Journal of Computational Physics 384 (2019) 270–288

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Neural-net-induced Gaussian process regression for function approximation and PDE solution

Guofei Pang, Liu Yang, George Em Karniadakis *

Division of Applied Mathematics, Brown University, Providence, RI 02912, USA

ARTICLE INFO

Article history:
Received 21 June 2018
Received in revised form 26 December 2018
Accepted 24 January 2019
Available online 22 February 2019

Keywords:
NN-induced Gaussian process
Neural network
Machine learning
Partial differential equation
Uncertainty quantification

ABSTRACT

Neural-net-induced Gaussian process (NNGP) regression inherits both the high expressivity of deep neural networks (deep NNs) as well as the uncertainty quantification property of Gaussian processes (GPs). We generalize the current NNGP to first include a larger number of hyperparameters and subsequently train the model by maximum likelihood estimation. Unlike previous works on NNGP that targeted classification, here we apply the generalized NNGP to function approximation and to solving partial differential equations (PDEs). Specifically, we develop an analytical iteration formula to compute the covariance function of GP induced by deep NN with an error-function nonlinearity. We compare the performance of the generalized NNGP for function approximations and PDE solutions with those of GPs and fully-connected NNs. We observe that for smooth functions the generalized NNGP can yield the same order of accuracy with GP, while both NNGP and GP outperform deep NN. For non-smooth functions, the generalized NNGP is superior to GP and comparable or superior to deep NN.

1. Introduction

The high expressivity of deep NNs [1] can be combined with the predictive ability of GP regression by taking advantage of the apparent equivalence between GPs and fully-connected, infinitely-wide, deep NNs with random weights and biases. This equivalence was first proved by [2] for shallow NNs. In a subsequent paper, the authors of [3] obtained analytically the covariance functions of the GPs induced by the shallow NNs having two specific activation functions: error and Gaussian functions. More recently, the authors of [4] extended [3]'s work to the case of deep NNs having generic activation functions, with reference to the mean field theory of signal propagation [1,5].

The term NNGP was coined by the authors of [4], but until now this method despite its potential merits has not been used extensively in applications. In particular, the authors of [4] validated the NNGP on two image recognition datasets: MNIST (handwritten digits data) and CIFAR-10 (color images data), and observed that NNGP outperforms finite width NNs in terms of classification accuracy. The primary objective of our work is to exploit the flexibility of NNGP in order to

* Corresponding author.
E-mail address: george_karniadakis@brown.edu (G.E. Karniadakis).

https://doi.org/10.1016/j.jcp.2019.01.045
0021-9991/

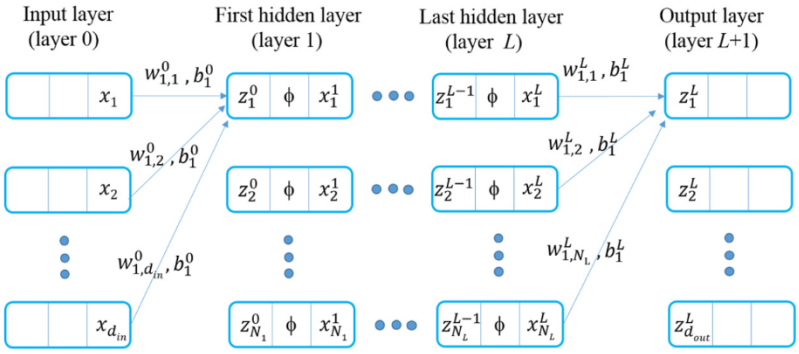

Fig. 1. A fully connected neural net with L hidden layers. For each unit or neuron in hidden layers, there exist one input z_i^{l-1} and one output x_i^l with $l = 1, 1 \cdots, L$. Layer 0 is the input layer. x_i is the i-th component of the input vector $\boldsymbol{x}$, and z_i^L is the i-th component of the output vector $\boldsymbol{z}$. The dimensions of input and output spaces are d_{in} and d_{out}, respectively. The width for hidden layer l is N_l. At the center of each unit is the activation function $\phi(\cdot)$ that transforms input to the corresponding output. Between two successive layers, the weight w_{ij}^l for $l = 0, 1, \cdots, L$ denotes the contribution of unit j in layer l to unit i in layer $l+1$. Layer $L+1$ is the output layer. The bias b_i^l is attached to unit i in layer $l+1$ for $l = 0, 1, \cdots, L$. Note that for clarity most of connecting arrows between layers are omitted.

approximate smooth and discontinuous functions, and furthermore in assessing the effectiveness of NNGP to solve linear and nonlinear PDEs, compared to the GP and deep NN methods formulated recently in [6–8]. Specifically, so far GP regression has been successfully applied to solution of PDEs including linear and nonlinear forward problems [6] as well as inverse problems [9,10]. Use of GP regression can bypass discretizing differential operators by properly placing GP priors, and can easily handle noisy data as well as the associated uncertainty propagation in time-dependent problems. Hence, the motivation of the current research is to evaluate if indeed NNGP possesses not only the high expressivity of deep NNs but also the uncertainty quantification property of GPs. To this end, we will perform three-way comparisons between NNGP, GP and deep NN for prototypical problems involving function approximation and PDE solution.

The present paper makes two main contributions.

First, to pursue high expressivity, we increase the number of hyperparameters of NNGP by assuming that the prior variances of weights and biases vary layer by layer and that the weight variances between the input layer and the first hidden layer are different for different neurons in the input layer. We also train the resulting NNGP regression model by using maximum likelihood estimation. We note that the authors of [4] assumed only two hyperparameters: one weight variance and one bias variance, which are kept fixed over different layers; the two hyperparameters are adjusted to achieve the best performance on validation dataset. However, no training was performed in [4].

Second, we derive the analytical covariance function for the NNGP induced by the deep NN with the error-function nonlinearity. This is motivated by the observation that the analytical covariance function induced by ReLU activated NN, given in [4], cannot be used to solve the two-dimensional Poisson equation. Denoting by $k(\boldsymbol{x}, \boldsymbol{x}')$ the analytical covariance function from the ReLU case where $\boldsymbol{x}$ and $\boldsymbol{x}'$ are input vectors of length two, we observed that the Laplacian of the covariance function, namely, $\Delta_{\boldsymbol{x}}\Delta_{\boldsymbol{x}'}k(\boldsymbol{x}, \boldsymbol{x}')$, which has to be computed in order to solve the problem (see Eq. (24)), will tend to infinity as $\boldsymbol{x} \to \boldsymbol{x}'$.

The paper is organized as follows. In Section 2, we review the equivalence between GPs and infinitely wide NNs, which is a cornerstone of NNGP. We change the notations for weight and bias variances compared to the previous work [4] in order to show a different hyperparameter setup. Section 3 introduces the analytically tractable covariance functions induced by the deep NNs with the ReLU and error-function nonlinearities. We employed a version of Gaussian quadrature, proposed in [4], to validate numerically the two analytical covariance functions. The methodology on GP regression for function approximation and PDE solution is briefly reviewed and summarized in Section 4. Section 5 compares the accuracy of GP, NNGP and NN in function approximation and PDE solution. The uncertainty estimates of GP and NNGP are also presented in the section. Finally, we conclude in Section 5 with a brief summary.

2. Equivalence between GPs and infinitely wide NNs

In this section, we adopt nearly the same notations as [4] except those for weight and bias variances. We add the subscript l to the variances σ_w^2 and σ_b^2 in order to represent index of layer and the subscript j to σ_w^2 in order to distinguish the weights coming from different neurons in input layer.

The fully-connected neural net can be represented as in Fig. 1. The equivalence relies on two assumptions: (1) Weight and bias parameters are all random variables. For $l \geq 1$ the weights w_{ij}^l are independent and identically distributed (i.i.d.) and obey the same distribution $\mathcal{D}_l$ with zero mean and same variance, namely, $w_{ij}^l \sim \mathcal{D}_l(0, \sigma_{w,l}^2/N_l), \forall i, j$. The biases b_i^l ($l \geq 1$) are i.i.d. Gaussian random variables, $b_i^l \sim \mathcal{N}(0, \sigma_{b,l}^2), \forall i$. The weights and biases are also independent across layers: the weights $w_{ij}^{l_1}$ and $w_{i'j'}^{l_2}$ ($l_1 \neq l_2$) are independent for arbitrary i, i', j, j', the biases $b_i^{l_1}$ and $b_{i'}^{l_2}$ ($l_1 \neq l_2$) are independent

for arbitrary i, i', and the weights $w_{ij}^{l_1}$ and the biases $b_{i'}^{l_2}$ $(l_1 \neq l_2)$ are independent for arbitrary i, j, i'. Note that in [4] the weight and bias variances are kept fixed over different layers, i.e., $\sigma_{w,l}^2 \equiv \sigma_w^2$ and $\sigma_{b,l}^2 \equiv \sigma_b^2$, but we here assume the variances vary layer by layer and increase the number of variances to be determined. For $l = 0$, the weights need to be i.i.d. with respect to the subscript i but only independent with respect to j, namely, $w_{ij}^0 \sim \mathcal{D}_0(0, \sigma_{w,0,j}^2), \forall i$; the biases are still i.i.d. Gaussian random variables, $b_i^0 \sim \mathcal{N}(0, \sigma_{b,0}^2), \forall i$. Unlike [4], we remove the assumption of being identically distributed for the weights w_{ij}^0 with respect to j for fixed i. These weights for different neurons (different j) in input layer do not have to be identically distributed, since the number of neurons in input layer is finite and thus we cannot use the Central Limit Theorem to conclude that the input of the first hidden layer, $z_i^0(\boldsymbol{x})$, is a Gaussian random variable for fixed $\boldsymbol{x}$. Allowing weight variances to vary for different neurons in input layer, we increase again the number of variances to be determined in NNGP. (2) The widths of all hidden layers tend to infinity, i.e., $N_l \to \infty, 1 \leq l \leq L$. This allows one to use the Central Limit Theorem to deduce the equivalence relation.

It follows from [3] and [11] that for analytical derivation of the covariance function the distribution $\mathcal{D}_0$ must be Gaussian. The specific forms of $D_l, l \geq 1$ do not really matter, since using the Central Limit Theorem one always obtains the Gaussian distribution whatever $\mathcal{D}_l$ is.

The i-th component of the input of the second hidden layer is computed as

$$z_i^1(\boldsymbol{x}) = \sum_{j=1}^{N_1} w_{ij}^1 x_j^1(\boldsymbol{x}) + b_i^1, \quad x_j^1(\boldsymbol{x}) = \phi\left(\sum_{k=1}^{d_{in}} w_{jk}^0 x_k + b_j^0\right). \tag{1}$$

In the rest of paper, we use bold letter, say $\boldsymbol{x}$ and $\boldsymbol{K}$, to represent column vector or matrix. The dependence on the input vector $\boldsymbol{x}$ is emphasized in the above equations. Since weights and biases w_{jk}^0 and b_j^0 are i.i.d., the output of the first hidden layer $x_j^1(\boldsymbol{x})$ is also i.i.d. with respect to the subscript j. Because $z_i^1(\boldsymbol{x})$ is a sum of i.i.d. terms, the use of the Central Limit Theorem yields that $z_i^1(\boldsymbol{x})$ is a Gaussian random variable when $N_1 \to \infty$. Similarly, from the multidimensional Central Limit Theorem, any finite collection of $\{z_i^1(\boldsymbol{x}_1), \cdots, z_i^1(\boldsymbol{x}_k)\}$ will form a joint multivariate Gaussian distribution, which is exactly the definition of a Gaussian process. Note that $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k$ are k arbitrary input vectors.

