SIAM J. SCI. COMPUT.
Vol. 42, No. 1, pp. A292–A317

PHYSICS-INFORMED GENERATIVE ADVERSARIAL NETWORKS FOR STOCHASTIC DIFFERENTIAL EQUATIONS[*]

LIU YANG[†], DONGKUN ZHANG[†], AND GEORGE EM KARNIADAKIS[‡]

Abstract. We developed a new class of physics-informed generative adversarial networks (PI-GANs) to solve forward, inverse, and mixed stochastic problems in a unified manner based on a limited number of scattered measurements. Unlike standard GANs relying solely on data for training, here we encoded into the architecture of GANs the governing physical laws in the form of stochastic differential equations (SDEs) using automatic differentiation. In particular, we applied Wasserstein GANs with gradient penalty (WGAN-GP) for its enhanced stability compared to vanilla GANs. We first tested WGAN-GP in approximating Gaussian processes of different correlation lengths based on data realizations collected from simultaneous reads at sparsely placed sensors. We obtained good approximation of the generated stochastic processes to the target ones even if there is a mismatch between the input noise dimensionality and the effective dimensionality of the target stochastic processes. We also studied the overfitting issue for both the discriminator and the generator, and we found that overfitting occurs also in the generator in addition to the discriminator as previously reported. Subsequently, we considered the solution of elliptic SDEs requiring approximations of three stochastic processes, namely the solution, the forcing, and the diffusion coefficient. Here again, we assumed data collected from simultaneous reads at a limited number of sensors for the multiple stochastic processes. Three generators were used for the PI-GANs: two of them were feed forward deep neural networks (DNNs), while the other one was the neural network induced by the SDE. For the case where we have one group of data, we employed one feed forward DNN as the discriminator, while for the case of multiple groups of data we employed multiple discriminators in PI-GANs. We solved forward, inverse, and mixed problems without changing the framework of PI-GANs, obtaining both the means and the standard deviations of the stochastic solution and the diffusion coefficient in good agreement with benchmarks. In this work, we have demonstrated the effectiveness of PI-GANs in solving SDEs for about 120 dimensions. In principle, PI-GANs could tackle very high dimensional problems given more sensor data with low-polynomial growth in computational cost.

Key words. WGAN-GP, multiplayer GANs, high-dimensional problems, inverse problems, elliptic stochastic problems

AMS subject classifications. 60H35, 34F05, 62M45

DOI. 10.1137/18M1225409

1. Introduction.

Generative adversarial networks (GANs) have achieved remarkable success within a short time for diverse tasks of generating synthetic data, such as images [3, 14, 19, 41], texts [38, 40, 7, 20], and even music [38, 22, 36, 11]. GANs can learn probability distributions from data, an attribute suggestive of its potential application to modeling the *inherent stochasticity* and *extrinsic uncertainty* in physical and biological systems. However, to the best of our knowledge, there is no work explicitly encoding the known physical laws into the framework of GANs so far in the spirit of physics-informed neural networks first introduced in [29, 30].

The specific data-driven approach to modeling physical and biological systems

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section November 7, 2018; accepted for publication (in revised form) November 13, 2019; published electronically February 4, 2020.
https://doi.org/10.1137/18M1225409
Funding: This work was funded by ARO grant W911NF-18-1-0301 and DOE grants DE-SC0019434 and DE-SC0019453.
[†]Division of Applied Mathematics, Brown University, Providence, RI 02912 (liu_yang@brown.edu, dongkun_zhang@brown.edu).
[‡]Corresponding author. Division of Applied Mathematics, Brown University, Providence, RI 02912 (george_karniadakis@brown.edu).

depends crucially on the amount of data available as well as on the complexity of the system itself, as illustrated in Figure 1. The classical paradigm for which many different numerical methods have been developed over the last 50 years is shown in the top plot of Figure 1, where we assume that the only data available are the boundary and initial conditions, while the specific governing partial differential equation (PDE) and its associate parameters are precisely known. On the other extreme (lower plot), we may have a lot of data, e.g., in the form of time series, but we may not know the governing physical law, e.g., the underlying PDE, at the continuum level; many problems in social dynamics fall under this category, although work so far has focused on recovering known PDEs from data only; e.g., see [33, 5, 27]. Perhaps the most interesting category is sketched in the middle plot, where we assume that we know the physics partially; e.g., in an advection-diffusion-reaction system, the reaction terms may be unknown, but we have several scattered measurements in addition to the boundary and initial conditions that we can use to infer the missing functional terms and other parameters in the PDE and simultaneously recover the solution. It is clear that this middle category is the most general case, and in fact it is representative of the other two categories, if the measurements are too few or too many. This is the *mixed* case that we address in this paper but with the significantly more complex scenario, where the solution is a stochastic process due to stochastic excitation or an uncertain material property, e.g., permeability or diffusivity in a porous medium, or uncertainty in the initial and boundary conditions. Hence, we employ stochastic differential equations (SDEs) to govern these stochastic solutions and other stochastic fields.

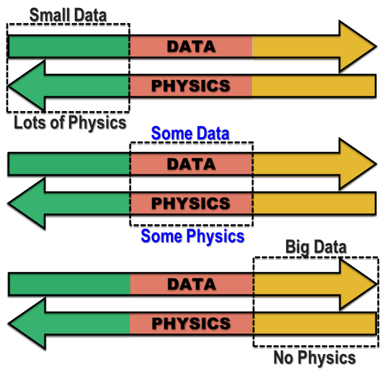


FIG. 1. *Schematic to illustrate three possible categories of physical problems and the associated available data. We use the term "PHYSICS" to imply the known physics for the target problem.*

Taking inspiration from the work on physics-informed neural networks for deterministic PDEs [29, 30], here we employ GANs for stochastic problems in the second category of Figure 1. We wish to encode the known physics, more specifically the SDEs, into the architecture of GANs, while at the same time exploiting the feature of GANs to model and learn the unknown stochastic terms in the equations from data. This approach represents a seamless integration of mathematical models and data for system inference/identification as well as mixed problems. This is an emerging paradigm in machine learning research addressing engineering applications, where typically the physics is very complex and we only have partial measurements of forcing or material properties. To this end, recent advances include using Gaussian process regression [10, 32, 4, 31, 24, 37] and deep neural networks (DNNs) [17, 18, 15, 29, 23, 34] to solve forward problems as well as Bayesian estimation [35] and DNNs [28, 42] for inverse problems. However, to the best of our knowledge, published works are mostly dealing with deterministic systems. There have only been very few works published on data-driven methods for SDEs, e.g., [6, 26] for forward problems, and they deal with parabolic PDEs, backward SDEs, or forward-backward SDEs, which are different from the problems we are going to deal with in this paper. For the inverse problem, Zhang et al. [39] have recently proposed a DNN based method that learns the modal functions of the quantity of interest, which could also be the unknown system parameters. While robust, this method suffers from the "curse of dimensionality," in that the number of polynomial chaos expansion terms grows exponentially as the effective dimension increases, leading to prohibitive computational costs for modeling stochastic systems in high dimensions. In this paper, we start from a new perspective and propose the physics-informed GANs (PI-GANs) to solve SDEs. Our method is flexible, in that without changing the general framework, it is capable of solving a wide range of problems, from forward problems to inverse problems and in between, i.e., mixed problems. Moreover, PI-GANs learn probability distributions from data and do not suffer from the curse of dimensionality. As a result, we can tackle SDEs involving stochastic processes with high effective dimensions. In addition, PI-GANs can make use of data collected from multiple groups consisting of multiple sensors with no alignment between data in the groups.

The organization of this paper is as follows. In section 2, we set up the data-driven problems. In section 3, we give a brief review of GANs and the specific version we use, namely Wasserstein GANs with gradient penalty (WGAN-GP). Our main algorithms are introduced in section 4, followed by a detailed study of the performance of our methods in section 5, where we first consider the learning of stochastic processes from a limited number of realizations and discuss the currently understudied issue of overfitting in WGAN-GP. Subsequently, we present solutions of stochastic PDEs for different types of available data and different dimensions. We conclude in section 6 with a brief summary and discussion of the current limitations of the method.

2. Problem setup.

To illustrate the main idea of PI-GANs, we consider the following SDE:

$$\begin{aligned} \mathcal{N}_{\boldsymbol{x}}[u(\boldsymbol{x};\omega);k(\boldsymbol{x};\omega)] &= f(\boldsymbol{x};\omega), \quad \boldsymbol{x}\in\mathcal{D}, \quad \omega\in\Omega, \\ B_{\boldsymbol{x}}[u(\boldsymbol{x};\omega)] &= b(\boldsymbol{x};\omega), \quad \boldsymbol{x}\in\Gamma, \end{aligned} \tag{1}$$

where $\mathcal{N}_{\boldsymbol{x}}$ is a general differential operator, $\mathcal{D}$ is the d-dimensional physical domain in $\mathbb{R}^d$, $\boldsymbol{x}$ is the d-dimensional spatial coordinate, Ω is the probability space, and ω is a random event. The coefficient $k(\boldsymbol{x};\omega)$ and the forcing term $f(\boldsymbol{x};\omega)$ are modeled as random processes, and thus the solution $u(\boldsymbol{x};\omega)$ will depend on both $k(\boldsymbol{x};\omega)$ and $f(\boldsymbol{x};\omega)$. $B_{\boldsymbol{x}}$ is the boundary condition operator acting on the domain boundary Γ. As a pedagogical example, in this paper we consider the one-dimensional elliptic SDE, which retains most of the main features of more complex SDEs:

$$\begin{gathered} -\frac{1}{10}\frac{d}{dx}\left[k(x;\omega)\frac{d}{dx}u(x;\omega)\right] = f(x;\omega), \quad x\in[-1,1], \\ u(-1)=u(1)=0, \end{gathered} \tag{2}$$

where $k(x;\omega)$ and $f(x;\omega)$ are independent stochastic processes, and $k(x;\omega)$ is strictly positive. For simplicity, we impose homogeneous Dirichlet boundary conditions on $u(x;\omega)$.

We consider a general scenario for (1), where we have a limited number of measurements from scattered sensors for the stochastic processes. Specifically, we place sensors at $\{\boldsymbol{x}_i^k\}_{i=1}^{n_k}$, $\{\boldsymbol{x}_i^u\}_{i=1}^{n_u}$, $\{\boldsymbol{x}_i^f\}_{i=1}^{n_f}$, and $\{\boldsymbol{x}_i^b\}_{i=1}^{n_b}$ to collect "snapshots" of $k(\boldsymbol{x};\omega)$, $u(\boldsymbol{x};\omega)$, $f(\boldsymbol{x};\omega)$, and $b(\boldsymbol{x};\omega)$, where n_k, n_u, n_f, and n_b are the numbers of sensors for $k(\boldsymbol{x};\omega)$, $u(\boldsymbol{x};\omega)$, $f(\boldsymbol{x};\omega)$, and $b(\boldsymbol{x};\omega)$, respectively. Here, one snapshot represents a simultaneous read of all the sensors, and we assume that the data in one snapshot correspond to the same random event ω in the random space Ω, while ω varies for different snapshots. Note that each snapshot is the concatenation of snapshots from $k(\boldsymbol{x};\omega)$, $u(\boldsymbol{x};\omega)$, $f(\boldsymbol{x};\omega)$, and $b(\boldsymbol{x};\omega)$, and thus it is actually a vector of size $(n_k + n_u + n_f + n_b)$. Suppose we have a group of N snapshots; then we denote the accessible data set by $\{T(\omega^{(j)})\}_{j=1}^N$, defined as

$$\begin{aligned}
\{T(\omega^{(j)})\}_{j=1}^N &= \{(K(\omega^{(j)}), U(\omega^{(j)}), F(\omega^{(j)}), B(\omega^{(j)}))\}_{j=1}^N,\\
K(\omega^{(j)}) &= (k(\boldsymbol{x}_i^k;\omega^{(j)}))_{i=1}^{n_k},\\
U(\omega^{(j)}) &= (u(\boldsymbol{x}_i^u;\omega^{(j)}))_{i=1}^{n_u},\\
F(\omega^{(j)}) &= (f(\boldsymbol{x}_i^f;\omega^{(j)}))_{i=1}^{n_f},\\
B(\omega^{(j)}) &= (b(\boldsymbol{x}_i^b;\omega^{(j)}))_{i=1}^{n_b}.
\end{aligned} \tag{3}$$

The corresponding terms in (3) are omitted if we put no sensors for that specific process.

