Journal of Computational Physics 397 (2019) 108850

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems

Dongkun Zhang [a], Lu Lu [a], Ling Guo [b,*], George Em Karniadakis [a]

[a] Division of Applied Mathematics, Brown University, Providence, RI, USA
[b] Department of Mathematics, Shanghai Normal University, Shanghai, China

ARTICLE INFO

Article history:
Received 29 September 2018
Received in revised form 8 May 2019
Accepted 21 July 2019
Available online 26 July 2019

Keywords:
Physics-informed neural networks
Uncertainty quantification
Stochastic differential equations
Arbitrary polynomial chaos
Dropout

ABSTRACT

Physics-informed neural networks (PINNs) have recently emerged as an alternative way of numerically solving partial differential equations (PDEs) without the need of building elaborate grids, instead, using a straightforward implementation. In particular, in addition to the deep neural network (DNN) for the solution, an auxiliary DNN is considered that represents the residual of the PDE. The residual is then combined with the mismatch in the given data of the solution in order to formulate the loss function. This framework is effective but is lacking uncertainty quantification of the solution due to the inherent randomness in the data or due to the approximation limitations of the DNN architecture. Here, we propose a new method with the objective of endowing the DNN with uncertainty quantification for both sources of uncertainty, i.e., the *parametric uncertainty* and the *approximation uncertainty*. We first account for the parametric uncertainty when the parameter in the differential equation is represented as a stochastic process. Multiple DNNs are designed to learn the modal functions of the *arbitrary polynomial chaos* (aPC) expansion of its solution by using stochastic data from sparse sensors. We can then make predictions from new sensor measurements very efficiently with the trained DNNs. Moreover, we employ *dropout* to quantify the uncertainty of DNNs in approximating the modal functions. We then design an *active learning* strategy based on the dropout uncertainty to place new sensors in the domain in order to improve the predictions of DNNs. Several numerical tests are conducted for both the forward and the inverse problems to demonstrate the effectiveness of PINNs combined with uncertainty quantification. This NN-aPC new paradigm of physics-informed deep learning with uncertainty quantification can be readily applied to other types of stochastic PDEs in multi-dimensions.

1. Introduction

How to make the best use of existing data while exploiting the information from classical mathematical models or even empirical correlations developed within a discipline is an important issue as *data-driven modeling* is emerging as a powerful paradigm for physical and biological systems. For example, in geophysics, researchers have been using the remote sensing data collected from multi-spectral satellites and the top-of-atmospheric reflectance model as a calibration of the data to study the soil salinization [1], or estimating the Earth heat loss based on the heat flow measurements and a model of the

* Corresponding author.
E-mail address: lguo@shnu.edu.cn (L. Guo).

https://doi.org/10.1016/j.jcp.2019.07.048
0021-9991/

hydrothermal circulation in the oceanic crust [2]. Data can be used to provide closures in nonlinear models or to estimate parameters or functions in mathematical models. Moreover, mathematical models can be used as additional knowledge to formulate "informative priors" in statistical estimation methods or be encoded in specially designed machine learning tools so that a smaller amount of data is required for inference of system identification. There has been recent progress for both forward (inference) and inverse (identification) problems using different methods. For example, for the forward problem, some of the popular choices of machine learning tools are Gaussian process [3–8] and deep neural networks (DNNs) [9–13]. For inverse problems, similar methods have been advanced, e.g., Bayesian estimation [14] and variational Bayes inference [15], and have been proposed for a wide variety of objectives, from parameter estimation [16] to discovering partial differential equations [17–20] to learning constitutive relationships [21].

In this work we focus on the DNNs, and in particular the *physics-informed neural networks* (PINNs) for forward and inverse problems, first introduced in [12,19]. However, in those works the mathematical models were deterministic differential equations, so here we consider stochastic differential equations with random parameters. There have only been very few works published on solving stochastic differential equations using DNNs, e.g., [22,23], for forward problems. Here we study the case where some of the physics is known, namely via the stochastic differential equations, and the random parameter in the equation is represented as a stochastic process, introducing *parametric uncertainty*. First, we solve the forward stochastic Poisson's equation, where there is uncertainty associated with the driving force. Subsequently, we consider the inverse stochastic elliptic equation of which the diffusivity is modeled as a random process, and an inverse time-dependent nonlinear diffusion-reaction equation with the forcing term modeled as a random process. In both cases, we have only partial information of the diffusivity/forcing term from scattered sensors but we have much more data available for the solution, and we aim to infer the stochastic processes of not only the solution but also the diffusivity/forcing term, and quantify their uncertainties given the randomness in the data.

In particular, in this paper we combine the *arbitrary polynomial chaos* (aPC) with PINNs for both the forward and the inverse stochastic problems. One of the most popular methods for uncertainty quantification studies is the *polynomial chaos* [24,25] because it has been very effective in representing stochastic fields. However, aPC [26–30] is more suitable for building the orthogonal basis from arbitrary random space, without the need of any assumption on the distribution of the data. Therefore, in the current work, we employ the aPC to develop a combined method that we call NN-aPC, where we use the DNNs to learn each individual mode of the aPC expansion. More importantly, after training, the proposed method can be used to predict new realizations of the solution based only on very few measurements.

An additional uncertainty is due to the DNN approximation, which we will refer to as the *approximation uncertainty*. Treatment of the DNN approximation uncertainty has been addressed using different methods in the past. The traditional way to estimate uncertainty in DNNs is using the Bayes' theorem, e.g., the Bayesian neural networks (BNNs) [31,32]. BNNs are standard DNNs with prior probability distributions placed over their weights, and given observed data, inference is then performed on weights. Because the inference is not tractable in general, variational inference or Markov Chain Monte Carlo (MCMC) are often used to approximate the inference [33–43]. However, these models have very high additional computational cost because they require more parameters for the same network size and more time for the DNN parameters to converge. Rivals and Personnaz [44] proposed a simple way to construct confidence intervals for a nonlinear regression based on least squares estimation. Specifically, they assumed that the measured output has random noise with zero mean, and considered the regression problem using NNs with quadratic loss function. However, our problem is not a standard regression problem, and our loss function is much more complicated than a quadratic loss. Hence, their method is not suitable for our problem. Recently, Gal et al. developed a new way to quantify uncertainty in DNNs by using *dropout* [45–47], which is largely used as a regularization technique [48,49] to address the problem of over-fitting. Gal et al. [45] showed that a DNN with dropout is mathematically equivalent to approximating a probabilistic deep Gaussian process [50], no matter what network architecture and non-linearities are used. Moreover, dropout does not induce much computation overhead and thus has been used as a practical tool to obtain uncertainty estimation effectively in real applications including language modeling [51], computer vision [52,53] and medical applications [54,55]. We also note that the physical laws can be straightforwardly imposed to the dropout networks in the same way as PINNs. In this paper, dropout is used to estimate the uncertainty in approximating each aPC mode. Based on the magnitude of this uncertainty we set up an active learning strategy and deploy additional sensors to obtain more measurements of the quantity of interest (QoI), in order to improve the predictability of PINNs. Taken together, we refer to the parametric uncertainty and the approximation uncertainty as the *total* uncertainty. To the best of our knowledge, the current work is the first to address total uncertainty in solving stochastic forward and inverse problems using DNNs.

The organization of this paper is as follows. In Section 2, we set up the data-driven forward and inverse problems. In Section 3, we introduce the PINNs for solving deterministic differential equations, followed by our main algorithm, the NN-aPC, and the method of dropout for uncertainty. In Section 4, we provide a detailed study of the accuracy and performance of the NN-aPC method for solving both the forward and inverse stochastic diffusion equation and demonstrate the effectiveness of active learning via dropout-induced uncertainty. Finally, we conclude with a brief discussion in Section 5.

2. Problem setup

Suppose we have a stochastic differential equation:

$$\begin{aligned} &\mathcal{N}_x[u(x;\omega);k(x;\omega)]=0, \quad x\in\mathcal{D}, \quad \omega\in\Omega,\\ &\text{B.C.:} \qquad \mathcal{B}_x[u(x;\omega)]=0, \quad x\in\Gamma, \end{aligned} \tag{1}$$

where $\mathcal{N}_x$ is the general form of a differential operator that could be nonlinear, $\mathcal{D}$ is a d-dimensional physical domain in $\mathbb{R}^d$, Ω is the random space, and $u(x;\omega)$ is the solution to this equation. The boundary condition is imposed through the generalized boundary condition operator $\mathcal{B}_x$ at the domain boundary Γ. The random parameter $k(x;\omega)$ is the source of parametric uncertainty, which could be represented by either a few random variables or by an infinite dimensional (in the random space) random process.

We solve two types of data-driven problem here: first, a *forward* problem, where we know exactly the distribution of $k(x;\omega)$ everywhere in the domain $\mathcal{D}$ and $u(x;\omega)$ is our QoI; and second, an *inverse* problem, where we assume that we have incomplete information on $k(x;\omega)$ but some extra knowledge on $u(x;\omega)$, and we are interested in inferring the full stochastic profile of $k(x;\omega)$. In practice, the information usually comes from data collected via sensor measurements for both types of problem. Here, we summarize the different scenarios of the sensors placement for each type of the problem:

- *Forward problem*: The u-sensors are placed only at the boundary Γ to provide boundary condition, while the k-sensors are virtual (since we know the distribution of k), thus we can have as many k-sensors as we want and they can be placed anywhere in $\mathcal{D}$.
- *Inverse problem*: In addition to having u-sensors at the boundary Γ, we have a limited number of extra u-sensors that can be placed in the domain $\mathcal{D}$, whereas we only have a limited number of k-sensors.

In this paper, we address both types of problems but we will focus more on solving the inverse problem.

3. Methodology

3.1. Physics-informed neural network

In this part, we briefly review using DNNs to solve deterministic differential equations [9,10,12], and its generalization for solving deterministic inverse problems in [19]. To this end, re-consider Equation (1) but replace the random input ω and approximate it with a finite set of parameters, leading to a parameterized differential equation:

$$\begin{aligned} &\mathcal{N}_x[u;\eta]=0, \quad x\in\mathcal{D},\\ &\text{B.C.:} \qquad \mathcal{B}_x[u;\eta]=0, \quad x\in\Gamma, \end{aligned} \tag{2}$$

where $u(x)$ is the solution and η denotes the parameters.

A DNN, denoted by $\hat{u}(x;\theta)$, is constructed as a surrogate of the solution $u(x)$, and it takes the coordinate x as the input and outputs a vector that has the same dimension as u. Here we use θ to denote the DNN parameters that will be tuned at the training stage, namely, θ contains all the weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$ in $\hat{u}(x;\theta)$. For this surrogate network $\hat{u}$, we can take its derivatives with respect to its input by applying the chain rule for differentiating compositions of functions using the automatic differentiation, which is conveniently integrated in many machine learning packages such as Tensorflow [56]. The restrictions on $\hat{u}$ are two-fold: first, given the set of scattered data of the $u(x)$ observations, the network should be able to reproduce the observed value, when taking the associated x as input; second, $\hat{u}$ should comply with the physics imposed by Equation (2). The second part is achieved by defining a residual:

$$\hat{r}(x;\theta,\eta) := \mathcal{N}_x[\hat{u}(x;\theta);\eta], \tag{3}$$

which is computed from $\hat{u}$ straightforwardly with automatic differentiation. This residual $\hat{r}(x;\theta,\eta)$ induces a PDE network, which together with the network $\hat{u}$ forms a physics-informed neural network (PINN) [12]. It shares the parameters θ with network $\hat{u}$ and should output the constant 0 for any input $x\in\mathcal{D}$. Fig. 1 shows a sketch of the PINN. At the training stage, the shared parameters θ (and also η, if it is also to be inferred) are fine-tuned to minimize a loss function that reflects the above two constraints. Suppose we have a total number of N_u observations on u, collected at location $\{x_u^{(i)}\}_{i=1}^{N_u}$, and N_r is the number of training points $\{x_r^{(i)}\}_{i=1}^{N_r}$ where we evaluate the residual $\hat{r}(x_r^{(i)};\theta,\eta)$. We shall use (x^*, y^*) to represent a

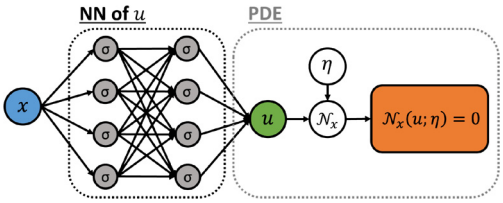

Fig. 1. Schematic of the PINN for solving differential equations.

single instance of training data, where the first entry x^* denotes the input and the second entry y^* denotes the anticipated output (also called "label"). The workflow of solving a differential equation with PINN can be summarized as follows:

Algorithm 1 PINN for solving differential equations with random inputs.

