Journal of Computational Physics 394 (2019) 263–279

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains

Nick Winovich [a], Karthik Ramani [b,1], Guang Lin [a,b,*,2]

[a] Department of Mathematics, 150 N. University Street, Purdue University, West Lafayette, 47907-2067, USA
[b] School of Mechanical Engineering, 585 Purdue Mall, Purdue University, West Lafayette, 47907-2088, USA

ARTICLE INFO

Article history:
Received 30 November 2018
Received in revised form 15 May 2019
Accepted 19 May 2019
Available online 29 May 2019

Keywords:
Partial differential equations
Uncertainty quantification
Convolutional encoder-decoder networks
Deep learning
Machine learning
Confidence interval

ABSTRACT

In this work, we introduce the ConvPDE-UQ framework for constructing light-weight numerical solvers for partial differential equations (PDEs) using convolutional neural networks. A theoretical justification for the neural network approximation to partial differential equation solvers on varied domains is established based on the existence and properties of Green's functions. These solvers are able to effectively reduce the computational demands of traditional numerical methods into a single forward-pass of a convolutional network. The network architecture is also designed to predict pointwise Gaussian posterior distributions, with weights trained to minimize the associated negative log-likelihood of the observed solutions. This setup facilitates simultaneous training and uncertainty quantification for the network's solutions, allowing the solver to provide pointwise uncertainties for its predictions. The associated training procedure avoids the computationally expensive Bayesian inference steps used by other state-of-the-art uncertainty models and allows training to be scaled to the large data sets required for learning on varied problem domains. The performance of the framework is demonstrated on three distinct classes of PDEs consisting of two linear elliptic problem setups and a nonlinear Poisson problem. After a single offline training procedure for each class, the proposed networks are capable of accurately predicting the solutions to linear and nonlinear elliptic problems with heterogeneous source terms defined on any specified two-dimensional domain using just a single forward-pass of a convolutional neural network. Additionally, an analysis of the predicted pointwise uncertainties is presented with experimental evidence establishing the validity of the network's uncertainty quantification schema.

1. Introduction

The theory of partial differential equations (PDEs) is central to our understanding of many physical systems. As a consequence, a substantial amount of research has been directed toward approximating the solutions of PDEs numerically. One

* Corresponding author at: Department of Mathematics, 150 N. University Street, Purdue University, West Lafayette, 47907-2067, USA.
E-mail addresses: nwinovic@purdue.edu (N. Winovich), ramani@purdue.edu (K. Ramani), guanglin@purdue.edu (G. Lin).
[1] School of Electrical and Computer Engineering (by courtesy).
[2] Department of Statistics (by courtesy).

https://doi.org/10.1016/j.jcp.2019.05.026
0021-9991/

of the most prevailing numerical frameworks for approximating PDE solutions is the finite element method (FEM) which serves as the basis for powerful computing platforms designed to accurately approximate solutions on complicated geometries. More recently, neural network models have been proposed as an alternative method for obtaining numerical solutions to PDEs [1–9] and have also been used to establish a physics-informed deep learning framework for modeling physical data with constraints described by PDEs [10,11]. These neural network frameworks have focused primarily on the construction of specialized networks, with each network dedicated to a specified PDE. This setup is motivated by the universal approximation property [12–14] of neural networks, with the trained network associated with a fixed PDE intended to serve as an embodiment of the solution itself. These networks are individually tailored to the specific problem and geometry into consideration, and by construction must be retrained for each new system. In effect, the neural network training procedure is employed as a substitute for explicitly forming and solving the linear system corresponding to the FEM framework.

One shortcoming of the conventional FEM approach, and the more recent neural network PDE solvers, however, is that neither provides any form of uncertainty estimate associated with their proposed solutions. This can result in overconfidence in the accuracy of the approximate solutions produced by the solvers, potentially leading to a reliance on inaccurate approximations without any indication of the risk. In an effort to mitigate this vulnerability, works in the field of probabilistic numerics have aimed to design Bayesian frameworks for approximating the solutions to PDEs using Gaussian process priors [15,16]. These solvers provide both mean and variance estimates, corresponding to a Bayesian posterior distribution associated with the solver's approximate solutions.

This Bayesian framework has the advantage of providing a natural form of uncertainty quantification; in particular, when uncertainties begin to accumulate in the numerical algorithm, and the approximation has the potential for significant inaccuracies in a certain region, the resulting uncertainty in the predicted solution is clearly indicated in the form of high variance estimates.

While the probabilistic numerics approaches provide accurate, rigorous uncertainty estimates, they also inherently rely on the explicit conditioning of Gaussian processes; this leads to cubic computational complexity and results in extremely slow inference speeds. The neural network-based methods, on the other hand, provide exceptionally fast estimates but do so without any attempt to quantify the associated uncertainty. To incorporate the benefits of uncertainty quantification [17–19] into deep learning models, Gal and Ghahramani [20] introduced a dropout procedure for representing model uncertainty without incurring the computational costs of fully Bayesian models. More recently, Lakshminarayanan et al. [21] introduced an ensemble method for predicting uncertainty estimates using deterministic networks, and Kendall and Gal [22] proposed a detailed framework for modeling epistemic and aleatoric uncertainties in deep learning models using Bayesian neural networks (BNNs) and loss functions derived from likelihood calculations.

Zhu and Zabaras [23] introduced a fully convolutional BNN designed to predict solutions to stochastic PDEs in an image-to-image manner; i.e. input arrays specifying the terms of the PDE are mapped directly to the associated solution array. Importantly, this setup allows the model to make predictions for new systems without requiring the model to be retrained. Moreover, since the network weights are realized as random variables in the Bayesian framework, uncertainty estimates are naturally provided by Monte Carlo approximations to the pointwise variance of the predicted solution (using samples from the posterior distributions of the network weights). The variational inference procedure used to train this network is specifically designed to account for the limited availability of training data on a fixed domain in consideration; generalizing this model to solve PDEs on varied domains requires extremely large data sets and would be infeasible due to the computational demands of the Bayesian training procedure.

In this work, we introduce a data-driven neural network framework, referred to as ConvPDE-UQ, for constructing numerical PDE solvers which provide accurate uncertainty estimates and are applicable to varied domains/geometries. These solvers are implemented as fully convolutional networks and are trained offline using a diverse collection of FEM solutions to PDEs on varying domains. To take advantage of recent advances in deep learning for computer vision [24,25], an image-to-image network architecture is introduced to construct a solver which does not require retraining for new problems or different domains; once training is complete, approximate solutions can be obtained using a single forward-pass of the convolutional network. A theoretical justification for the approximation of direct PDE input-to-solution mappings by neural networks is established based on the existence and properties of Green's functions. In addition, a probabilistic training procedure is proposed which casts network predictions as pointwise Gaussian estimates and leads to a natural form of uncertainty quantification with virtually no additional computational demands. The performance of the proposed framework is demonstrated on a collection of three distinct classes of PDEs consisting of two linear elliptic problems and a nonlinear Poisson problem. The inference speed of these solver networks are shown to be 30 times faster than the parallelized FEM implementations using FEniCS [26–53]. Moreover, a careful analysis of the network uncertainty estimates is carried out, and the interpretation of the network predictions as pointwise Gaussian estimates is shown to be strongly supported by the experimental evidence.

The presentation of this work is organized as follows: in Section 2, the mathematical problem statement is provided along with a theoretical justification for the proposed model. The network architecture, probabilistic training procedure, and data generation details are described in Section 3, followed by a discussion of the numerical results in Section 4 and a brief set of concluding remarks in Section 5.