In this paper, we assume that we always have a sufficient number of sensors for $f(x;\omega)$ in (2). The type of the problems varies depending on the number of sensors on $k(x;\omega)$ and $u(x;\omega)$: as we decrease the number of sensors on $k(x;\omega)$ while increasing the number of sensors on $u(x;\omega)$, the problems gradually transform from forward to mixed and finally to inverse problems.

Moreover, we could have more than one group of snapshots of measurements. For example, in addition to the existing old sensors, we may place some new sensors, collecting and utilizing data from both the new and the old sensors. In this scenario, we have two groups of data: the newly collected data, and the previously collected data solely from the old sensors. In the case of multiple groups, our accessible data are

$$\{\{T_t(\omega^{(t,j)})\}_{j=1}^{N_t}\}_{t=1}^M = \{\{(K_t(\omega^{(t,j)}), U_t(\omega^{(t,j)}), F_t(\omega^{(t,j)}), B_t(\omega^{(t,j)}))\}_{j=1}^{N_t}\}_{t=1}^M, \tag{4}$$

where M is the number of groups, t is the index for the groups, and N_t is the number of snapshots in group t. The random event $\omega^{(t,j)}$ denotes the random instance of the jth snapshot in group t. Note that the sensor setups in different groups could be different, and we use $\{\boldsymbol{x}_i^{k,t}\}_{i=1}^{n_{k,t}}$ to denote the position setup of $n_{k,t}$ sensors for k in group t (similarly for other terms). More importantly, snapshots from different groups could be collected independently, so that the snapshots in different groups do not have to be aligned or paired.

3. A brief review of GANs and WGANs.

Before moving to our main algorithms, we briefly review GANs and WGANs. Consider the problem of learning a distribution P_r on $\mathbb{R}^d$, given data sampled from P_r. Suppose we have a DNN $G_\theta(\cdot)$ parameterized by θ that takes the random variable $\boldsymbol{\xi} \in \mathbb{R}^m$ as input and outputs a sample $G_\theta(\boldsymbol{\xi}) \in \mathbb{R}^d$. The random input $\boldsymbol{\xi}$ is sampled from a prescribed distribution Q_z (e.g., uniform, Gaussian), and we denote the distribution of $G_\theta(\boldsymbol{\xi})$ as P_g. We want to approximate P_r with P_g.

GANs deal with this problem by defining a two-player zero-sum game between the generator $G_\theta(\cdot)$ and the discriminator $D_\rho(\cdot)$, which is another DNN parameterized by ρ. The discriminator takes a sample $\boldsymbol{x} \in \mathbb{R}^d$ as input and aims to tell whether it is sampled from P_r or P_g. Meanwhile, the generator tries to "deceive" the discriminator by mimicking the true distribution P_r. In vanilla GANs [9], the two-player zero-sum game is defined as follows:

$$\inf_\theta \sup_\rho \mathbb{E}_{\boldsymbol{x}\sim P_r}[\log(D_\rho(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{\xi}\sim Q_z}[\log(1 - D_\rho(G_\theta(\boldsymbol{\xi})))]. \tag{5}$$

Correspondingly, the loss functions for the generator and discriminator are

$$\begin{aligned} L_g &= \mathbb{E}_{\boldsymbol{\xi}\sim Q_z}[\log(1 - D_\rho(G_\theta(\boldsymbol{\xi})))], \\ L_d &= -\mathbb{E}_{\boldsymbol{x}\sim P_r}[\log(D_\rho(\boldsymbol{x}))] - \mathbb{E}_{\boldsymbol{\xi}\sim Q_z}[\log(1 - D_\rho(G_\theta(\boldsymbol{\xi})))]. \end{aligned} \tag{6}$$

If the discriminator is optimal, L_g measures the Jensen–Shannon (JS) divergence between P_r and P_g up to multiplication and addition by constants:

$$JS(P_r, P_g) = \frac{1}{2}KL(P_r||M) + \frac{1}{2}KL(P_g||M), \tag{7}$$

where $M = (P_r + P_g)/2$, and $KL(\cdot||\cdot)$ is the Kullback–Leibler divergence [9]. By training the generator and discriminator iteratively, ideally we can make P_g approach P_r in the sense of the JS divergence.

However, the JS divergence does not always provide a usable gradient for the generator, especially when the two distributions concentrate on low-dimensional manifolds [1]. As a consequence, training vanilla GANs is quite a delicate process and could be unstable [1]. To fix this problem, Wasserstein GANs with clips on weights (weight-clipped WGANs) [1] and WGANs with gradient penalty (WGAN-GP) [12] were proposed. Similar to vanilla GANs, the two-play zero-sum game in WGANs is formulated as

$$\inf_\theta \sup_{\substack{\rho \\ D_\rho \text{ is 1-Lipschitz}}} \mathbb{E}_{\boldsymbol{x}\sim P_r}[D_\rho(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{\xi}\sim Q_z}[D_\rho(G_\theta(\boldsymbol{\xi}))]. \tag{8}$$

The difference between weight-clipped WGANs and WGAN-GP is mainly on the technique of imposing the Lipschitz constraint for D_ρ: weight-clipped WGANs force ρ to be bounded during the training, while WGAN-GP adds a gradient penalty to the loss function for the discriminator.[1] To be specific, in WGAN-GP the loss functions for the generator and discriminator are defined as

$$\begin{aligned} &L_g = -\mathbb{E}_{\boldsymbol{\xi}\sim Q_z}[D_\rho(G_\theta(\boldsymbol{\xi}))], \\ &L_d = \mathbb{E}_{\boldsymbol{\xi}\sim Q_z}[D_\rho(G_\theta(\boldsymbol{\xi}))] - \mathbb{E}_{\boldsymbol{x}\sim P_r}[D_\rho(\boldsymbol{x})] + \lambda\mathbb{E}_{\hat{\boldsymbol{x}}\sim P_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D_\rho(\hat{\boldsymbol{x}})\|_2 - 1)^2], \end{aligned} \tag{9}$$

where $P_{\hat{\boldsymbol{x}}}$ is the distribution generated by uniform sampling on straight lines between pairs of points sampled from P_r and P_g, and λ is the gradient penalty coefficient.

Instead of the JS divergence in vanilla GANs, in WGANs the loss function for the generator corresponds to the earth mover's distance or Wasserstein-1 distance ($\mathbb{W}_1$) between P_r and P_g:

$$\mathbb{W}_1(P_r, P_g) = \inf_{\gamma\in\Gamma(P_r,P_g)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma}[\|\boldsymbol{x} - \boldsymbol{y}\|], \tag{10}$$

[1]In [1, 12], the discriminators are names of "critics," but in this paper we still use the name of discriminators for consistency with other versions of GANs, including vanilla GANs.

where $\Gamma(P_r, P_g)$ denotes the set of all joint distributions $\gamma(\boldsymbol{x}, \boldsymbol{y})$ whose marginals are P_r and P_g, respectively. $\mathbb{W}_1$ distance is continuous and differentiable almost everywhere with respect to the parameters in the generator under a mild constraint [1]. As a result, WGANs do not suffer from the problem of mode collapse, as would occur to vanilla GANs [1]. Moreover, according to [12], training WGAN-GP is more stable than weight-clipped WGANs.

To demonstrate the effectiveness of WGAN-GP, we apply it on four toy problems of approximating distributions in $\mathbb{R}^{10}$, as illustrated in Figure 2. We consider two types of P_r: one is a uniform distribution on a hypercube in $\mathbb{R}^{10}$, while the other one is a uniform distribution concentrated on a curve embedded in $\mathbb{R}^{10}$. For both cases of P_r, we test with an input of either one-dimensional or 10-dimensional standard Gaussian noise, i.e., $Q_z = N(0, 1)$ or $Q_z = N(0, I_{10})$. In all four cases, the generator converges and generates samples with distribution P_g close to the real distribution P_r, even for the cases where the dimension of Q_z mismatches the support of P_r. Due to its robustness, we use WGAN-GP as our default version of GANs in this paper.

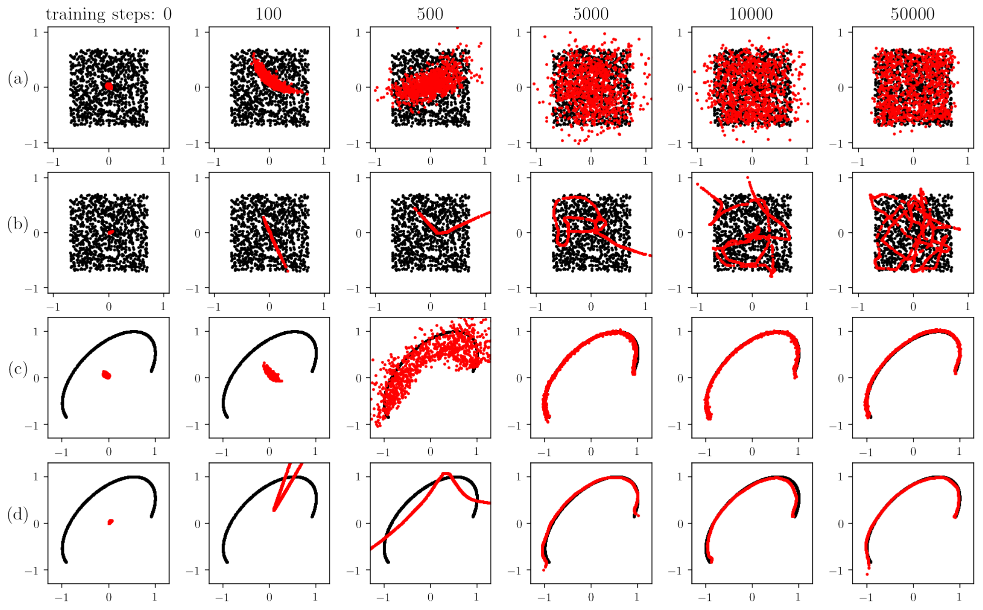


FIG. 2. *Evolution of generated distributions P_g for two different input dimensions and two different target distributions during training. The red dots (color available online only) are samples from P_g, and the black dots are samples from P_r. We visualize the distributions in $\mathbb{R}^{10}$ by plotting sample projections in the first two dimensions.* (a) *The support of P_r is a hypercube, $Q_z = N(0, I_{10})$.* (b) *The support of P_r is a hypercube, $Q_z = N(0, 1)$. Notice that the support of P_g is always a curve, but it gets increasingly twisted in filling the hypercube and minimizing $\mathbb{W}_1(P_r, P_g)$.* (c) *The support of P_r is a curve, $Q_z = N(0, I_{10})$. Notice that P_g gets increasingly concentrated in fitting the curve and minimizing $\mathbb{W}_1(P_r, P_g)$.* (d) *The support of P_r is a curve, $Q_z = N(0, 1)$.*

4. Methodology.

4.1. Approximating stochastic processes with GANs. As a pedagogical problem, let us first consider the problem of modeling a stochastic process $f(\boldsymbol{x}; \omega)$ on the domain $D \in \mathbb{R}^d$ with GANs. We use the total N snapshots of $f(\boldsymbol{x}; \omega)$ sensor data as our training set:

$$\{F(\omega^{(j)})\}_{j=1}^N = \{(f(\boldsymbol{x}_i;\omega^{(j)}))_{i=1}^{n_s}\}_{j=1}^N, \tag{11}$$

where n_s is the number of sensors and $\{\boldsymbol{x}_i\}_{i=1}^n$ are positions of the sensors. We use a feed forward DNN $\tilde{f}_\theta(\boldsymbol{x};\boldsymbol{\xi})$ parameterized by θ as our generator to model the stochastic process $f(\boldsymbol{x};\omega)$. The generator takes the concatenation of a noise vector $\boldsymbol{\xi} \sim Q_z$ from $\mathbb{R}^m$ and the coordinate $\boldsymbol{x}$ as the input, while the output is a real number representing $\tilde{f}_\theta(\boldsymbol{x};\boldsymbol{\xi})$. Let

$$\{\tilde{F}(\boldsymbol{\xi}^{(j)})\}_j = \{(\tilde{f}_\theta(\boldsymbol{x}_i;\boldsymbol{\xi}^{(j)}))_{i=1}^{n_s}\}_j \tag{12}$$

be the generated "fake" snapshots, where $\{\boldsymbol{\xi}^{(j)}\}_j$ are instances of $\boldsymbol{\xi}$ with j as the index.