Denoting by $\mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$ the GP with the mean function $\mu(\cdot)$ and the covariance function $k(\cdot, \cdot)$, we learn that $z_i^1(\boldsymbol{x}) \sim \mathcal{GP}(\mu^1(\boldsymbol{x}), k^1(\boldsymbol{x}, \boldsymbol{x}'))$ when $N_1 \to \infty$. Also, $\{z_1^1(\boldsymbol{x}), z_2^1(\boldsymbol{x}), \cdots, z_{N_2}^1(\boldsymbol{x})\}$ are independent GPs with same mean and covariance functions. The mean function of GP $z_i^1(\boldsymbol{x})$ is zero due to the zero mean of the distribution $\mathcal{D}_1$, and the covariance function is computed as

$$\begin{aligned}
k^1(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}(z_i^1(\boldsymbol{x}) z_i^1(\boldsymbol{x}')) &= \mathbb{E}\left(\left(\sum_{j=1}^{N_1} w_{ij}^1 x_j^1(\boldsymbol{x}) + b_i^1\right)\left(\sum_{k=1}^{N_1} w_{ik}^1 x_k^1(\boldsymbol{x}') + b_i^1\right)\right) \\
&= \mathbb{E}\left(\left(\sum_{j=1}^{N_1} w_{ij}^1 \phi(z_j^0(\boldsymbol{x})) + b_i^1\right)\left(\sum_{k=1}^{N_1} w_{ik}^1 \phi(z_k^0(\boldsymbol{x}')) + b_i^1\right)\right) \\
&= \sum_{j=1}^{N_1} \mathbb{E}(w_{ij}^1)^2 \mathbb{E}\left[\phi(z_j^0(\boldsymbol{x}))\phi(z_j^0(\boldsymbol{x}'))\right] + \mathbb{E}(b_i^1)^2 \\
&= \frac{\sigma_{w,1}^2}{N_1} \sum_{j=1}^{N_1} \mathbb{E}\left[\phi(z_j^0(\boldsymbol{x}))\phi(z_j^0(\boldsymbol{x}'))\right] + \sigma_{b,1}^2 \\
&= \sigma_{w,1}^2 \mathbb{E}\left[\phi(z_j^0(\boldsymbol{x}))\phi(z_j^0(\boldsymbol{x}'))\right] + \sigma_{b,1}^2.
\end{aligned} \tag{2}$$

In the above derivation, we use the independence relations between weights $\mathbb{E}(w_{ij}^1 w_{ik}^1) = \delta_{jk}\sigma_{w,1}^2$ (δ_{ij} is the Kronecker delta), between weight and bias $\mathbb{E}(w_{ij}^1 b_i^1) = \mathbb{E}(w^1)\mathbb{E}(b_i^1) = 0$, and between weight and activation function $\mathbb{E}\left[(w_{ij}^1)^2 \phi(z_j^0(\boldsymbol{x}))\phi(z_j^0(\boldsymbol{x}'))\right] = \mathbb{E}\left(w_{ij}^1\right)^2 \mathbb{E}\left[\phi(z_j^0(\boldsymbol{x}))\phi(z_j^0(\boldsymbol{x}'))\right]$. The last equality in Eq. (2) comes from the independence of $\phi(z_j^0(\boldsymbol{x}))\phi(z_j^0(\boldsymbol{x}'))$ with respect to j. The input of the first hidden layer is denoted by $z_j^0(\boldsymbol{x}) = \sum_{k=1}^{d_{in}} w_{jk}^0 x_k + b_j^0$, and the expectation $\mathbb{E}(\cdot)$ is taken over the distribution of w_{ij}^0 and b_i^0. Actually, $z_j^0(\boldsymbol{x})$ is a GP only if the distribution D_0 is Gaussian. In this case, the covariance function of the GP $z_j^0(\boldsymbol{x})$ is

$$
\begin{aligned}
k^0(\boldsymbol{x},\boldsymbol{x}') = \mathbb{E}(z_i^0(\boldsymbol{x})z_i^0(\boldsymbol{x}')) &= \mathbb{E}\left(\left(\sum_{j=1}^{d_{in}} w_{ij}^0 x_j + b_i^0\right)\left(\sum_{k=1}^{d_{in}} w_{ik}^0 x_k' + b_i^0\right)\right) \\
&= \sum_{j=1}^{d_{in}} \mathbb{E}(w_{ij}^0)^2 x_j x_j' + \mathbb{E}(b_i^0)^2 \\
&= \sum_{j=1}^{d_{in}} \sigma_{w,0,j}^2 x_j x_j' + \sigma_{b,0}^2.
\end{aligned} \tag{3}
$$

The input vector components x_j and x_j' are deterministic and thus $\mathbb{E}(x_j x_j') = x_j x_j'$.

3. Two NN-induced covariance kernels of GP

The arguments of the previous section can be extended to deeper layers by induction. For layer l (≥ 2), using the multidimensional Central Limit Theorem, we learn that $z_i^l(\boldsymbol{x})$ and $z_i^{l-1}(\boldsymbol{x})$ are both GPs. Specifically, $z_i^l(\boldsymbol{x})$ has zero mean function and the covariance function computed as

$$
k^l(\boldsymbol{x},\boldsymbol{x}') = \mathbb{E}(z_i^l(\boldsymbol{x})z_i^l(\boldsymbol{x}')) = \sigma_{w,l}^2 \mathbb{E}\left[\phi(z_i^{l-1}(\boldsymbol{x}))\phi(z_i^{l-1}(\boldsymbol{x}'))\right] + \sigma_{b,l}^2. \tag{4}
$$

The expectation in Eq. (4) is taken over the GP governing $z_i^{l-1}(\boldsymbol{x})$, and this amounts to integrating over the joint distribution of two Gaussian random variables $z_i^{l-1}(\boldsymbol{x})$ and $z_i^{l-1}(\boldsymbol{x}')$ given $\boldsymbol{x}$ and $\boldsymbol{x}'$. The covariance function is rewritten as

$$
k^l(\boldsymbol{x},\boldsymbol{x}') = \sigma_{w,l}^2 \int_{\mathbb{R}^2} \phi(z_i^{l-1}(\boldsymbol{x}))\phi(z_i^{l-1}(\boldsymbol{x}'))p(z_i^{l-1}(\boldsymbol{x}), z_i^{l-1}(\boldsymbol{x}'))dz_i^{l-1}(\boldsymbol{x})dz_i^{l-1}(\boldsymbol{x}') + \sigma_{b,l}^2, \tag{5}
$$

where the density function of the joint distribution of $z_i^l(\boldsymbol{x})$ and $z_i^l(\boldsymbol{x}')$ is

$$
p(z_i^{l-1}(\boldsymbol{x}), z_i^{l-1}(\boldsymbol{x}')) = \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\begin{bmatrix} z_i^{l-1}(\boldsymbol{x}) \\ z_i^{l-1}(\boldsymbol{x}') \end{bmatrix}^T \Sigma^{-1} \begin{bmatrix} z_i^{l-1}(\boldsymbol{x}) \\ z_i^{l-1}(\boldsymbol{x}') \end{bmatrix}\right) \tag{6}
$$

with the covariance matrix

$$
\Sigma = \begin{bmatrix} \mathbb{E}(z_i^{l-1}(\boldsymbol{x}))^2 & \mathbb{E}(z_i^{l-1}(\boldsymbol{x})z_i^{l-1}(\boldsymbol{x}')) \\ \mathbb{E}(z_i^{l-1}(\boldsymbol{x}')z_i^{l-1}(\boldsymbol{x})) & \mathbb{E}(z_i^{l-1}(\boldsymbol{x}'))^2 \end{bmatrix} = \begin{bmatrix} k^{l-1}(\boldsymbol{x},\boldsymbol{x}) & k^{l-1}(\boldsymbol{x},\boldsymbol{x}') \\ k^{l-1}(\boldsymbol{x}',\boldsymbol{x}) & k^{l-1}(\boldsymbol{x}',\boldsymbol{x}') \end{bmatrix}. \tag{7}
$$

Eq. (5) is actually an iteration formula, given by

$$
\begin{aligned}
&k^l(\boldsymbol{x},\boldsymbol{x}') = \sigma_{w,l}^2 F_\phi\left(k^{l-1}(\boldsymbol{x},\boldsymbol{x}), k^{l-1}(\boldsymbol{x}',\boldsymbol{x}'), k^{l-1}(\boldsymbol{x},\boldsymbol{x}')\right) + \sigma_{b,l}^2, l = 1, 2, \cdots, L, \\
&k^0(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{\Lambda} \boldsymbol{x}' + \sigma_{b,0}^2,
\end{aligned} \tag{8}
$$

where the diagonal matrix $\boldsymbol{\Lambda}$ is defined by $\boldsymbol{\Lambda} = diag(\sigma_{w,0,1}^2, \sigma_{w,0,2}^2, \cdots, \sigma_{w,0,d_{in}}^2)$. Note that $k^{l-1}(\boldsymbol{x},\boldsymbol{x}') = k^{l-1}(\boldsymbol{x}',\boldsymbol{x})$ due to symmetry of the kernel. The trivariate function $F_\phi()$ represents the two-fold integral in Eq. (5), which depends on specific form of activation function $\phi(\cdot)$. For certain $\phi(\cdot)$, the function $F_\phi()$ is analytically tractable. Two iteration formulas that are analytically derivable will be given in the next two subsections.

3.1. Kernel from the ReLU nonlinearity

Letting the activation function be a rectifier $\phi(x) = \max(x, 0)$, we can derive the analytical expression for $F_\phi()$, and the corresponding iteration formula is [4,11]

$$
\begin{aligned}
&k^l(\boldsymbol{x},\boldsymbol{x}') = \frac{\sigma_{w,l}^2}{2\pi}\sqrt{k^{l-1}(\boldsymbol{x},\boldsymbol{x})k^{l-1}(\boldsymbol{x}',\boldsymbol{x}')}\left(\sin\theta^{l-1} + (\pi - \theta^{l-1})\cos\theta^{l-1}\right) + \sigma_{b,l}^2, l = 1, 2, \cdots, L, \\
&\theta^{l-1} = \cos^{-1}\left(\frac{k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{k^{l-1}(\boldsymbol{x},\boldsymbol{x})k^{l-1}(\boldsymbol{x}',\boldsymbol{x}')}}\right), \\
&k^0(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{x}^T \Lambda \boldsymbol{x}' + \sigma_{b,0}^2.
\end{aligned} \tag{9}
$$

Here we have increased the number of undetermined variances compared to the original formula in [4] in which $\sigma_{w,l}^2 = \sigma_w^2$, $\sigma_{b,l}^2 = \sigma_b^2$, and $\boldsymbol{\Lambda} = \frac{\sigma_w^2}{d_{in}}\boldsymbol{I}$. There are totally $d_{in} + 1 + 2L$ undetermined parameters $\boldsymbol{\theta}$: $\sigma_{w,0,j}^2$ ($j = 1, 2, \cdots, d_{in}$), $\sigma_{b,0}^2, \sigma_{w,l}^2$

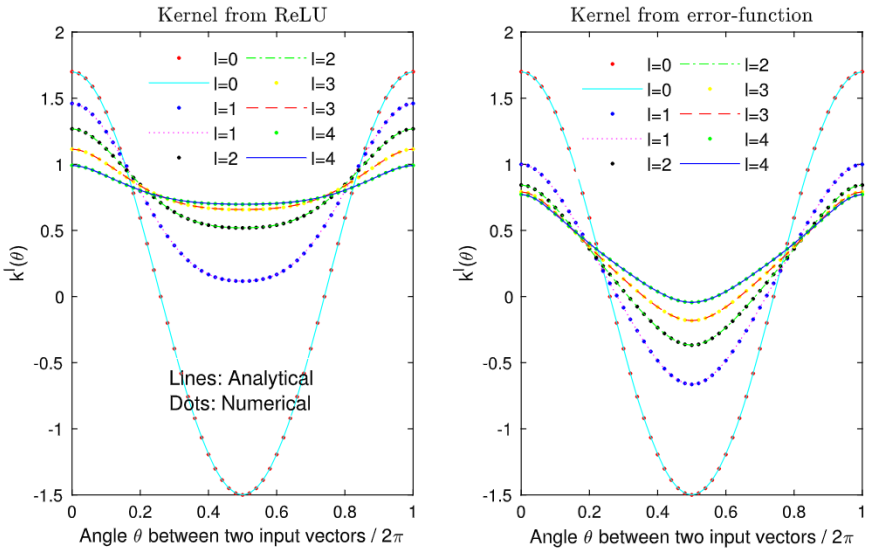

Fig. 2. Comparison of the analytical and numerical iteration formulas for the covariance functions from the ReLU (left) and error-function (right) nonlinearities. The covariance function $k^l(\boldsymbol{x},\boldsymbol{x}')$ of the GP $z_i^l(\boldsymbol{x})$ is computed analytically and numerically for $l=0,1,2,3,4(L=4)$. For ease of comparison, the input vectors $\boldsymbol{x},\boldsymbol{x}'\in\mathbb{R}^2$ ($d_{in}=2$) are normalized to $||\boldsymbol{x}||_2=||\boldsymbol{x}'||_2=1$. The angle between the two vectors is $\theta=\cos^{-1}(\boldsymbol{x}^T\boldsymbol{x}')$. All the weight variances are set to be 1.6 and the bias variances are 0.1.

($l=1,2,\cdots,L$), and $\sigma_{b,l}^2$ ($l=1,2,\cdots,L$). These parameters (a.k.a. hyperparameters in GP regression) can be learned by using maximum likelihood estimation.