2. Problem setup

2.1. Mathematical framework

To begin, we consider the task of solving the Poisson equation with the Dirichlet boundary condition:

$$\begin{cases} \Delta u = f & \text{in} \quad \Omega \\ u = 0 & \text{on} \quad \partial\Omega \end{cases} \tag{1}$$

where the domain $\Omega = D$, corresponding to the unit disk in $\mathbb{R}^2$, is fixed and we aim to solve the system for a family of *source terms* $f \in C^\infty(\overline{\Omega})$. The solution to this problem can be obtained explicitly in terms of the fundamental solution to the Laplace operator $\Delta = \sum_{i=1}^2 \frac{\partial^2}{\partial x_i^2}$ and the Green's function associated with the domain Ω. We recall that the fundamental solution Γ to the Laplacian in $\mathbb{R}^2$ is given by:

$$\Gamma(x) \;=\; \frac{1}{2\pi}\,\ln(|x|) \quad \forall x \in \mathbb{R}^2 \setminus \{0\} \tag{2}$$

and the Green's function for the unit disk in $\mathbb{R}^2$ is defined by:

$$G(x,y) = \begin{cases} \Gamma(x-y) \;-\; \Gamma\big(\,|y|\,(x-\overline{y})\,\big) & \text{for} \quad y \neq 0 \\ \Gamma(x) & \text{for} \quad y = 0 \end{cases} \tag{3}$$

where $\overline{y} = y/|y|^2$ corresponds to a scaled reflection of interior points $y \in D \setminus \{0\}$ across the boundary of the disk [54]. The solution to the homogeneous Dirichlet problem posed above can then be expressed using the representation formula:

$$u(x) \;=\; \int_\Omega G(x,y)\,f(y)\,dy \quad \forall x \in \Omega \tag{4}$$

Accordingly, we have an explicit solution mapping:

$$\mathcal{G} : C^\infty(\overline{\Omega}) \longrightarrow C^\infty(\overline{\Omega}) \tag{5}$$

$$f \;\mapsto\; \textstyle\int_\Omega G(-,y)f(y)dy \tag{6}$$

which assigns to each admissible source term f the associated solution $\mathcal{G}f = u$. This solution mapping takes the form of an integral operator with kernel $G(x,y)$ and satisfies the well-known *maximum principle* [54]: the unique solution $u \in C^\infty(\overline{\Omega})$ to Equation (1) with source term $f \in C^\infty(\overline{\Omega})$ satisfies:

$$\sup_\Omega |\,\mathcal{G}f\,| \;\leq\; C \cdot \sup_\Omega |\,f\,| \tag{7}$$

In particular, the solution mapping $\mathcal{G} : C^\infty(\overline{\Omega}) \to C^\infty(\overline{\Omega})$ is continuous with respect to the supremum norm.

More generally, given a domain Ω with piecewise smooth boundary $\partial\Omega$, an associated Green's function can be constructed by finding a solution $h(x)$ to the system:

$$\begin{cases} \Delta h = 0 & \text{in} \quad \Omega \\ h = -\Gamma & \text{on} \quad \partial\Omega \end{cases} \tag{8}$$

and setting $G(x,y) = \Gamma(x-y) + h(x-y)$. Repeating the argument above, we arrive at analogous estimates for the continuity of the new solution mapping $\mathcal{G}$ for the domain Ω. In this general case, however, the explicit Green's function is no longer available (with the exception of a select number of simple geometries).

Although an explicit Green's function is not available for general domains, the maximum principle can still be applied since the constant C in Equation (7) depends only on the diameter of the domain Ω. In particular, when the diameters of the domains in consideration are bounded by a fixed constant, the maximum principle gives a uniform, domain-independent bound for the continuity of the solution mapping.

2.2. Discretization

To cast the problem in discrete form for numerical approximation, we introduce a regular, rectangular grid Λ which covers the closure $\overline{\Omega}$ of each domain Ω in consideration (see Fig. 1). For our purposes, we consider domains contained inside the square $[-1,1]^{\times 2}$ and take Λ to be a uniform grid on the square with fixed resolution R. Given a domain Ω, we denote by Ω_d the collection of mesh points in Λ which lie inside of the closure of the domain Ω (i.e. $\Omega_d = \Lambda \cap \overline{\Omega}$). We then define the associated interpolation and projection mappings:

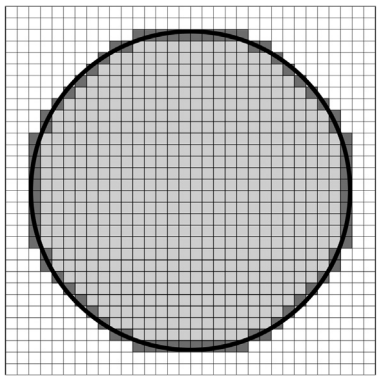


Fig. 1. Discretization of the domain Ω (light gray) and boundary $\partial\Omega$ (dark gray) for the Poisson equation on the circle. The interior and boundary locations are used to define the network loss function and tailor training to the domain in consideration.

$$\mathcal{I} : \left\{ (x,\ f(x)) \right\}_{x \in \Omega_d} \mapsto \ \text{interp}_{\overline{\Omega}} \left\{ (x,\ f(x)) \right\} \tag{9}$$

$$\mathcal{P} :\ u\,|_{\overline{\Omega}} \mapsto \ \left\{ (x,\ u(x)) \right\}_{x \in \Omega_d} \tag{10}$$

where $\text{interp}_{\overline{\Omega}} \left\{ (x,\ f(x)) \right\}$ denotes the function defined on $\overline{\Omega}$ resulting from a fixed interpolation procedure (e.g. bilinear or bicubic interpolation). By construction, these mappings satisfy the norm estimates:

$$\sup_{\Omega} \left| \mathcal{I}\left\{ (x,\ f(x)) \right\} \right| \ \leq \ C(\text{interp}) \cdot \max\{|f(x)|\} \tag{11}$$

$$\max \left\{ |u(x)| : (x,\ u(x)) \in \mathcal{P}u \right\} \ \leq \ \sup_{\Omega} |u| \tag{12}$$

where the constant $C(\text{interp})$ depends only on the fixed interpolation procedure which has been selected. It follows that the discretized, composite mapping:

$$\left\{ (x,\ f(x)) \right\}_{x \in \Omega_d} \xrightarrow{\ \mathcal{I}\ } f\,|_{\overline{\Omega}} \xrightarrow{\ \mathcal{G}\ } u\,|_{\overline{\Omega}} \xrightarrow{\ \mathcal{P}\ } \left\{ (x,\ u(x)) \right\}_{x \in \Omega_d} \tag{13}$$

is continuous and can, therefore, be approximated by a multi-layer feedforward neural network according to the *universal approximation theorem* [12–14]. In light of this, we pursue the construction of an approximate PDE solver using a neural network designed to predict discrete arrays $\{(x,\ u(x))\}$ of the solution point values on a grid provided a discrete array $\{(x,\ f(x))\}$ of specified source term values as input.

2.3. Approximation by convolutional networks

Our aim is to define a neural network which approximates the discretized solution mapping from Equation (13). The convolutional form of the integral operator $\mathcal{G}$ naturally inspires the use of convolutional layers within this neural network approximation. As outlined in Section 2, the task is rather trivial in the setting where the domain Ω in Equation (1) is fixed to the unit disk D. In this case, the true Green's function is known and can be used to directly apply the linear solution operator in Equation (5). For more general domains, however, the explicit Green's function is unavailable and alternative numerical methods must be employed. We first consider the problem setup on the circle as a proof-of-concept for the proposed neural network PDE solver framework; after establishing the performance of the network for this simple setup, we proceed to a more careful analysis of the following more complex scenarios:

i.) *Varying domain:* construct a network to approximate the solution u to Equation (1) when both the source term f and domain Ω are permitted to vary.

ii.) *Nonlinear Poisson:* construct a network to approximate the solution u when both the source term f and domain Ω are permitted to vary, and Equation (1) is replaced with the following nonlinear PDE:

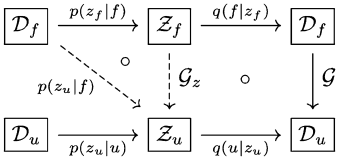
$$\begin{cases} \text{div}\left(\left(1 + |u|^2\right) \cdot \nabla u \right) \ = \ f & \text{in} \quad \Omega \\ u \ = \ 0 & \text{on} \quad \partial\Omega \end{cases} \tag{14}$$

The proposed framework can also be naturally extended to work with more general PDE systems. In particular, variable coefficient differential operators and homogeneous Neumann boundary conditions can both be accounted for with minimal changes to the network architecture. The implementation details for these additional experiments are provided in Appendix A along with the associated experimental results.

