bar{I} = \emptyset$, (b) $\bar{I} = [1/3, 2/3]$, and (c) $\bar{I} = [3/4, 1]$. See online version for color.

TABLE 2: Q^2_{T} for different methods and segments $\bar{I}$ of missing low-fidelity values. Here $S = 5$

$\bar{I}$	GP 1F	AR(1)	Deep GP	MBK	GPBNN
$\emptyset$	0.12	–0.29	0.99	0.99	0.99
$[1/3, 2/3]$	0.13	–0.34	0.98	0.90	0.98
$[3/4, 1]$	0.12	–0.29	0.90	0.51	0.93

TABLE 3: Coverage probability CP_α for $\alpha = 80\%$ and for different methods and segments $\bar{I}$ of missing low-fidelity values. Here $S = 5$

$\bar{I}$	GP 1F	AR(1)	Deep GP	MBK	GPBNN
$\emptyset$	0.82	0.82	0.99	0.76	0.88
$[1/3, 2/3]$	0.78	0.79	0.60	0.84	0.83
$[3/4, 1]$	0.78	0.82	0.62	0.86	0.78

TABLE 4: Mean predictive interval width MPIW_α for $\alpha = 80\%$ and for different methods and segments $\bar{I}$ of missing low-fidelity values. Here $S = 5$

$\bar{I}$	GP 1F	AR(1)	Deep GP	MBK	GPBNN
$\emptyset$	0.44	0.55	0.002	0.037	0.083
$[1/3, 2/3]$	0.42	0.45	0.010	0.36	0.082
$[3/4, 1]$	0.44	0.45	0.097	0.31	0.084

assumption in Deep GP that is hard to verify in practical applications. When $\bar{I}$ is nonempty, the MBK method gives less accurate predictions because the method assumes strong knowledge of the low-fidelity code level. However, its uncertainty interval seems realistic for this example although too large. The Deep GP has reasonable errors but Tables 3 and 4 show that the predictive uncertainty of the Deep GP method does not fit the actual uncertainty of the prediction (it has poor coverage probability, either too large or too small). The GPBNN method has the smallest error (best Q^2_{T}), and it is predicting its accuracy precisely (it has good and nominal coverage probability; here, $S = 5$) and the predictive variance and prediction interval width are small compared to the other methods: The GP 1F and AR(1) models have reasonable coverage probabilities but large mean predictive interval widths. For this simple illustrative example, the GPBNN method seems to be the most suitable method.

5.2 2D Function Approximation

The Currin function is a two-dimensional function, with $\boldsymbol{x} \in [0, 1]^2$. This function is commonly used to simulate computer experiments [9]. The high- and low-fidelity functions are

$$f_H(\boldsymbol{x}) = \left[1 - \exp\left(-\frac{1}{2x_2}\right)\right] \frac{2300x_1^3 + 1900x_1^2 + 2092x_1 + 60}{100x_1^3 + 500x_1^2 + 4x_1 + 20} \tag{32}$$

$$\begin{aligned} f_L(\boldsymbol{x}) = {} & \frac{1}{4}[f_H(x_1 + \delta, x_2 + \delta) + f_H(x_1 + \delta, \max(0, x_2 - \delta))] \\ & + \frac{1}{4}[f_H(x_1 - \delta, x_2 + \delta) + f_H(x_1 - \delta, \max(0, x_2 - \delta))] \end{aligned} \tag{33}$$

with $\boldsymbol{x} = [x_1, x_2]$ and δ the filter parameter. In [9], we have $\delta = 0.05$, and this gives very small differences between the two functions, and the prediction of the high-fidelity function by the low-fidelity one has $Q^2_{l\to h} = 0.98$. In the following, we set $\delta = 0.1$ and then $Q^2_{l\to h} = 0.87$. An additive Gaussian noise is added to the low-fidelity code. The noise has a zero mean and a variance equal to the empirical variance of the signal 0.08.