Since the trivariate function $F_\phi()$ is not analytically tractable for a generic nonlinearity, such as hyperbolic tangent, sigmoid, exponential linear unit (ELU), the authors of [4] developed an efficient Gaussian quadrature scheme to approximate the two-fold integral in Eq. (5). Fig. 2 (left) compares the numerical results computed by the Gaussian quadrature scheme with the analytical results computed by Eq. (9), showing good agreement.

3.2. Kernel from the error-function nonlinearity

The authors of [11] have proposed a general strategy to construct kernels for multilayer inducing NNs, based on the base kernel in single-layer NNs. Nevertheless, the authors only considered the base kernel induced by the ReLU-activated NNs, and derived the corresponding analytical iteration formula (9). In this paper, we consider a different base kernel, which is induced by the error-function-activated NNs. Actually, the base kernel has already been derived in [3], but it has not been realized that this base kernel can also be incorporated in the aforementioned general strategy to generate an analytical iteration formula similar to (9).

We first briefly review the general strategy proposed in [11]. It is well known that a kernel function can be represented by the inner product of two vectors in feature space: $k(\boldsymbol{x},\boldsymbol{x}')=\psi(\boldsymbol{x})\cdot\psi(\boldsymbol{x}')$, where $\psi(\boldsymbol{x})$ projects the input $\boldsymbol{x}$ into feature space (see section 2.1.2 of [12]). Letting the base kernel $k^1(\boldsymbol{x},\boldsymbol{x}')=\psi(\boldsymbol{x})\cdot\psi(\boldsymbol{x}')$ and assuming $\sigma_{w,l}^2\equiv 1,\sigma_{b,l}^2\equiv 0$ for brevity of notation, we can construct the kernel function for the next hidden layer by composing the projection operator $\psi(\cdot)$:

$$k^2(\boldsymbol{x},\boldsymbol{x}')=\psi(\psi(\boldsymbol{x}))\cdot\psi(\psi(\boldsymbol{x}'))=k^1(\psi(\boldsymbol{x}),\psi(\boldsymbol{x}')). \tag{10}$$

Generally, we have the recursive formula

$$k^l(\boldsymbol{x},\boldsymbol{x}')=k^{l-1}(\psi(\boldsymbol{x}),\psi(\boldsymbol{x}')),\quad l=2,3,\cdots,L. \tag{11}$$

If the base kernel $k^1(\cdot,\cdot)$ is analytically tractable, we can derive the kernels for all other hidden layers using the above formula. For the ReLU nonlinearity, the base kernel reads [11]

$$k^1(\boldsymbol{x},\boldsymbol{x}')=\frac{1}{2\pi}\|\boldsymbol{x}\|_2\|\boldsymbol{x}'\|_2(\sin\theta^0+(\pi-\theta^0)\cos\theta^0), \tag{12}$$

where $\theta^0=\cos^{-1}\left(\frac{k^0(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{k^0(\boldsymbol{x},\boldsymbol{x})k^0(\boldsymbol{x}',\boldsymbol{x}')}}\right)$ and $k^0(\boldsymbol{x},\boldsymbol{x}')=\boldsymbol{x}\cdot\boldsymbol{x}'$. Due to $\|\psi(\boldsymbol{x})\|_2=\sqrt{k^1(\boldsymbol{x},\boldsymbol{x})}$, it follows from (10) that

$$k^2(\boldsymbol{x},\boldsymbol{x}')=k^1(\psi(\boldsymbol{x}),\psi(\boldsymbol{x}'))=\frac{1}{2\pi}\sqrt{k^1(\boldsymbol{x},\boldsymbol{x})}\sqrt{k^1(\boldsymbol{x}',\boldsymbol{x}')}(\sin\gamma+(\pi-\gamma)\cos\gamma), \tag{13}$$

where

$$\gamma = \cos^{-1}\left(\frac{k^0(\psi(\boldsymbol{x}),\psi(\boldsymbol{x}'))}{\sqrt{k^0(\psi(\boldsymbol{x}),\psi(\boldsymbol{x}))k^0(\psi(\boldsymbol{x}'),\psi(\boldsymbol{x}'))}}\right) = \cos^{-1}\left(\frac{k^1(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{k^1(\boldsymbol{x},\boldsymbol{x})k^1(\boldsymbol{x}',\boldsymbol{x}')}}\right) = \theta^1. \tag{14}$$

It is straightforward to obtain the kernels $k^l(\cdot,\cdot)$ for $l > 2$. We see that we have derived the iteration formula (9) for $\sigma^2_{w,l} \equiv 1, \sigma^2_{b,l} \equiv 0$.

When $\sigma^2_{w,l} \equiv 1, \sigma^2_{b,l} \equiv 0$, the base kernel for the error-function nonlinearity reads [3]

$$k^1(\boldsymbol{x},\boldsymbol{x}') = \frac{2}{\pi}\sin^{-1}\left(\frac{2k^0(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{(1+2\|\boldsymbol{x}\|_2^2)(1+2\|\boldsymbol{x}'\|_2^2)}}\right). \tag{15}$$

Analogous to the ReLU case, we can finally derive the iteration formula for the error-function case:

$$\begin{aligned} k^l(\boldsymbol{x},\boldsymbol{x}') &= \frac{2\sigma^2_{w,l}}{\pi}\sin^{-1}\left(\frac{2k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{(1+2k^{l-1}(\boldsymbol{x},\boldsymbol{x}))(1+2k^{l-1}(\boldsymbol{x}',\boldsymbol{x}'))}}\right) + \sigma^2_{b,l}, l = 1,2,\cdots,L, \\ k^0(\boldsymbol{x},\boldsymbol{x}') &= \boldsymbol{x}^T\Lambda\boldsymbol{x}' + \sigma^2_{b,0}. \end{aligned} \tag{16}$$

To validate the above formula, we compare the analytical results from the above formula with the numerical results from the aforementioned Gaussian quadrature scheme, and as we see in Fig. 2 (right) the new iteration formula (16) is correct.

We specifically call the GPs with the kernels that are induced iteratively by NNs "NNGP" for short. NNGP is actually a specific case of GP, but we still distinguish it from the standard GP involving an *ad hoc* kernel (e.g., square exponential and Matern kernels). An exception is the output of the first hidden layer, $x_i^1(\boldsymbol{x})$. The output is also a GP and its covariance function is [3,12]

$$k(\boldsymbol{x},\boldsymbol{x}') = \frac{2}{\pi}\sin^{-1}\left(\frac{2k^0(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{\left(1+2k^0(\boldsymbol{x},\boldsymbol{x})\right)\left(1+2k^0(\boldsymbol{x}',\boldsymbol{x}')\right)}}\right). \tag{17}$$

We still regard the GP with this kernel as a standard GP rather than a NNGP, as the kernel $k(\boldsymbol{x},\boldsymbol{x}')$ corresponds to an incomplete, shallow NN without output layer. In Section 5.2.2 we will compare the predictive ability of this kernel with our generic kernel $k^l(\boldsymbol{x},\boldsymbol{x}')$ computed by formula (16).

4. GP regression

The essence of GP regression is the conditional distribution of quantities of interest, $\boldsymbol{q}$, given the known observations $\boldsymbol{o}$, where $\boldsymbol{q}$ and $\boldsymbol{o}$ are both Gaussian random vectors. Thanks to the analytical tractability of Gaussian distribution, we can derive analytically mean vector and covariance matrix of the conditional distribution. Supposed that $\boldsymbol{q}$ and $\boldsymbol{o}$ have the joint multivariate Gaussian distribution with zero mean vector and block-wise covariance matrix

$$\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{o}\end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix}\boldsymbol{K}_{qq} & \boldsymbol{K}_{qo}\\ \boldsymbol{K}_{oq} & \boldsymbol{K}_{oo}\end{bmatrix}\right), \tag{18}$$

the conditional distribution $p(\boldsymbol{q}|\boldsymbol{o})$ is also a multivariate Gaussian distribution with the mean vector

$$\boldsymbol{m}(\boldsymbol{q}) = \boldsymbol{K}_{qo}\boldsymbol{K}_{oo}^{-1}\boldsymbol{o} \tag{19}$$

and the covariance matrix

$$\boldsymbol{K}(\boldsymbol{q}) = \boldsymbol{K}_{qq} - \boldsymbol{K}_{qo}\boldsymbol{K}_{oo}^{-1}\boldsymbol{K}_{oq} \tag{20}$$

(see (A.5) and (A.6) of [12]). Zero-mean and σ^2_{noise}-variance Gaussian white noise is assumed in observations.

Note that observations and quantities of interest are usually evaluated at specific time-space coordinates. To preserve a multivariate Gaussian distribution (18) at arbitrary time-space coordinates, the observations and the quantities of interest can be seen as samples of GPs. In other words, $\boldsymbol{o} = [X_o(\boldsymbol{x}_1),\cdots,X_o(\boldsymbol{x}_{N_o})]^T$ is a sample of $X_o(\boldsymbol{x}) \sim \mathcal{GP}(0, k_o(\boldsymbol{x},\boldsymbol{x}'))$ and $\boldsymbol{q} = [X_q(\boldsymbol{x}_1),\cdots,X_q(\boldsymbol{x}_{N_q})]^T$ is a sample of $X_q(\boldsymbol{x}) \sim \mathcal{GP}(0, k_q(\boldsymbol{x},\boldsymbol{x}'))$, where the index set $\boldsymbol{x}$ represents arbitrary time and/or space coordinates. Particularly, for solving PDEs, the observations $\boldsymbol{o}$ consist of samples from different GPs. Different covariance function $k_o(\cdot,\cdot)$ will capture different trends and features that the observations $\boldsymbol{o}$ exhibit.

Before computing mean and covariance of conditional distribution from formulas (19) and (20), we need to first learn parameters of covariance function as well as the noise variance σ^2_{noise}. These parameters are also known as hyperparameters and can be learned by using maximum likelihood estimation (MLE) [12,13]. In practice, we find the optimal hyperparameters (including undetermined parameters $\boldsymbol{\theta}$ in covariance function and the noise variance σ^2_{noise}) that minimize the negative log marginal-likelihood [12]

$$\frac{1}{2}\boldsymbol{o}^T \boldsymbol{K}_{oo}^{-1}(\boldsymbol{\theta}, \sigma_{noise}^2)\boldsymbol{o} + \frac{1}{2}\log|\boldsymbol{K}_{oo}(\boldsymbol{\theta}, \sigma_{noise}^2)| + \frac{N_o}{2}\log 2\pi, \tag{21}$$

where $|\cdot|$ denotes the determinant and the covariance matrix for observations (or training data) $\boldsymbol{K}_{oo}$ depends on $\boldsymbol{\theta}$ and σ_{noise}^2.

In what follows, we will show how the GP regression can be adapted to different problems of function approximation and PDE solution by selecting different forms of observations $\boldsymbol{o}$.

4.1. Function approximation

Let $\boldsymbol{q} = [f(\boldsymbol{x})]$ and $\boldsymbol{o} = [f(\boldsymbol{x}_1) + \epsilon, \cdots, f(\boldsymbol{x}_{N_o}) + \epsilon]^T$ where ϵ is Gaussian white noise having variance σ_{noise}^2. The objective is to approximate the unknown scalar-valued function $f(\boldsymbol{x})$ given the observations $\boldsymbol{o}$. After prescribing the covariance function $k_o(\cdot,\cdot)$ for computing the covariance matrix $\boldsymbol{K}_{oo}$ and then training the regression model by the MLE, we derive the function approximation by using formula (19). Moreover, the use of the variance computed from (20) yields the confidence interval. Denoting by $\boldsymbol{X}$ (N_o by d_{in} matrix) the collection of the evaluation points $\{\boldsymbol{x}_j\}$ (usually called training inputs), we formulate the covariance matrix in (18) by:

$$\begin{aligned}
&\boldsymbol{K}_{oo} = \left[k_o(\boldsymbol{X}, \boldsymbol{X}) + \sigma_{noise}^2 \boldsymbol{I}\right],\\
&\boldsymbol{K}_{qo} = \boldsymbol{K}_{oq}^T = [k_o(\boldsymbol{x}, \boldsymbol{X})],\\
&\boldsymbol{K}_{qq} = [k_o(\boldsymbol{x}, \boldsymbol{x})].
\end{aligned} \tag{22}$$