3. Methodology

3.1. Bayesian framework

For the training procedure and theoretical foundation of the proposed model, we adopt the probabilistic framework introduced by Kingma and Welling [55]. In particular, we consider a setting in which two variational autoencoder (VAE) models have been trained: one for the space of source terms and one for the space of solutions. In this setup, the source data space $\mathcal{D}_f$ and solution data space $\mathcal{D}_u$ are mapped to associated latent variable spaces $\mathcal{Z}_f$ and $\mathcal{Z}_u$ by recognition models $p(z_f|f)$ and $p(z_u|u)$, respectively. The encoded variables are then decoded back to the original data spaces by generative models $q(f|z_f)$ and $q(u|z_u)$. The associated theoretical framework is summarized in the following commutative diagram:



where the mapping $\mathcal{G}_z$ corresponds to the latent space transformation which sends the encoded latent variables z_f associated with the source term f to the latent variables z_u corresponding to the solution u.

The proposed neural network model incorporates a recognition model $p(z_u|f)$, trained to map functions from the source data space $\mathcal{D}_f$ directly to the solution representations z_u, along with a generative model $q(u|z_u)$, designed to produce approximations to the true solutions from the encoded representations. This has the benefit of resulting in a one-shot training process (i.e. separate training procedures are not required for each VAE model) as well as enabling the learned latent space structure to be optimized simultaneously for the data space encoder and solution space decoder.

3.2. Probabilistic training procedure

One shortcoming of traditional numerical methods is the lack of any explicit uncertainty quantification associated with the resulting approximations. Though rigorously grounded in asymptotic convergence results, the uncertainty for each individual approximation is known only up to a single, global bound. The approximate solutions produced by a FEM solver, for example, do not provide any indication of specific regions in a domain where the convergence/accuracy of the numerical solution is uncertain. Conventional neural network architectures follow this format as well, predicting pointwise values without providing estimates gauging how far off the prediction may be from the true solution in certain regions.

To address this issue, we construct a network with the capacity to quantify the *heteroscedastic* uncertainty associated with its predictions; that is, the uncertainty specific to individual examples (e.g. the uncertainty resulting from a sharp corner in a given domain). As noted by [21,22], this can be achieved by casting the network's predictions in the form of Gaussian posterior distributions, as opposed to singular point-estimates. The network is designed to predict pointwise statistics $\widehat{\mu}[i,j]$ and $\widehat{\sigma}[i,j]$ corresponding to a posterior distribution $\mathcal{N}(\widehat{\mu}[i,j],\widehat{\sigma}[i,j])$ over the possible true solution values $\{u(x_i,y_j)\}$. This allows the network to attach to each point both an estimate $\widehat{\mu}[i,j]$ for the value of the solution and a standard deviation $\widehat{\sigma}[i,j]$ reflective of its confidence in that estimate. For numerical stability, and to avoid enforcing constraints on the network's variance predictions, it is convenient to predict the log standard deviations $\widehat{\sigma}_{\log}$ and compute the associated statistics for the loss function using the exponential function:

$$\widehat{\sigma}[i,j] \;=\; \exp\left(\widehat{\sigma}_{\log}[i,j]\right) \tag{15}$$

Training is then designed to maximize the associated likelihood of observing the known true solution values. Assuming, for example, independent pointwise posterior distributions, we aim to maximize:

$$p(y\,;\,\widehat{\mu},\widehat{\sigma}) \;=\; \prod_{i,j=1}^{R} \frac{1}{\sqrt{2\pi\cdot\widehat{\sigma}[i,j]^2}}\,\exp\Big(\,-\tfrac{1}{2}\left(y[i,j]-\widehat{\mu}[i,j]\right)^2/\widehat{\sigma}[i,j]^2\,\Big) \tag{16}$$

or, equivalently, to minimize the negative log-likelihood of the observed solution values:

$$-\log p(y\,;\,\widehat{\mu},\widehat{\sigma}) \;=\; \sum_{i,j=1}^{R} \tfrac{1}{2}\left(y[i,j]-\widehat{\mu}[i,j]\right)^2/\widehat{\sigma}[i,j]^2 \;+\; \sum_{i,j=1}^{R} \tfrac{1}{2}\,\log(2\pi\cdot\widehat{\sigma}[i,j]^2) \tag{17}$$

This framework allows the network to begin by making coarse mean estimates while admitting relatively high variance estimates (i.e. low-precision predictions) and to gradually increase the network's predicted confidence by lowering variance estimates as the training procedure progresses. This interpretation is indeed motivated by the design of the network's loss function, which is fundamentally responsible for guiding the training process, as illustrated in Fig. 2. More importantly, the interpretation is seen to coincide with what happens in practice, as will be discussed further in Section 4.

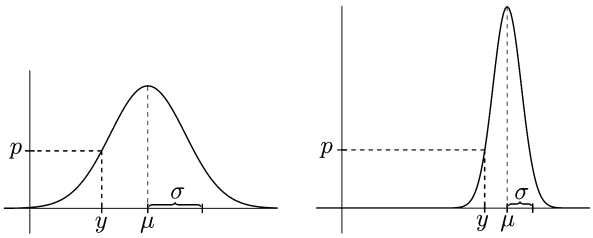


Fig. 2. The probabilistic prediction framework allows the network to begin with coarse, low-confidence predictions (left) and to gradually build confidence by lowering the predicted standard deviations. High confidence predictions (right) allow the network to attain far better losses when correct, but have steep drop-offs penalizing any inaccuracy in the prediction.

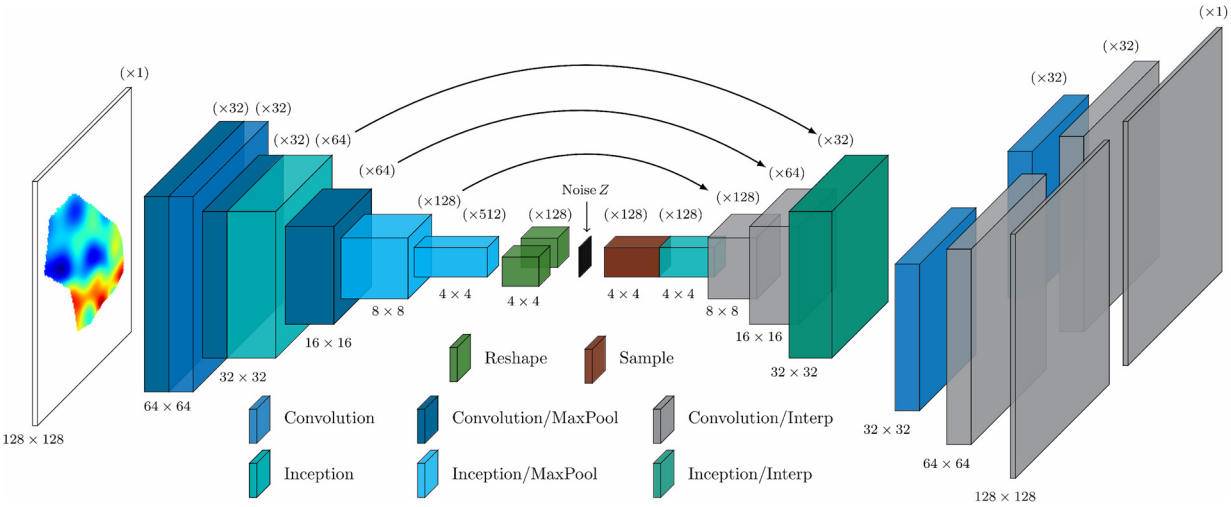


Fig. 3. Proposed network architecture. Features have been color-coded according to the type of layer which has been used to produce them. The arrows indicate feature concatenation, whereby the encoder features are shared/reused in the decoding process in accordance with the U-Net architecture. For the probabilistic network, the stem of the decoder is split into two branches corresponding to mean and variance predictions (as shown above); the MSE network architecture is obtained by simply dropping the variance branch of the decoder (since variance predictions are not required for the MSE model). (For an interpretation of the colors in the figure, the reader is referred to the web version of this article.)