The discriminator $D_\rho(\cdot)$ is implemented as another feed forward DNN parameterized by ρ. The GAN strategy shall be applied by feeding both the real and the generated snapshots into the discriminator, and training the generator and discriminator iteratively with the Adam optimizer [16] according to (9). The detailed algorithm is presented in Algorithm 1.

Algorithm 1 GANs for approximating stochastic processes.

```
Require: training steps $n_t$, the gradient penalty coefficient $\lambda$, the number of discriminator iterations per generator iteration $n_d$, the batch size $n$, Adam hyperparameters $\alpha, \beta_1, \beta_2$, initial values $\theta_0$ and $\rho_0$ for the parameters generator $\theta$ and $\rho$.
  for $s_t = 1,2,\dots,n_t$ do
    for $s_d = 1,2,\dots,n_d$ do
      Sample $n$ snapshots $\{F^{(j)}\}_{j=1}^n$ from the training data set.
      Sample $n$ random vectors $\{\boldsymbol{\xi}^{(j)}\}_{j=1}^n \sim Q_z$.
      Sample $n$ uniform random numbers $\{\epsilon^{(j)}\}_{j=1}^n \sim U[0,1]$.
      for $j = 1,2,\dots,n$ do
        $\hat{F}^{(j)} \leftarrow \epsilon^{(j)} F^{(j)} + (1-\epsilon^{(j)})\tilde{F}(\boldsymbol{\xi}^{(j)})$
        $L^{(j)} \leftarrow D_\rho(\tilde{F}(\boldsymbol{\xi}^{(j)})) - D_\rho(F^{(j)}) + \lambda(\|\nabla_{\hat{F}^{(j)}} D_\rho(\hat{F}^{(j)})\|_2 - 1)^2$
      end for
      $\rho \leftarrow \mathrm{Adam}(\nabla_\rho \frac{1}{n}\sum_{j=1}^n L^{(j)}, \rho, \alpha, \beta_1, \beta_2)$
    end for
    Sample $n$ random vectors $\{\boldsymbol{\xi}^{(j)}\}_{j=1}^n \sim Q_z$.
    $\theta \leftarrow \mathrm{Adam}(\nabla_\theta \frac{1}{n}\sum_{j=1}^n -D_\rho(\tilde{F}(\boldsymbol{\xi}^{(j)})), \theta, \alpha, \beta_1, \beta_2)$
  end for
```

4.2. Solving SDEs with PI-GANs. Consider (1), as illustrated in Figure 3: solving SDEs with PI-GANs consists of the following three steps.

First, we use two independent feed forward DNNs, namely $\tilde{k}_{\theta_k}(\boldsymbol{x};\boldsymbol{\xi})$ and $\tilde{u}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi})$ parameterized by θ_k and θ_u, to represent the stochastic processes $k(\boldsymbol{x};\omega)$ and $u(\boldsymbol{x};\omega)$ in the aforementioned way.

Second, inspired by the physics-informed neural networks for deterministic PDEs [29], we encode the equation into the neural networks system by applying the operator $\mathcal{N}_{\boldsymbol{x}}$ and $B_{\boldsymbol{x}}$ on the feed forward DNNs $\tilde{k}_{\theta_k}(\boldsymbol{x};\boldsymbol{\xi})$ and $\tilde{u}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi})$ to generate "induced" neural networks, which are formulated as

$$\tilde{f}_{\theta_u,\theta_k}(\boldsymbol{x};\boldsymbol{\xi}) = \mathcal{N}_{\boldsymbol{x}}[\tilde{u}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi}); \tilde{k}_{\theta_k}(\boldsymbol{x};\boldsymbol{\xi})] \tag{13}$$

and

$$\tilde{b}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi}) = B_{\boldsymbol{x}}[\tilde{u}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi})]. \tag{14}$$

Differentiation in $\mathcal{N}_{\boldsymbol{x}}$ and $B_{\boldsymbol{x}}$ are performed by automatic differentiation [2]. We then use $\tilde{f}_{\theta_u,\theta_k}(\boldsymbol{x};\boldsymbol{\xi})$ and $\tilde{b}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi})$ as the generators of $f(\boldsymbol{x};\omega)$ and $b(\boldsymbol{x};\omega)$, respectively. Note that both $\tilde{f}_{\theta_u,\theta_k}(\boldsymbol{x};\boldsymbol{\xi})$ and $\tilde{b}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi})$ are induced from $\tilde{u}_{\theta_u}(\boldsymbol{x};\boldsymbol{\xi})$ and $\tilde{k}_{\theta_k}(\boldsymbol{x};\boldsymbol{\xi})$, and thus the parameters θ_k and θ_u are directly inherited from $\tilde{u}$ and $\tilde{k}$.

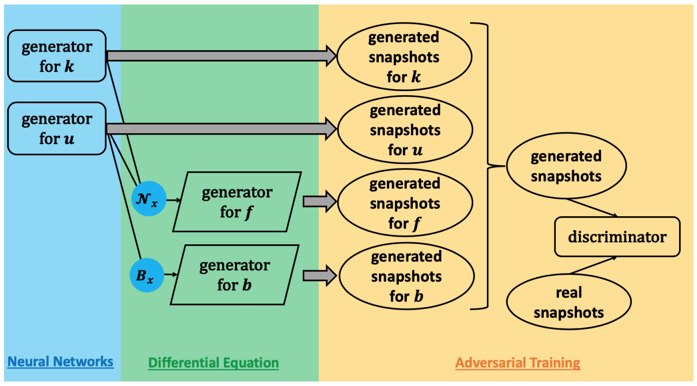

generator for $\boldsymbol{k}$
generator for $\boldsymbol{u}$
$\mathcal{N}_x$
B_x
generator for $\boldsymbol{f}$
generator for $\boldsymbol{b}$
generated snapshots for $\boldsymbol{k}$
generated snapshots for $\boldsymbol{u}$
generated snapshots for $\boldsymbol{f}$
generated snapshots for $\boldsymbol{b}$
generated snapshots
discriminator
real snapshots
Neural Networks
Differential Equation
Adversarial Training

FIG. 3. *Schematic of solving SDEs with PI-GANs. The round corner rectangles in the blue and yellow boxes represent feed forward neural networks. The parallelograms in the green box represent the neural networks induced by operators ($\mathcal{N}_{\boldsymbol{x}}$ and $B_{\boldsymbol{x}}$). The ellipses represent snapshots from sensors, and the gray arrows represent sampling procedures. The bracket in the yellow box represents concatenation. Color is available online only.*

In the third step, we incorporate our training data to conduct adversarial training. The training data are collected as a group of snapshots in (3). The corresponding generated "fake" snapshots are

$$\begin{aligned}
\{G(\boldsymbol{\xi}^{(j)})\}_j &= \{(\tilde{K}(\boldsymbol{\xi}^{(j)}), \tilde{U}(\boldsymbol{\xi}^{(j)}), \tilde{F}(\boldsymbol{\xi}^{(j)}), \tilde{B}(\boldsymbol{\xi}^{(j)}))\}_j,\\
\tilde{K}(\boldsymbol{\xi}^{(j)}) &= (\tilde{k}_{\theta_k}(\boldsymbol{x}_i^k;\boldsymbol{\xi}^{(j)}))_{i=1}^{n_k},\\
\tilde{U}(\boldsymbol{\xi}^{(j)}) &= (\tilde{u}_{\theta_u}(\boldsymbol{x}_i^u;\boldsymbol{\xi}^{(j)}))_{i=1}^{n_u},\\
\tilde{F}(\boldsymbol{\xi}^{(j)}) &= (\tilde{f}_{\theta_k,\theta_u}(\boldsymbol{x}_i^f;\boldsymbol{\xi}^{(j)}))_{i=1}^{n_f},\\
\tilde{B}(\boldsymbol{\xi}^{(j)}) &= (\tilde{b}_{\theta_u}(\boldsymbol{x}_i^b;\boldsymbol{\xi}^{(j)}))_{i=1}^{n_b},
\end{aligned} \tag{15}$$

where $\{\boldsymbol{\xi}^{(j)}\}_j$ are instances of $\boldsymbol{\xi}$ with j as the index. We could then feed the generated snapshots and real snapshots into the discriminator and train the generators and discriminators iteratively. With well-trained generators, we can then calculate all the statistics with sample paths created by the generators.

If the snapshots are collected in M groups ($M > 1$), we will also need to generate M groups of "fake" snapshots:

$$\begin{aligned}
\{\{G_t(\boldsymbol{\xi}^{(t,j)}\}_j\}_{t=1}^M &= \{\{(\tilde{K}_t(\boldsymbol{\xi}^{(t,j)}), \tilde{U}_t(\boldsymbol{\xi}^{(t,j)}), \tilde{F}_t(\boldsymbol{\xi}^{(t,j)}), \tilde{B}_t(\boldsymbol{\xi}^{(t,j)}))\}_j\}_{t=1}^M,\\
\tilde{K}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{k}_{\theta_k}(\boldsymbol{x}_i^{k,t};\boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{k,t}},\\
\tilde{U}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{u}_{\theta_u}(\boldsymbol{x}_i^{u,t};\boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{u,t}},\\
\tilde{F}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{f}_{\theta_k,\theta_u}(\boldsymbol{x}_i^{f,t};\boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{f,t}},\\
\tilde{B}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{b}_{\theta_u}(\boldsymbol{x}_i^{b,t};\boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{b,t}},
\end{aligned} \tag{16}$$

where t's are the indices for the groups, $\boldsymbol{\xi}^{(t,j)}$ is an instance of $\boldsymbol{\xi}$ for each (t,j), and $\{\boldsymbol{x}_i^{k,t}\}_{i=1}^{n_{k,t}}$ is the position setup of $n_{k,t}$ sensors for k in group t (similarly for other terms). We will also use multiple discriminators $\{D_{\rho_t}(\cdot)\}_t^M$, with each discriminator focusing on one group of snapshots, while the generators need to "deceive" all the discriminators simultaneously.

We give a formal and detailed description of our method in Algorithm 2. For the case where we only have one group of data, we set $M = 1$. For simplicity, here we set the weight in the generator loss function $a_t = 1$ for each t, which works well, but the method of setting $\{a_t\}_{t=1}^M$ requires further study.

Algorithm 2 PI-GANs for solving SDEs.

```
Require: training steps $n_t$, the gradient penalty coefficient $\lambda$, the number of discriminator iterations per generator iteration $n_d$, the batch size $n$, Adam hyperparameters $\alpha, \beta_1$, and $\beta_2$, initial values $\theta_{k0}$, $\theta_{u0}$, and $\{\rho_{t0}\}_{t=1}^M$ for the parameters $\theta_k$, $\theta_u$ and $\{\rho_t\}_{t=1}^M$, the weights in the loss function for the generators $\{a_t\}_{t=1}^M$.
  for $s_t = 1,2,\dots,n_t$ do
    for $s_d = 1,2,\dots,n_d$ do
      for $t = 1,2,\dots,M$ do
        Sample $n$ snapshots $\{T_t^{(j)}\}_{j=1}^n$ from training data group $t$.
        Sample $n$ random vectors $\{\boldsymbol{\xi}^{(t,j)}\}_{j=1}^n \sim Q_z$.
        Sample $n$ uniform random numbers $\{\epsilon^{(t,j)}\}_{j=1}^n \sim U[0,1]$.
        for $j = 1,2,\dots,n$ do
          $\hat{G}_t^{(j)} \leftarrow \epsilon^{(j)} T_t^{(j)} + (1-\epsilon^{(j)}) G_t(\boldsymbol{\xi}^{(t,j)})$
          $L_t^{(j)} \leftarrow D_{\rho_t}(G_t(\boldsymbol{\xi}^{(t,j)})) - D_{\rho_t}(T^{(t,j)})$
                $+ \lambda(\|\nabla_{\hat{G}_t^{(j)}} D_{\rho_t}(\hat{G}_t^{(j)})\|_2 - 1)^2$
        end for
        $\rho_t \leftarrow \mathrm{Adam}(\nabla_{\rho_t} \frac{1}{n}\sum_{j=1}^n L_t^{(j)}, \rho_t, \alpha, \beta_1, \beta_2)$
      end for
    end for
    Sample $n$ random vectors $\{\boldsymbol{\xi}^{(j)}\}_{j=1}^n \sim Q_z$.
    $\theta \leftarrow \mathrm{Adam}(\nabla_\theta \sum_{t=1}^M a_t(\frac{1}{n}\sum_{j=1}^n -D_{\rho_t}(G_t(\boldsymbol{\xi}^{(j)}))), \theta, \alpha, \beta_1, \beta_2)$,
      where $\theta = (\theta_k, \theta_u)$
  end for
```

Note that our method does not explicitly distinguish the three types of problems described in section 2. Solving forward problems, inverse problems, or mixed problems actually uses the same framework.