Step 1: Specify the training set:

$$\hat{u} \text{ network: } \{(x_u^{(i)}, u(x_u^{(i)}))\}_{i=1}^{N_u}, \quad \hat{r} \text{ network: } \{(x_r^{(i)}, 0)\}_{i=1}^{N_r}.$$

Step 2: Construct a DNN $\hat{u}(x;\theta)$ with randomly initialized parameters θ.

Step 3: Calculate the residual $\hat{r}(x;\theta,\eta)$ by substituting the surrogate $\hat{u}$ into the governing equation (Equation (3)) via automatic differentiation and arithmetic operations.

Step 4: Specify a loss function by summing the mean squared error of both the u observations and the residual:

$$\mathcal{L}(\theta,\eta) = \frac{1}{N_u}\sum_{i=1}^{N_u}[\hat{u}(x_u^{(i)};\theta) - u(x_u^{(i)})]^2 + \frac{1}{N_r}\sum_{i=1}^{N_r}\hat{r}(x_r^{(i)};\theta,\eta)^2. \tag{4}$$

Step 5: Train the DNNs to find the best parameters θ and η by minimizing the loss function:

$$\theta = \arg\min \mathcal{L}(\theta,\eta) \tag{5}$$

3.2. NN-aPC: combining arbitrary polynomial chaos with neural networks

We generalize the PINN method to solve stochastic differential equations for both forward and inverse problems, i.e., we aim to infer continuous random processes. Assume that a sensor will generate a sequence of measurements after being installed, and when the data is recorded, all sensors are read simultaneously. We denote the measurements from all the sensors at the same instant by a *snapshot* of the sensor data. Although the measurement results change from one measurement to the next due to randomness, it is reasonable to believe that every snapshot of sensor data corresponds to the same random event in the random space. We also assume that when the number of snapshots is big enough, the empirical distribution approximates the true distribution.

Let us consider Equation (1). Suppose we have N_k sensors for $k(x;\omega)$ placed at $\{x_k^{(i)}\}_{i=1}^{N_k}$, N_u sensors for $u(x;\omega)$ placed at $\{x_u^{(i)}\}_{i=1}^{N_u}$, and N_r training points at $\{x_r^{(i)}\}_{i=1}^{N_r}$ that are used to calculate the residual of Equation (1). A total number of N snapshots of measurements are made from all these sensors. Let $k_s^{(i)}$ and $u_s^{(i)}$ $(s = 1, 2, ..., N)$ be the s-th measurement of k and u at location $x_k^{(i)}$ and $x_u^{(i)}$ respectively, and ω_s is the random instance at the s-th measurement, i.e., $k_s^{(i)} = k(x_k^{(i)};\omega_s)$ and $u_s^{(i)} = u(x_u^{(i)};\omega_s)$. The training data set can be represented by

$$\mathcal{S}_t = \{\{(x_k^{(i)}, k_s^{(i)})\}_{i=1}^{N_k}, \{(x_u^{(i)}, u_s^{(i)})\}_{i=1}^{N_u}, \{(x_r^{(i)}, 0)\}_{i=1}^{N_r}\}_{s=1}^{N}. \tag{6}$$

The proposed NN-aPC method consists of the following steps:

1. dimension reduction;
2. constructing the aPC basis;
3. building the NN-aPC as a surrogate model of aPC modes and train the network for each mode.

The trained NN-aPC can then be used to calculate the statistics of our QoI and to predict new instances of the continuous trajectories of the QoI, with newly collected sensor data. We will explain each of three steps and the prediction procedure below.

3.2.1. Dimension reduction with principal component analysis

As the first step, we find a lower dimensional random space spanned by a set of hidden random variables. We analyze the data of k, which is the source of randomness in Equation (1), using principal component analysis (PCA) to obtain a set of uncorrelated random variables $\boldsymbol{\xi}_s$, which are the coordinates of the QoI in the stochastic space. Let K be the $N_k \times N_k$ covariance matrix for the sensor measurements on k, i.e.,

$$K_{i,j} = \mathrm{Cov}(k^{(i)}, k^{(j)}). \tag{7}$$

Let λ_l and ϕ_l be the l-th largest eigenvalue and its associated normalized eigenvector of K. Therefore, PCA yields

$$K = \mathbf{\Phi}^T \mathbf{\Lambda} \mathbf{\Phi}, \tag{8}$$

where $\mathbf{\Phi} = [\phi_1, \phi_2, ..., \phi_{N_k}]$ is an orthonormal matrix and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, ...\lambda_{N_k})$ is a diagonal matrix. Let $\boldsymbol{k}_s = [k_s^{(1)}, k_s^{(2)}, ..., k_s^{(N_k)}]^T$ be the results of the k measurements of the s-th snapshot, then

$$\boldsymbol{\xi}_s = \mathbf{\Phi}^T \sqrt{\mathbf{\Lambda}}^{-1} \boldsymbol{k}_s \tag{9}$$

is an uncorrelated random vector, and hence $\boldsymbol{k}_s$ can be rewritten as a reduced dimensional expansion

$$\boldsymbol{k}_s \approx \boldsymbol{k}_0 + \sqrt{\mathbf{\Lambda}^M} \mathbf{\Phi}^M \boldsymbol{\xi}_s^M, \quad M < N_k, \tag{10}$$

where $\boldsymbol{k}_0 = \mathbb{E}[\boldsymbol{k}]$ is the mean of each sensor's measurements. The choice of M depends on how much energy should be maintained in the low-dimensional random space. For reasons of simplicity, we shall always use the reduced dimensional representation and omit the superscript M. We note that Equation (10) can also be written in the form of the Karhunen-Loève expansion:

$$k(x_k^{(i)}; \omega_s) \approx k_0(x_k^{(i)}) + \sum_{l=1}^{M} \sqrt{\lambda_l} k_l(x_k^{(i)}) \xi_{s,l}, \quad M < N_k, \tag{11}$$

where $k_0(x_k^{(i)})$ is the mean of k measurements at $x_k^{(i)}$, $k_l(x)$ is the l-th mode function of $k(x;\omega)$ whose value at $x_k^{(i)}$ coincides with the i-th entry of the eigenvector ϕ_l, and $\xi_{s,l}$ is the l-th entry of the random vector $\boldsymbol{\xi}_s$. We want to extend the range of k_l to the entire domain to approximate the continuous samples of $k(x;\omega)$.

3.2.2. Arbitrary polynomial chaos

Assume that we have a set of M-dimensional samples of random vectors

$$S := \{\boldsymbol{\xi}_s\}_{s=1}^N$$

with hidden probability measure $\rho(\boldsymbol{\xi})$. Given a sufficiently large number of snapshots, we can approximate the underlying probability measure $\rho(\boldsymbol{\xi})$ by the discrete measure $\nu_S(\boldsymbol{\xi})$,

$$\rho(\boldsymbol{\xi}) \approx \nu_S(\boldsymbol{\xi}) = \frac{1}{N} \sum_{\boldsymbol{\xi}_s \in S} \delta_{\boldsymbol{\xi}_s}(\boldsymbol{\xi}), \tag{12}$$

where $\delta_{\boldsymbol{\xi}_s}$ is the Dirac measure. Then, a set of multivariate orthonormal polynomial basis functions $\{\psi_\alpha(\boldsymbol{\xi})\}_{\alpha=0}^P$ can be constructed via the Gram-Schmidt orthogonalization process following [30,29]. The subscript α is the graded lexicographic multi-index and the number of basis, $P+1$, depends on the highest allowed polynomial order r in $\psi_\alpha(\boldsymbol{\xi})$, following the formula

$$P + 1 = \frac{(r+M)!}{r!\,M!}. \tag{13}$$

Specifically, the basis $\{\psi_\alpha(\boldsymbol{\xi})\}_{\alpha=0}^P$ are constructed using the recursive algorithm

$$\psi_\alpha(\boldsymbol{\xi}) = w_\alpha^\alpha \psi_\alpha^*(\boldsymbol{\xi}) - \sum_{\beta \prec \alpha} w_\beta^\alpha \psi_\beta(\boldsymbol{\xi}), \tag{14}$$

where $\psi_\alpha^*(\boldsymbol{\xi}) := \prod_{i=1}^M \boldsymbol{\xi}_i^{\alpha_i}$ represents the multivariate monomial basis function; the coefficients w_β^α are determined by imposing the orthonormal condition with respect to the discrete measure ν_S, i.e.,

$$\begin{aligned} \int \psi_\alpha(\boldsymbol{\xi}) \psi_\beta(\boldsymbol{\xi}) \mathrm{d}\rho(\boldsymbol{\xi}) &\approx \int \psi_\alpha(\boldsymbol{\xi}) \psi_\beta(\boldsymbol{\xi}) \mathrm{d}\nu_S(\boldsymbol{\xi}) \\ &= \frac{1}{N} \sum_{s=1}^{N} \psi_\alpha(\boldsymbol{\xi}_s) \psi_\beta(\boldsymbol{\xi}_s) \\ &\equiv \delta_{\alpha,\beta}, \quad \beta \preceq \alpha. \end{aligned} \tag{15}$$

Note that the construction of aPC bases is purely data-driven, and the orthogonality holds exactly under the discrete measure $\nu_S(\boldsymbol{\xi})$ generated by data. With the polynomial basis $\{\psi_\alpha(\boldsymbol{\xi})\}$ that are automatically adapted to the distribution of $\boldsymbol{\xi}$, we can write any function $g(x;\boldsymbol{\xi})$ in the form of the aPC expansion,

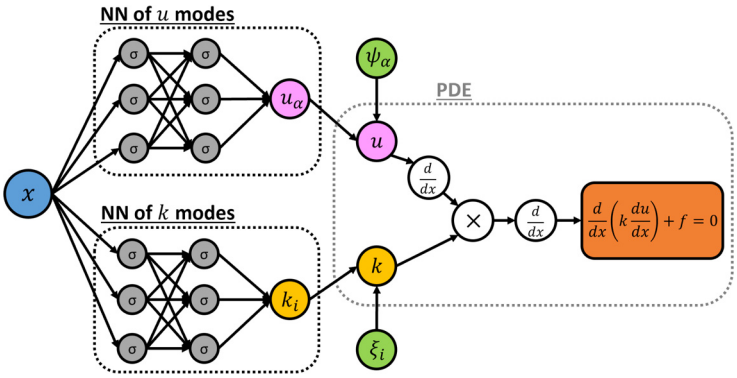

Fig. 2. Schematic of the NN-aPC for solving the stochastic elliptic equation $-\frac{\mathrm{d}}{\mathrm{dx}}(k(x;\omega)\frac{\mathrm{d}}{\mathrm{dx}}u)=f$.