In this example we also consider that the low-fidelity code is costly and we only have a small number of low-fidelity points: $N_L = 25$ and $N_H = 15$. The high- and low-fidelity points are chosen by maximin Latin hypercube sampling (LHS). The test set is composed of 1000 independent points following a random uniform law on $[0, 1]^2$.

The kernel used for GP regression low fidelity is a Matérn 5/2 covariance function. The predictive error for the GP regressor of the low-fidelity model is $Q^2 = 0.91$ using a nugget effect in the Gaussian process regression. The BNN is defined with $N_l = 40$ neurons, and the mean and variance are evaluated by Eqs. (26) and (27) with $N_v = 500$. N_l could be chosen arbitrarily, but the fact that we were in a small data context led us to choose a small value. It is possible to use cross-validation to choose N_l, but the computer cost here would be prohibitive. The previous example discussed in Section 5.1 suggests choosing S between 3 and 5 for the low-fidelity surrogate model sampling. In this example, due to the large low-fidelity error, the model needs a large value of S to account appropriately for the uncertainty and we choose $S = 5$.

All methods are compared in Table 5. The GPBNN model in this example seems to be accurate in terms of Q^2_{T} and in uncertainty quantification. It is much better than the GP 1F model (single-fidelity GP model built with the high-fidelity data). We presume that it is due to the lack of high-fidelity data. AR(1) model gives better but not satisfying results. This is expected due to the nonlinearity between codes. The results of the Deep GP and MBK methods are

TABLE 5: Comparison of the multi-fidelity methods on the CURRIN function via Q^2_{T}, $\mathrm{CP}_{80\%}$ and $\mathrm{MPIW}_{80\%}$

	GP 1F	**AR(1)**	**Deep GP**	**MBK**	**GPBNN**
Q^2_{T}	0.73	0.80	0.29	0.27	0.88
$\mathrm{CP}_{80\%}$	0.68	0.80	0.62	0.57	0.80
$\mathrm{MPIW}_{80\%}$	0.5	1.0	0.13	1.9	0.51

worse than the one of the GP 1F in error and in uncertainty. For the Deep GP, this can be understood by the fact that the covariance is not well adapted, see [9]. And for the MBK, the lack of points leads to a very poor optimization of the hyperparameters.

5.3 Double Pendulum

The double pendulum system can be seen as a dual-oscillator cluster, see Fig. 7. The system is presented in [31]. The inputs of the system are of dimension 5, including $(k, M, \theta, \dot{\theta}, y_0)$. The output is of dimension 1, it is the maximum along the axis y of the position of the mass m in the first 10 s. We have two codes: (i) the high-fidelity code numerically solves Newton's equation and (ii) the low-fidelity code simplifies the equation, by linearization for small angles of the pendulum motion, and solves the system. Our goal is to build a surrogate model of the high-fidelity code using $N_L = 100$ and $N_H = 20$, with maximin LHS sampling. The input parameters are the mass $M \in [10, 12]$, the spring stiffness $k \in [1, 1.4]$, the initial angle of the pendulum $\theta_0 \in [\pi/4, \pi/3]$, the initial derivative $\dot{\theta}_0 \in [0, 1/10]$, and the initial position of the mass $y_0 \in [0, 0.2]$. The fixed parameters are $\dot{y_0} = 0$, the gravitational acceleration $g = 9.81$, the length of the pendulum $l = 2$, and its mass $m = 0.5$. The output is the maximum in time of the amplitude of the mass m. The error is computed on a test set, different for each learning set, of 64 samples uniformly distributed on the input space. To evaluate each model, we use five independent learning and test sets.