5. Numerical results.

The following settings are commonly shared by all the test cases. We use *tanh* as the activation function instead of the commonly used ReLU activation function for generator neural networks, because piecewise linear functions are not suitable for solving SDEs, where we may need to take high order derivatives. For discriminator neural networks, we also use *tanh* as the activation function, if not specifically mentioned. The feed forward DNNs for generators in the following numerical experiments have four hidden layers of width 128, if not specifically mentioned. The sizes of the input layer into the generators vary and will be specified case by case. As the default settings, the discriminators have hidden layers of the same size as that of the generators, except for the ones in subsection 5.1 with four hidden layers of width 64, the one in the 120-dimensional problem in subsection 5.2.2 whose size we will specify later, and the one for the additional group of snapshots on $u(x;\omega)$ in subsection 5.3 that has four hidden layers of width 16. To initialize the DNNs, we use the uniform Xavier initializer for weights and the zero initializer for biases. The distribution Q_z of the noise input into the generators is an independent multivariate standard Gaussian distribution. For the hyperparameters in loss functions and optimizers, we use the default values of $\lambda = 0.1$, $n_d = 5$, $\alpha = 0.0001$, $\beta_1 = 0.5$, $\beta_2 = 0.9$, as in the toy problem in [12]. The sensors are placed equidistantly in the domain. Our algorithms are implemented with Tensorflow. For the cases of solving PDEs, in order to generate the data as well as the references solutions, we numerically simulated the sensor measurements with a Monte Carlo method using the finite difference scheme. All the references solutions are generated from 1×10^5 Monte Carlo sample paths.

5.1. A pedagogical problem: Approximating stochastic processes. In this section, we test the problem of approximating stochastic processes. Consider the following Gaussian processes with zero mean and squared exponential kernel:

$$\begin{aligned} f(x) &\sim \mathcal{GP}\left(0, \exp\left(\frac{-(x-x')^2}{2l^2}\right)\right), \\ x &\in D = [-1,1], \end{aligned} \tag{17}$$

where l is the correlation length.

5.1.1. Effect of the number of sensors and snapshots. We consider three different choices of correlation length l, $l = 1, 0.5, 0.2$, and two sensor numbers for $f(x;\omega)$, 11 and 6, and we fix the number of snapshots to be 1000. For $l = 0.2$, we also consider a supplementary case, where we have 1×10^4 snapshots. The training sample paths and positions of sensors are illustrated in Figure 4. For each case, we run the code three times with different random seeds. The batch size in all the cases is 1000. The input layer of the generators has a width of 5; i.e., the input noise is a four-dimensional random vector.

To decide when to stop the training process, we calculate the $\mathbb{W}_1$ distances between the empirical distributions of generated snapshots and training snapshots $\mathbb{W}_1(\hat{P_g^n}, \hat{P_t^n})$, where $\hat{P_g^n}$ is the empirical distribution of n generated snapshots from Monte Carlo sampling and $\hat{P_t^n}$ is the empirical distribution of n snapshots from the training set, using the python POT package [8]. In our tests, we set $n = 1000$. We plot the $\mathbb{W}_1$ distances in Figure 5. Note that the $\mathbb{W}_1$ distances decay and converge during the training, indicating that the generated distributions approach the real distributions. We stop the training after 1×10^5 steps since $\mathbb{W}_1(\hat{P_g^n}, \hat{P_t^n})$ is stable. In each run, we select the 11 generators at training step in the last 10001 steps with a stride of 1×10^3; all together, we have 33 generators for each case. From each selected generator, we generate 1×10^4 sample paths based on the Halton quasi–Monte Carlo method and calculate its spectra, i.e., the eigenvalues of the covariance matrices from the principal component analysis.

The results are illustrated in Figure 6, from which we conclude the following:

1. When we fix the number of snapshots to be 1000, as the correlation length decreases, the gap between our generated processes and the reference becomes wider. This is because a smaller l results in higher effective dimension and more subtle local behavior of the stochastic processes; however, this gap could be narrowed if we increase the number of snapshots.

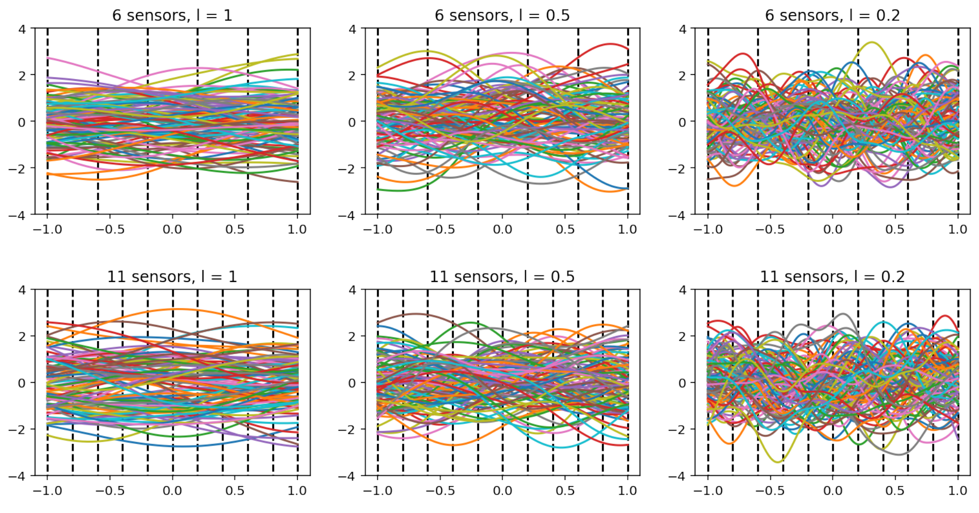


FIG. 4. *Sample paths of Gaussian processes with the sensor locations denoted by the vertical dashed lines. We use the correlation lengths* $l = 1$ *(left),* 0.5 *(middle), and* 0.2 *(right) and* 6 *sensors (top) and* 11 *sensors (bottom).*

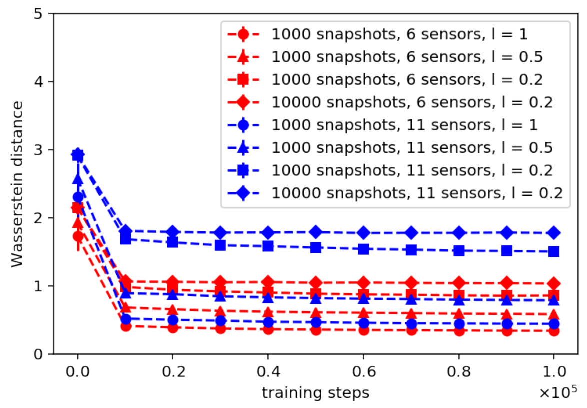


FIG. 5. $\mathbb{W}_1$ *distances* $\mathbb{W}_1(\hat{P_g^n}, \hat{P_t^n})$ *versus training steps for different correlation lengths and number of sensors. The decay of* $\mathbb{W}_1$ *distances becomes slow after about* 2×10^4 *steps and hard to see after* 8×10^4 *steps. We set the number of sample snapshots for empirical distribution to be* $n = 1000$. *The means and two standard deviations are from three independent runs and* 10 *batches of* n *generated snapshots and training snapshots in each run.*

2. The approximations in the cases with 11 sensors are better than the approximations in the cases with 6 sensors if we have sufficient training data. This is reasonable since we need more sensors to describe the stochastic processes with small correlation length.
3. Despite the fact that the input noise into the generator is a four-dimensional vector, the spectra of the generated processes approximate the spectra of the target processes with much higher effective dimensionality. This makes sense since the low-dimensional manifold could fold and twist itself to fill the high-dimensional region, as discussed earlier.

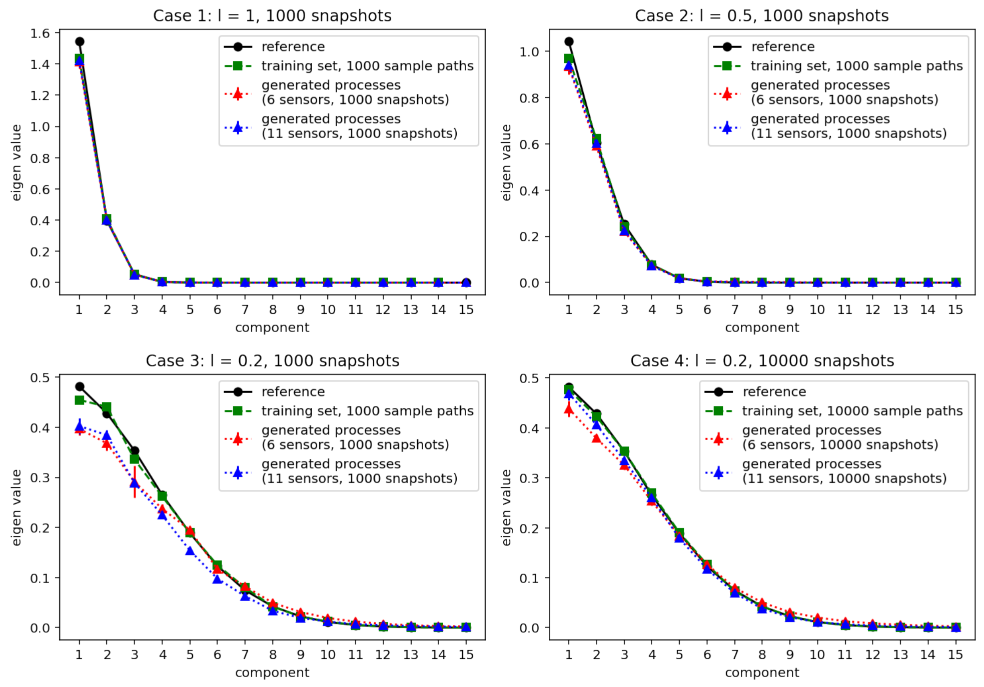


FIG. 6. *Spectra of the correlation structure for the generated processes of different correlation lengths with the squared exponential kernel. The training sets consist of the processes from where the training snapshots are collected, and the reference is calculated from* 1×10^5 *independent sample paths. The means and two standard deviations are calculated from the selected* 33 *generators for each case; the standard deviations are very small.*

5.1.2. Different kernels. To show that our method can learn various spatial correlation structures, apart from the squared exponential kernel shown above, we also test the problem of approximating stochastic processes with an exponential kernel:

$$f(x) \sim \mathcal{GP}\left(0, \exp\left(-\frac{|x-x'|}{l}\right)\right), \quad x, x' \in D = [-1, 1], \tag{18}$$

where l is the correlation length. Similar to the squared exponential kernel above, we test the four cases with the different correlation lengths and number of snapshots. The results are illustrated in Figure 7, and we can see the improvement as we increase the number of sensors and snapshots, as well as that the spectra of generated processes fit the reference values given sufficient sensors and snapshots, just like in the squared exponential kernel case.

To further illustrate that our method can learn different spatial correlation structures, in Figure 8 we plot the covariance of the generated processes evaluated at randomly selected pairs of coordinates for both types of covariance kernels. To be more specific, for each case we randomly select 1000 pairs of (x, x') from the domain, calculate the covariance of the generated processes at x and x', and then show the scatter plots of the covariance against the distance between x and x'. We can see that for both cases, the scatter plots fit well with the analytic reference, showing that our method can reveal the different spatial correlation structures.