$$g(x;\boldsymbol{\xi})=\sum_{\alpha=0}^{P} g_\alpha(x)\psi_\alpha(\boldsymbol{\xi}), \tag{16}$$

where the functions $g_\alpha(x)$ are called the aPC modes of g and can be calculated by

$$g_\alpha(x)=\frac{1}{N}\sum_{s=1}^{N}\psi_\alpha(\boldsymbol{\xi}_s)g(x;\boldsymbol{\xi}_s). \tag{17}$$

3.2.3. Learning stochastic modes

The key to our method is to train DNNs that predict the stochastic modes of our QoI. In this section, we focus on the inverse problem where we have to learn both the modes of u and k. (Solving a forward problem is similar and more straightforward, and will be briefly discussed in Section 4.1.1.) Two disjoint DNNs are constructed, i.e., the network $\widehat{u_\alpha}$, which takes the coordinate x as the input and outputs a $(P+1)\times 1$ vector of the aPC modes of u evaluated at x, and the network $\widehat{k_i}$ that also takes the coordinate x as the input and outputs a $(M+1)\times 1$ vector of the modes of k (we take k_0 in Equation (11) as the 0-th mode of k). Then, we can approximate k and u at the s-th snapshot by

$$\tilde{k}(x;\omega_s)=\widehat{k_0}(x)+\sum_{i=1}^{M}\sqrt{\lambda_i}\widehat{k_i}(x)\xi_{s,i}, \tag{18}$$

and

$$\tilde{u}(x;\omega_s)=\sum_{\alpha=0}^{P}\widehat{u_\alpha}(x)\psi_\alpha(\boldsymbol{\xi_s}). \tag{19}$$

Similar to the PINN method, we calculate the residual via automatic differentiation and arithmetic operations of DNNs by substituting $u(x;\omega)$ and $k(x;\omega)$ in Equation (1) with $\tilde{u}(x;\omega_s)$ and $\tilde{k}(x;\omega_s)$. This residual is designed to reflect the essence of the aPC expansion (see Fig. 2 for a schematic of the NN-aPC).

In practice, for $\widehat{k_i}$, we separate the mean from the rest of the modes and learn it with a small scale DNN, and for $\widehat{u_\alpha}$, we group the modes corresponding to the same order of aPC expansion together and learn each group of modes with a separate DNN, as depicted in Fig. 3. This is due to fact that the mean and the modes of different orders often correspond to different scales that we do not know *a priori*.

The loss function is defined as a sum of the mean squared errors (MSEs):

$$\mathcal{L}(\mathcal{S}_t)=MSE_u+MSE_k+MSE_r, \tag{20}$$

where

$$MSE_u=\frac{1}{NN_u}\sum_{s=1}^{N}\sum_{i=1}^{N_u}\left[\left(\tilde{u}(x_u^{(i)};\omega_s)-u(x_u^{(i)};\omega_s)\right)^2\right], \tag{21}$$

$$MSE_k=\frac{1}{NN_k}\sum_{s=1}^{N}\sum_{i=1}^{N_k}\left[\left(\tilde{k}(x_k^{(i)};\omega_s)-k(x_k^{(i)};\omega_s)\right)^2\right], \tag{22}$$

and

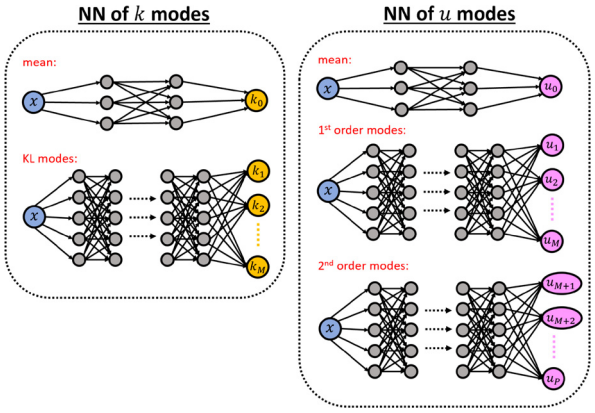

Fig. 3. Schematic of the DNNs used for learning the stochastic modes of k (left-hand side plot) and u (right-hand side plot). The mean functions are modeled separately using small scale DNNs. For k, all its rest modal functions are modeled using one DNN. For u, the modes that correspond to the same order of aPC expansion are grouped together and are modeled with a single DNN.

$$MSE_r = \frac{1}{NN_r}\sum_{s=1}^{N}\sum_{i=1}^{N_r}\left[\left(\mathcal{N}_x[\tilde{u}(x_r^{(i)};\omega_s);\tilde{k}(x_r^{(i)};\omega_s)]\right)^2\right]. \tag{23}$$

It is worth noting that one may write the loss function as a weighted sum of MSEs, as the MSE with a larger weight puts more emphasis on minimizing its associated discrepancy at the training stage. The distribution of weights in the loss function remains an open research topic and is not the focus point in this paper. Our numerical tests indicate that a moderate distribution of weights does not have a significant impact on the performance of the proposed algorithm. Besides, lacking extra information, it would be too artificial to put arbitrary weights in front of the loss function components. Therefore, in the numerical examples we use the plain summation of MSEs as our loss function, just as written in Equation (20). So far, we have specified the training data, constructed the DNNs and formalized the loss function, and we are now ready to train the DNNs.

3.2.4. Predicting stochastic realizations

In real applications, the training set could be generated from historical data, and the training process should be performed at the offline stage. The fine-tuned model shall be used to predict new random instances provided with new snapshots of sensor data, at the online stage. Suppose we have a snapshot of sensor data $\{\{(x_k^{(i)}, k_{\text{new}}^{(i)})\}_{i=1}^{N_k}, \{(x_u^{(i)}, u_{\text{new}}^{(i)})\}_{i=1}^{N_u}\}$; the first step is to extract the hidden random variables $\boldsymbol{\xi}_{\text{new}}$ from $\{k_{\text{new}}^{(i)}\}_{i=1}^{N_k}$ using Equation (9). Then, for any assigned location x, we can predict the modal functions for both $k(x;\omega)$ and $u(x;\omega)$ from the trained DNNs $\widehat{k_i}$ and $\widehat{u_\alpha}$. Finally, the prediction of k and u for the new random instance can be made via Equation (18) and (19), respectively.

3.3. Dropout for uncertainty

Although DNNs can be used to approximate any measurable function accurately, standard DNNs do not convey model uncertainty. Dropout is one convenient way to quantify the approximation uncertainty in DNNs. The key idea of dropout is to drop units from the DNN independently and randomly with a pre-selected probability $p \in (0, 1)$. In the original work, dropout was only used during training, while no units were dropped at test time, i.e., the prediction of the DNN for the unknown data was deterministic. In the dropout for uncertainty, the units are also dropped randomly at test time, resulting in stochastic predictions for each forward pass. The loss in the dropout inference is the summation of the original loss and a l_2 regularization term over the DNN parameters, as suggested in [45]:

$$\mathcal{L} = \frac{1}{N_t}\sum_{i=1}^{N_t} l(\widehat{y}_i, y_i) + \lambda\sum_i \theta_i^2, \tag{24}$$

where N_t is the number of training points, $\widehat{y}_i$ is the prediction, y_i is the true value, $l(\cdot,\cdot)$ is the loss for a single prediction, θ_i is any weight or bias, and λ is the l_2 regularization rate.

During prediction, to compute the output y for an input x, we need to run the forward propagation of neural networks independently with dropout for $T \gg 1$ times, generating T different estimates of y: $\{\mathcal{NN}_1(x), \mathcal{NN}_2(x), \ldots, \mathcal{NN}_T(x)\}$. The mean and variance of these outputs are directly estimated as

$$\mathbb{E}(y) \approx \frac{1}{T}\sum_{t=1}^{T}\mathcal{NN}_t(x), \tag{25}$$

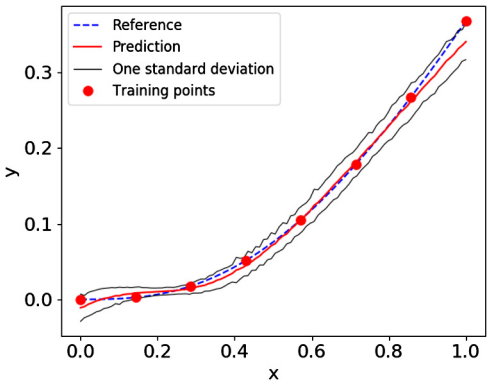

Fig. 4. Dropout for uncertainty: An example of using the dropout in DNN to approximate the function $y = x^3 e^{-x}$ in the domain $[0, 1]$, where we use four hidden layers and 20 neurons per hidden layer, and we choose $p = 0.01, \lambda = 10^{-6}$. The mean and standard deviation are calculated from $T = 1000$ runs.

$$\mathrm{Var}(y) \approx \frac{1}{T}\sum_{t=1}^{T}[\mathcal{NN}_t(x) - \mathbb{E}(y)]^2. \tag{26}$$

We use $\mathrm{Var}(y)$ to quantify the approximation uncertainty of DNNs.

Fig. 4 shows an example of using the dropout DNN for regression. In this example, we use a small dropout rate and a weak regularization strength, and thus the DNN with four hidden layers and 20 neurons per layer is large enough to approximate the target function. This example is used to demonstrate the ability of dropout to estimate the uncertainty, rather than to obtain a good approximation. In Fig. 4, we show that even if the approximation obtained from a DNN is bad, we are still able to estimate a good uncertainty using dropout. We plan to use the dropout strategy in the NN-aPC method to estimate the uncertainty of our DNN model and as a guidance for active learning.

4. Numerical examples

4.1. NN-aPC for solving stochastic PDEs

We first demonstrate the effectiveness of solving stochastic differential equations with the NN-aPC method for the forward and inverse problems.

4.1.1. Forward problem: stochastic Poisson's equation

Consider the following one-dimensional stochastic Poisson's equation with homogeneous boundary conditions:

$$\begin{aligned} -\frac{\mathrm{d}^2}{\mathrm{d}x^2}u &= f(x;\omega), \quad x \in [-1,1] \text{ and } \omega \in \Omega, \\ u(-1) &= u(1) = 0. \end{aligned} \tag{27}$$

Here Ω is the random space, the forcing term $f(x;\omega) \sim \mathcal{GP}(f_0(x), \mathrm{Cov}(x,x'))$ is a Gaussian random process with mean $f_0(x) = 10\sin(\pi x)$ and a squared exponential covariance function

$$\mathrm{Cov}(x,x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{l_c^2}\right), \tag{28}$$

where the standard deviation $\sigma = 1.0$ and the correlation length $l_c = 0.5$.

In this and the following examples, all data are generated by the Monte Carlo sampling method. Specifically, we sample $N = 1000$ snapshots of continuous $f(x;\omega)$ trajectories $\{f_s = f(x;\omega_s)\}_{s=1}^N$ and extract from $\{f_s\}_{s=1}^N$ the values of the forcing term at $x_f^{(i)}$ $(i = 1, 2, ..., N_f)$, where we place N_f (virtual) f-sensors. Therefore, we are able to calculate the SDE residual at $x_f^{(i)}$ (similar to $x_r^{(i)}$ in Equation (23)). For every $f(x;\omega)$ trajectory, we solve for its corresponding solution trajectories $u(x;\omega)$ using the finite difference method, and will use the statistics of these u trajectories as our reference. To evaluate the performance of the trained model, we collect another $N_s = 500$ snapshots of continuous $f(x;\omega)$ and $u(x;\omega)$ trajectories independently from the training data. The N_s pairs of (f,u) trajectories form our test sample set. Similarly, we extract the f-sensor data from every snapshot in the test set as the input at the predicting stage. We shall use the same N and N_s in the following tests, if not explicitly mentioned.