3.3. Network architecture

The proposed network architecture consists of two primary components; an encoder designed to map high-level input functions to low-dimensional latent features, and a decoder used to map these latent features to approximate solutions. The encoder consists of a series of convolutional layers which gradually reduce the resolution of input features. The output features of the encoder are then partitioned into mean and standard deviation values and used to sample latent features. The standard 'reparameterization trick' [55] is used to maintain a differentiable network structure; the entries of the latent features are sampled independently via $z = \mu + \sigma \cdot \varepsilon$ where μ and σ are the corresponding entries of the reshaped encoded features and $\varepsilon \sim \mathcal{N}(0, 1)$.

The decoder maps the sampled latent features back to the original resolution using a series of convolutional layers followed by bilinear interpolation. In addition, the network incorporates aspects of the U-Net architecture [25], passing features extracted in the encoding component directly over to the decoder. In particular, the encoder features with spatial resolutions 32×32, 16×16, and 8×8 are concatenated with the features of the same resolution in the network's decoder (as indicated with arrows in Fig. 3). To accommodate the probabilistic prediction framework described in Section 3.2, the stem of the decoder is split into two branches: one for the mean predictions and another for log standard deviation predictions.

For improved performance, some layers have been split into a collection of collaborative filters [24] which are referred to as 'inception blocks' in Fig. 3 for convenience. These blocks consist of a max-pooling layer along with 1×1, 3×3, and two stacked 3×3 convolutional layers implemented in parallel; the features of these layers are then concatenated channel-wise to produce the final set of features sent to the next network layer. Dropout layers with drop-rate 0.045 have also been included before and after the first inception block in the decoder to help avoid over-fitting to the training data set.

3.4. Network loss functions and training procedure

Based on the problem statement provided in Equation (1), it is natural to define the network's loss function in terms of two components; one to enforce the differential equation on the interior, and another to enforce the boundary condition. To

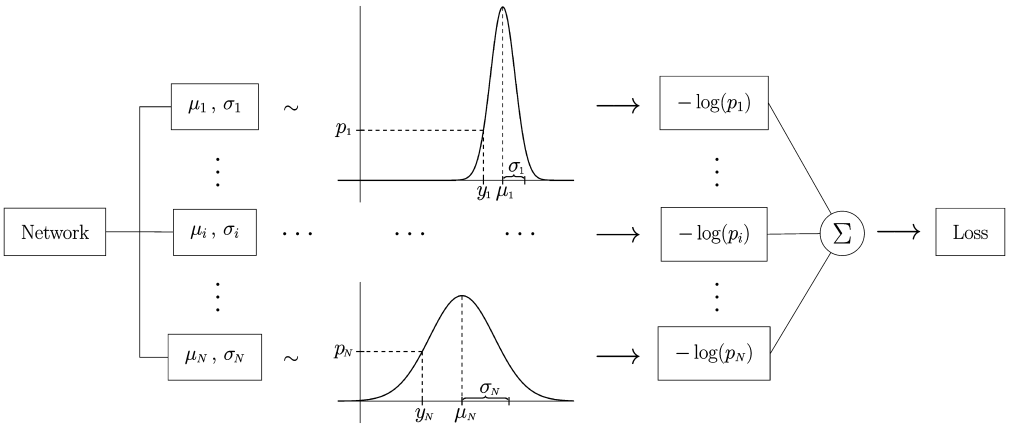


Fig. 4. Overview of the proposed probabilistic network design; point estimates $\widehat{y}_i$ are replaced with the statistics μ_i and σ_i of a normal distribution, and the negative log-likelihoods of observed solution values y_i are summed to define the network loss. To align with the numerical implementation, we have flattened the summation operation in Equation (20) and omitted terms in which $\mathbb{1}_\Omega[i,j]=0$ so that only the $N=|\Omega|$ points inside of the current domain are included in the loss calculation.

this end, the mean squared error (MSE) can be calculated on the interior and the boundary of the domain using a template file to indicate where these errors should be calculated:

$$\text{Loss}_{\texttt{MSE}}(\widehat{y}\,;\,y,\Omega) \;=\; \frac{1}{|\Omega|}\sum_{i,j=1}^{R}\mathbb{1}_\Omega[i,j]\cdot\big(\widehat{y}[i,j]-y[i,j]\big)^2 \;+\; \frac{\lambda}{|\partial\Omega|}\sum_{i,j=1}^{R}\mathbb{1}_{\partial\Omega}[i,j]\cdot\big(\widehat{y}[i,j]-y[i,j]\big)^2 \tag{18}$$

Here R denotes the selected output resolution, $|D|$ corresponds to the number of pixels inside the region D, and $\mathbb{1}_D$ denotes the indicator function for the discretized, Boolean template file corresponding to the domain D (e.g. $\mathbb{1}_\Omega[i,j]=1$ when the (i,j)th pixel lies inside of the domain Ω, and $\mathbb{1}_\Omega[i,j]=0$ otherwise).

The hyperparameter λ in Equation (18) is intended to balance the contributions of the interior and boundary terms. In practice, however, it has been observed that this explicit boundary loss term is not strictly necessary. Setting $\lambda=0$, we arrive at the alternative, simplified expression for the MSE loss:

$$\text{Loss}_{\texttt{MSE}}(\widehat{y}\,;\,y,\Omega) \;=\; \frac{1}{|\Omega|}\sum_{i,j=1}^{R}\mathbb{1}_\Omega[i,j]\cdot\big(\widehat{y}[i,j]-y[i,j]\big)^2 \tag{19}$$

The probabilistic training procedure described in Section 3.2, and illustrated in Fig. 4, is implemented by replacing the traditional MSE loss function with the following negative log-likelihood calculation:

$$\text{Loss}_{\texttt{PROB}}(\widehat{\mu},\widehat{\sigma}\,;\,y,\Omega) \;=\; \frac{1}{|\Omega|}\sum_{i,j=1}^{R}\mathbb{1}_\Omega[i,j]\cdot\Big[\tfrac{1}{2}\big(\widehat{\mu}[i,j]-y[i,j]\big)^2/\widehat{\sigma}[i,j]^2 \;+\; \tfrac{1}{2}\log(2\pi\cdot\widehat{\sigma}[i,j]^2)\Big] \tag{20}$$

Intuitively, the two terms of this summand can be seen to account for a notion of *model fitness* and *model confidence*, respectively; the training procedure is thus directed toward balancing the trade-offs between the two competing notions in order to minimize the probabilistic loss function.

In addition to the primary MSE/probabilistic loss term, the network incorporates a secondary loss term corresponding to the Kullback-Leibler (KL) divergence prescribed by the VAE framework. As noted in [55], assuming independent normal priors on the latent means z_μ and variances z_{σ^2}, this can be computed via:

$$\text{Loss}_{\texttt{KL}}(z_\mu,z_\sigma) \;=\; \frac{1}{2}\sum_{i=1}^{L}\Big(z_{\mu,i}^2+z_{\sigma^2,i}-\log z_{\sigma^2,i}-1\Big) \tag{21}$$

where L denotes the number of latent variables at the sampling stage of the model, and $z_{\mu,i}$ and $z_{\sigma^2,i}$ denote the encoded mean and variance associated with the ith latent variable. In particular, for the experiments presented in this work, there are $L=2048$ latent variables corresponding to the individual entries of the 128 channels of 4×4 features at the bottleneck of the network architecture, as illustrated in Fig. 3.