4.2. PDE solution

First we consider the following time-independent linear PDE

$$\begin{aligned}
&L_{\boldsymbol{x}} u(\boldsymbol{x}) = f(\boldsymbol{x}), \boldsymbol{x} \in \Omega \in \mathbb{R}^{d_{in}},\\
&u(\boldsymbol{x}) = g(\boldsymbol{x}), \boldsymbol{x} \in \partial\Omega,
\end{aligned} \tag{23}$$

where $L_{\boldsymbol{x}}$ is a linear differential operator. The source term $f(\cdot)$ and the boundary condition term $g(\cdot)$ are the only information we know, and thus we put them in the observations by letting $\boldsymbol{o} = [\boldsymbol{o}_u^T, \boldsymbol{o}_f^T]^T = [g(\boldsymbol{x}_{u,1}) + \epsilon_u, \cdots, g(\boldsymbol{x}_{u,N_u}) + \epsilon_u, f(\boldsymbol{x}_{f,1}) + \epsilon_f, \cdots, f(\boldsymbol{x}_{f,N_f}) + \epsilon_f]^T$, where we select N_u boundary points as the evaluation points for $\boldsymbol{o}_u$ and N_f domain points as the evaluation points for $\boldsymbol{o}_f$. The variances of the Gaussian white noises ϵ_u and ϵ_f are σ_u^2 and σ_f^2, respectively. We assume that the solution $\boldsymbol{q} = [u(\boldsymbol{x})]$ is a GP with zero mean and covariance function $k_u(\boldsymbol{x}, \boldsymbol{x}')$, and the source term is another GP that depends on u, whose covariance function is written as [6]

$$k_f(\boldsymbol{x}, \boldsymbol{x}') = L_{\boldsymbol{x}} L_{\boldsymbol{x}'} k_u(\boldsymbol{x}, \boldsymbol{x}'). \tag{24}$$

Denoting by $\boldsymbol{X}_u$ (N_u by d_{in} matrix) the collection of the boundary evaluation points $\{\boldsymbol{x}_{u,j}\}$ and by $\boldsymbol{X}_f$ (N_f by d_{in} matrix) the collection of domain evaluation points $\{\boldsymbol{x}_{f,j}\}$, we formulate the covariance matrix in (18) as [6]

$$\begin{aligned}
&\boldsymbol{K}_{oo} = \begin{bmatrix} k_u(\boldsymbol{X}_u, \boldsymbol{X}_u) + \sigma_{noise,u}^2 \boldsymbol{I} & L_{\boldsymbol{x}'} k_u(\boldsymbol{X}_u, \boldsymbol{X}_f) \\ L_{\boldsymbol{x}} k_u(\boldsymbol{X}_f, \boldsymbol{X}_u) & L_{\boldsymbol{x}} L_{\boldsymbol{x}'} k_u(\boldsymbol{X}_f, \boldsymbol{X}_f) + \sigma_{noise,f}^2 \boldsymbol{I} \end{bmatrix},\\
&\boldsymbol{K}_{qo} = \boldsymbol{K}_{oq}^T = \begin{bmatrix} k_u(\boldsymbol{x}, \boldsymbol{X}_u) & L_{\boldsymbol{x}'} k_u(\boldsymbol{x}, \boldsymbol{X}_f) \end{bmatrix},\\
&\boldsymbol{K}_{qq} = [k_u(\boldsymbol{x}, \boldsymbol{x})].
\end{aligned} \tag{25}$$

For time-dependent problem, the above GP regression model also works as we can regard the time variable as the $(d_{in}+1)$-th dimension of space-time domain (see Section 4.3.1 of [6]).

An alternative way to handle the time-dependent problems is numerical GP regression [7]. Consider the following time-dependent PDE

$$\begin{aligned}
&\frac{\partial u(\boldsymbol{x}, t)}{\partial t} = \tilde{L}_{\boldsymbol{x}} u(\boldsymbol{x}, t), \boldsymbol{x} \in \Omega \in \mathbb{R}^{d_{in}},\\
&u(\boldsymbol{x}, 0) = h(\boldsymbol{x}),\\
&u(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \boldsymbol{x} \in \partial\Omega.
\end{aligned} \tag{26}$$

The differential operator $\tilde{L}_{\boldsymbol{x}}$ can be nonlinear (say, Burgers' equation). We denote by $L_{\boldsymbol{x}}$ the linearized counterpart of $\tilde{L}_{\boldsymbol{x}}$. For linear operator, $L_{\boldsymbol{x}} = \tilde{L}_{\boldsymbol{x}}$. The use of Eular backward scheme leads to $u^n(\boldsymbol{x}) - \Delta t L_{\boldsymbol{x}} u^n(\boldsymbol{x}) = u^{n-1}(\boldsymbol{x})$. According to the known information from initial-boundary conditions, we define the observations at n-th time step as $\boldsymbol{o}^n = [\boldsymbol{o}_n^T, \boldsymbol{o}_{n-1}^T]^T = [u^n(\boldsymbol{x}_{b,1}) +$

$\epsilon_n, \cdots, u^n(\boldsymbol{x}_{b,N_b}) + \epsilon_n, u^{n-1}(\boldsymbol{x}_1) + \epsilon_{n-1}, \cdots, u^{n-1}(\boldsymbol{x}_{N^{n-1}}) + \epsilon_{n-1}]^T$ and $\boldsymbol{q}^n = [u^n(\boldsymbol{x})]$. Note that $u^n(\boldsymbol{x}_{b,j}) = g(\boldsymbol{x}_{b,j}, n\Delta t)$ and $u^0(\boldsymbol{x}_j) = h(\boldsymbol{x}_j)$. Thus, we actually apply GP regression at each time step. Assuming $u^n(\boldsymbol{x})$ is a GP with zero mean and covariance kernel $k_{u^n}(\boldsymbol{x}, \boldsymbol{x}')$ and denoting by $\boldsymbol{X}_b^n$ (N_b by d_{in} matrix) the collection of the boundary evaluation points selected at n-th time step and by $\boldsymbol{X}^{n-1}$ (N^{n-1} by d_{in} matrix) the collection of the domain evaluation points sampled at $(n-1)$-th time step, we formulate the covariance matrix as [7]

$$\begin{aligned}
\boldsymbol{K}_{o^n o^n} &= \begin{bmatrix} k_{u^n}(\boldsymbol{X}_b^n, \boldsymbol{X}_b^n) + \sigma_{noise,n}^2 \boldsymbol{I} & (\boldsymbol{I} - \Delta t L_{\boldsymbol{x}'}) k_{u^n}(\boldsymbol{X}_b^n, \boldsymbol{X}^{n-1}) \\ (\boldsymbol{I} - \Delta t L_{\boldsymbol{x}}) k_{u^n}(\boldsymbol{X}^{n-1}, \boldsymbol{X}_b^n) & (\boldsymbol{I} - \Delta t L_{\boldsymbol{x}})(\boldsymbol{I} - \Delta t L_{\boldsymbol{x}'}) k_{u^n}(\boldsymbol{X}^{n-1}, \boldsymbol{X}^{n-1}) + \sigma_{noise,n-1}^2 \boldsymbol{I} \end{bmatrix}, \\
\boldsymbol{K}_{q^n o^n} &= \boldsymbol{K}_{o^n q^n}^T = \begin{bmatrix} k_{u^n}(\boldsymbol{x}, \boldsymbol{X}_b^n) & (\boldsymbol{I} - \Delta t L_{\boldsymbol{x}'}) k_{u^n}(\boldsymbol{x}, \boldsymbol{X}^{n-1}) \end{bmatrix}, \\
\boldsymbol{K}_{q^n q^n} &= [k_{u^n}(\boldsymbol{x}, \boldsymbol{x})].
\end{aligned} \tag{27}$$

Generally, we let the covariance function be the same for all n, namely $k_{u^n}(\boldsymbol{x}, \boldsymbol{x}') = k_o(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}_n)$, except with varied undetermined parameters $\boldsymbol{\theta}_n$. Considering the propagation of the uncertainty in time marching, we need to take into account the uncertainty of the predictions in previous time step (i.e., $u^{n-1}(\boldsymbol{x})$ predicted at $\boldsymbol{X}^{n-1}$). We rewrite the predictive or posterior variance in (20) as [7]

$$\begin{aligned}
\boldsymbol{K}(\boldsymbol{q}^n) = &\boldsymbol{K}_{q^n q^n} - \boldsymbol{K}_{q^n o^n} \boldsymbol{K}_{o^n o^n}^{-1} \boldsymbol{K}_{o^n q^n} \\
&+ \boldsymbol{K}_{q^n o^n} \boldsymbol{K}_{o^n o^n}^{-1} \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{K}(u^{n-1}(\boldsymbol{X}^{n-1})) \end{bmatrix} \boldsymbol{K}_{o^n o^n}^{-1} \boldsymbol{K}_{o^n q^n}.
\end{aligned} \tag{28}$$

5. Numerical results

We perform a three-way comparison between NNGP, NN, and GP for function approximation and PDE solution for prototypical problems. The hyperparameters of NNGP and GP are trained by minimizing the negative log marginal-likelihood function. The conjugate gradient algorithm (see the function *minimize*() in GPML Matlab code (version 3.6) [14]) is employed for the optimization.

Because of the non-convex nature of the objective function, to avoid trapping into the local minima, we run our optimization algorithm code ten times with different initializations. Among the ten groups of optimized hyperparameters, for GP/NNGP we choose the one yielding the smallest negative log marginal-likelihood, and for NN we select the one producing the lowest loss for NN. The initializations of hyperparameters for GP/NNGP are taken as the first ten entries of the Halton quasi-random sequence [15] and these initializations are kept deterministic. The initializations of network weights are obtained from Xavier's initialization algorithm [16], which is provided by the TensorFlow package. Also, for each initialization at most 200 function evaluations are allowed in conjugate gradient search in GP/NNGP. For NNGP, the central finite difference scheme with the step size 10^{-4} is used in computing the gradients of covariance function with respect to hyperparameters while for GP the analytical derivation is employed. Additionally, we compute l_2 relative error $||\boldsymbol{u}_{approximate} - \boldsymbol{u}_{exact}||_2 / ||\boldsymbol{u}_{exact}||_2$ in quantifying accuracy of function approximation and PDE solution, where $\boldsymbol{u}$ is a vector formed by the function values or PDE's solutions evaluated at test points. For function approximation examples and solving the Poisson equation, we adopt in NN the Adam algorithm implemented in TensorFlow package to minimize loss functions. For solving the Burgers' equation, we adopt the continuous-time NN approach instead of the discrete-time one since the former can yield slightly high accuracy compared to the latter [8]. The L-BFGS-B optimizer in the TensorFlow package is employed to minimize the loss function for the Burgers' equation in order to achieve faster convergence compared to the Adam optimizer case.

5.1. Function approximation

The covariance matrices of GP/NNGP are formulated according to (22).

5.1.1. Step function

We approximate the step function $f(x) = 1, x \geq 0$; $f(x) = 0$, otherwise. Ten evenly distributed training inputs and 100 equispaced test points are chosen. Fig. 3 shows the approximation by the GPs with two commonly used kernels: squared exponential (SE) and Matern [12]. The first kernel can describe well the data exhibiting smooth trend while the second one is suitable for finite regularity. However for the step function, a non-smooth function, the performance of these kernels is poor. The NNGP predictions shown in Figs. 4 and 5 are much better. In the domain away from the discontinuity, the NNGP approximation is good with small uncertainty. Particularly, the ReLU-induced kernel even succeeds in capturing the discontinuity. The uncertainty around $x = 0$ is obviously larger than that in other regions. This is probably caused by the missing information between two training points adjacent to $x = 0$. This explanation also works for the error-function-induced kernel in Fig. 4. An important observation is that the ReLU induced kernel is more suitable for non-smooth functions than the error-function induced kernel. Another observation regarding NNGP is that deepening the inducing NN does not change the approximation accuracy (kept in the same order of magnitude) for the step function. One possible explanation for this

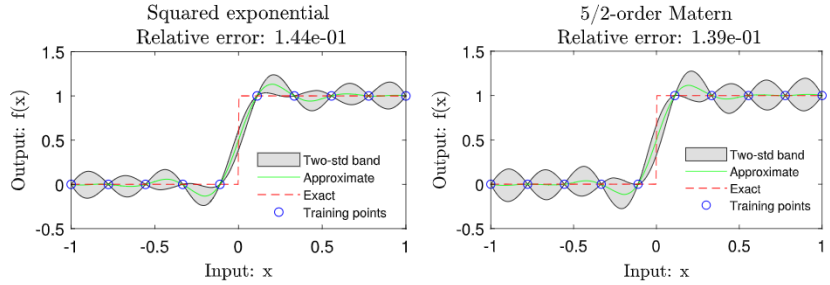

Fig. 3. Approximating the step function: GPs with SE (left) and Matern (right) kernels. "Two-std band" denotes the mean vector of the conditional distribution ($m(\boldsymbol{q})$ computed from (19)) plus/minus twice the standard deviation (diagonals of $\sqrt{K(\boldsymbol{q})}$ computed from (20)). The quantities of interest are $\boldsymbol{q} = f(\boldsymbol{X}_t)$ where $\boldsymbol{X}_t$ (100 by 1 matrix) is the collection of test points.