We place $N_f = 13$ sensors of $f(x;\omega)$ in the $[-1,1]$ domain equidistantly and keep 6 principal random variables (corresponding to 99% stochastic energy) after performing PCA. The solution $u(x;\omega)$ is approximated with a first-order aPC expansion. Fig. 5 shows the scattered plots of the measurements from the first three f-sensors and the first three arbitrary

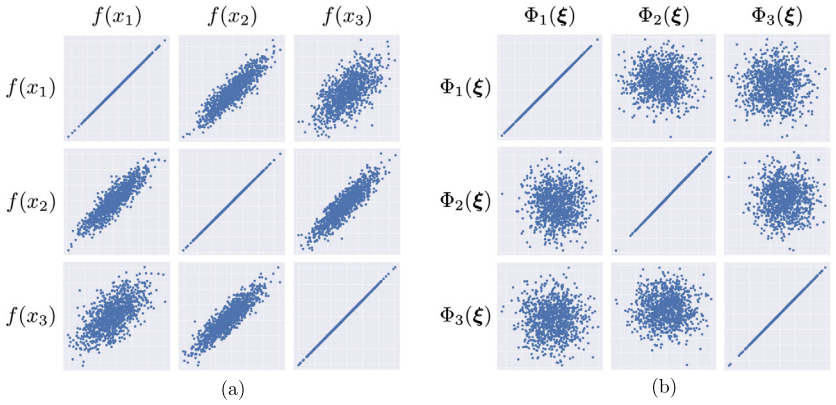

Fig. 5. Correlation structure: (a) Scattered plots of data collected from the first three f-sensors. The correlation between different sensors is significant, and the closer the sensors are, the more correlated measurements they produce. (b) Scattered plots of the first three aPC basis evaluations. There is no correlation between different aPC basis functions.

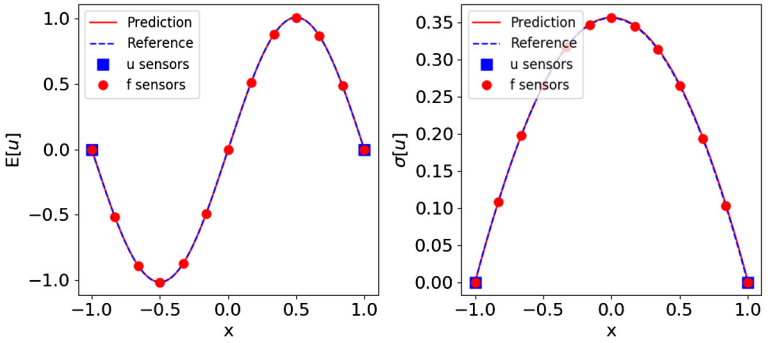

Fig. 6. Forward problem: The mean function (left) and the standard deviation (right) of u predicted using the trained DNN. The reference is calculated from the 1000 snapshots of continuous u samples. In this case, 13 f-sensors are employed, while only two u-sensors are placed at the domain boundaries.

polynomial bases. It is evident that the raw data from the measurements are correlated while their induced polynomials are not, thus the induced polynomials would serve as a valid set of basis in the random space.

The DNNs used to approximate the modes of u are constructed as in Fig. 3, where we use an isolated small scale DNN of two hidden layers with four neurons per hidden layer to approximate the mean profile, and a DNN of four hidden layers with 32 neurons per hidden layer to model the modes. The *tanh* function is selected as the default activation function since it is second order differentiable, so that the residual is well-posed. Then, a DNN for the residual can be constructed via auto-differentiation and arithmetic operations. The training set $\mathcal{S}_t$ is

$$\mathcal{S}_t = \left\{ \{(-1,0),(1,0)\}, \{(x_f^{(i)},0)\}_{i=1}^{N_f} \right\}_{s=1}^{N},$$

where the u data is collected only at the boundaries to provide boundary conditions. The loss function is slightly modified based on Equation (20) to add a l_2 regularization term. At the training stage, we choose the l_2 regularization rate $\lambda = 0.001$, and use the Adam [57] optimizer with learning rate 0.001 to train our model for 20000 epochs. Fig. 6 shows the predicted mean and standard deviation of the solution u versus the reference. Fig. 7 shows our DNN prediction of three modes of u where the reference modes are calculated by Equation (17). We can see that the NN-aPC method makes accurate predictions of the mean and standard deviation of the solution $u(x;\omega)$ and learns the arbitrary polynomial chaos modes.

The trained model is then used to predict the QoI at any location x_o given new snapshots of sensor data, with only one forward evaluation of the DNN with x_o as the input, as described in Section 3.2.4. Fig. 8(a) illustrates the prediction of solution for three different snapshots of the sensor data in the test samples; our prediction recovers the true solution very well. In Fig. 8(b), we use an increasing value of N_f to study the effect of the number of f-sensors on the accuracy of the predicted solutions; we observe that when more f-sensors are deployed, we can achieve better accuracy. This demonstrates that the f-sensors help with the training of u-networks by bringing into the information of the governing equation via the NN-aPC method.

4.1.2. Inverse problem: stochastic elliptic equation

We solve the one-dimensional stochastic elliptic equation as an inverse problem, where we have some extra information on the solution $u(x;\omega)$ but incomplete information of the diffusion coefficient $k(x;\omega)$. The equation reads

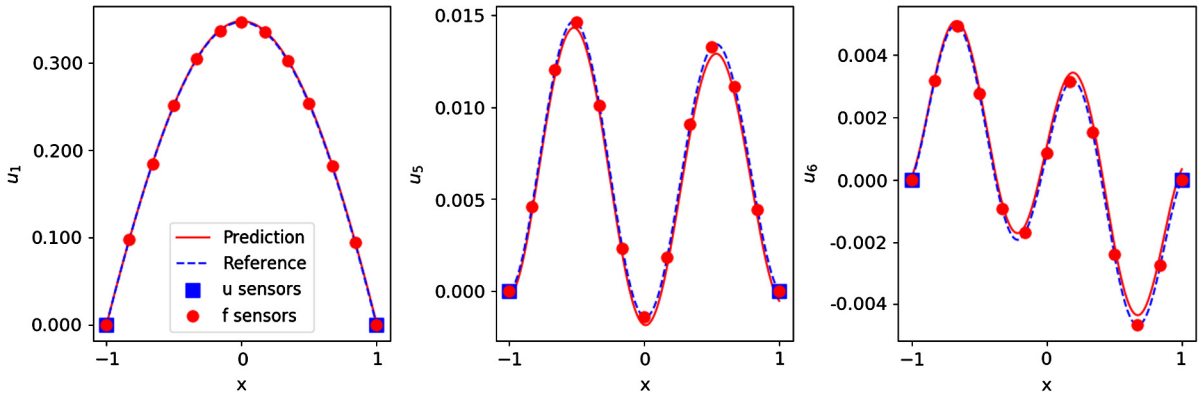

Fig. 7. Forward problem: Plots of three (out of six) aPC modes of u versus the reference, which is calculated from Equation (17) using the 1000 continuous u samples. The predicted modes match the true modes closely.

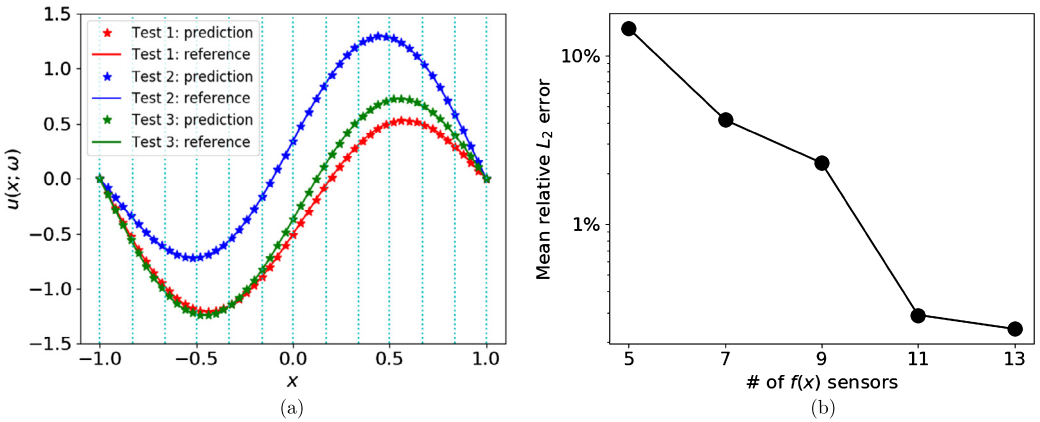

Fig. 8. Making predictions for the forward problem using the NN-aPC: (a) For three different snapshots in the test data set, we compare the predicted solution u, calculated using the measurements of 13 f-sensors whose locations are marked with the dashed lines, versus the true u. (b) Relative L_2 error of the predicted u, averaged for all snapshots in the test data set, versus the number of f-sensors placed in the domain.

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(k(x;\omega)\frac{\mathrm{d}}{\mathrm{d}x}u\right) = f(x), \quad x \in [-1,1] \text{ and } \omega \in \Omega, \\ u(-1) = u(1) = 0. \tag{29}$$

In this example, we use a constant forcing term $f(x) = 10$. The randomness comes from the diffusion coefficient $k(x;\omega)$, for which we only have limited information at the locations where we place the k-sensors. Here in this example, k is modeled and sampled from a non-Gaussian random process such that

$$\log(k(x;\omega)) \sim \mathcal{GP}(k_0(x), \mathrm{Cov}(x,x')), \tag{30}$$

where the mean $k_0(x) = \sin(3\pi x/2)/5$ and the covariance function has the same form as in Equation (28), where we set the standard deviation $\sigma = 0.1$ and correlation length $l_c = 1.0$. We use the same strategy as in Section 4.1.1 to generate the training and testing samples, and the training set is constructed as in Equation (6).

For this case, two groups of DNNs, i.e., $\widehat{k_i}$ and $\widehat{u_\alpha}$ are built to calculate the modes for k and u. Again, we use the Adam optimizer with learning rate 0.001 to train the DNNs for 50000 epochs. In Fig. 9(a), we use the 1st-order aPC expansion and we study the impact of using different l_2 regularization rate λ and different shapes of DNNs. We only change the DNNs that learn the stochastic modes, while the DNNs that learn the mean profiles are fixed to have two hidden layers and four neurons per hidden layer; this is the default setting for the future examples as well. The results indicate that a moderate choice of λ (0.0005) gives us the most accurate predictions, since a too small/large choice of λ causes over-/under-fitting. A suitable choice of DNN shape (four hidden layers with 32 neurons per hidden layer) produces the best trained model. For the rest of this numerical example, we shall adopt these optimal DNN setting.