The GP regression on low fidelity is performed without trend with a Matérn 5/2 as kernel, and the hyperparameters are estimated with maximum likelihood, using [29]. The Q^2 [given in Eq. (29)] for the low-fidelity surrogate model for the GP is 0.98. The model is efficient to predict the low-fidelity code output, but the prediction interval is underevaluated with a coverage probability interval of 69% and mean predictive interval width of 0.041. The BNN is defined with the number of neurons $N_l = 30$. The BNN is one hidden layer fully connected with Gaussian priors for weights and bias, as described in Section 2.2. Therefore, the number of parameters in the BNN is $N_l \times d + 2 \times N_l + 1 = 211$. We estimate the posterior distribution of the weights and bias using the HMC algorithm with NUTS. This relatively small number of parameters allows one to use Pyro's NUTS algorithm (see [30]) with standard parameters. The number of estimation samples for mean and prediction intervals is $N_v = 500$. The high-fidelity code output is model by GPBNN. A Gauss-Hermite sample size $S = 5$ is chosen according to the results of Section 5.1.

The results are presented in Table 6. The prediction of the MBK method is not accurate compared to all the other methods. We think this is due to the small data set regarding the dimension of the BNN's input. However, the uncertainty of prediction is still accurate even if the uncertainty interval is large compared to the other methods (i.e., the coverage probability is close to the target value). This result is very surprising for us in regard to the poor quality

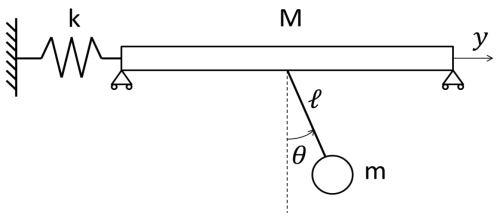


FIG. 7: Illustration of the double pendulum

TABLE 6: Comparison of the multi-fidelity methods on the pendulum example via Q^2_{T}, $\mathrm{CP}_{80\%}$, and $\mathrm{MPIW}_{80\%}$

	GP 1F	**AR(1)**	**Deep GP**	**MBK**	**GPBNN**
Q^2_{T}	0.93	0.94	0.95	0.54	0.95
$\mathrm{CP}_{80\%}$	0.81	0.78	0.62	0.88	0.80
$\mathrm{MPIW}_{80\%}$	0.154	0.146	0.069	0.859	0.101

of the low-fidelity model, which has a $Q^2_{l\to l}$ of 0.7. The Deep GP model shares the best predictive error with the GPBNN. The single fidelity and the AR(1) models display slightly larger errors. The $\mathrm{CP}_{80\%}$ values are in the target area for GP1F and AR(1), but they are associated with large prediction intervals. The Deep GP clearly underestimates the uncertainty of its predictions. The $\mathrm{CP}_{80\%}$ value is good for the GPBNN and close to the target value. Moreover, the uncertainty interval is the smallest of all methods. On this real-life example, the GPBNN is competitively compared to other state-of-the-art methods.

6. CONCLUSION

Our main focus in this paper was to give the Gaussian process regression posterior distribution of a low-fidelity model as input to a Bayesian neural network for multi-fidelity regression. Different methods are proposed and studied to transfer the uncertain predictions of the low-fidelity emulator to the high-fidelity one, which is crucial to obtain minimal predictive errors and accurate predictive uncertainty quantification. The Gauss-Hermite quadrature method is shown to significantly improve the predictive properties of the BNN. The conducted experiments show that the GPBNN method is able to process noisy and real-life problems. Moreover, the comparison to some state-of-the-art methods for multi-fidelity surrogate model highlight the precision in prediction and in uncertainty quantification.

It is possible to extend the GPBNN method to a hierarchical multi-fidelity framework with more than two levels of codes sorted by increasing accuracy. The natural approach is to consider the output of a low-fidelity metamodel as an input to the next metamodel. The problem that arises from this naive extension is that the input sample size increases from one metamodel to the other one. To solve this problem, two different methods can be considered. For simplicity, let us consider the case of three code levels. The first method is to consider that the metamodel for the lowest fidelity level carries no uncertainty. The low-fidelity metamodel prediction is then added to the low-fidelity inputs of a GPBNN modeling the two highest fidelity code levels. The second method consists of approximating the mid-fidelity output uncertainty of the BNN as Gaussian. This means modeling the system by one GP for the low-fidelity level and two different BNNs for the other fidelity levels. It becomes possible, with the Gaussian approximation, to use less expensive sampling methods, such as the Gauss-Hermite quadrature. Of course, other methods could be considered to control the sample size. We also anticipate that the GPBNN method could be extended beyond the hierarchical case of a sequence of simulators ranked from lowest to highest fidelity. Indeed, it should be possible to deal with a general Markov case in which the different fidelity levels are connected via a directed acyclic graph [32].