Once the loss function for the network is specified, training is carried out using the Adam optimization algorithm [56] to minimize the loss function evaluated on batches of data from the training set. A batch size of 64 has been used, and the learning rate of the Adam optimizer has been initialized at 0.00075, with an exponential decay applied every 10,000 steps by a factor of 0.95. The network architecture and hyperparameters have been tuned to optimize the MSE training procedure.

Table 1
Summary of the network errors for each of the three problem setups under consideration. The L^1 and L^2 relative errors are provided for both the training dataset (left) and validation dataset (right). The relative errors are seen to gradually increase as we move from the simple setup on the circle to the more complex nonlinear equation on varying domains. In all cases, however, we see that the probabilistic training procedure outperforms the traditional mean squared error training.

Problem Setup	Model	L^1 Relative Error		L^2 Relative Error	
Poisson on Circle	**Probability**	**9.19e−3**	**1.00e−2**	**1.18e−4**	**1.50e−4**
	MSE ($\lambda = 0.1$)	1.23e−2	1.28e−2	2.60e−4	3.06e−4
	MSE ($\lambda = 0.0$)	1.23e−2	1.29e−2	2.48e−4	2.90e−4
Varying Domain	**Probability**	**1.82e−2**	**2.11e−2**	**1.21e−3**	**1.45e−3**
	MSE ($\lambda = 0.1$)	3.43e−2	3.57e−2	2.25e−3	2.62e−3
	MSE ($\lambda = 0.0$)	3.60e−2	3.75e−2	2.43e−3	2.86e−3
Nonlinear Poisson	**Probability**	**1.94e−2**	**2.24e−2**	**1.32e−3**	**1.58e−3**
	MSE ($\lambda = 0.1$)	3.21e−2	3.46e−2	1.84e−3	2.46e−3
	MSE ($\lambda = 0.0$)	3.37e−2	3.61e−2	2.09e−3	2.69e−3

The same architecture and hyperparameters were observed to be optimal for the probabilistic training procedure as well (with the final layers of the decoding component duplicated to produce uncertainty estimates), and this same network setup is shown to be suitable for each of the three problem setups into consideration.

3.5. Data generation

The source terms for each experiment are taken to be samples from mean-zero Gaussian random fields:

$$f \sim \mathcal{G}(0, k_l(x, y)) \tag{22}$$

where the covariance kernels $k_l(x, y) = \exp(-\|x - y\|^2/2l^2)$ correspond to squared exponential kernels with *length-scales* l varying in a fixed interval $[l_{min}, l_{max}]$. In particular, 20 distinct length-scales were selected between $l_{min} = 0.2$ and $l_{max} = 0.6$ were selected for data creation in the experiments presented in this work. We note that this setup explicitly defines the universal data distribution, in accordance with the proposed Bayesian framework for training described in Section 3.1. In particular, the data samples for the source terms f are drawn from a distribution of functions in $C^\infty(\Omega)$ with an average expected rate of change controlled by the length-scale parameters l in the specified collection of Gaussian random fields.

The domain generation procedure has been designed to sample from a family of polygonal domains in the interior of the region $[-1, 1]^{\times 2} \subset \mathbb{R}^2$ with vertex counts between $v_{min} = 4$ and $v_{max} = 16$. This generation was carried out by sampling a vertex count $v \sim \text{Unif}(\{v_{min}, \dots, v_{max}\})$, randomly sampling angles for vertices to be placed at $\{\theta_k\} \sim \text{Unif}(0, 2\pi)$, and lastly sampling the corresponding radii for the vertices $\{r_k\} \sim \text{Unif}(r_{min}, 1)$ with $r_{min} = 0.25$. After re-indexing to ensure $\theta_1 < \dots < \theta_v$, the boundary of the generated domain was taken to correspond to the cycle $[(r_1 \cos\theta_1, r_1 \sin\theta_1), \dots, (r_v \cos\theta_v, r_v \sin\theta_v)]$.

Once the source terms and domains have been generated, it remains only to produce the corresponding solution to each system. The approximate solutions $\{u(x)\}$ used for the training datasets have been computed numerically using the finite element solver FEniCS with a mesh resolution of 35 (corresponding to finite element cells of approximate diameter $1/35 \approx 0.0286$) and Lagrange basis functions of polynomial order 1. The generated source terms and meshes have also been implemented within FEniCS to facilitate the solution procedure; the final source terms, meshes, and solutions are then evaluated on a regular grid and converted to rectangular arrays to accommodate for a more natural neural network architecture. Of note is the fact that the data generation process outlined above is trivially parallelizable, as the solutions to distinct systems are independent. This fact has indeed been leveraged in the code provided to take full advantage of all processors available, and can further be implemented on distributed systems with minimal changes.

The numerical results presented in this work have been obtained using datasets consisting of 100,000 examples, with 80% used for training and 20% used for validation. These datasets have been generated following the procedures above with 5,000 source term samples drawn from each of the 20 length-scale classes. The mesh, source term, and solution arrays are also rotated 180 degrees and flipped horizontally during training to augment the effective dataset size to 400,000 examples for each problem setup.

4. Numerical results

4.1. Comparison of training procedures

Following the methodology outlined in Section 3, a convolutional network has been trained to serve as a numerical solver for each of the three problem setups. The networks have each been trained independently using the traditional MSE loss as well as the proposed probabilistic training procedure. As shown in Table 1, the probabilistic framework outperforms the conventional MSE training in each of the three problem setups.