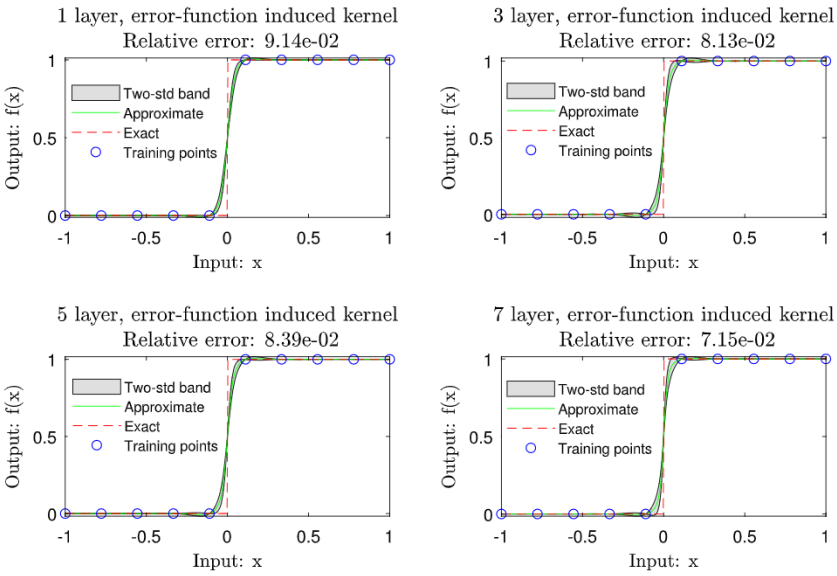

Fig. 4. Approximating the step function: NNGP with error-function induced kernel. "One-layer" indicates that there is totally one hidden layer in the inducing NN ($L = 1$).

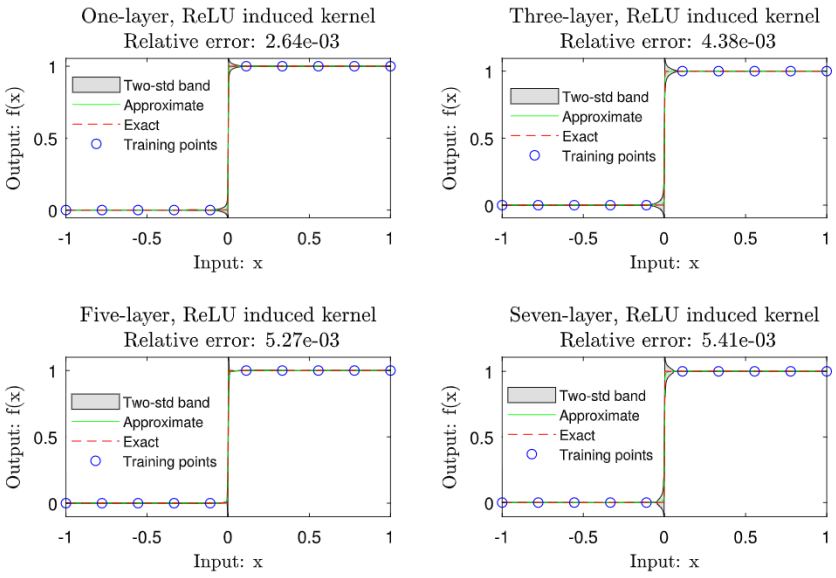

Fig. 5. Approximating the step function: NNGP with the ReLU induced kernel.

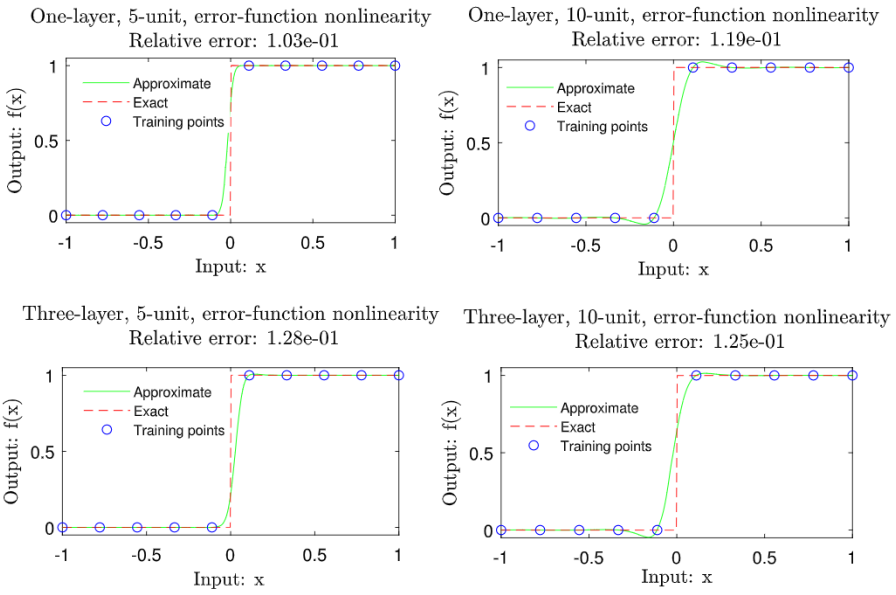

Fig. 6. Approximating the step function: NN with the error-function nonlinearity ("5-unit" indicates that there exist 5 units/neurons in each hidden layer).

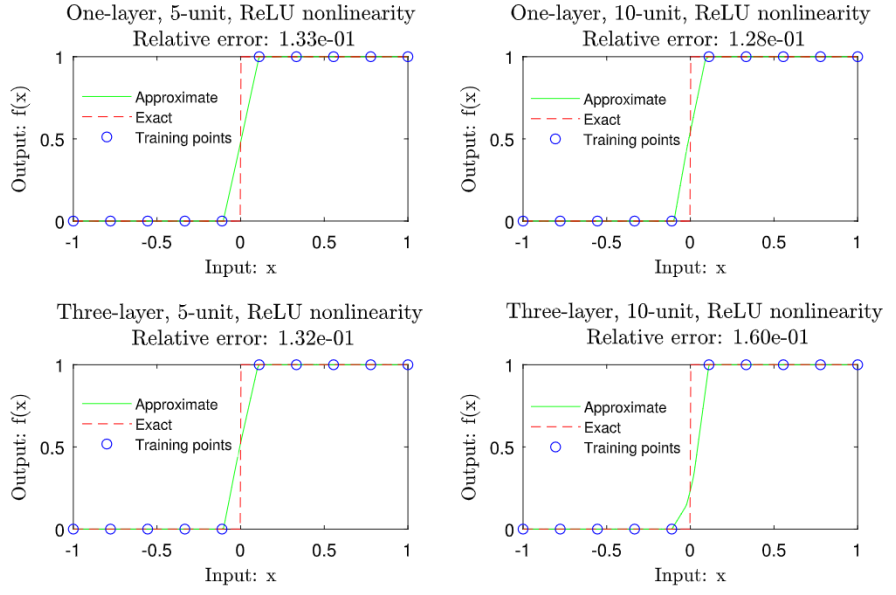

Fig. 7. Approximating the step function: NN with the ReLU nonlinearity.

observation is that choosing a deeper inducing NN will produce more hyperparameters, which are slightly more difficult to train. In principle, the deeper the inducing NN is, the more random initial guesses for the hyperparameters we need to select for optimizing likelihood function. Figs. 6 and 7 show the predictions of NN, which are less accurate than NNGP's. Unlike GP/NNGP, NN does not quantify any uncertainty for approximation.

It should also be noted that if letting weight and bias variances be kept the same across layers, which was considered in [4], we can also obtain accurate results using the ReLU nonlinearity. However, this is only the case for shallow inducing NNs. As Fig. 8 demonstrates, with the increasing depth of inducing NNs, the performance of NNGP degrades. This could imply that as the inducing NNs become deeper, we need to introduce more free parameters to release the approximation ability of NNGP. Only two parameters σ_w^2 and σ_b^2 were considered in [4] and this is inconsistent with the high expressivity of a deep NN. Therefore, in the following numerical examples, we will only consider the weights and biases whose variances vary across layers.

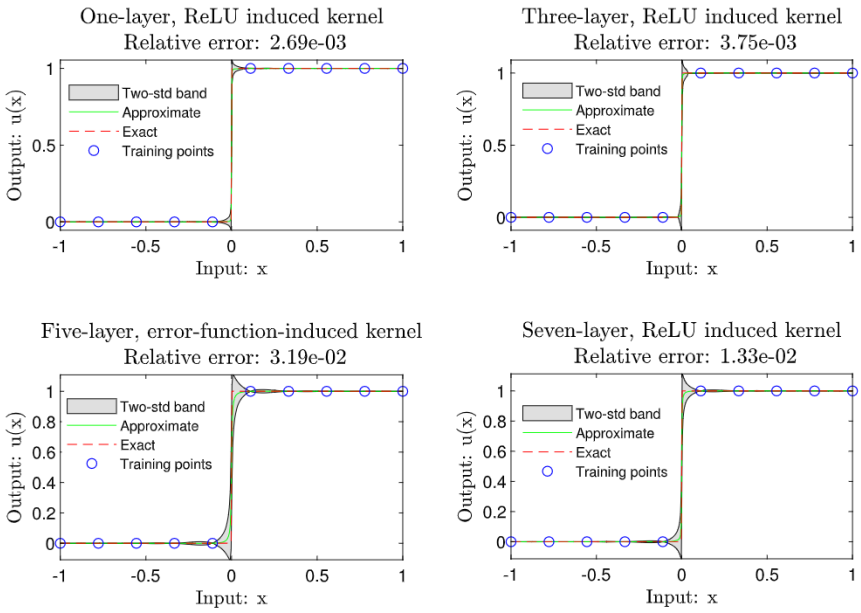

Fig. 8. Approximating the step function: NNGP with the ReLU induced kernel. Here, following [4], the weights and biases are assumed to have the same variances across layers, namely $\sigma_{w,l}^2 \equiv \sigma_w^2$ for $l > 0$, $\sigma_{b,l}^2 \equiv \sigma_b^2$ for $l \geq 0$ and $\sigma_{w,0,j}^2 \equiv \sigma_w^2$.

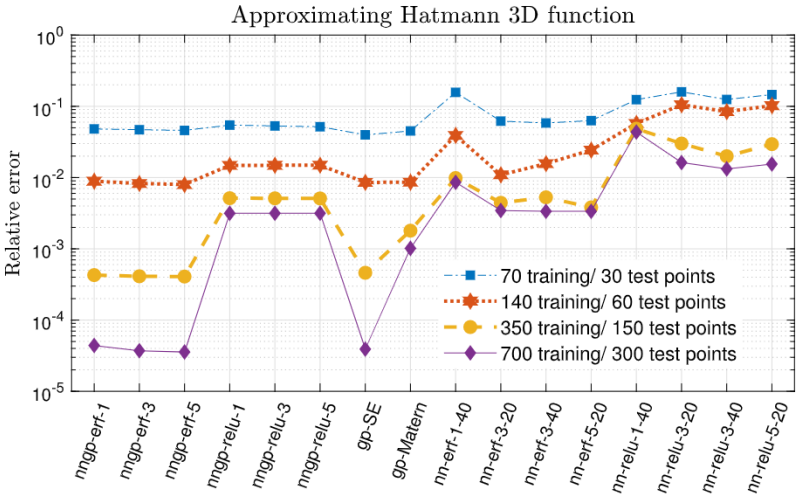

Fig. 9. Approximating the Hartman 3D function: error comparison. ("nngp-erf-1" means the NNGP with the one hidden layer and error-function induced kernel; "gp-SE" denotes the GP with the squared exponential kernel; "nn-erf-3-20" indicates the three-layer, 20-unit-wide NN with the error-function nonlinearity.)

5.1.2. Hartmann 3D function

The Hartmann function is frequently used for testing global optimization algorithms. Here we consider the trivariate case. The function is much smoother than the step function and thus we can expect GP to perform well. To test the approximation accuracy, we first generate N ($N = 100, 200, 500$, and 1000) points by choosing the first N entries of the Halton sequence, and then randomly permutate the points, followed by selecting the first 70% points as training points and the remaining 30% points as the test points.

Fig. 9 compares NN, NNGP, and GP in terms of approximation accuracy. Since the Hartmann 3D function is smooth, the NNGP with error-function induced kernel and the GP with squared exponential kernel are both very accurate. ReLU is not suitable for smooth function in contrast to approximating the non-smooth step function. Additionally, the GP and NNGP both outperform NN. It is observed again that NN does not give any uncertainty estimate, whereas GP and NNGP do. Another observation is that increasing the depth of the inducing NN in NNGP does not change the accuracy.