The interest of combining the regression method for multi-fidelity surrogate modeling is not to be proved, but this paper adds the heterogeneity of models for multi-fidelity modeling. To be able to combine heterogeneous models into one model and to consider the uncertainty between them is one of the keys to adapt the multi-fidelity surrogate model to a real-life regression problem.

The number of elements in the learning set is not a problem any more, thanks to [3]. This approach could be used for the GP part of the GPBNN. With more time in the training, the BNN part will be able to deal with many points, even if this ability is less critical because $N_L \gg N_H$. Consequently, the GPBNN can be extended in order to tackle larger data sets.

Existing autoregressive models and Deep GP can only be used for low-dimensional outputs. We wish to extend the method to high-dimensional outputs. Dimension reduction techniques have already been used as principal component analysis or autoencoder, as well as tensorized covariance methods [31] that remain to be extended to the

multi-fidelity context. However, neural networks are known to adapt to high-dimensional outputs. We should further investigate how to build multi-fidelity surrogate models with functional input/output in the context of small data. The ability to construct models that are tractable for high-dimensional input and output is key for further research.

REFERENCES

1. Williams, C.K. and Rasmussen, C.E., *Gaussian Processes for Machine Learning*, Cambridge, MA: MIT Press, 2006.
2. Santner, T.J., Williams, B.J., Notz, W., and Williams, B.J., *The Design and Analysis of Computer Experiments*, New York, NY: Springer, 2003.
3. Rullière, D., Durrande, N., Bachoc, F., and Chevalier, C., Nested Kriging Predictions for Datasets with a Large Number of Observations, *Stat. Comput.*, **28**(4):849–867, 2018.
4. Kennedy, M. and O'Hagan, A., Predicting the Output from a Complex Computer Code when Fast Approximations Are Available, *Biometrika*, **87**(1):1–13, 03 2000.
5. O'Hagan, A., A Markov Property for Covariance Structures, *Stat. Res. Rep.*, **98**(13):510, 1998.
6. Forrester, A.I., Sóbester, A., and Keane, A.J., Multi-Fidelity Optimization via Surrogate Modelling, *Proc. R. Soc. A*, **463**(2088):3251–3269, 2007.
7. Le Gratiet, L. and Garnier, J., Recursive Co-Kriging Model for Design of Computer Experiments with Multiple Levels of Fidelity, *Int. J. Uncertainty Quantif.*, **4**(5):364–386, 2014.
8. Perdikaris, P., Raissi, M., Damianou, A., Lawrence, N.D., and Karniadakis, G.E., Nonlinear Information Fusion Algorithms for Data-Efficient Multi-Fidelity Modelling, *Proc. R. Soc. A*, **473**(2198):20160751, 2017.
9. Cutajar, K., Pullin, M., Damianou, A., Lawrence, N., and González, J., Deep Gaussian Processes for Multi-Fidelity Modeling, *Stat. Mach. Learn.*, arXiv:1903.07320v1, 2019.
10. Song, J., Chen, Y., and Yue, Y., A General Framework for Multi-Fidelity Bayesian Optimization with Gaussian Processes, *Proc. of 22nd In. Conf. on Artificial Intelligence and Statistics*, PMLR, Okinawa, Japan, pp. 3158–3167, 2019.
11. Pilania, G., Gubernatis, J.E., and Lookman, T., Multi-Fidelity Machine Learning Models for Accurate Bandgap Predictions of Solids, *Comput. Mater. Sci.*, **129**:156–163, 2017.
12. Ng, L.W.T. and Eldred, M., Multifidelity Uncertainty Quantification Using Non-Intrusive Polynomial Chaos and Stochastic Collocation, *Proc. of 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conf. 20th AIAA/ASME/AHS Adaptive Structures Conf. 14th AIAA*, Honolulu, HI, p. 1852, 2012.
13. Palar, P.S., Zuhal, L.R., Shimoyama, K., and Tsuchiya, T., Global Sensitivity Analysis via Multi-Fidelity Polynomial Chaos Expansion, *Reliab. Eng. Syst. Saf.*, **170**:175–190, 2018.
14. Li, S., Xing, W., Kirby, R., and Zhe, S., Multi-Fidelity Bayesian Optimization via Deep Neural Networks, in *Advances in Neural Information Processing Systems*, Vol. 33, Curran Associates, Inc., pp. 8521–8531, 2020.
15. Meng, X. and Karniadakis, G.E., A Composite Neural Network That Learns from Multi-Fidelity Data: Application to Function Approximation and Inverse PDE Problems, *J. Comput. Phys.*, **401**:109020, 2020.
16. Zhang, X., Xie, F., Ji, T., Zhu, Z., and Zheng, Y., Multi-Fidelity Deep Neural Network Surrogate Model for Aerodynamic Shape Optimization, *Comput. Methods Appl. Mech. Eng.*, **373**:113485, 2021.
17. MacKay, D.J., A Practical Bayesian Framework for Backpropagation Networks, *Neural Comput.*, **4**(3):448–472, 1992.
18. Meng, X., Babaee, H., and Karniadakis, G.E., Multi-Fidelity Bayesian Neural Network: Algorithms and Applications, *Comput. Sci. Mach. Learn.*, arXiv:2012.13294, 2020.
19. Kabir, H.M.D., Khosravi, A., Hosen, M.A., and Nahavandi, S., Neural Network-Based Uncertainty Quantification: A Survey of Methodologies and Applications, *IEEE Access*, **6**:36218–36234, 2018.
20. Cigizoglu, H.K. and Alp, M., Generalized Regression Neural Network in Modelling River Sediment Yield, *Adv. Eng. Software*, **37**(2):63–68, 2006.
21. Tripathy, R.K. and Bilionis, I., Deep UQ: Learning Deep Neural Network Surrogate Models for High Dimensional Uncertainty Quantification, *J. Comput. Phys.*, **375**:565–588, 2018.
22. Jospin, L.V., Buntine, W., Boussaid, F., Laga, H., and Bennamoun, M., Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users, *Comput. Sci. Mach. Learn.*, arXiv:2007.06823, 2020.