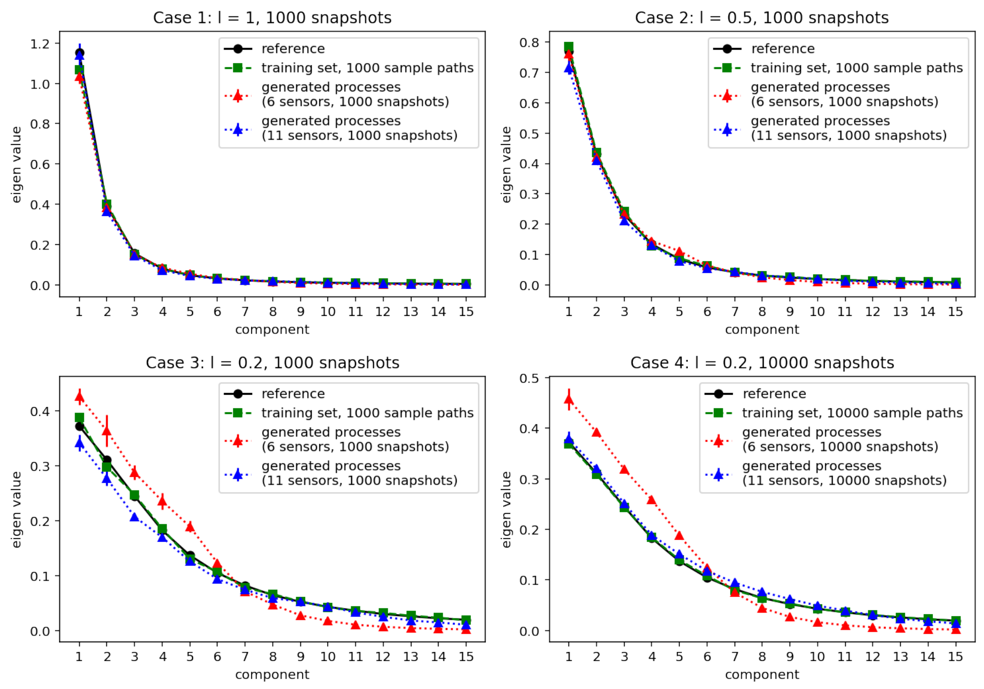


Fig. 7. *Spectra of the correlation structure for the generated processes for different correlation lengths with the exponential kernel. The training sets consist of the processes from where the training snapshots are collected, and the reference solution is calculated from* 1×10^5 *independent sample paths. The means and two standard deviations are calculated from the selected* 33 *generators for each case; the standard deviations are very small.*

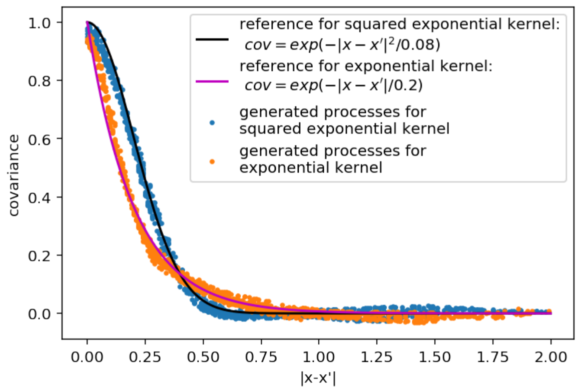


Fig. 8. *Scatter plot of covariance of the generated processes evaluated at randomly selected pairs of coordinates for both the squared exponential kernel and the exponential kernel. Each blue or orange dot represents the covariance of generated random processes with different structures, evaluated at a randomly selected pair of points* (x, x')*. The black and purple lines indicate the reference covariance from the exact expressions of the squared exponential kernel and the exponential kernel, respectively. Color is available online only.*

5.1.3. WGAN-GP versus vanilla GANs.

We compare the performance of WGAN-GP and the vanilla GANs in approximating stochastic processes for the following two cases:

1. A Gaussian process in (17) with $l = 0.2$.
2. A stochastic process with fixed 0 boundary condition. Specifically, we consider $f(x) = (x^2 - 1)g(x)$, where $g(x)$ is a Gaussian process:

$$g(x) \sim \mathcal{GP}\left(0, \exp\left(\frac{-(x-x')^2}{2 \times 0.2^2}\right)\right), \quad x, x' \in D = [-1, 1].$$

In both cases, we use 1×10^4 snapshots collected from 11 sensors as training data. In Figure 9, we show the means and standard deviations of generated processes trained by WGAN-GP and the vanilla GANs. We can see that both versions of GANs produce good approximations for case 1. However, the vanilla GANs fail in case 2, while WGAN-GP still generates a good approximation. This agrees with the theory in [12] that vanilla GANs are not suitable for approximating distributions concentrated on low-dimensional manifolds (the two fixed boundaries in this case) since the JS divergence cannot provide a usable gradient for the generators.

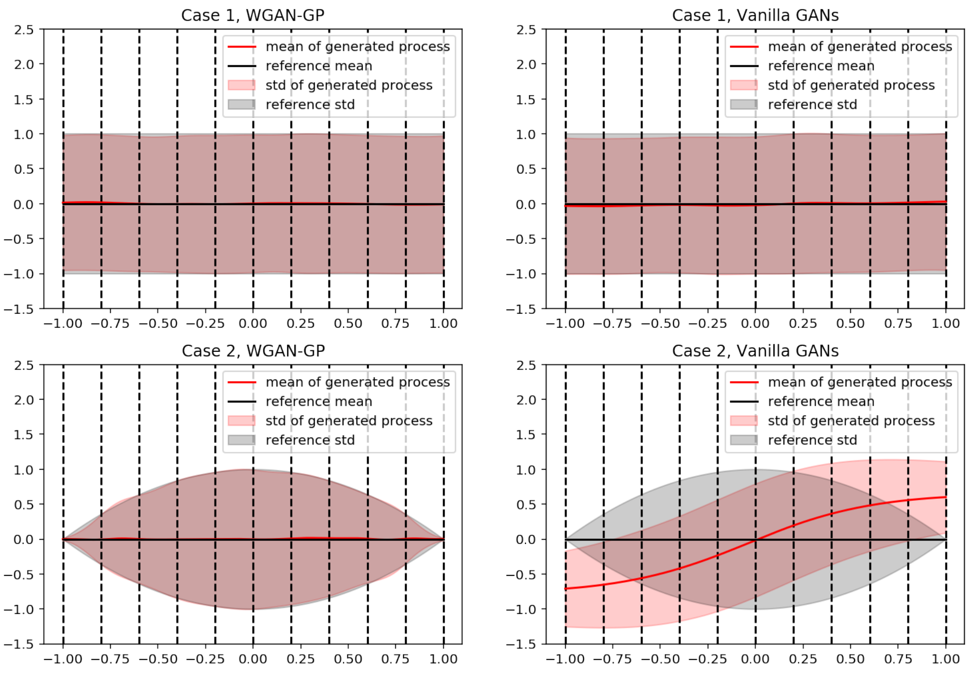


Fig. 9. *Mean and standard deviation of random processes generated by WGAN-GP and vanilla GANs: For case* 1*, both versions of GANs perform well, but the vanilla GANs failed in case* 2*, while WGAN-GP can still provide accurate results. The vertical dashed lines represent the positions of sensors.*

5.1.4. Overfitting issues. It was pointed out in [12] that the discriminator can overfit the training data given sufficient capacity but too little training data. Here we report the same issue in our method. Take the case of approximating the stochastic process where the correlation length is $l = 0.2$ with 11 sensors and 10000 training snapshots as an example. We plot the negative discriminator loss $-L_d$ for the training set and the validation set in Figure 10a. As training goes on, the negative discriminator loss gradually increases for the training set while still decreasing for the validation set. The gap between them implies that the discriminator overfits the training data, and gives a biased estimation of the $\mathbb{W}_1$ distance between the real distribution and generated distribution, just as is reported in [12].

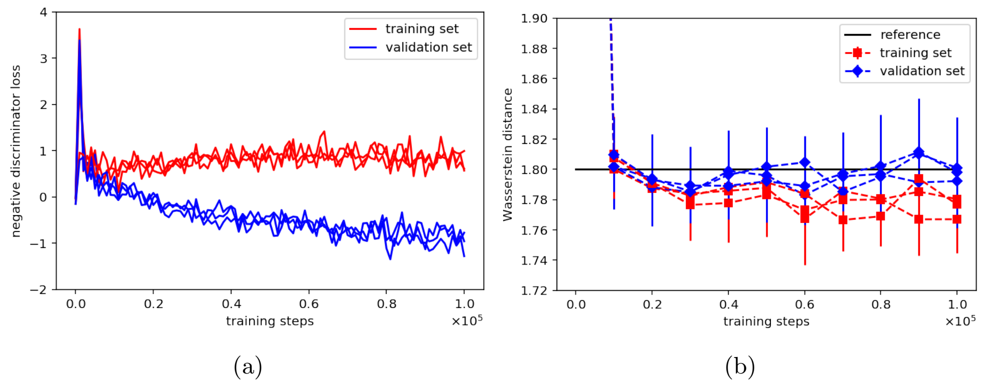


Fig. 10. *Overfitting of the discriminator* and *generator.* (a) *Negative discriminator loss on the training set and validation set versus the training steps in three independent runs.* (b) $\mathbb{W}_1$ *distances* $\mathbb{W}_1(\hat{P_g^n}, \hat{P_t^n})$ *and* $\mathbb{W}_1(\hat{P_g^n}, \hat{P_v^n})$ *versus the training steps in three independent runs. The means (markers) and two standard deviations (vertical lines on the marker) come from* 10 *groups of* $(\hat{P_g^n}, \hat{P_t^n}, \hat{P_v^n})$*. The black horizontal line is the empirical expectation of* $\mathbb{W}_1(\hat{P_r^n}_1, \hat{P_r^n}_2)$ *from* 50 *groups of* $(\hat{P_r^n}_1, \hat{P_r^n}_2)$*.*

How about the overfitting of generators? In our problem, the overfitting of generators comes in two types:

Type-1: Overfitting in the random space: The distribution of generated snapshots recovers the empirical distribution of the training snapshots instead of the true underlying distribution. In the worst case, the generated snapshots are concentrated on or near the support of training data.

Type-2: Overfitting in the physical space: The generated stochastic processes become worse after extensive training, and tend to match the real processes only at the sensor locations and display large variations where there is no sensor.

We first report that we did not detect Type-2 overfitting in our experiments. Actually, this type of overfitting would be reflected on the mean and standard deviation of generated processes, which fit the reference values pretty well in our experiments. We attribute this to the property of GANs that the target for the generator is to approximate a distribution rather than a single point on the sensors. As a result, the generator does not need to overfit a specific value on the sensors in order to decrease the loss.

As for the Type-1 overfitting, we could detect it in our experiments. As depicted in Figure 10b, we can see this from the $\mathbb{W}_1$ distances between empirical distributions of the generated snapshots and the training snapshots or the validation snapshots, i.e., $\mathbb{W}_1(\hat{P_g^n}, \hat{P_t^n})$ or $\mathbb{W}_1(\hat{P_g^n}, \hat{P_v^n})$, where $\hat{P_g^n}$, $\hat{P_t^n}$, and $\hat{P_v^n}$ are empirical distributions of generated snapshots, training snapshots, and validation snapshots, and n is the number of snapshots. Here we set $n = 1000$. We can see that as the training goes on, $\mathbb{W}_1(\hat{P_g^n}, \hat{P_v^n})$ converges around the expectation of $\mathbb{W}_1(\hat{P_r^n}_1, \hat{P_r^n}_2)$, where $\hat{P_r^n}_1$ and $\hat{P_r^n}_2$ are independent empirical distributions of n snapshots from real distribution P_r. However, $\mathbb{W}_1(\hat{P_g^n}, \hat{P_t^n})$ goes down below the reference line, indicating that the generated snapshots tend to approximate the empirical distribution of the training snapshots instead of the true underlying distribution; in other words, type-1 overfitting actually happened.

Finally, we point out that Type-1 overfitting is less harmful than underfitting: in our problems, even in the worst case, where the generated distribution is concentrated on or near the support of training data, we can still recover the sample paths whose snapshots are concentrated on or near the training snapshots, and based on these sample paths we can still obtain decent estimations.