In Fig. 9(b), we use the 1st-order aPC expansion and compare the averaged relative L_2 error in predictions when different numbers of k- and u-sensors are deployed to collect the training data. In general, the proposed method makes more accurate predictions after training with data collected from more k- and u-sensors. One reason is that a larger number of sensors supports a greater variety of input x in the training data, thus feeding more information to the model to reduce

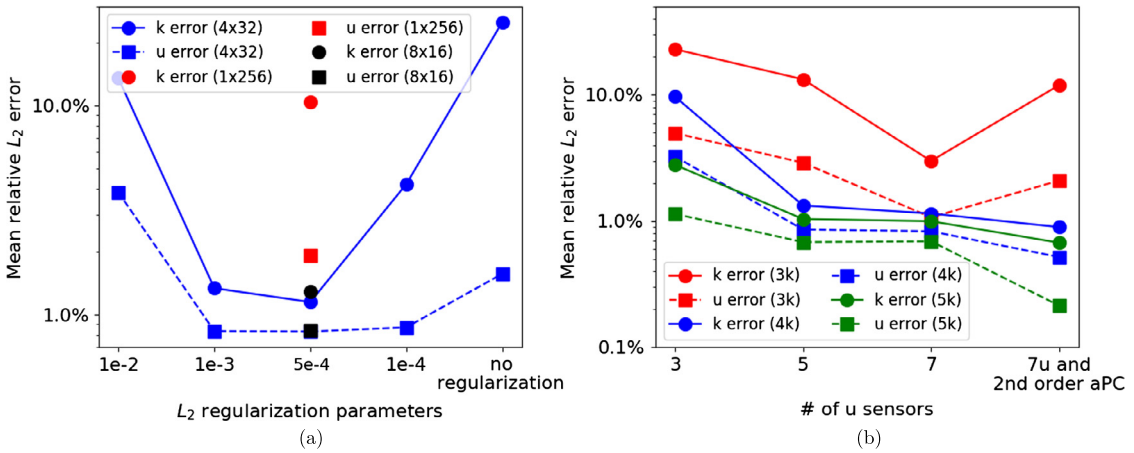

Fig. 9. Comparing prediction accuracy for the inverse problem using the NN-aPC: (a) The mean of relative L_2 error in predicting u and k trajectories in the test set when using different sizes of DNNs and different l_2 regularization rate. In this case, we use 1st-order aPC expansion four k-sensors and seven u-sensors are deployed and both DNNs have the same size. (b) The mean relative L_2 error in predicting u- and k-trajectories versus the number of sensors deployed. In this case we use the 1st-order aPC expansion, choose $\lambda = 0.0005$, and the DNNs have four hidden layers with 32 neurons per hidden layer.

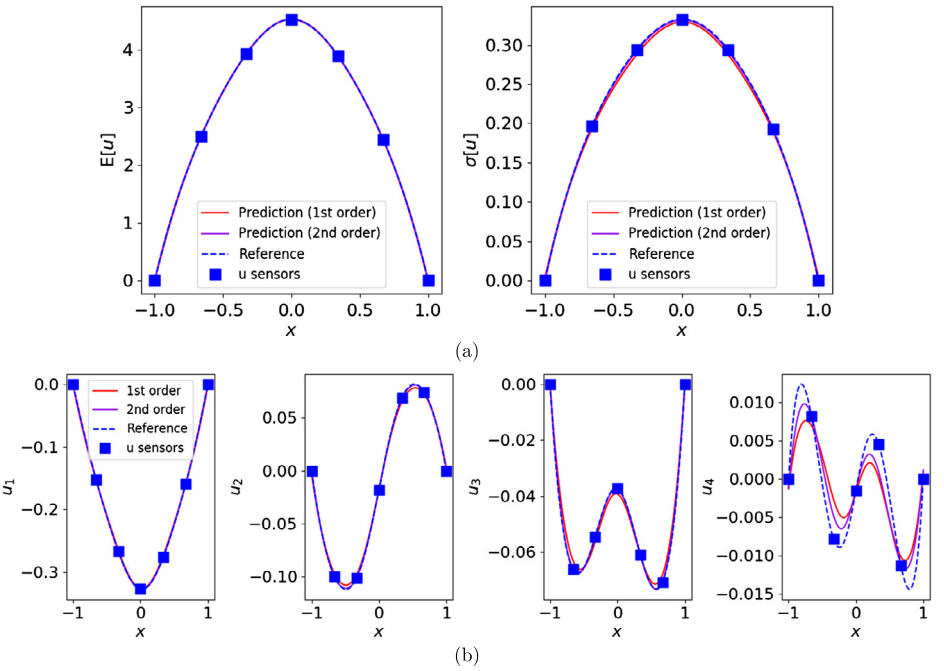

Fig. 10. Testing 2nd-order aPC expansions: (a) The predicted mean/standard deviation of u calculated with a 1st- and 2nd-order aPC expansions versus the reference. (b) The first four modes of u calculated by the 1st- and 2nd-order aPC expansion. The reference solutions in all plots are calculated with the continuous trajectories that produced the training data. The results are generated with seven u-sensors (blue squares) and four k-sensors (red dots in Fig. 11).

the probability of over-fitting. Also, more k-sensors allows for a higher effective random dimension of the aPC expansion for better approximation. Fig. 9(b) also shows that a 2nd-order aPC expansion helps to improve predictions. We note that this is not the case when we use only three k-sensors. The bottleneck here is insufficient random dimension, so without enough training information, adopting the 2nd-order aPC expansion doubles the number of $\widehat{u_\alpha}$ net outputs and would only increase the risk of over-fitting.

We use a combination of seven u-sensors and four k-sensors. Fig. 10(a) and 11(a) compare the mean and standard deviation of u and k calculated by the trained DNNs when we use the 1st- and 2nd-order aPC expansions. Fig. 10(b) shows the first four common aPC modes of u for both expansions, where we can see that the 2nd-order aPC expansion helps to improve the accuracy of the predicted lower-order modes. The 2nd-order aPC expansion would also help with learning the modes of k, as depicted in Fig. 11(b), even if the $\widehat{k_i}$ network is disjoint with the $\widehat{u_\alpha}$ network. Table 1 lists the relative L_2 error of the trained model in calculating the mean, standard deviation and all the common modes of k and u. We can

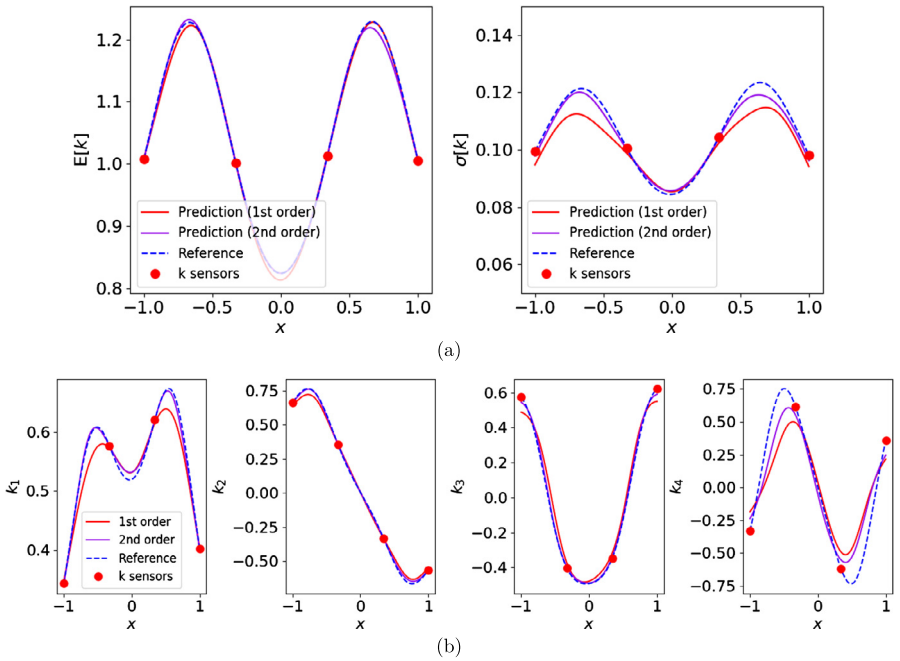

Fig. 11. Testing 2nd-order aPC expansions: (a) The predicted mean/standard deviation of k calculated with a 1st- and 2nd-order aPC expansions versus the reference. (b) The first 4 modes of k calculated by the 1st- and 2nd-order aPC expansion. The references in all plots are calculated with the continuous trajectories that produced the training data. The results are generated with the same setup of sensors as that of Fig. 10. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 1
Comparing the relative L_2 error when using the 1st- and 2nd-order aPC expansion, and using data from four k-sensors and seven u-sensors for training.

		mean	std	mode 1	mode 2	mode 3	mode 4
k	1st-order	0.54%	5.26%	4.15%	5.39%	12.03%	42.81%
	2nd-order	0.45%	1.87%	1.28%	1.95%	3.67%	29.95%
u	1st-order	0.14%	1.83%	0.98%	3.60%	4.34%	45.56%
	2nd-order	0.14%	0.51%	0.08%	0.80%	1.04%	32.16%

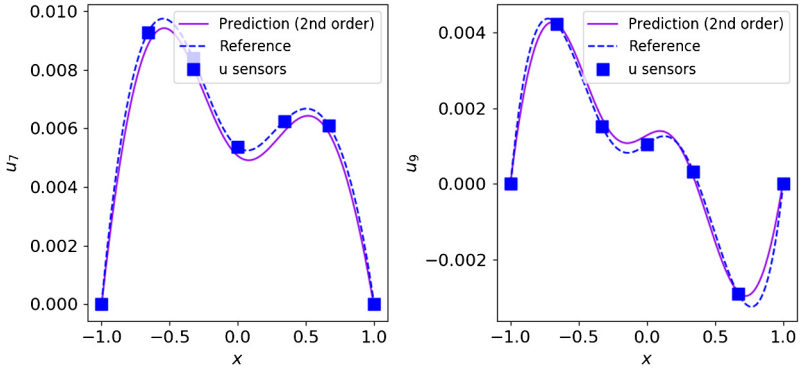

Fig. 12. Testing 2nd-order aPC expansions: Two higher order modes of u calculated with the 2nd-order aPC expansion. The magnitude of the higher-order modes is smaller compared to the lower-order modes, but the NN-aPC method is still able to capture the small magnitude modes. The results are generated with the same setup of sensors as that of Fig. 10.

see that using a 2nd-order aPC significantly improves the accuracy. We also note that in Fig. 11(a), due to the placement of k-sensors, the measurements from each sensor will yield almost the same mean (1.0) and standard deviation (0.1), but the trained k net would reveal the non-trivial wavy structure of its mean and standard deviation in the entire domain, containing information more than just the training data could provide. This indicates that there is information fusion of all three types of training data and the stochastic differential equation, rather than a simple interpolation. Fig. 12 shows that

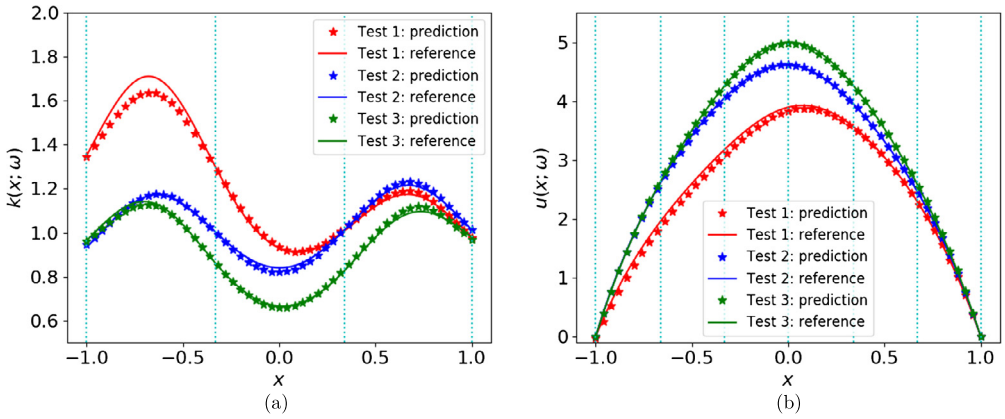

Fig. 13. Making predictions for the inverse problem using the NN-aPC: For three different snapshots in the test data set, we compare the predicted solution k (in plot (a)) and u (in plot (b)), calculated using the measurements of seven u-sensors and four k-sensors (denoted by the dashed lines).

the higher-order aPC modes can also be captured even if they exist in a much smaller scale compared to the lower-order more energetic modes.