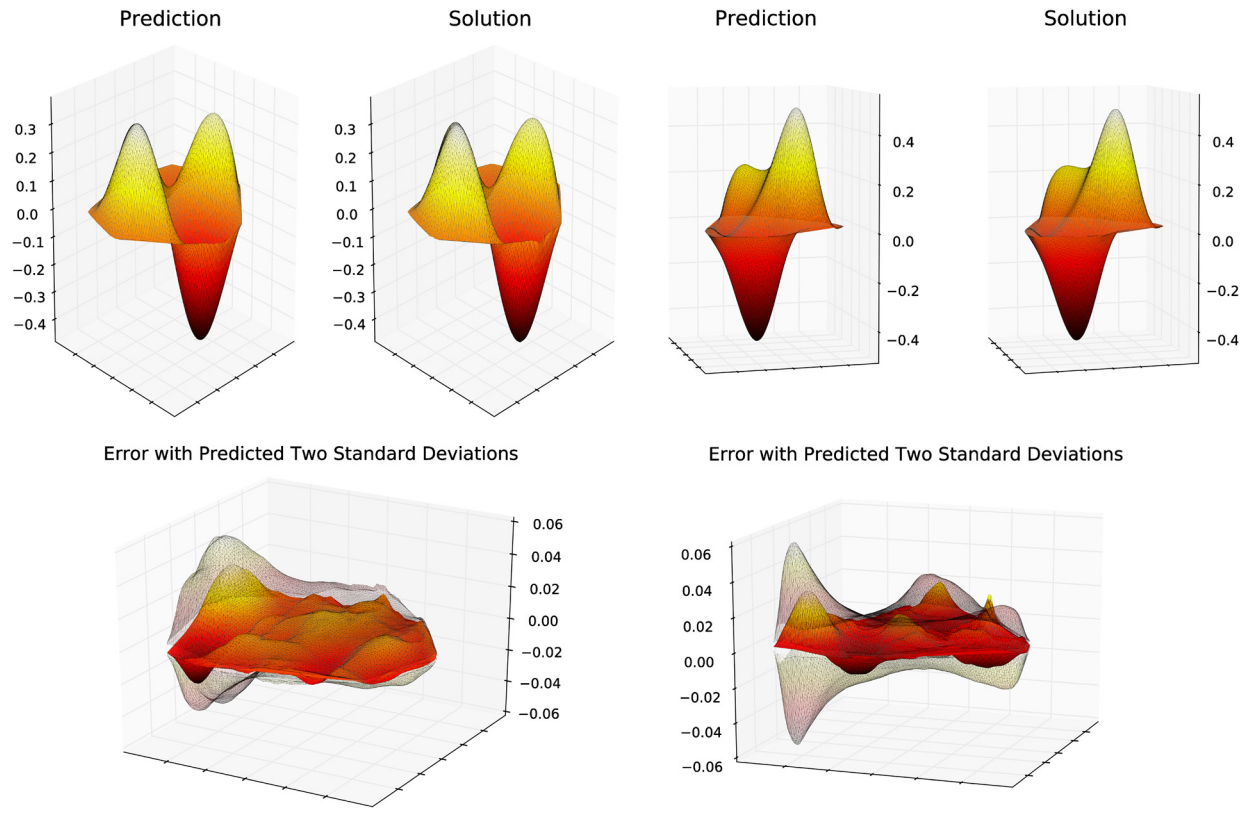


Fig. 5. Qualitative results demonstrating the network's predictions for two problems in the "Varying Domain Poisson Equation" setup. Both examples have been taken from the validation data set and have not been seen by the network during training. Network predictions and true solutions are shown at the top, with the corresponding absolute errors plotted below along with the network's predicted two standard deviations plus/minus bounds (shown as transparent wireframes). The predicted pointwise standard deviations are seen to provide accurate error estimates for the network predictions.

This is a rather striking result considering the fact that the mean estimates of the probabilistic networks (which are used to calculate the errors in Table 1) are produced using an identical architecture to those used for the MSE networks. Introducing an independent variance branch in the decoder and redefining the loss function in terms of likelihood estimates is actually seen to achieve lower L^2 errors, the precise errors the MSE network is specifically designed to minimize. The consistent outperformance of the probabilistic network is also observed when the data set is partitioned according to the length-scales of the input functions, as shown in Fig. 6. As one would expect, both networks admit a gradual reduction in accuracy as the length-scales of the inputs decrease, corresponding to more oscillatory source terms.

In addition to the unexpected improvement in performance, the probabilistic networks have the added benefit of providing predictive uncertainties associated with their predictions. This gives the probabilistic networks the capacity to clearly indicate potential inaccuracies in its predictions and is seen to occur in practice as shown in Fig. 5. Of note is the fact that these examples have been taken from the validation data set, and have therefore never been seen by the network during the training procedure. The fact that the uncertainty estimates remain accurate for the validation problems suggests that the network has learned an uncertainty quantification schema capable of generalization, and is not simply fit to model the observed errors during training. A closer, more quantitative analysis of these uncertainties is provided below.

4.2. Uncertainty quantification

An essential requirement of any meaningful measure of uncertainty is that the predicted uncertainties are correlated with the associated prediction errors. As shown in Fig. 7, this is indeed the case for the proposed framework; the predicted uncertainties are seen to gradually reduce throughout the training process, closely following the progression of the network's L^2 error on the validation data set. A strong relationship between the model uncertainty and the absolute L^2 error is observed for the full duration of training, with a Pearson correlation coefficient of 0.921.

By construction, the trained network is designed to predict pointwise Gaussian posterior distributions for the true solution values. To evaluate how effectively the network conforms to this design in practice, we compare the trained network predictions with true solutions. We should expect the fraction of pixels falling within a specified standard deviation range to coincide with that of a Gaussian distribution:

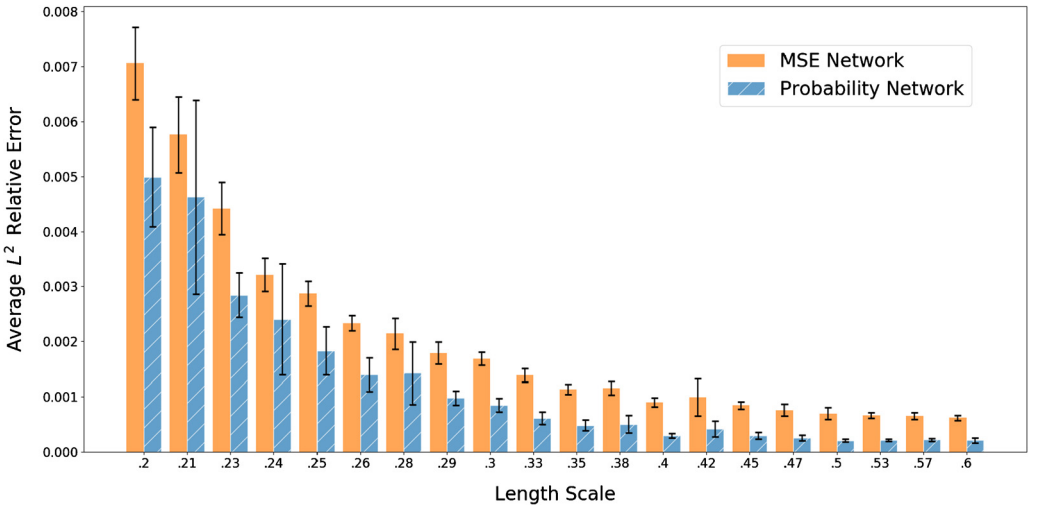


Fig. 6. Average L^2 relative error per length-scale class in the "Varying Domain Poisson Equation" setup. The probabilistic network is seen, all else unchanged, to outperform the traditional MSE network for each length-scale in the sampling procedure. As may be expected, both networks concede higher losses for input functions sampled from lower length-scales, i.e. those corresponding to more oscillatory inputs.

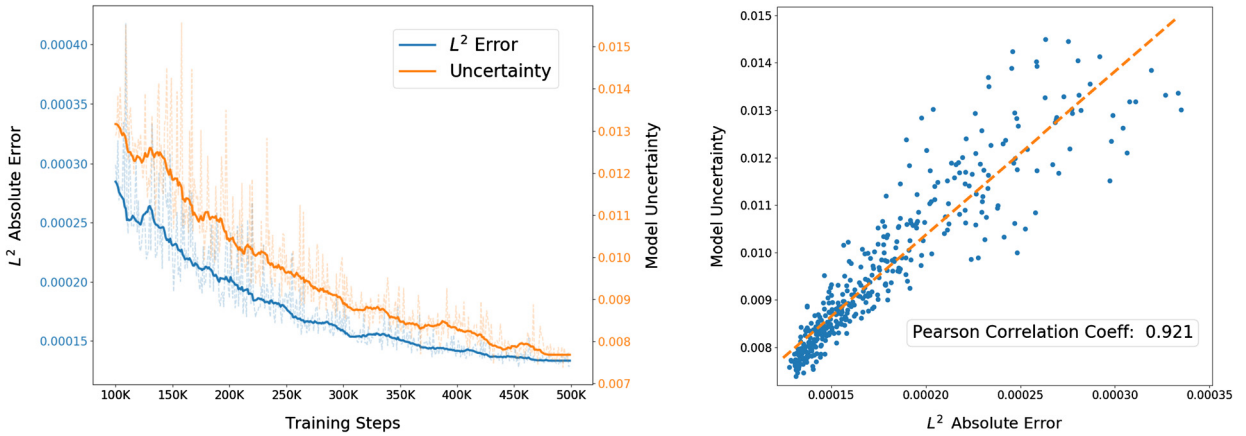


Fig. 7. The absolute L^2 loss and magnitude of predicted uncertainties in the "Varying Domain Poisson Equation" setup throughout the training process (left) along with a plot of the absolute L^2 error against the predicted model uncertainty (right). A high correlation between the predicted model uncertainty and error in network predictions is clearly visible, with a Pearson correlation coefficient of 0.921.

$$\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}\sum_{[i,j]\in\Omega_d}\frac{1}{|\Omega_d|}\,\mathbb{1}\Big(\big|\widehat{y}_d[i,j]-y_d[i,j]\big|\leq x\cdot\widehat{\sigma}_d[i,j]\Big)\;\approx\;\mathbb{P}\big(|\mathcal{N}(0,1)|\leq x\big) \tag{23}$$

Here the left-hand-side corresponds to the average number of points where the true solution differs from the predicted mean by less than a factor x of the predicted standard deviation:

$$\text{i.e.}\quad \widehat{y}_d[i,j]-x\cdot\widehat{\sigma}_d[i,j]\;\leq\;y_d[i,j]\;\leq\;\widehat{y}_d[i,j]+x\cdot\widehat{\sigma}_d[i,j]\quad d\in\mathcal{D},\;[i,j]\in\Omega_d \tag{24}$$

As shown in Fig. 8, the trained model follows the Gaussian design almost perfectly in practice. In particular, the observed errors between the network predictions and true solutions are seen to closely approximate the empirical 68-95-99.7 rule for normal distributions. This experimental evidence suggests that the network's predicted error bounds do, in fact, provide an accurate measure of the model's uncertainty.