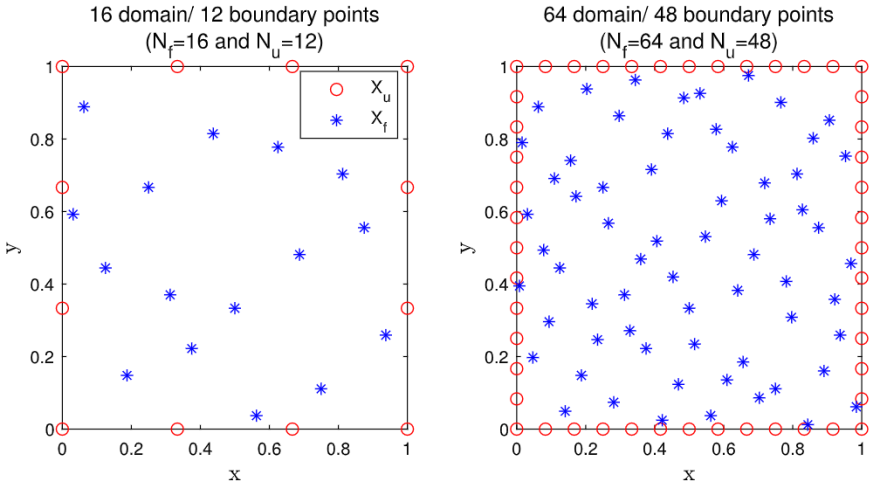

Fig. 10. Solving 2D Poisson equation: Two setups of training inputs for both fabricated solutions.

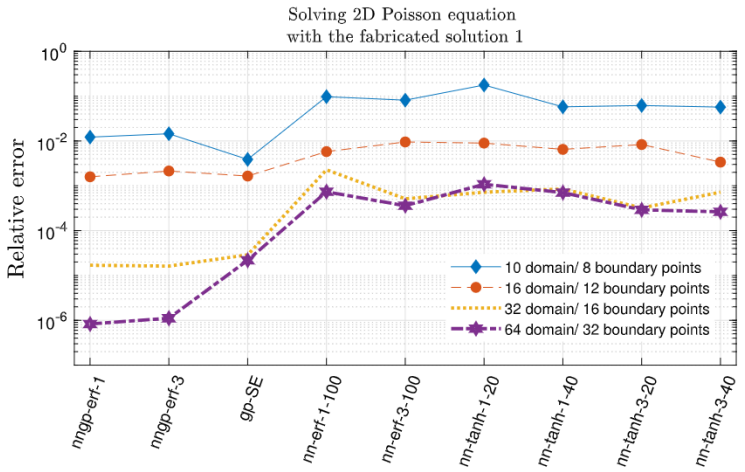

Fig. 11. Solving 2D Poisson equation with the exact solution $u(x, y) = \sin(\pi x)(y^2 + \exp(-y))$. ("nn-tanh-3-20" represents the three-layer, 20-unit-wide NN with the hyperbolic tangent nonlinearity.

5.2. PDE solution

We use the proposed covariance function from error-function nonlinearity to solve the following two PDEs. We replace the covariance function $k_u(\boldsymbol{x}, \boldsymbol{x}')$ in Eq. (25) as well as $k_{u^n}(\boldsymbol{x}, \boldsymbol{x}')$ in Eq. (27) with the new kernel (16). The derivation of kernel's derivatives for the case of Poisson equation is given in Appendix.

5.2.1. 2D Poisson equation

Consider the equation

$$-\Delta u(x, y) = f(x, y), (x, y) \in (0, 1)^2, \tag{29}$$

with two fabricated solutions (1) $\sin(\pi x)(y^2 + \exp(-y))$ and (2) $\sin(\pi x)\cos(2\pi(y^2 + x))$. Dirichlet boundary conditions are assumed. The second solution is more complex than the first one and thus we use more training points in the second solution case. The training inputs $\boldsymbol{X}_f$ are chosen from the first N_f entries of the Halton sequence and $\boldsymbol{X}_u$ is equispaced on the boundary, which are shown in Fig. 10.

The covariance matrices of GP/NNGP are obtained by (25). Figs. 11 and 12 give the error plots of NN, NNGP, and GP for fabricated solution 1 and 2, respectively. Totally 441 evenly distributed test points are taken to evaluate the relative error. The NN results are obtained according to the approach proposed in [8]. To keep a fair comparison, the same training inputs are selected in the NN case. It is observed from the figures that for different fabricated solutions, the approximation accuracy of the GP and NNGP is comparable. Also, the NN accuracy is nearly one order of magnitude lower than that of GP/NNGP.

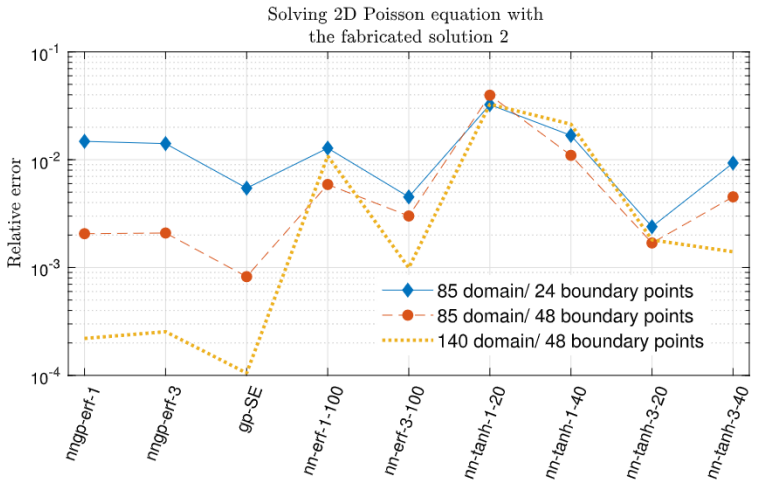

Fig. 12. Solving 2D Poisson equation with the exact solution $u(x, y) = \sin(\pi x)\cos(2\pi(y^2 + x))$.

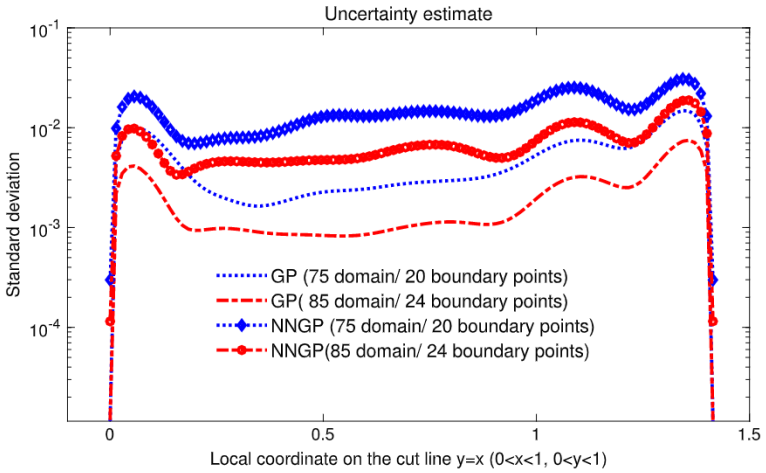

Fig. 13. Solving 2D Poisson equation with the fabricated solution 2: uncertainty estimates for GP and one-layer NNGP evaluated on the cut line $y = x$. The y-axis represents the standard deviation of the conditional or posterior distribution computed by (20). The relative errors evaluated on the cut line are 0.049, 0.029, 0.022, and 0.0081 from the top curve to the bottom one. It is seen that the uncertainty is strongly correlated with the prediction error.

Fig. 13 shows the uncertainty estimates of GP and one-layer NNGP evaluated on the cut line $y = x$ of the square domain, for the case of fabricated solution 2. We see that for both methods uncertainty is reduced with increasing number of training points. Moreover, the uncertainty is strongly correlated with the prediction error as smaller uncertainty corresponds to lower error. Note that the uncertainty at the endpoints of the cut line, namely, $(x, y) = (0, 0)$ and $(x, y) = (1, 1)$, is exactly zero, due to the boundary condition.

5.2.2. 1D Burgers' equation

Consider the equation [7]

$$\frac{\partial u(x,t)}{\partial t} + u(x,t)\frac{\partial u(x,t)}{\partial x} = \frac{0.01}{\pi}\frac{\partial^2 u(x)}{\partial x^2}, (x,t) \in [0,1]^2, \tag{30}$$

with $u(-1, t) = u(1, t) = 0$. The initial condition is $u(x, 0) = -\sin(\pi x)$. After the linearization through replacing the nonlinear term $u^n \frac{\partial u^n}{\partial x}$ by $\mu^{n-1}\frac{\partial u^n}{\partial x}$, where μ^n is the mean vector computed for the previous time step by (19), we derive the differential operator $L_x = \frac{0.01}{\pi}\frac{\partial^2}{\partial x^2} - \mu^{n-1}\frac{\partial}{\partial x}$. The covariance matrices of GP/NNGP are formulated by (27). It should be noted that the NN approach does not need linearization since it directly assumes the nonlinear term to be the product of two correlated neural networks [8]. Thus, the NN approach is more flexible in solving nonlinear PDEs.

The temporal step-size in the Eular backward scheme is fixed to be $\Delta t = 0.01$. According to the numerical study in [7], the temporal discretization error is generally of the order $O(\Delta t)$. Decreasing Δt from 0.01 to 0.001 will improve solution accuracy, but after that point the accuracy saturates. Nevertheless, we observe that setting $\Delta t = 0.01$ can already produce accurate solutions. To evaluate the relative error at each time step, we place 400 equispaced test points in the space domain

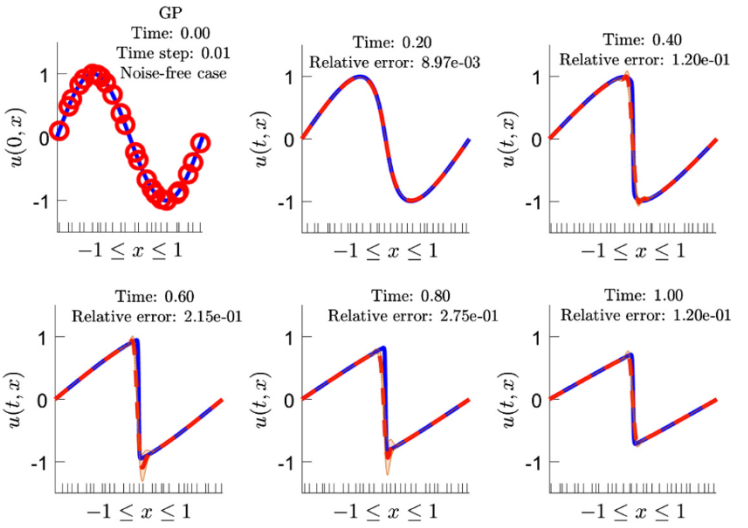

Fig. 14. Solving 1D Burgers' equation with noise-free initial condition: GP with kernel (17). The blue solid curve is the exact solution computed according to [17], the red circle is the training points at $n = 0$ (namely $N^0 = 24$), and the red dashed curve is the numerical solution. The x-axis ticks denote the locations of 31 randomly sampled training points at $n \geq 1$ ($N^n = 31$). The orange shadowed region is the two-standard-deviation band. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

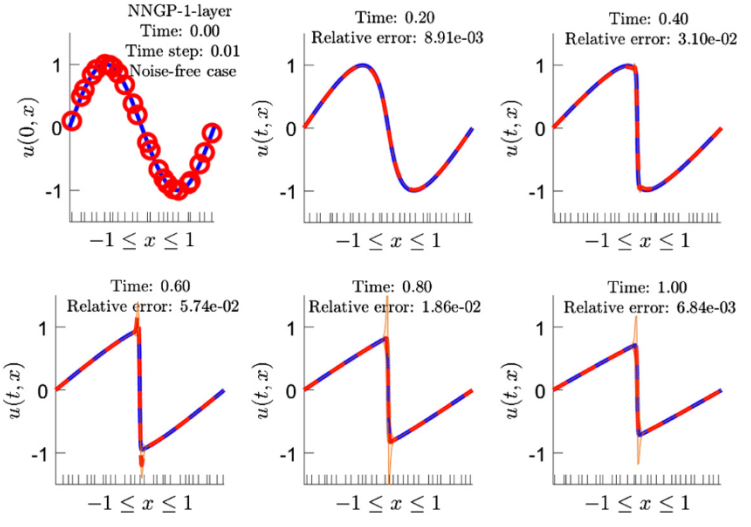

Fig. 15. Solving 1D Burgers' equation with noise-free initial condition: NNGP with one-layer, error-function induced kernel. Number of training points at $n = 0$ is $N^0 = 24$ and for $n \geq 1$ is $N^n = 31$.