Fig. 13 shows the prediction of k and u for three arbitrary snapshots in the test sample set. Every pair of predictions are made based on a single time measurement from 4 k- and 7 u-sensors, and they agree with the true reference value. Again, the predicting stage takes little computational time since only one forward evaluation of the well-trained DNNs is needed for every input x, no matter how many snapshots of predictions are to be made.

4.1.3. Inverse problem: stochastic diffusion-reaction equation

As another demonstration, we solve the time-dependent stochastic diffusion-reaction equation with a nonlinear reaction term, i.e.,

$$\begin{aligned} &\frac{\mathrm{d}u}{\mathrm{d}t} = \frac{\mathrm{d}^2u}{\mathrm{d}x^2} - \frac{\mathrm{d}u}{\mathrm{d}x} - \frac{u^2}{2} + k(x;\omega), \\ &\text{BC: } u(-1,t) = u(1,t) = 0, \quad \text{IC: } u(x,0) = 1 - x^2. \end{aligned} \tag{31}$$

Here, $u(x,t;\omega)$ is the solution to this reaction-diffusion equation, where $x \in [-1,1]$, $t \in [0,1]$ and $\omega \in \Omega$. Again, $\log k(x;\omega)$ is modeled as a Gaussian random process as in the previous example, but with a larger standard deviation $\sigma = 0.5$. We place four k-sensors and seven u-sensors equidistantly in the spatial domain and try to infer the mean and the modes of $k(x;\omega)$. When collecting training data, compared to the previous case, the only difference is that we measure u at three different times: $t_1 = 0.125$, $t_2 = 0.5$ and $t_3 = 0.875$, generating three groups of data, each of which contains 1000 snapshots of u measurements. The DNN for the modes of u, $\widehat{u_\alpha}$, takes both the spatial coordinate x and the temporal coordinates t as the input. The contribution of u to the loss function is the MSE of all u measurements at the three different times, i.e.,

$$MSE_u = \frac{1}{3NN_u} \sum_{l=1}^{3} \sum_{s=1}^{N} \sum_{i=1}^{N_u} \left[\left(\tilde{u}(x_u^{(i)}, t_l; \omega_s) - u(x_u^{(i)}, t_l; \omega_s) \right)^2 \right]. \tag{32}$$

We adopt a second order aPC expansion for $u(x,t;\omega)$, and use a DNN with three hidden layers (16 neurons per hidden layer) to approximate the mean of u, a DNN with three hidden layers (32 neurons per hidden layer) to approximate the first order modes of u, and a DNN with three hidden layers (64 neurons per hidden layer) to approximate the second order modes of u. The DNNs used to approximate the mean and modes of k are the same as in the previous example. The neural networks are trained with the Adam optimizer for 500000 steps. Fig. 14 shows the mean and standard deviation of solution $u(x,t;\omega)$ at five different times, and Fig. 15 shows the mean, standard deviation and the four modes of the random forcing $k(x;\omega)$. The NN-aPC method solves the time-dependent nonlinear inverse problem and makes accurate predictions for the mean functions of both u and k, and the standard deviations of u. The prediction of the standard deviation of k captures the overall shape but is less accurate compared to that of the previous example. Note that for nonlinear problems, the stochastic structure evolves with time and thus a finite set of stationary bases, e.g., the aPC bases, are not the optimal choice for time-dependent nonlinear stochastic problems. To gain more accurate results, we need to make use of non-stationary expansions such as the dynamically orthogonal [58] or the bi-orthogonal [59,60] decomposition which are beyond the scope of this paper and will be investigated in future research.

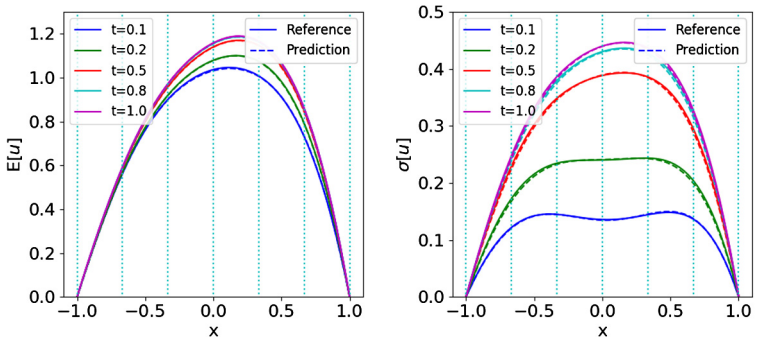

Fig. 14. Nonlinear diffusion-reaction equation: Mean and standard deviation of solution u at five different times. The vertical dashed lines indicate the location of u sensors.

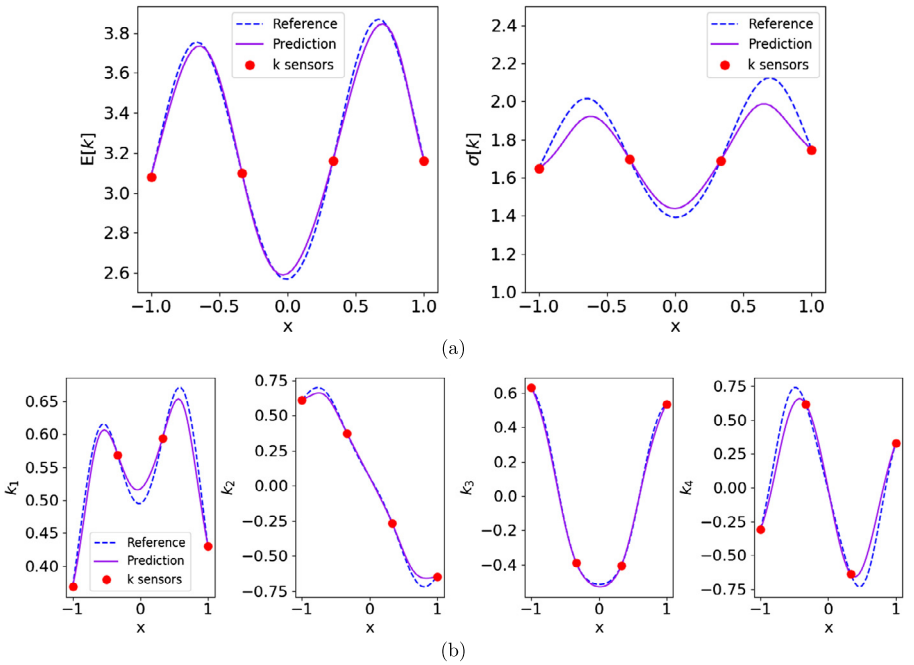

Fig. 15. Nonlinear diffusion-reaction equation: (a) The predicted mean/standard deviation of k versus the reference. (b) The first four modes of k. The reference solutions in all plots are calculated with the continuous trajectories that produced the training data. The results are generated with the same setup of sensors as that of Fig. 14.

4.2. Active learning using dropout uncertainty

In Appendix A, we demonstrate that dropout can reduce over-fitting in solving forward and inverse differential equations. In this part, we mainly show that the dropout-induced uncertainty serves as useful guidance for active learning.

4.2.1. Active learning for deterministic problem

As a pedagogical example, we first solve an inverse deterministic elliptic equation (the deterministic version of Equation (29)). To start with, we use five k-sensors and seven u-sensors, and suppose that we are provided with additional k-sensors to be placed in the domain. We choose $f(x) = 10$ and $k(x) = \exp(\sin(3\pi x/2)/5)$ as the hidden diffusion coefficient. Two separate DNNs are constructed: a small scale regular DNN that has two hidden layers with four neurons per hidden layer to model the function $u(x)$, and a large scale dropout neural network with six hidden layers and 100 neurons per hidden layer to model the function $k(x)$, with the dropout rate fixed at 0.01. As for the choice of dropout rate, we remark that a large dropout rate significantly suppresses the over-fitting issue but makes the DNNs hard to train, while in the physics-informed training process, over-fitting is usually not the main issue because the PDEs serve as a form of regularization. Here, we use the dropout strategy mainly to estimate the approximation uncertainty, not to suppress over-fitting, and therefore, we choose a small dropout rate 0.01. An l_2 regularization term with $\lambda = 10^{-6}$ is added to the loss function. At the predicting stage, we evaluate the dropout network for 10000 times to estimate the mean and standard deviation. Next, a new k-sensor is placed where the standard deviation reaches its maximum. If it happens that the location to add the new sensor is close

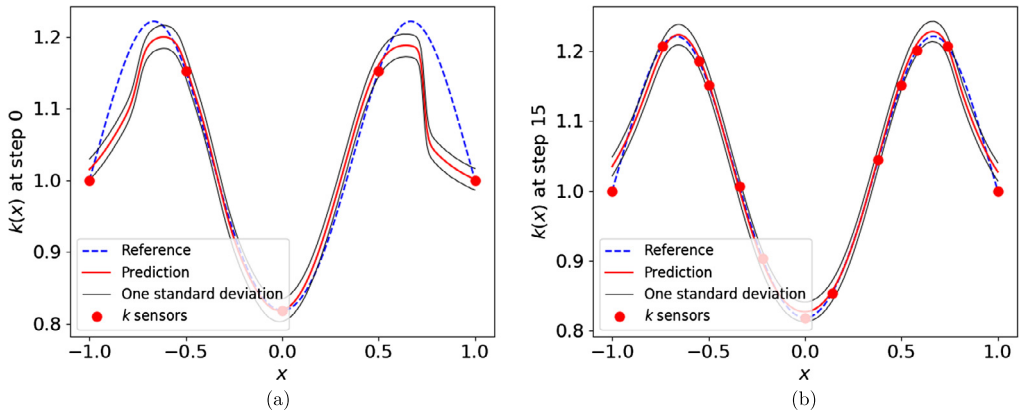

Fig. 16. Active learning for deterministic problem: (a) The prediction of k and its dropout-induced uncertainty of the starting step with 5 k-sensors; (b) The prediction of k and its dropout-induced uncertainty of the last step with 13 k-sensors.

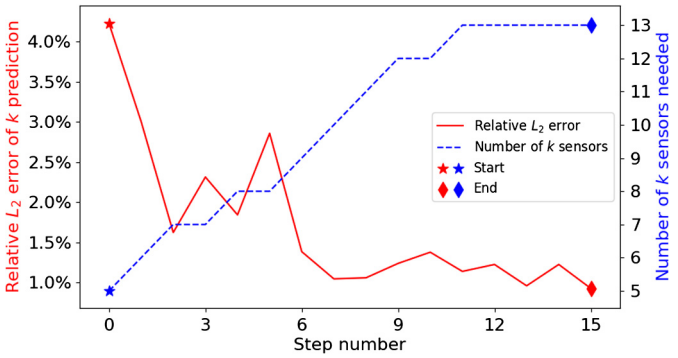

Fig. 17. Active learning for deterministic problem: The red solid line shows the relative L_2 error of the predicted k revealing a decaying trend. The blue dashed line shows the number of k-sensors deployed in each step.

to an existing k-sensor by a threshold distance of ρ, we do not add the new sensor, but instead, we count the nearest existing sensor twice as if we had added a virtual sensor at the same location of the existing one. In practice, we choose $\rho = 0.03$. Note that this may not be the most efficient way of adding new sensors, and the optimal way of adding sensors remains an open topic for the future research.