4.3. Inference speed

The numerical results for each experiment have been obtained using an 8-core Intel Xeon 3.60GHz processor and a single NVIDIA GeForce GTX 1080 GPU. To compare the inference/prediction speed of the proposed convolutional networks with the traditional FEM approach, the average inference times have been evaluated for solving 10,000 distinct PDE systems. As shown in Table 2, the convolutional networks achieve faster inference times for each problem setup, with a speed-up by a

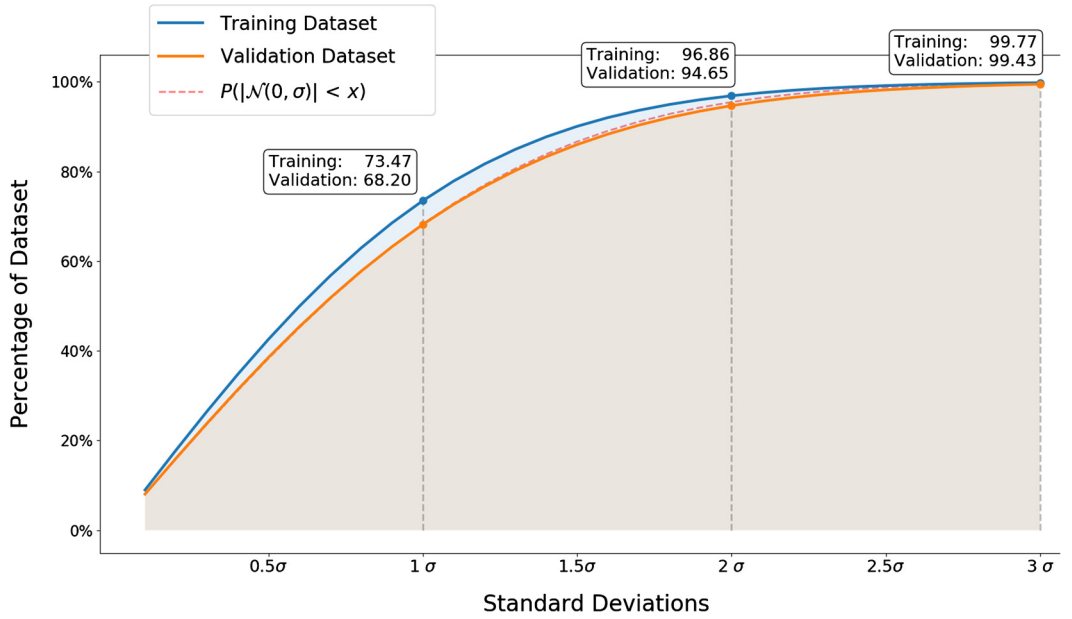


Fig. 8. Quantitative analysis of the network's predicted standard deviation values in the "Varying Domain Poisson Equation" setup. The percentage of points where the difference between the network prediction and true solution is less than a multiple of the predicted pointwise standard deviation values is shown for the training (top curve) and validation (bottom curve) data sets. The curve for the validation data set is shown to closely follow the cumulative distribution of the half-normal distribution (dashed, partially covered by validation), supporting the claim that the predicted standard deviation values do, in fact, provide accurate pointwise uncertainty estimates.

Table 2
Comparison of inference/prediction times for the convolutional and FEM solvers. The "FEniCS (Coarse)" results correspond to FEM implementations in which the meshes have been coarsened until the accuracy coincides with that of the neural network models. The accuracies of the coarse FEM implementation and neural network models have been measured with respect to the refined mesh "FEniCS" results which serve as proxies for the true solutions.

Problem Setup	FEniCS	FEniCS (Coarse)	Network	Network (GPU)
Poisson on Circle	0.05064 seconds	0.03872 seconds	0.01157 seconds	0.00171 seconds
Varying Domain	0.04432 seconds	0.03395 seconds	0.01160 seconds	0.00170 seconds
Nonlinear Poisson	0.04863 seconds	0.03473 seconds	0.01152 seconds	0.00164 seconds
Average	0.04786 seconds	0.03580 seconds	0.01156 seconds	0.00168 seconds

factor of 3.1 using the CPU alone and a speed-up by a factor of 21.3 using a single GPU. For a fair comparison with the batch evaluation of the convolutional networks, the FEM implementation in FEniCS has been parallelized to use all 8 cores.

5. Summary

In this paper, we propose the preliminary ConvPDE-UQ framework for the construction of approximate PDE solvers on varied two-dimensional domains by leveraging existing numerical methods and recent advances in data-driven deep learning. A theoretical justification for the use of neural networks as approximations to PDE solution mappings on varied domains is established using the theory of Green's functions and the universal approximation property of neural networks. This framework replaces online tasks such as mesh generation, finite element space construction, and linear/nonlinear system solvers with a one-shot, offline training session. After training, predictions can be obtained on any specified two-dimensional domain via a single forward-pass through a light-weight convolutional network. The networks are also designed to provide pointwise error bounds along with the approximate solutions; this is achieved using a simplified deterministic training procedure which avoids the computationally expensive inference steps of BNNs and allows training to be scaled up to work with the large data sets required for learning on varied geometries.

The performance of the framework has been demonstrated for both linear and nonlinear heterogeneous elliptic PDEs on varied two-dimensional domains with homogeneous Dirichlet boundary conditions. To accommodate more general PDE systems, future work will be directed toward modeling inhomogeneous and mixed boundary conditions. The proposed framework can also be naturally extended to handle inhomogeneous PDE coefficients, such as stiffness terms, by simply adding additional channels to the network's input. The experimental results of this work provide a foundation for the construction of more general neural network solvers in the future; it is envisioned that the long-term development of this

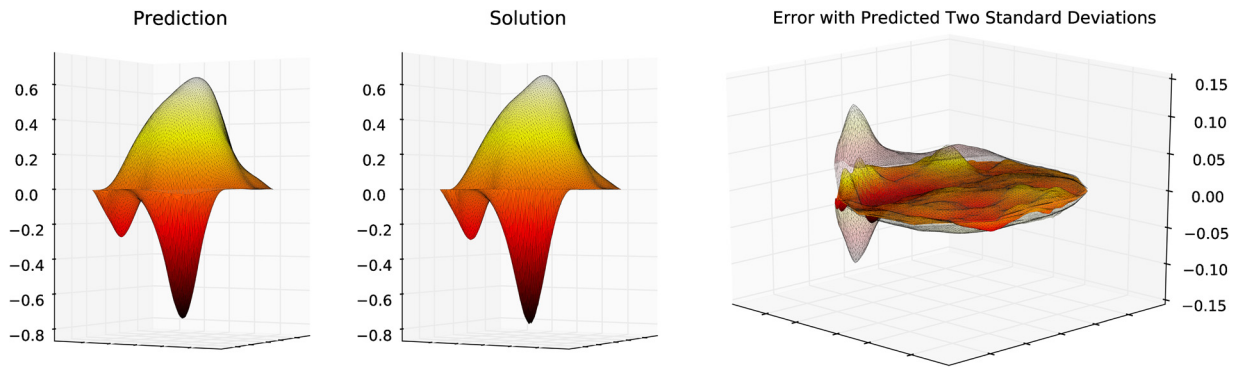


Fig. A.9. Qualitative results for the "Variable Coefficient" problem setup.