$[-1, 1]$. We also solve the same equation using the NN approach proposed in [8]. One main difference between the NN approach and the numerical GP/NNGP regression is the sampling strategy for training inputs. For NN, we sample the training inputs in the time-space domain $(x, t) \in [0, 1]^2$, whereas for GP/NNGP, we sample the training inputs merely in the space domain and then perform time-marching. To make the comparison as fair as possible, we sample 10000 training points for NN, as in GP/NNGP we have at most 100 (number of time step) $\times$ 100 (number of training inputs in space) = 10000 (number of time-space sampling points). The Latin hypercube sampling strategy is adopted in sampling the training inputs in NN, GP, and NNGP. To accelerate training of numerical GP/NNGP, the initial guess of hyperparameters for current time step is taken as the optimized hyperparameters attained at previous time step. This strategy is reasonable since for a small time step two successive GPs are highly correlated and therefore the respective hyperparameters should be close.

The kernel (17), rather than the SE considered in solving Poisson equation, is taken in GP. We choose a non-stationary kernel instead of a stationary one, because the solution exhibits discontinuity that cannot be well captured by stationary kernel. Additionally, the NN with four hidden layers of 40-unit width and the hyperbolic tangent nonlinearity is employed in the NN approach. The width and depth of NNs and the type of nonlinearity are carefully tuned in order to make the NN approach achieve its best performance.

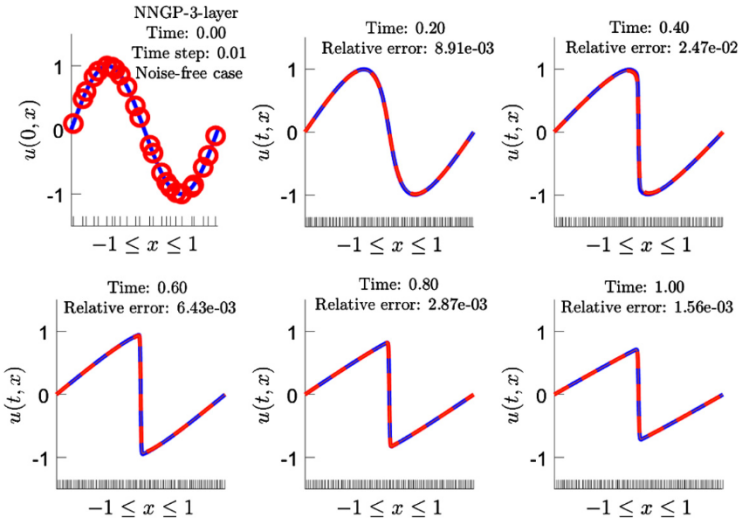

Fig. 16. Solving 1D Burgers' equation with noise-free initial condition: NNGP with three-layer, error-function induced kernel. Number of training points at $n = 0$ is $N^0 = 24$ and at $n \geq 1$ is $N^n = 101$.

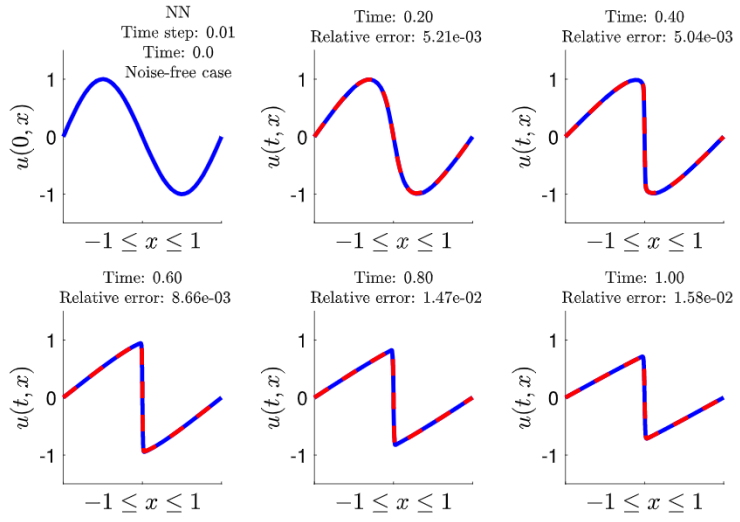

Fig. 17. Solving 1D Burgers' equation with noise-free initial condition: NN with four hidden layers of 40-unit width as well as hyperbolic tangent nonlinearity. The maximum error appears near the discontinuity point.

We first consider the case where the initial condition is noise-free. Figs. 14, 15, and 16 show the numerical solutions computed by the GP, the NNGP with one-hidden-layer ($L = 1$), and the NNGP with three-hidden-layer ($L = 3$), respectively. We can see that the NNGP has higher solution accuracy than the GP. Importantly, different from previous examples, increasing the depth of inducing NN in NNGP improves the results in the current example. The high accuracy of NNGP is attributed to a larger number of hyperparameters compared to the GP case. More hyperparameters presumably implies higher expressivity for function approximation, which means that we can use the kernel to approximate a wider spectrum of functions. For simple function or solution (as in the Hartmann 3D function and Poisson equation examples), the advantage of higher expressivity is not fully realized, but for complex ones, such as the step function and the Burgers' equation here, the advantage can be clearly seen.

The NN results are shown in Fig. 17. Although NN can derive sufficiently accurate solutions, it does not give any uncertainty estimate. Fig. 18 collects the errors from GP, NNGP, and NN for noise-free case. The NNGP and NN achieve errors of the same order of magnitude, while the GP performs worst. This is because NNGP and NN both have a large set of parameters to be tuned and thus possess higher expressivity.

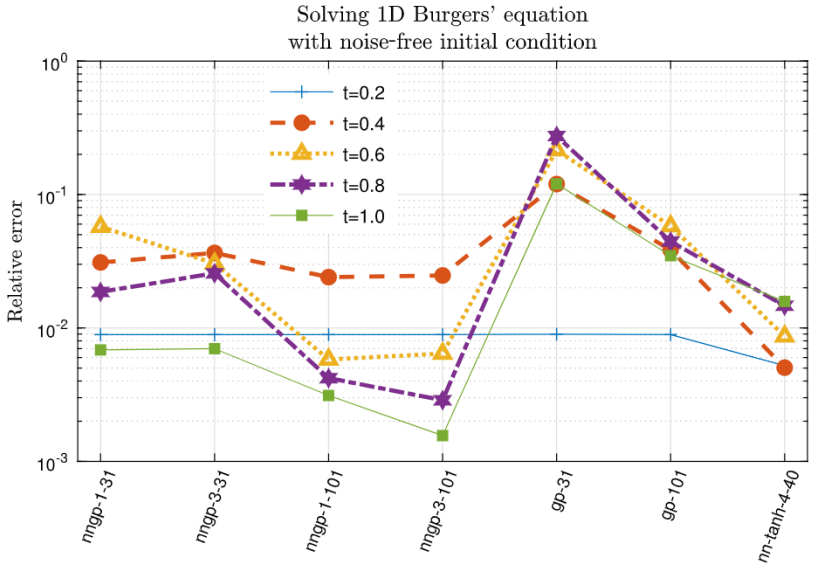

Fig. 18. Solving 1D Burgers' equation with noise-free initial condition: Accuracy comparison of NN, NNGP, and GP. "nngp-1" means the NNGP with one hidden layer and error-function induced kernel. "31" and "101" denote the numbers of training points $N^n = 31$ and 101 for $n \geq 1$, respectively. It is fixed that $N^0 = 24$ for all the NNGP examples. Kernel (17) is adopted in the GP. "nn-tanh-4-40" represents the NN with four hidden layers of 40-unit width as well as hyperbolic tangent nonlinearity.

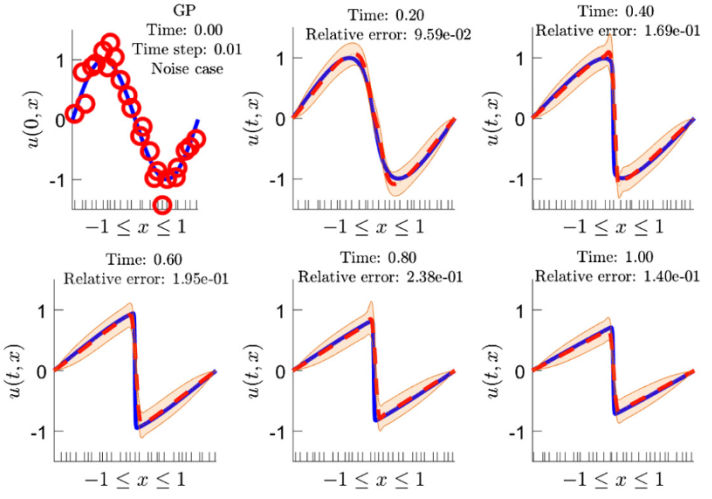

Fig. 19. Solving 1D Burgers' equation with noisy initial condition (noise variance $\sigma^2_{noise} = 0.15^2$): GP with kernel (17). Number of training points at $n = 0$ is $N^0 = 24$ and at $n \geq 1$ is $N^n = 31$.

Next we consider the case where the initial condition is contaminated by Gaussian white noise of zero mean and variance 0.15^2. The NN approach for the noise case is beyond the scope of the present paper, since without proper regularization methods (such as dropout [18], early stopping, and weighted L1/L2 penalty), the NN will easily encounter overfitting. Unlike the NN case, the GP and NNGP inherently include the model complexity penalty in the negative log-marginal likelihood and have less risk in overfitting (see the discussion in Section 5.4.1 of the book [12]).

Numerical solutions computed by the GP and the NNGP are plotted in Figs. 19, 20, and 21. NNGP still has higher solution accuracy than the GP due to the higher expressivity. Data noise amplifies uncertainty represented by the orange shadowed region. The GP and the NNGP can handle noise well, because noise variance can be directly learned from the training data and the corresponding uncertainty is quantified by the conditional (or posterior) distribution. Fig. 22 compares the solution accuracy of GP and NNGP. For long-term simulations, the accuracy of the NNGP is roughly one order higher than that of GP.

For NNGP, increasing the depth, L, of inducing NN does not guarantee the increase of accuracy. For example, one-layer NNGP with 101 training points outperforms three-layer NNGP with 101 training points, as is shown in Fig. 22. A larger L amounts to a larger number of hyperparameters, which merely increases the possibility in fitting complex functions better. Actually, L can also be seen as another hyperparameter of NNGP.

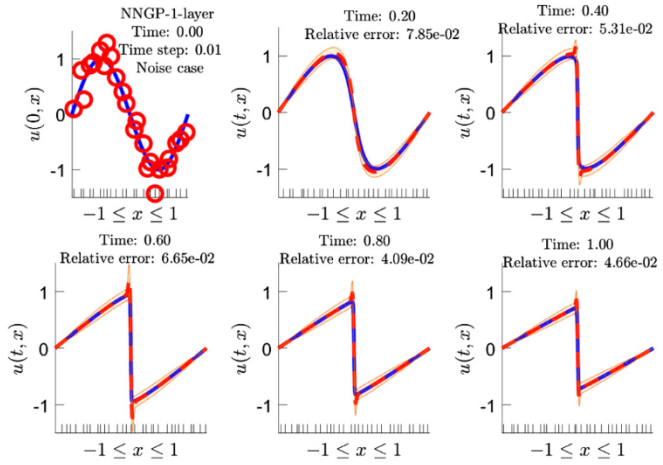

Fig. 20. Solving 1D Burgers' equation with noisy initial condition (noise variance $\sigma^2_{noise} = 0.15^2$): NNGP with the one hidden layer and error-function induced kernel. Number of training points at $n = 0$ is $N^0 = 24$ and at $n \geq 1$ is $N^n = 31$.

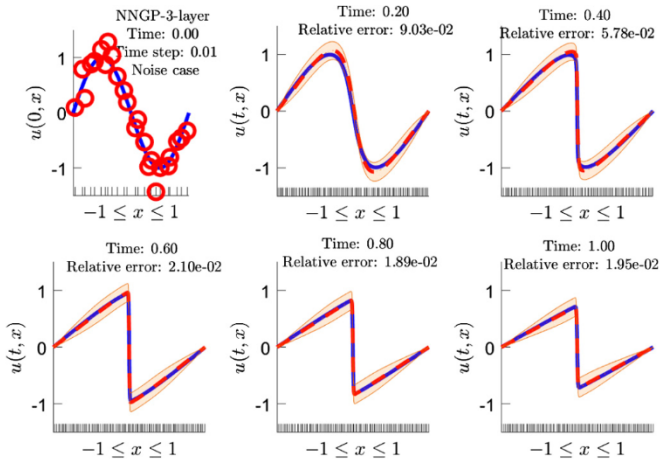

Fig. 21. Solving 1D Burgers' equation with noisy initial condition (noise variance $\sigma^2_{noise} = 0.15^2$): NNGP with the three hidden layers and error-function induced kernel. Number of training points at $n = 0$ is $N^0 = 24$ and at $n \geq 1$ is $N^n = 101$.