5.2. Forward problem.

5.2.1. Effects of input noise dimension and number of training snapshots. We consider the elliptic SDE (2) and $k(x;\omega)$ and $f(x;\omega)$ as the following independent stochastic processes:

$$\begin{aligned} &k(x) = \exp\left[\frac{1}{5}\sin\left(\frac{3\pi}{2}(x+1)\right) + \hat{k}(x)\right],\\ &\hat{k}(x) \sim \mathcal{GP}\left(0, \frac{4}{25}\exp\bigl(-(x-x')^2\bigr)\right),\\ &f(x) \sim \mathcal{GP}\left(\frac{1}{2}, \frac{9}{400}\exp\bigl(-25(x-x')^2\bigr)\right). \end{aligned} \tag{19}$$

In this case, we need 13 dimensions to retain 99% energy of $f(x;\omega)$. We put 13 k-sensors , 21 f-sensors, and 2 u-sensors on the boundary of physical domain $\mathcal{D}$. The positions of the sensors and some sample paths of $k(x;\omega)$, $u(x;\omega)$, and $f(x;\omega)$ are illustrated in Figure 11.

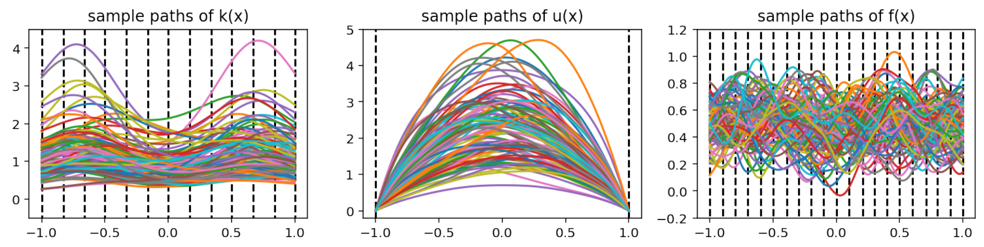


FIG. 11. *Forward problem: the sample paths and sensor locations of k, u, and f. The vertical dashed lines denote the locations of sensors.*

To study the influence of input noise dimension, we fix the number of training snapshots to be 1000 and vary the input noise dimension to be 2, 4, 20, and 50. Subsequently, we fix the input noise dimension as 20 and vary the number of training snapshots to be 300, 1000, and 3000 to study the influence of the number of training snapshots. During the training process, we keep the batch size to be the total number of training snapshots. One notable condition here is the independence of $k(x;\omega)$ and $f(x;\omega)$. To reflect this, we shuffle the alignment of snapshots from $k(x;\omega)$ and $f(x;\omega)$ in each training step. For each case, we run the code three times with different random seeds. We stop the training after 1×10^5 steps, and then select 33 generators and generate 1×10^4 sample paths from each generator in the same way as in subsection 5.1 to calculate the following statistics.

Our main quantity of interest in this problem is the mean and standard deviation of $u(x;\omega)$, i.e., $\mu(x) = \mathbb{E}_\omega[u(x;\omega)]$ and $\sigma(x) = \sqrt{\mathbb{E}_w[(u(x;\omega)-\mu(x))^2]}$. We look into the relative error defined as

$$\begin{aligned} &\text{relative error in mean: } \frac{||\hat{\mu}(x) - \mu(x)||_2}{||\mu(x)||_2}, \\ &\text{relative error in standard deviation: } \frac{||\hat{\sigma}(x) - \sigma(x)||_2}{||\sigma(x)||_2}, \end{aligned} \tag{20}$$

where $||\cdot||_2$ is the L_2 norm, and $\hat{\mu}(x)$ and $\hat{\sigma}(x)$ are the inferred mean and standard deviation of $u(x;\omega)$. In Figure 12, we show the relative error of inference from generated processes and compare it with the relative error of the stochastic collocation method and the Monte Carlo full trajectory sample paths of $u(x;\omega)$. We can see the following:

1. Our errors are of the same magnitude with the errors calculated from 1000 full trajectory sample paths of $u(x;\omega)$, showing the effectiveness of our method considering that we only have 1000 snapshots on sparsely placed sensors for $k(x;\omega)$, $f(x;\omega)$, and the boundary of $u(x;\omega)$.
2. The stochastic collocation method gives a better solution, but it requires a full knowledge of $k(x;\omega)$ and $f(x;\omega)$, including the covariance kernel function, which is far beyond our accessible data.
3. When we increase the dimension of input noise, we can see that the error of mean does not change too much, but the error of the standard deviation decreases. We attribute this to the fact that although a low-dimensional manifold could twist itself to fill in the high-dimensional region, higher-dimensional manifolds produce better approximations by filling in the high-dimensional region more efficiently.
4. With fixed input noise dimension, the error decreases as the number of training snapshots increases.

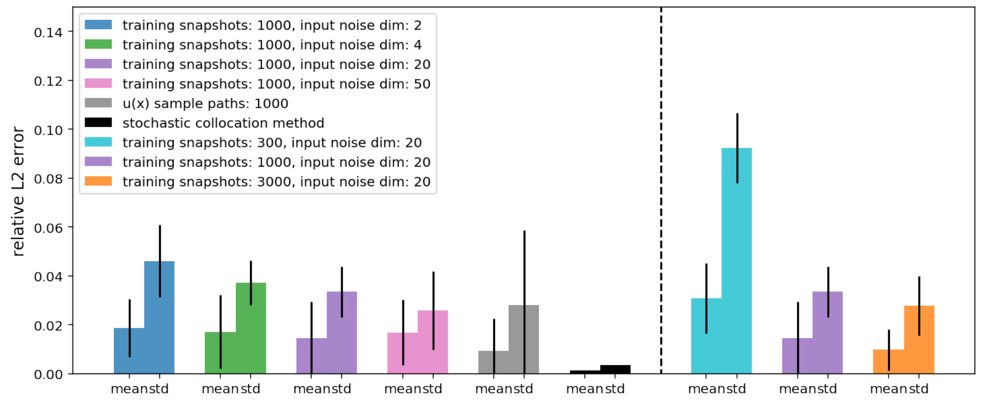


FIG. 12. *Forward problem for different input noise dimensionality (left part of the panel) and the number of training snapshots (right part of the panel): relative errors of inferred mean and standard deviation of the stochastic solution $u(x;\omega)$. The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. The grey bars and the black lines represent the expectation and two standard deviations of the relative error if we calculate the mean and standard deviation of $u(x;\omega)$ from a random draw of 1000 $u(x;\omega)$ sample paths. The black bar (before the vertical dashed line) is the relative error from the stochastic collocation method with 630 collocation points. The reference solutions are calculated from 1×10^5 $u(x;\omega)$ sample paths. Color is available online only.*

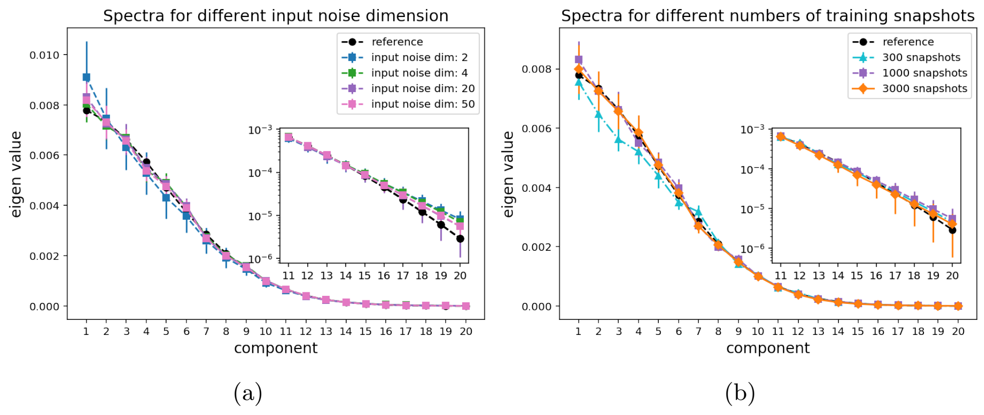


FIG. 13. *Forward problem: spectra of generated right-hand-side $f(x;\omega)$ processes.* (a) *varying input noise dimension for a fixed number of snapshots at* 1000. (b) *varying the number of training snapshots for a fixed dimension of input noise at* 20. *The inset plots are the zoom-ins of the eigenvalues of high frequency modes. The means (markers) and two standard deviations (vertical lines on the markers) are calculated from the selected* 33 *generators. The reference curves are calculated from another* 1×10^5 *independent* $f(x;\omega)$ *sample paths.*

We also illustrate the spectra of the generated processes for $f(x;\omega)$ in Figure 13 to verify that the generated processes captured the covariance structure of $f(x;\omega)$. We can see that the spectra of our generated processes fit the reference solution well. With a fixed number of training snapshots, as we increase the input noise dimension, the gap between our generated processes and the reference solutions narrows. With fixed input noise dimension, the gap narrows as we increase the number of training snapshots. These observations on the spectra are similar to those about the error of u.

We further check the correlation between the generated processes. The correlation coefficient $C(f_1(x;\omega), f_2(x;\omega))$ between two stochastic processes $f_1(x;\omega)$ and $f_2(x;\omega)$ $(x \in D)$ is defined as

$$C(f_1(x;\omega), f_2(x;\omega)) = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\operatorname{Cov}(f_1(x_i;\omega), f_2(x_i;\omega))}{\sqrt{\operatorname{Var}(f_1(x_i;\omega))\operatorname{Var}(f_2(x_i;\omega))}}\right|, \tag{21}$$

where x_i are the uniform grid points in D. Here we set $n = 201$ for $C(k, f)$, while $n = 199$ for $C(k, u)$ to exclude the boundary points for u.

From Table 1 we can see that our generated processes for $k(x;\omega)$ and $f(x;\omega)$ are weakly correlated, while the generated processes for $k(x;\omega)$ and $u(x;\omega)$ are strongly correlated. All the correlation coefficients are close to the reference values.

5.2.2. High-dimensional problems. Here we solve the SDE where the correlation length of $f(x;\omega)$ is relatively small. Let $k(x;\omega)$ and $f(x;\omega)$ be independent stochastic processes as follows:

$$\begin{aligned} k(x) &= \exp\left[\frac{1}{5}\sin\left(\frac{3\pi}{2}(x+1)\right) + \hat{k}(x)\right], \\ \hat{k}(x) &\sim \mathcal{GP}\left(0, \frac{4}{25}\exp(-(x-x')^2)\right), \\ f(x) &\sim \mathcal{GP}\left(\frac{1}{2}, \frac{9}{400}\exp\left(\frac{-(x-x')^2}{a^2}\right)\right). \end{aligned} \tag{22}$$

Table 1

Forward problem: correlation coefficient between the generated random processes. The mean and two standard deviations are calculated from the selected 33 *generators for each case. The reference for* $C(k, f)$ *comes from the assumption that* $k(x;\omega)$ *and* $f(x;\omega)$ *are independent, while the reference for* $C(k, u)$ *is calculated from* (21) *with* 1×10^5 *independent samples of* $k(x;\omega)$ *and* $u(x;\omega)$ *paths.*

	2 dim	4 dim	20 dim	40 dim
$C(k,f)$	3.4±2.1 %	2.3±1.6 %	2.5±2.3 %	2.7±1.8 %
$C(k,u)$	74.3±1.3 %	73.6±1.0 %	73.8± 0.9 %	73.2±0.8 %

	300 snapshots	3000 snapshots	Reference
$C(k,f)$	2.5±1.8 %	2.5±1.5 %	0
$C(k,u)$	74.0±0.9 %	72.4±1.5 %	72.5 %

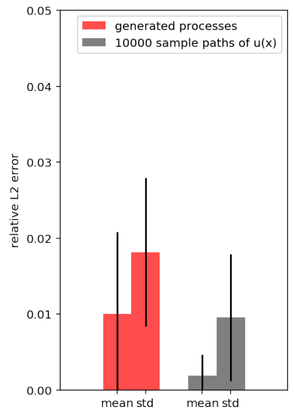
Two cases are considered, i.e., case 1: $a = 0.08$, and case 2: $a = 0.02$. Note that to retain 99% energy of $f(x;\omega)$, we need 30 dimensions for case 1 and about 120 dimensions for case 2. For case 1, we put 13 k-sensors, 41 f-sensors, and 2 u-sensors on the boundaries of our domain of interest $\mathcal{D}$ and set the input noise dimension to be 20. For case 2, we put 13 k-sensors, 161 f-sensors, and 2 u-sensors on the boundaries and set the input noise dimension to be 100. In both cases, we use 1×10^4 snapshots as training data, while we set the batch size to be 1000. Again, we shuffle the alignment of snapshots of $k(x;\omega)$ and $f(x;\omega)$ during the training to reflect their independence.