23. Hoffman, M.D. and Gelman, A., The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo., *J. Mach. Learn. Res.*, **15**(1):1593–1623, 2014.
24. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., and Teller, E., Equation of State Calculations by Fast Computing Machines, *J. Chem. Phys.*, **21**(6):1087–1091, 1953.
25. Robert, C.P., *The Bayesian Choice*, New York: Springer, 2004.
26. Roberts, G.O., Gelman, A., and Gilks, W.R., Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms, *Ann. Appl. Probab.*, **7**(1):110–120, 1997.
27. Duane, S., Kennedy, A.D., Pendleton, B.J., and Roweth, D., Hybrid Monte Carlo, *Phys. Lett. B*, **195**(2):216–222, 1987.
28. Gautschi, W., *Numerical Differentiation and Integration*, New York: Springer, pp. 159–251, 2012.
29. GPy, GPy: A Gaussian Process Framework in Python, from http://github.com/SheffieldML/GPy, 2012.
30. Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P.A., Horsfall, P., and Goodman, N.D., Pyro: Deep Universal Probabilistic Programming, *J. Mach. Learn. Res.*, **20**(1):973–978, 2019.
31. Perrin, G., Adaptive Calibration of a Computer Code with Time-Series Output, *Reliab. Eng. Syst. Safety*, **196**:106728, 2020.
32. Ji, Y., Mak, S., Soeder, D., Paquet, J., and Bass, S.A., A Graphical Multi-Fidelity Gaussian Process Model, with Application to Emulation of Expensive Computer Simulations, *Stat. Methodol.*, arXiv:2108.00306, 2021.