Fig. 16 shows the initial prediction of k and the prediction after iterating the above algorithm for 15 steps. In Fig. 16(b), the sensors are clustered where the curvature of k is big, which is consistent with our intuition that we should put more sensors where the function changes rapidly. In Fig. 17, we see that during these 15 iterations, only 8 new sensors are deployed while the relative error of k predictions reduces from more than 5% to less than 1%. So far, we have designed an automatic iterative procedure for active learning that reduces the cost of adding new sensors by efficiently re-using the old ones.

4.2.2. Active learning for stochastic inverse problem

We consider the inverse stochastic elliptic problem as described in Section 4.1.2 for active learning, but in this example, $\log(k(x;\omega))$ is modeled by a Gaussian random process with correlation length $l_c = 0.5$. At the beginning, we have 1000 snapshots of data from three k-sensors, seven u-sensors and 21 f-sensors that are equidistantly distributed in the physical domain, and our goal is to infer $k(x;\omega)$ in the entire domain. Suppose that we are then provided with additional sensors of k; we shall allocate them according to the uncertainty induced by the dropout neural networks. In practice, we learn the modes of $k(x;\omega)$ with a dropout neural network that has four hidden layers with 128 neurons per hidden layer, and a dropout rate 0.01. The solution $u(x;\omega)$ is expanded with the 1st-order aPC expansion, while the modes are modeled by a regular DNN with four layers and 32 neurons per hidden layer. As an inverse problem, our main goal is to identify $k(x,\omega)$ and then identify where we should add more k-sensors to enhance the accuracy of prediction. Therefore, we implement dropout only on $\widehat{k_i}$ to estimate the dropout uncertainty in predicting k_i.

We use an Adam optimizer with learning rate 5×10^{-4} to train the networks for 50000 epochs. The mean and standard deviation of the modes of k are evaluated from 10000 independent evaluations of the dropout neural network. The active learning is performed in the same manner as the previous example, based on the standard deviation of the first mode of k. This is because the first mode is associated with the largest eigenvalue and brings the largest impact to the stochastic structure, and thus, it is the most important to learn the first mode accurately. The network parameters are reset after each step.

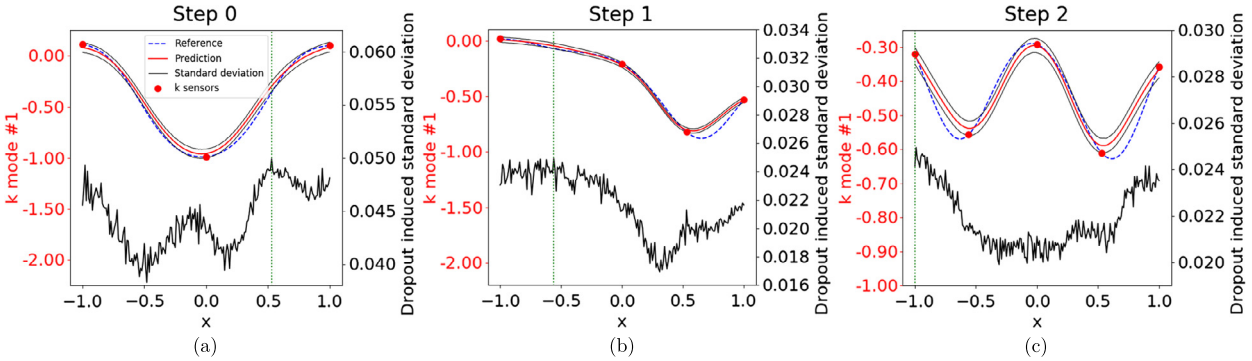

Fig. 18. Active learning for stochastic inverse problem: The first modes of k in the three steps and their associated standard deviation induced from the dropout neural network. The green dashed line indicates the location where the standard deviation reaches its maximum, and where the new sensor will be added in the next step.

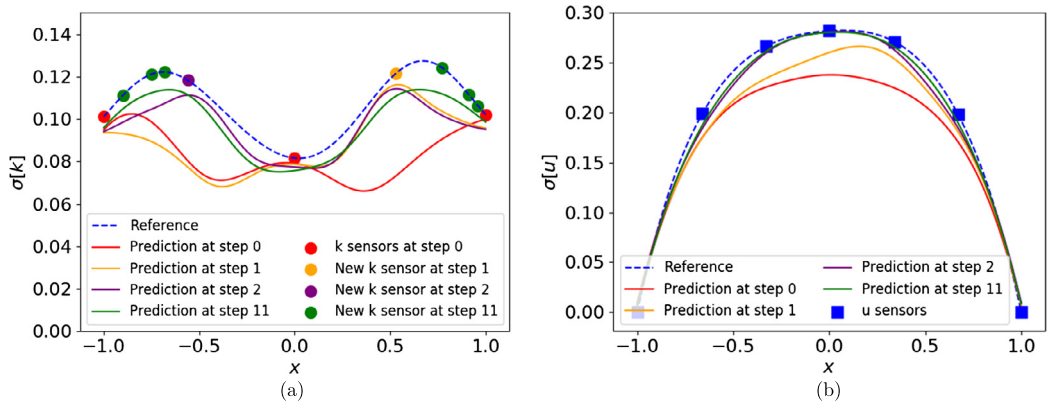

Fig. 19. Active learning for stochastic inverse problem: The predicted standard deviation of k (plot (a)) and u (plot (b)) in the first three and the last steps. In plot (a) the k-sensors at step 0 are colored in red and the newly added k-sensors are denoted in different colors. The u-sensors (as depicted in plot (b)) are kept the same. The reference solutions are calculated using the continuous trajectories that generate the training data.

Table 2
Active learning for stochastic inverse problem: Comparison of the relative L_2 error at different steps.

	k mean	k std	k prediction	u mean	u std	u prediction
Step 0	0.87%	26.57%	6.07%	0.18%	15.23%	3.06%
Step 1	2.79%	20.19%	5.29%	0.24%	9.33%	2.41%
Step 2	1.48%	10.21%	2.58%	0.14%	3.89%	1.08%
Step 11	0.46%	8.49%	2.01%	0.04%	2.93%	0.67%

We carry out the active learning steps until the k prediction error does not decrease after a new sensor is added. To better illustrate the process of adding new sensors, in Fig. 18(a), 18(b) and 18(c) we show the learned first mode of k and its associated standard deviation from dropout uncertainty. Three steps of active learning are displayed. Note that the shapes of the first modes do not stay the same due to the fact that every time a new k-sensor is added, the principal components of K in Equation (8) are therefore changing. Fig. 19(a) and Fig. 19(b) show the comparison of the predicted standard deviation of k and u in the first three steps and the last step, respectively. It is evident that adding extra k-sensors automatically according to the dropout-induced uncertainty will improve the accuracy of standard deviation prediction for both k and u. Finally, the trained model is used to predict continuous trajectories of k and u in the test samples. We can conclude from Table 2 that adding extra k-sensors based on active learning with the dropout helps us to make better predictions.

5. Summary

We have presented a new approach to quantify the parametric uncertainty in the physics-informed neural networks (PINNs) by employing the arbitrary polynomial chaos (aPC) expansion to represent the stochastic solution. We use the data collected from sensor measurements to build a set of arbitrary polynomial basis and learn the modal functions of the aPC expansion through the PINNs, i.e., DNNs that encode the underlying stochastic differential equation. The proposed data-driven method can be used to solve forward problems, but more importantly, it deals with stochastic *inverse* problems.

In the classical inverse problem, typically all the information available is for the solution, and we aim to identify the parameters. Here we selected to solve the inverse problems, where we have partial information available both for the solution and the parameter, which is a stochastic process. Once the model is trained with existing sensor data, i.e., historical data, it can be used to predict new instances of trajectories of the quantity of interest (solution or parameter) at very small additional computational cost.

We aim at quantifying two different types of uncertainties, i.e., the *parametric* uncertainty due to the stochastic equation, as well as the *approximation* uncertainty of the PINN. The latter represents how well the PINN is trained and how robust it is at the predicting stage. To this end, we adopt the dropout strategy for estimating the approximation uncertainty. Dropout is typically used to deal with over-fitting problems but it can also be exploited to quantify the approximation uncertainty at no additional cost. In our examples, we use DNNs to learn the modal functions of the stochastic parametric modes and the dropout strategy to quantify their associated uncertainty. Based on this, we propose an iterative method of actively learning where to place new sensors to enhance the approximation accuracy of PINNs. The numerical results exhibit the effectiveness of such an *active learning* strategy, not only in placing new sensors but also in making better use of the existing ones.

There are other possible methods of quantifying parametric and approximation uncertainties. For example, we can abandon aPC and use directly the stochastic data as the input. One possible approach is to consider the random space as the extension of the physical space. Hence, standard DNNs, and in particular the deterministic PINNs developed in [12,19] can be directly used to solve stochastic differential equations. Here we did not choose this approach for two reasons: first, it does not lead to explicit expressions for stochasticity of the quantity of interest (QoI); and second, different from sampling in the physical space where we can always mark the location by their coordinates, marking random instances with random variables is much harder. Another viable approach is the Gaussian process regression, as it can be used to construct data-efficient and physics-informed learning machines. However, the treatment using Gaussian process for nonlinear problems requires local linearization, and the Bayesian nature of the Gaussian process regression requires certain prior assumptions that may limit the representation capacity of the model and give rise to robustness/brittleness issues, as mentioned in [12] and [19].

The limitations of the proposed method are as follows: First, as with most spectral expansion based methods, the NN-aPC method suffers from the "curse of dimensionality" when dealing with problems with high-dimensional stochastic input, since it requires the high-dimensional DNN outputs, making the training process harder. A promising method that may be able to deal with the high-dimensional stochastic problems are the generative adversarial networks (GANs) [61,62] as it avoids the spectral decomposition and is targeted directly on the distribution of data. Second, the proposed method is not suitable for solving time-dependent problems with strong non-linearities, since the stationary aPC bases does not capture the dynamical evolution of the stochastic structure. In this situation, we may integrate PINNs with the dynamically orthogonal or bi-orthogonal decomposition. In addition, the dropout strategy for Bayesian neural networks (BNNs) could suffer from undefined or pathological behavior [63], while the application of other BNN strategies (e.g., variational inference, Markov Chain Monte Carlo) in PINNs is yet to be developed. The future work will involve a systematic comparison between the performance of various Bayesian neural network techniques, especially when applied in quantifying the approximation uncertainty in PINNs.

Acknowledgement

This work is supported by DARPA N66001-15-2-4055, Air Force FA9550-17-1-0013, Army Research Laboratory W911NF-12-2-0023 and NSF of China (No. 11671265). In addition, we would like to thank Liu Yang for his generous advice.