In practice, for case 2, we increase the number of hidden layer to eight, and follow the design of ResNet [13] by adding shortcut connections in the neural networks, since it has been reported that shortcut connections help the training of DNNs [13]. More specifically, for two adjacent hidden layers $\boldsymbol{z}_l$ and $\boldsymbol{z}_{l+1}$, $\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \sigma(\boldsymbol{W}_l \boldsymbol{z}_l + \boldsymbol{b}_l)$, where $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ are weights and biases, respectively, and σ is the activation function. While still using the *tanh* activation function in the generator neural network for case 2, in the discriminator neural network we use a leaky ReLu activation function [21] with a negative slope of 0.01 since it works well especially for higher resolution modeling in other applications of GANs [25].

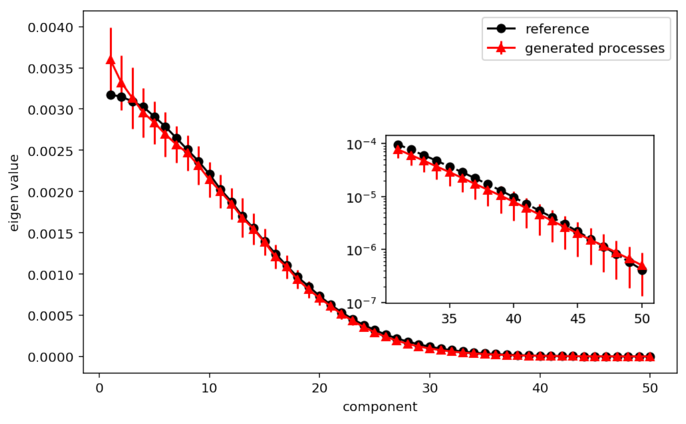
For case 1, we run the code three times with different random seeds and stop the training after 2×10^5 steps. We then select 33 generators and generate 1×10^4 sample paths from each generator, in the same way as in subsection 5.1, to calculate the error of $u(x;\omega)$ as well as the spectra of $f(x;\omega)$. For case 2, we run the code one time (due to the high computational cost) and stop the training after 4×10^5 steps. We then select 11 generators in the last 10001 training steps, with the stride of 1000 steps, and generate 1×10^4 sample paths from each generator to calculate the error of $u(x;\omega)$ as well as the spectra of $f(x;\omega)$. We report that for a single run, compared with case 1, the wall clock computational time in case 2 is about 18 times (about $\times 9$ for each step and $\times 2$ for the total number of steps), while the number of stochastic dimensions in case 2 is about four times larger than in case 1.

The results are illustrated in Figure 14. For case 1, we can see that the spectra fit the reference values well. The error is in the same magnitude with the error calculated from 1×10^4 sample paths of $u(x;\omega)$. In case 2, we can see that we managed to capture the spatial correlation roughly for the first 120 components, while after the first 120 components the eigenvalues of generated processes deviated from the reference values. Compared with case 1, in case 2 the error in mean is similar, while the error in standard deviation is larger but still within 8%.

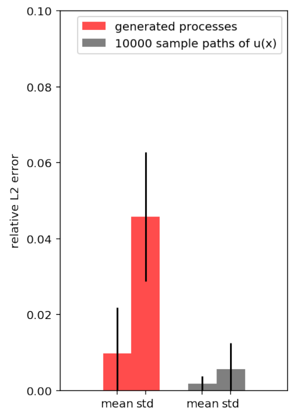
5.3. Inverse and mixed problems. In this section, we show that our method can manage a wide range of problems, from forward problems to inverse problems,



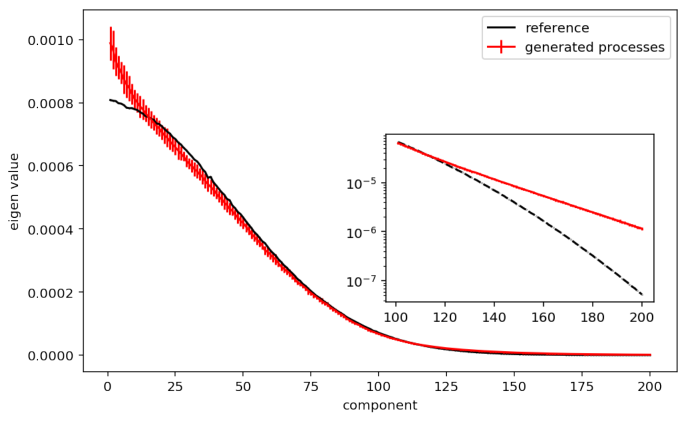
(a) Case 1, $a = 0.08$



(b) Case 1, $a = 0.08$



(c) Case 2, $a = 0.02$



(d) Case 2, $a = 0.02$

FIG. 14. *Forward problem with stochastic right-hand-side process $f(x;\omega)$ of high dimensionality. Figures* (a) *and* (b) *correspond to case* 1*, where the effective dimension is* 30*, while figures* (c) *and* (d) *correspond to case* 2*, where the effective dimension is about* 120*. Left: relative errors of inferred mean and standard deviation of $u(x;\omega)$. The red bars and their associated black lines represent the mean and two standard deviations of relative errors from the selected generators. The grey bars and their associated black lines represent the expectation and two standard deviations of relative errors calculated from 1×10^4 sample paths of $u(x;\omega)$. Right: the spectra of generated process versus the reference values. The means (red markers) and two standard deviations (red vertical lines on the markers) are calculated from the selected generators. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $u(x;\omega)$ and $f(x;\omega)$. Color is available online only.*

and mixed problems in between. In particular, we solve the three types of problems governed by (2), where $k(x;\omega)$ and $f(x;\omega)$ are independent processes, as follows:

$$\begin{aligned} k(x) &= \exp\left[\frac{1}{5}\sin\left(\frac{3\pi}{2}(x+1)\right) + \tilde{k}(x)\right], \\ \tilde{k}(x) &\sim \mathcal{GP}\left(0, \frac{4}{25}\exp\big(-(x-x')^2\big)\right), \\ f(x) &\sim \mathcal{GP}\left(\frac{1}{2}, \frac{9}{400}\exp\big(-(x-x')^2\big)\right). \end{aligned} \tag{23}$$

We consider the following four cases of sensor placement:

Case 1: 1 k-sensor, 13 u-sensors (including 2 on the boundary), 13 f-sensors.
Case 2: 5 k-sensors, 9 u-sensors (including 2 on the boundary), 13 f-sensors.
Case 3: 9 k-sensors, 5 u-sensors (including 2 on the boundary), 13 f-sensors.
Case 4: 13 k-sensors, 2 u-sensors on the boundary of u, 13 f-sensors.

Note that Case 1 is an inverse problem and Case 4 is a forward problem, while Cases 2 and 3 represent mixed problems. For each case, we use 1000 snapshots for training. Note that in Cases 1–3, we *cannot* shuffle the alignment of snapshots from $k(x;\omega)$ and $f(x;\omega)$ as in subsection 5.2 since both $k(x;\omega)$ and $f(x;\omega)$ are correlated with $u(x;\omega)$. For consistency, in this section, we do not shuffle the alignment of snapshots for any of the four cases. We set the batch size to be 1000 and the input noise dimension to be 20. For each case, we run the code three times with different random seeds. The training stops after 2×10^5 steps, and we select 33 generators and generate 1×10^4 sample paths from each generator in the same way as in subsection 5.1 to calculate the mean and standard deviation of $k(x;\omega)$ and $u(x;\omega)$.

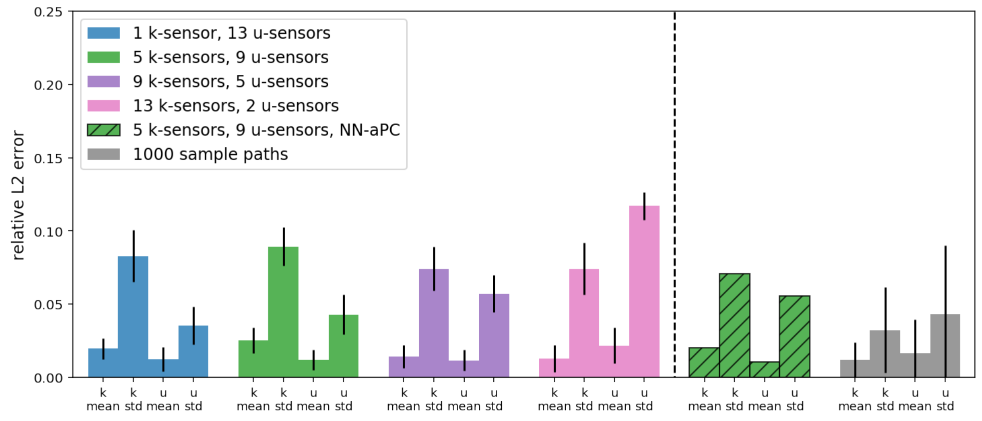


FIG. 15. *Relative errors of inferred mean and two standard deviations for both $k(x;\omega)$ and $u(x;\omega)$ in the inverse problem (blue leftmost bars), mixed problems (green and purple bars), and the forward problem (pink bars). The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. Also shown on the right part of the panel are two reference cases, the green shaded bars corresponding to the NN-aPC method* [39] *for Case 2 on the left, and the grey bars and their associated black lines corresponding to the expectation and two standard deviations of relative error calculated from 1000 independent sample paths of $k(x;\omega)$ or $u(x;\omega)$. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $k(x;\omega)$ and $u(x;\omega)$. Color is available online only.*

In Figure 15, we compare the relative errors with reference solutions calculated from Monte Carlo sample paths as well as the method proposed in [39]. The relative errors of $u(x;\omega)$ and $k(x;\omega)$ are defined as in (20). We can see that our errors are in the same order of magnitude with errors from 1000 sample paths, showing the effectiveness of our method in solving all three types of problems. Also, for the case of 5 k-sensors and 9 u-sensors, our method achieves comparable accuracy with the method in [39].

5.4. Multiple groups of training data. Finally, we test our method for the case where we have multiple groups of snapshots as training data. In particular, we perform our test based on Case 4 in subsection 5.3. Apart from the sensors in that case, we put one additional u-sensor at position $x = 0$ and collect another group of 1000 snapshots at this additional sensor. Hence, we have two groups of training data:

Group 1 : 13 k-sensors, 2 u-sensors on the boundary of u, 13 f-sensors, 1000 snapshots.

Group 2 : 1 u-sensor at $x = 0$, 1000 snapshots.

Again, we emphasize that the two groups of snapshots cannot be aligned. Note that a single snapshot in Group 2 is almost useless since we cannot align it with another snapshot in Group 1. However, the *ensemble* of the snapshots in Group 2 actually can tell the *distribution* for $u(x;\omega)$ at $x = 0$.

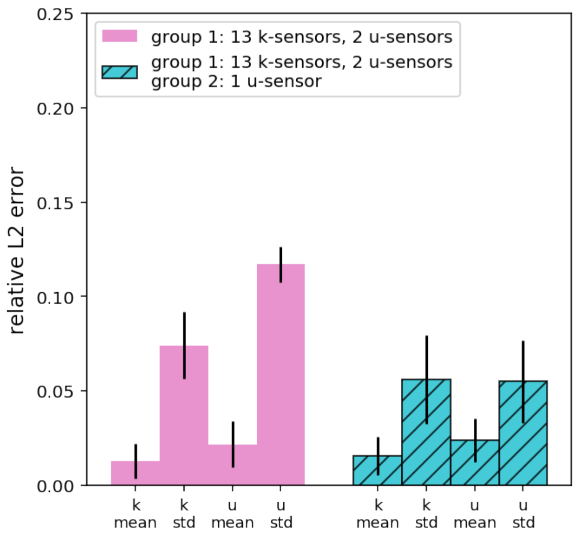


FIG. 16. *Relative errors of inferred mean and two standard deviations for both $k(x;\omega)$ and $u(x;\omega)$ using one group of data (pink) and two groups of data (light green). The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $k(x;\omega)$ and $u(x;\omega)$. Color is available online only.*

To utilize the data in the two groups, we apply Algorithm 2 with the number of discriminators $M = 2$. In this case, we set the training batch size as 1000, while the input noise dimension as 20. We run the code three times with different random seeds. The training stops after 2×10^5 steps, and we select 33 generators and generate 1×10^4 sample paths from each generator in the same way as in subsection 5.1 to calculate the mean and standard deviation of $k(x;\omega)$ and $u(x;\omega)$. In Figure 16, we plot the errors of the inferred $k(x;\omega)$ and $u(x;\omega)$ in this case as well as errors from Case 4 in subsection 5.3. Compared with the results obtained by only using one group of data, in this case, the error of the standard deviation of $u(x)$ decreases significantly, showing the capability of our method of *learning from the ensemble* of snapshots in Group 2. To the best of our knowledge, our method is so far the only one that can manage this case.