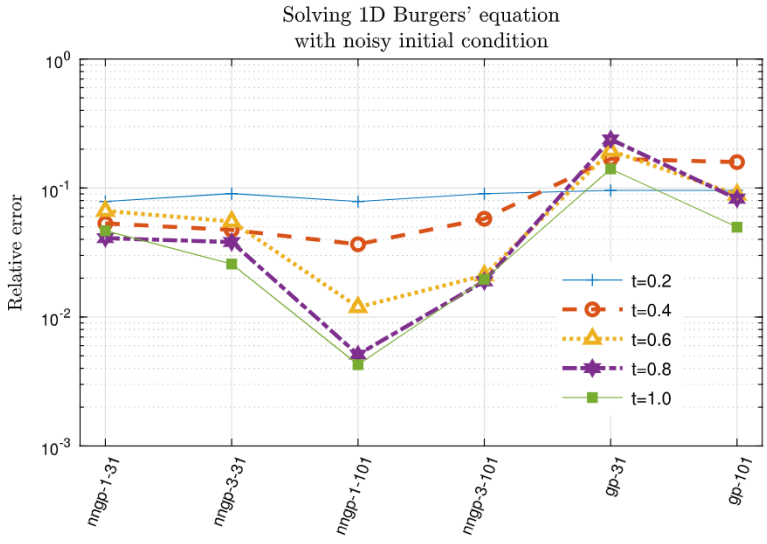

Fig. 22. Solving 1D Burgers' equation with noisy initial condition (noise variance $\sigma_n^2 = 0.15^2$): Accuracy comparison of GP and NNGP.

Table 1
Comparison of GP, NNGP, and NN (Accuracy) for function approximation and PDE solution.

Function/Solution	Feature	Accuracy comparison	Kernel/nonlinearity		
			GP	NNGP	NN
Step function	Non-smooth	NNGP> {GP, NN}	SE/Matern	erf/ReLU	erf/ReLU
Hartmann 3D	Smooth	{NNGP, GP}> NN	SE/Matern	erf/ReLU	erf/ReLU
Poisson eq.	Smooth	{NNGP, GP}> NN	SE	erf	erf/tanh
Burgers' eq.	Non-smooth	{NNGP, NN}> GP	kernel (17)	erf	tanh

Table 2
Comparison of GP, NNGP, and NN (whether to estimate uncertainty and the computational cost for training) for function approximation and PDE solution.

	GP	NNGP	NN
Uncertainty	✓	✓	×
Cost	$O(m_{GP}N_{training}^3)$	$O(m_{NNGP}N_{training}^3)$	$O(m_{NN}N_{weight}N_{training})$

6. Summary

A larger number of hyperparameters enables NNGP to achieve higher or comparable accuracy for both smooth and non-smooth functions in comparison with GP, which can be seen from Table 1. The deep NN that induces NNGP contributes its prior variances of network parameters (weights and biases) as well as its depth to the hyperparameter list of NNGP, which endows NNGP with high expressivity. On the other hand, NNGP is able to estimate uncertainty of predictions, which is crucial to noisy-data handling and active learning [19]. Unlike NN, Bayesian NN can provide uncertainty estimate. However, the conventional Bayesian NN [20] could be time-consuming to train due to approximation of a high-dimensional integral over weight space.

Due to the need for inverting the covariance matrix, NNGP has cubic time complexity for training (see Table 2). In Table 2, for GP and NNGP $N_{training}$ is the size of training set, and m_{GP} and m_{NNGP} are numbers of evaluations of negative log marginal-likelihood in conjugate gradient algorithm for GP and NNGP, respectively. For NN, the accurate estimate of time complexity for training is still an open question [21]. Generally, the training of a fully-connected NN is faster than that of GP, because one does not need to invert a matrix. For each training point, the forward and backward propagation only need linear cost, namely $O(N_{weight})$, where N_{weight} is the total number of weights in the network. $N_{training}$ for NN means batch size; in this paper we fed the whole training set to optimization algorithm and thus the batch size is exactly the size of training set. m_{NN} is number of iterations in optimization algorithm. In numerical examples of this paper, training set size does not exceed 1000. However, for very large datasets, GP and NNGP will be less attractive compared to NN. In future work, we intend to leverage scalable GPs recently developed in [22–24] to tackle large dataset problems.

Acknowledgements

We thank Mr. Dongkun Zhang and Dr. Maziar Raissi for useful discussions. This work was supported by AFOSR (FA9550-17-1-0013) and National Science Foundation (DMS-1736088). The first author was also supported by the National Natural Science Foundation of China (11701025). The second author was also supported by a gift from Swiss company BH-Robotics.

Appendix A. Derivatives of NNGP kernel from the error-function nonlinearity

Absorbing the constant coefficients $\frac{2}{\pi}$ and 2 into the variances, the kernel derived from the error-function nonlinearity, namely (16), can be further simplified to

$$\begin{aligned} k^l(\boldsymbol{x},\boldsymbol{x}') &= \sigma_{w,l}^2 \sin^{-1}\left(\frac{k^l(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{(1+k^{l-1}(\boldsymbol{x},\boldsymbol{x}))(1+k^{l-1}(\boldsymbol{x}',\boldsymbol{x}'))}}\right)+\sigma_{b,l}^2, l=1,2,\cdots,L, \\ k^0(\boldsymbol{x},\boldsymbol{x}') &= \boldsymbol{x}^T\Lambda\boldsymbol{x}'+\sigma_{b,0}^2. \end{aligned} \tag{A.1}$$

To solve PDEs, we need to compute the derivatives of the kernel $k^l(\boldsymbol{x},\boldsymbol{x}')$. We take the 2D Poisson equation as an example. We need to know the explicit forms of $-\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)k^l(\boldsymbol{x},\boldsymbol{x}')$ and $\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\left(\frac{\partial^2}{\partial x'^2}+\frac{\partial^2}{\partial y'^2}\right)k^l(\boldsymbol{x},\boldsymbol{x}')$ where $\boldsymbol{x}=[x,y]^T$ and $\boldsymbol{x}'=[x',y']^T$. The partial derivatives up to fourth order needs to be derived. Denoting by the trivariate function $F_\phi()$ the arcsin() term in the above iteration formula, the use of chain rule yields

$$
\begin{aligned}
\frac{\partial k^l(\boldsymbol{x},\boldsymbol{x}')}{\partial x} &= \sigma_{w,l}^2\left[\frac{\partial F_\phi}{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})}\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})}{\partial x}+\frac{\partial F_\phi}{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\partial x}+\frac{\partial F_\phi}{\partial k^{l-1}(\boldsymbol{x}',\boldsymbol{x}')}\times 0\right],\\
\frac{\partial^2 k^l(\boldsymbol{x},\boldsymbol{x}')}{\partial x^2} &= \sigma_{w,l}^2\left[\left(\frac{\partial^2 F_\phi}{\partial^2 k^{l-1}(\boldsymbol{x},\boldsymbol{x})}\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})}{\partial x}+\frac{\partial^2 F_\phi}{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})}\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\partial x}\right)\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})}{\partial x}\right.\\
&\quad+\frac{\partial F_\phi}{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})}\frac{\partial^2 k^{l-1}(\boldsymbol{x},\boldsymbol{x})}{\partial x^2}\\
&\quad+\left(\frac{\partial^2 F_\phi}{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x})}{\partial x}+\frac{\partial^2 F_\phi}{\partial^2 k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\partial x}\right)\frac{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\partial x}\\
&\quad\left.+\frac{\partial F_\phi}{\partial k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}\frac{\partial^2 k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\partial x^2}\right]\\
&\quad\cdots\cdots\cdots\cdots
\end{aligned}
\tag{A.2}
$$

Generally, the iteration formulas for kernel and its derivatives can be written as

$$
\frac{\partial^{i+j}k^l(\boldsymbol{x},\boldsymbol{x}')}{\partial x^i\partial x'^j}=F_{i,j}\left(\frac{\partial^{i'+j'}k^{l-1}(\boldsymbol{x},\boldsymbol{x}')}{\partial x^{i'}\partial x'^{j'}},\frac{\partial^{i'+j'}k^{l-1}(\boldsymbol{x},\boldsymbol{x})}{\partial x^{i'}\partial x'^{j'}},\frac{\partial^{i'+j'}k^{l-1}(\boldsymbol{x}',\boldsymbol{x}')}{\partial x^{i'}\partial x'^{j'}},0\le i'\le i,0\le j'\le j\right),
\tag{A.3}
$$

where $i,j\in 0,1,2$ and $F_{0,0}=F_\phi$. The initial values for the iteration are the known kernel k^0 and its derivatives. It should be noted that the explicit form of the function $F_{i,j}$ will become rather lengthy and ugly for higher order derivatives, but fortunately, we have derived for the readers these lengthy formulas using Maple. The Matlab code computing the covariance functions and its partial derivatives up to fourth order can be downloaded at https://github.com/Pang1987/nngp_kernel_derivative.

References

[1] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, Surya Ganguli, Exponential expressivity in deep neural networks through transient chaos, in: Advances in Neural Information Processing Systems, 2016, pp. 3360–3368.
[2] Radford M. Neal, Bayesian Learning for Neural Networks, PhD thesis, University of Toronto, 1995.
[3] Christopher KI Williams, Computing with infinite networks, in: Advances in Neural Information Processing Systems, 1997, pp. 295–301.
[4] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, Jascha Sohl-Dickstein, Deep neural networks as Gaussian processes, preprint, arXiv:1711.00165, 2017.
[5] Haim Sompolinsky, Andrea Crisanti, Hans-Jurgen Sommers, Chaos in random neural networks, Phys. Rev. Lett. 61 (3) (1988) 259.
[6] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, J. Comput. Phys. 335 (2017) 736–746.
[7] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Numerical Gaussian processes for time-dependent and non-linear partial differential equations, preprint, arXiv:1703.10230, 2017.
[8] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Physics informed deep learning (Part I): data-driven solutions of nonlinear partial differential equations, preprint, arXiv:1711.10561, 2017.
[9] Guofei Pang, Paris Perdikaris, Wei Cai, George Em Karniadakis, Discovering variable fractional orders of advection–dispersion equations from field data using multi-fidelity Bayesian optimization, J. Comput. Phys. 348 (2017) 694–714.
[10] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. 348 (2017) 683–693.
[11] Youngmin Cho, Lawrence K. Saul, Kernel methods for deep learning, in: Advances in Neural Information Processing Systems, 2009, pp. 342–350.
[12] Carl Edward Rasmussen, Christopher K.I. Williams, Gaussian Processes for Machine Learning, the MIT press, 2006.
[13] Loic Le Gratiet, Multi-Fidelity Gaussian Process Regression for Computer Experiments, PhD thesis, Université Paris-Diderot-Paris VII, 2013.
[14] Carl Edward Rasmussen, Hannes Nickisch, Gaussian processes for machine learning (GPML) toolbox, J. Mach. Learn. Res. 11 (2010) 3011–3015.
[15] Ladislav Kocis, William J. Whiten, Computational investigations of low-discrepancy sequences, ACM Trans. Math. Softw. 23 (2) (1997) 266–294.
[16] Xavier Glorot, Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[17] Cea Basdevant, M. Deville, P. Haldenwang, J.M. Lacroix, J. Ouazzani, R. Peyret, P. Orlandi, A.T. Patera, Spectral and finite difference solutions of the burgers equation, Comput. Fluids 14 (1) (1986) 23–41.
[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[19] Eric Brochu, Vlad M. Cora, Nando De Freitas, A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, preprint, arXiv:1012.2599, 2010.
[20] Radford M. Neal, Bayesian Learning for Neural Networks, vol. 118, Springer Science & Business Media, 2012.
[21] Le Song, Santosh Vempala, John Wilmes, Bo Xie, On the complexity of learning neural networks, preprint, arXiv:1707.04615, 2017.
[22] James Hensman, Nicolo Fusi, Neil D. Lawrence, Gaussian processes for big data, preprint, arXiv:1309.6835, 2013.
[23] Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W. Hogg, Michael O'Neil, Fast direct methods for Gaussian processes, IEEE Trans. Pattern Anal. Mach. Intell. 38 (2) (2016) 252–265.
[24] Alexander Litvinenko, Ying Sun, Marc G. Genton, David Keyes, Likelihood approximation with hierarchical matrices for large spatial datasets, preprint, arXiv:1709.04419, 2017.