Appendix A. Reducing over-fitting via dropout in solving PDEs

To show how dropout reduces over-fitting, we implement the dropout neural networks to solve both a forward Poisson's equation:

$$-\frac{\mathrm{d}^2}{\mathrm{d}x^2}u = f(x), \quad x \in [-1,1], \quad u(-1) = u(1) = 0, \tag{A.1}$$

and an inverse elliptic equation:

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(k(x)\frac{\mathrm{d}}{\mathrm{d}x}u\right) = f(x), \quad x \in [-1,1], \quad u(-1) = u(1) = 0. \tag{A.2}$$

For the forward Poisson's equation, we choose $f(x) = 9\pi^2 \sin(3\pi x/2)/4$ as the forcing term. A dropout neural network with six hidden layers and 100 neurons per hidden layer is constructed to model the solution $u(x)$. The training data consists of two u measurements at both boundaries and six f-sensors in the domain. For the inverse elliptic equation, we choose, as before, $f(x) = 10$ and $k(x) = \exp(\sin(3\pi x/2)/5)$ as the hidden diffusion coefficient, and we use five k-sensors and seven u-sensors. Two separate DNNs are constructed: a small scale regular DNN that has two hidden layers with four neurons per

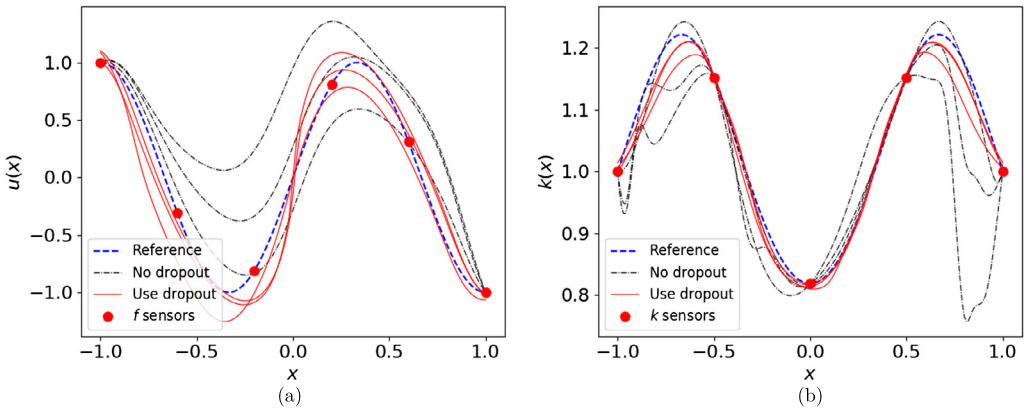

Fig. A.20. Dropout to reduce over-fitting: A comparison of the predicted QoI using the PINN/dropout and the regular PINN. For each case, three independent runs are conducted. (a) Forward problem: we solve the Poisson's equation with 6 f-sensors (red dots) and 2 u-sensors (at the domain boundary, not shown in the plot). (b) Inverse problem: we solve an elliptic equation with 5 k-sensors (red dots) and 7 u-sensors (equidistantly placed in the domain, not shown in the plot).

hidden layer to model the function $u(x)$, and a large scale dropout neural network with six hidden layers and 100 neurons per hidden layer to model the function $k(x)$. The dropout rate in both examples is fixed at 0.01.

For both the forward and inverse problem, we train the DNNs as described in Section 3.3, using an Adam optimizer with learning rate 0.001 for 30000 epochs. Due to the lack of sufficient number of sensors, over-fitting could occur at the training stage. Fig. A.20 shows a comparison of the results when we train the networks with and without using dropout. As we can see in the plots, the results from a regular DNN are very different from each other, showing irregular jumps of large amplitudes, while the results from dropout DNNs are similar to each other, and they are closer to the truth. This shows that dropout works as an effective means of reducing over-fitting.

References

[1] X. Fan, Y. Liu, J. Tao, Y. Weng, Soil salinity retrieval from advanced multi-spectral sensor with partial least square regression, Remote Sens. 7 (2015) 488–511.
[2] H.N. Pollack, S.J. Hurter, J.R. Johnson, Heat flow from the Earth's interior: analysis of the global data set, Rev. Geophys. 31 (1993) 267–280.
[3] T. Graepel, Solving noisy linear operator equations by Gaussian processes: application to ordinary and partial differential equations, in: International Conference on Machine Learning, 2003, pp. 234–241.
[4] S. Särkkä, Linear operators and stochastic partial differential equations in Gaussian process regression, in: International Conference on Artificial Neural Networks, Springer, 2011, pp. 151–158.
[5] I. Bilionis, Probabilistic solvers for partial differential equations, preprint, arXiv:1607.03526, 2016.
[6] M. Raissi, P. Perdikaris, G.E. Karniadakis, Numerical Gaussian processes for time-dependent and nonlinear partial differential equations, SIAM J. Sci. Comput. 40 (2018) A172–A198.
[7] G. Pang, L. Yang, G.E. Karniadakis, Neural-net-induced Gaussian process regression for function approximation and PDE solution, preprint, arXiv:1806.11187, 2018.
[8] X. Yang, G. Tartakovsky, A. Tartakovsky, Physics-informed Kriging: a physics-informed Gaussian process regression method for data-model convergence, preprint, arXiv:1809.03461, 2018.
[9] I.E. Lagaris, A.C. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (1998) 987–1000.
[10] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (2000) 1041–1049.
[11] Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks, preprint, arXiv:1707.03351, 2017.
[12] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, preprint, arXiv:1711.10561, 2017.
[13] M.A. Nabian, H. Meidani, A deep neural network surrogate for high-dimensional random partial differential equations, preprint, arXiv:1806.02957, 2018.
[14] A.M. Stuart, Inverse problems: a Bayesian perspective, Acta Numer. 19 (2010) 451–559.
[15] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.
[16] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. 348 (2017) 683–693.
[17] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (2017) e1602614.
[18] S. Rudy, A. Alla, S.L. Brunton, J.N. Kutz, Data-driven identification of parametric partial differential equations, preprint, arXiv:1806.00732, 2018.
[19] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, preprint, arXiv:1711.10566, 2017.
[20] M. Raissi, G. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[21] A.M. Tartakovsky, C.O. Marrero, P. Perdikaris, G.D. Tartakovsky, D. Barajas-Solano, Learning parameters and constitutive relationships with physics informed deep neural networks, preprint, arXiv:1808.03398v2, 2018.
[22] W. E, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, preprint, arXiv:1706.04702, 2017.

[23] M. Raissi, Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations, preprint, arXiv:1804.07010, 2018.
[24] R. Ghanem, P.D. Spanos, Polynomial chaos in stochastic finite elements, J. Appl. Mech. 57 (1990) 197–202.
[25] D. Xiu, G.E. Karniadakis, The Wiener-Askey polynomial chaos for stochastic differential equations, SIAM J. Sci. Comput. 24 (2002) 619–644.
[26] M. Zheng, X. Wan, G.E. Karniadakis, Adaptive multi-element polynomial chaos with discrete measure: algorithms and application to SPDEs, Appl. Numer. Math. 90 (2015) 91–110.
[27] X. Wan, G. Karniadakis, Multi-element generalized polynomial chaos for arbitrary probability measures, SIAM J. Sci. Comput. 28 (2006) 901–928.
[28] S. Oladyshkin, W. Nowak, Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion, Reliab. Eng. Syst. Saf. 106 (2012) 179–190.
[29] H. Lei, J. Li, P. Gao, P. Stinis, N. Baker, Data-driven approach of quantifying uncertainty in complex systems with arbitrary randomness, preprint, arXiv:1804.08609, 2018.
[30] J.A. Witteveen, H. Bijl, Modeling arbitrary uncertainties using Gram-Schmidt polynomial chaos, in: 44th AIAA Aerospace Sciences Meeting and Exhibit, 2006, p. 896.
[31] D.J. MacKay, A practical Bayesian framework for backpropagation networks, Neural Comput. 4 (1992) 448–472.
[32] R.M. Neal, Bayesian Learning for Neural Networks, Lecture Notes in Statistics, vol. 118, Springer, 1996.
[33] M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, L.K. Saul, An introduction to variational methods for graphical models, in: Learning in Graphical Models, Springer, 1998, pp. 105–161.
[34] J. Paisley, D. Blei, M. Jordan, Variational Bayesian inference with stochastic search, preprint, arXiv:1206.6430, 2012.
[35] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, preprint, arXiv:1312.6114, 2013.
[36] M.D. Hoffman, D.M. Blei, C. Wang, J. Paisley, Stochastic variational inference, J. Mach. Learn. Res. 14 (2013) 1303–1347.
[37] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, preprint, arXiv:1401.4082, 2014.
[38] M. Titsias, M. Lázaro-Gredilla, Doubly stochastic variational Bayes for non-conjugate inference, in: International Conference on Machine Learning, 2014, pp. 1971–1979.
[39] D. Madigan, J. York, D. Allard, Bayesian graphical models for discrete data, Int. Stat. Rev. (1995) 215–232.
[40] D. Koller, N. Friedman, F. Bach, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[41] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, preprint, arXiv:1505.05424, 2015.
[42] J.T. Springenberg, A. Klein, S. Falkner, F. Hutter, Bayesian optimization with robust Bayesian neural networks, in: Advances in Neural Information Processing Systems, pp. 4134–4142.
[43] C. Su, M.E. Borsuk, Improving structure MCMC for Bayesian networks through Markov blanket resampling, J. Mach. Learn. Res. 17 (2016) 1–20.
[44] I. Rivals, L. Personnaz, Construction of confidence intervals for neural networks based on least squares estimation, Neural Netw. 13 (2000) 463–484.
[45] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: representing model uncertainty in deep learning, in: International Conference on Machine Learning, 2016, pp. 1050–1059.
[46] J.H. Yarin Gal, A. Kendall, Concrete dropout, in: Advances in Neural Information Processing Systems, 2017, pp. 3584–3593.
[47] Y. Li, Y. Gal, Dropout inference in Bayesian neural networks with alpha-divergences, in: International Conference on Machine Learning, 2017, pp. 2052–2061.
[48] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, preprint, arXiv:1207.0580, 2012.
[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[50] A. Damianou, N. Lawrence, Deep Gaussian processes, in: Artificial Intelligence and Statistics, 2013, pp. 207–215.
[51] Y. Gal, Z. Ghahramani, A theoretically grounded application of dropout in recurrent neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 1019–1027.
[52] A. Kendall, V. Badrinarayanan, R. Cipolla, Bayesian SegNet: model uncertainty in deep convolutional encoder-decoder architectures for scene understanding, preprint, arXiv:1511.02680, 2015.
[53] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision? in: Advances in Neural Information Processing Systems, 2017, pp. 5580–5590.
[54] C. Angermueller, H.J. Lee, W. Reik, O. Stegle, DeepCpG: accurate prediction of single-cell DNA methylation states using deep learning, Genome Biol. 18 (2017) 67.
[55] X. Yang, R. Kwitt, M. Niethammer, Fast predictive image registration, in: Deep Learning and Data Labeling for Medical Applications, Springer, 2016, pp. 48–57.
[56] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: a system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation, 2016, pp. 265–283.
[57] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, preprint, arXiv:1412.6980, 2014.
[58] T.P. Sapsis, P. Lermusiaux, Dynamically orthogonal field equations for continuous stochastic dynamical systems, Physica D 238 (2009) 2347–2360.
[59] M. Cheng, T.Y. Hou, Z. Zhang, A dynamically bi-orthogonal method for time-dependent stochastic partial differential equations i: derivation and algorithms, J. Comput. Phys. 242 (2013) 843–868.
[60] M. Cheng, T.Y. Hou, Z. Zhang, A dynamically bi-orthogonal method for time-dependent stochastic partial differential equations II: adaptivity and generalizations, J. Comput. Phys. 242 (2013) 753–776.
[61] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.
[62] L. Yang, D. Zhang, G.E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, preprint, arXiv:1811.02033, 2018.
[63] J. Hron, A.G.d.G. Matthews, Z. Ghahramani, Variational Bayesian dropout: pitfalls and fixes, preprint, arXiv:1807.01969, 2018.