6. Summary and future work.

We proposed physics-informed generative adversarial networks (PI-GANs) as a data-driven method for solving stochastic differential equations (SDEs) based on a limited number of scattered measurements. PI-GANs are composed of a discriminator, which is represented by a simple feed forward deep neural network (DNN), and of generators, which are a combination of feed forward DNNs and a neural network induced by the SDE. We assumed that partial data are available in terms of different realizations of the stochastic process obtained simultaneously at different locations in the domain. We trained the generators and discriminator iteratively with the loss functions employed in WGAN-GP [12] so that the joint distribution of generated processes approximates the target stochastic processes. We also proposed a more general architecture with multiple discriminators to deal with cases where data are collected in multiple groups, i.e., data collected independently from different sets of sensors.

We first tested WGAN-GP in approximating Gaussian processes for different correlation lengths. As shown in Figure 6, we obtained good approximation of the generated stochastic processes to the target ones even for a mismatch between the input noise dimensionality and the effective dimensionality of the target stochastic processes. The approximations were improved by increasing the number of sensors and snapshots. We also compared WGAN-GP and vanilla GANs, and concluded that vanilla GANs are not suitable for approximating stochastic processes with a deterministic boundary condition, as shown in Figure 9. We further studied the overfitting issue by monitoring the negative discriminator loss (Figure 10a) and Wasserstein distance between empirical distributions (Figure 10b). We found that overfitting occurs also in the generator in addition to the discriminator, as previously reported.

Subsequently, we considered the solution of elliptic SDEs requiring approximations of three stochastic processes, namely the solution $u(x;\omega)$, the forcing $f(x;\omega)$, and the diffusion coefficient $k(x;\omega)$. Without changing the framework, we were able to solve a wide range of problems, from forward to inverse problems, and in between, i.e., mixed problems where we have incomplete information for both the solution and the diffusion coefficient. As shown in Figures 12 and 15, we obtained both the means and the standard deviations of the stochastic solution and the diffusion coefficient in good agreement with benchmarks. In the case of the forward problem, we studied the influence of dimensionality of the input noise into the generators in Figure 12 and found that although small dimensionality can also work, high dimensionality for the input noise leads to better results. Moreover, we tested PI-GANs for a relatively high dimensional problem with $f(x;\omega)$ of stochastic dimension 30. In Figure 14, we showed that the inferred mean and standard deviation of $u(x;\omega)$ as well as the spectra of $f(x;\omega)$ match the reference values well. Finally, we applied PI-GANs consisting of two discriminators to a stochastic problem with two groups of snapshots available for training, where the second group includes snapshots from only a single sensor for $u(x;\omega)$. We could see in Figure 16 that the error decreases compared with the error of the case where we only have the first group of data. This demonstrates the capability of PI-GANs to utilize information from multiple groups of data and learn from an *ensemble* of snapshots, even when a single snapshot is useless.

We also point out some limitations of the current version of PI-GANs. Since our method is based on data collected from sparse sensors, when learning stochastic processes the local structures smaller than the distance between adjacent sensors cannot be captured. For example, we report that in subsection 5.1.2, where we use the exponential kernel, the eigenvalues of the generated processes decay faster than the reference values for high order components (whose eigenvalues are very small). This indicates that although our method can accurately learn the large scale spatial correlation structure, the generated processes are smoother than the processes generated from the exponential kernel, missing some small scale details. Also, since the computational cost of training GANs is much higher than training a single feed forward neural network, the PI-GAN method has higher computational cost than the physics-informed neural networks for deterministic PDEs [29, 30] and SDEs [39] for low stochastic dimensional problems. Also, in our numerical experiments, overfitting was detected in both discriminators and generators when training data are limited. Although overfitting of generators is less harmful than underfitting, we wish to address this issue systematically in future work. Moreover, in the current work we only take into consideration the uncertainty described in SDEs, but not from measurements, nor do we account for the uncertainty of the approximability of GANs, as was done in [39] for DNNs using the dropout method. In future work, we wish to endow PI-GANs with uncertainty quantification coming from diverse sources and hence provide estimates of the total uncertainty of the unknown distribution. In this paper, we only consider static problems, but the proposed method can be extend to solving time-dependent stochastic problems, by treating time as "an extra dimension of spatial coordinate," as in [29]. Moreover, although in this paper the snapshots fed into the discriminator are just the concatenation of the snapshots from various terms, in some of other experiments we found that preprocessing, such as scaling and translating snapshots from some terms for both the real ones and the generated ones, helps improve the learning accuracy, and we wish to look into the principle for this trick in the future. We shall also aim to optimize PI-GANs in terms of the architecture, depth, and width of the generators and discriminators as well as the other hyperparameters. Finally, we comment on the computational cost due to the high dimensionality of stochastic problems. In subsection 5.2.2, we investigate the two cases with 30 and about 120 stochastic dimensions, respectively. For a single run, the wall clock computational time in case 2 is about $\times 18$ times of that in case 1 and the stochastic dimension is about $\times 4$ times, while the relative errors in both cases are of the same magnitude, suggesting that the overall computational cost increases with a low-polynomial growth, and hence PI-GANs can, in principle, tackle very high dimensional stochastic problems. We will systematically investigate the scalability of PI-GANs for more complex stochastic PDEs in very high dimensions in future work.

REFERENCES

[1] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein GAN*, preprint, https://arxiv.org/abs/1701.07875, 2017.

[2] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: A survey*, J. Mach. Learn. Res., 18 (2017), 153.

[3] D. Berthelot, T. Schumm, and L. Metz, *BEGAN: Boundary Equilibrium Generative Adversarial Networks*, preprint, https://arxiv.org/abs/1703.10717, 2017.

[4] I. Bilionis, *Probabilistic Solvers for Partial Differential Equations*, preprint, https://arxiv.org/abs/1607.03526, 2016.

[5] S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proc. Natl. Acad. Sci. USA, 113 (2016), pp. 3932–3937.

[6] W. E, J. Han, and A. Jentzen, *Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations*, preprint, https://arxiv.org/abs/1706.04702, 2017.

[7] W. Fedus, I. Goodfellow, and A. M. Dai, *MaskGAN: Better Text Generation via Filling in the ______*, preprint, https://arxiv.org/abs/1801.07736, 2018.

[8] R. Flamary and N. Courty, *POT Python Optimal Transport Library*, https://github.com/rflamary/POT, 2017.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2014, pp. 2672–2680.

[10] T. Graepel, *Solving noisy linear operator equations by Gaussian processes: Application to*

ordinary and partial differential equations, in Proceedings of the International Conference on Machine Learning, 2003, pp. 234–241.

[11] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik, *Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models*, preprint, https://arxiv.org/abs/1705.10843, 2017.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, *Improved training of Wasserstein GANs*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2017, pp. 5767–5777.

[13] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[14] T. Karras, T. Aila, S. Laine, and J. Lehtinen, *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, preprint, https://arxiv.org/abs/1710.10196, 2017.

[15] Y. Khoo, J. Lu, and L. Ying, *Solving Parametric PDE Problems with Artificial Neural Networks*, preprint, https://arxiv.org/abs/1707.03351, 2017.

[16] D. P. Kingma and J. Ba, *Adam: A method for Stochastic Optimization*, preprint, https://arxiv.org/abs/1412.6980, 2014.

[17] I. E. Lagaris, A. C. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Netw., 9 (1998), pp. 987–1000, https://doi.org/10.1109/72.712178.

[18] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, *Neural-network methods for boundary value problems with irregular boundaries*, IEEE Trans. Neural Netw., 11 (2000), pp. 1041–1049, https://doi.org/10.1109/72.870037.

[19] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, *Photo-realistic single image super-resolution using a generative adversarial network*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4681–4690.

[20] X. Liang, Z. Hu, H. Zhang, C. Gan, and E. P. Xing, *Recurrent Topic-Transition GAN for Visual Paragraph Generation*, preprint, https://arxiv.org/abs/1703.07022, 2017.

[21] A. L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, in Proceedings of the 30th International Conference on Machine Learning, 2013.

[22] O. Mogren, *C-RNN-GAN: Continuous Recurrent Neural Networks with Adversarial Training*, preprint, https://arxiv.org/abs/1611.09904, 2016.

[23] M. A. Nabian and H. Meidani, *A Deep Neural Network Surrogate for High-Dimensional Random Partial Differential Equations*, preprint, https://arxiv.org/abs/1806.02957, 2018.

[24] G. Pang, L. Yang, and G. E. Karniadakis, *Neural-net-induced Gaussian process regression for function approximation and PDE solution*, J. Comput. Phys., 384 (2019), pp. 270–288, https://doi.org/10.1016/j.jcp.2019.01.045.

[25] A. Radford, L. Metz, and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, preprint, https://arxiv.org/abs/1511.06434, 2015.

[26] M. Raissi, *Forward-Backward Stochastic Neural Networks: Deep Learning of High-Dimensional Partial Differential Equations*, preprint, https://arxiv.org/abs/1804.07010, 2018.

[27] M. Raissi and G. E. Karniadakis, *Hidden physics models: Machine learning of nonlinear partial differential equations*, J. Comput. Phys., 357 (2018), pp. 125–141.

[28] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Machine learning of linear differential equations using Gaussian processes*, J. Comput. Phys., 348 (2017), pp. 683–693, https://doi.org/10.1016/j.jcp.2017.07.050.

[29] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics Informed Deep Learning (Part* I*): Data-Driven Solutions of Nonlinear Partial Differential Equations*, preprint, https://arxiv.org/abs/1711.10561, 2017.

[30] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics Informed Deep Learning (Part* II*): Data-Driven Discovery of Nonlinear Partial Differential Equations*, preprint, https://arxiv.org/abs/1711.10566, 2017.

[31] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Numerical Gaussian processes for time-dependent and nonlinear partial differential equations*, SIAM J. Sci. Comput., 40 (2018), pp. A172–A198, https://doi.org/10.1137/17M1120762.

[32] S. Särkkä, *Linear operators and stochastic partial differential equations in Gaussian process regression*, in Artificial Neural Networks and Machine Learning — ICANN 2011, Springer, Berlin, Heidelberg, 2011, pp. 151–158.

[33] M. Schmidt and H. Lipson, *Distilling free-form natural laws from experimental data*, Science, 324 (2009), pp. 81–85.

[34] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364.

[35] A. M. Stuart, *Inverse problems: A Bayesian perspective*, Acta Numer., 19 (2010), pp. 451–559.

[36] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, *Midinet: A Convolutional Generative Adversarial Network for Symbolic-Domain Music Generation*, preprint, https://arxiv.org/abs/1703.10847, 2017.

[37] X. Yang, G. Tartakovsky, and A. Tartakovsky, *Physics-Informed Kriging: A Physics-Informed Gaussian Process Regression Method for Data-Model Convergence*, preprint, https://arxiv.org/abs/1809.03461, 2018.

[38] L. Yu, W. Zhang, J. Wang, and Y. Yu, *Seqgan: Sequence generative adversarial nets with policy gradient*, in Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[39] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis, *Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems*, J. Comput. Phys., 397 (2019), 108850.

[40] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin, *Adversarial Feature Matching for Text Generation*, preprint, https://arxiv.org/abs/1706.03850, 2017.

[41] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, *Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks*, preprint, https://arxiv.org/abs/1703.10593, 2017.

[42] Y. Zhu and N. Zabaras, *Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification*, J. Comput. Phys., 366 (2018), pp. 415–447.