Table A.3
Quantitative results for the "Variable Coefficient" problem setup.

Problem Setup	Model	L^1 Relative Error		L^2 Relative Error	
Variable Coefficient	**Probability**	**2.06e−2**	**2.62e−2**	**7.95e−4**	**1.33e−3**
	MSE ($\lambda = 0.1$)	4.25e−2	4.54e−2	2.82e−3	3.42e−3
	MSE ($\lambda = 0.0$)	3.81e−2	4.19e−2	2.28e−3	3.00e−3

framework will result in networks capable of solving linear and nonlinear heterogeneous PDEs with variable coefficients and mixed boundary conditions on arbitrary domains.

Acknowledgements

We gratefully acknowledge the support from the National Science Foundation (DGE-1144843, DMS-1555072, DMS-1736364, and DMS-1821233).

Appendix A. Additional experiments

A.1. Variable coefficient differential operators

The proposed framework provides a natural extension for handling variable coefficient differential operators as well. In particular, the coefficients of the operator can be passed to the network by simply concatenating the coefficient arrays with the source term array. Each channel of the network's input then corresponds to a specific coefficient or source term in the differential equation. For example, we may consider the elliptic differential equation:

$$\begin{cases} \operatorname{div}\left(a \cdot \nabla u\right) = f & \text{in} \quad \Omega \\ u = 0 & \text{on} \quad \partial\Omega \end{cases} \tag{A.1}$$

where the coefficient $a \in C^{\infty}(\overline{\Omega})$ is subject to the elliptic coercivity constraint $\inf_{\Omega} a > \kappa$ for some fixed $\kappa > 0$. The same randomization procedure used to generate the source terms f can be applied to sample coefficient terms a with one additional step designed to enforce the coercivity constraint: the values of each coefficient array are rescaled and shifted to have a mean value of 1.0 and satisfy $\inf_{\Omega} a > \kappa$ with $\kappa = 0.2$.

In this case, the network input consists of two channels: one for the source term f and another for the coefficient a. Conveniently, no additional modification to the network architecture is required; the first convolutional layer identifies the additional input channel, adds the necessary filters for parsing the channel, and passes the extracted features to subsequent hidden layers without further modification. The qualitative and quantitative results for the variable coefficient problem setup defined in Equation (A.1) are provided in Fig. A.9 and Table A.3, respectively.

A.2. Neumann boundary conditions

The proposed framework is also compatible with homogeneous Neumann boundary conditions. In particular, denoting the outward unit normal vector along the boundary of the domain Ω by $\vec{n}$, we consider the following problem:

$$\begin{cases} \Delta u = f & \text{in} \quad \Omega \\ \dfrac{\partial u}{\partial \vec{n}} = 0 & \text{on} \quad \partial\Omega \end{cases} \tag{A.2}$$

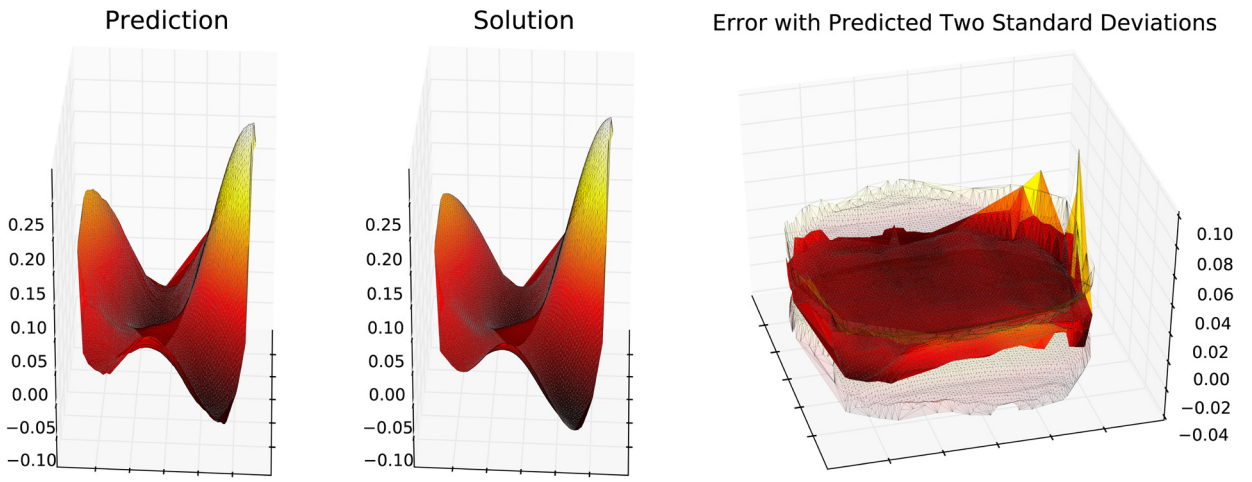


Fig. A.10. Qualitative results for the "Neumann" problem setup. The network's predictive uncertainty is observed to increase sharply near the boundary of the domain, forming a bowl-like shape. The boundary of the domain is also where the most severe inaccuracies in the network's predictions occur, and this information is reflected in the uncertainty estimates provided by the network.

Table A.4
Quantitative results for the "Neumann" problem setup.

Problem Setup	Model	L^1 Relative Error		L^2 Relative Error	
	Probability	**4.23e−2**	**4.64e−2**	**1.78e−2**	**1.84e−2**
Neumann BC	MSE ($\lambda = 0.1$)	1.29e−1	1.33e−1	3.18e−2	3.38e−2
	MSE ($\lambda = 0.0$)	1.31e−1	1.34e−1	3.26e−2	3.46e−2

In this situation, no modification to the network architecture is required. The training dataset is simply modified to reflect the Neumann boundary conditions, and the network naturally adapts its predictions to approximate the target boundary behavior.

In practice, however, the network is observed to perform poorly on domains with sharp corners in the Neumann boundary condition setup. This is a rather natural difficulty since the outward normal derivative constraints at domain vertices include severe discontinuities. To minimize the impact this has on the network's training procedure, the dataset has been generated with an additional domain smoothing step intended to remove sharp corners from the randomized geometries.

The qualitative and quantitative results for the Neumann boundary condition setup defined in Equation (A.2) are provided in Fig. A.10 and Table A.4, respectively.

Appendix B. Alternative distributions for modeling network uncertainty

A natural extension to the proposed uncertainty schema can be obtained by replacing the normally distributed network predictions with an alternative distribution. For example, the likelihood associated with independent pointwise normal distributions:

$$p_{\text{Normal}}(y\,;\,\mu,\sigma) \;=\; \prod_{i,j=1}^{R} \frac{1}{\sqrt{2\pi\cdot\sigma[i,j]^2}} \exp\Big(-\tfrac{1}{2}\big(y[i,j]-\mu[i,j]\big)^2/\sigma[i,j]^2\Big) \tag{B.1}$$

can be replaced with the likelihood associated with independent pointwise Laplace or Cauchy distributions:

$$p_{\text{Laplace}}(y\,;\,\mu,b) \;=\; \prod_{i,j=1}^{R} \frac{1}{2\cdot b[i,j]} \exp\big(-\big|y[i,j]-\mu[i,j]\big|/b[i,j]\big) \tag{B.2}$$

$$p_{\text{Cauchy}}(y\,;\,\mu,\gamma) \;=\; \prod_{i,j=1}^{R} \left(\pi\cdot\gamma[i,j]\cdot\left(1+\frac{1}{\gamma[i,j]^2}\big(y[i,j]-\mu[i,j]\big)^2\right)\right)^{-1} \tag{B.3}$$

Experiments with these alternative implementations show that the network is still able to achieve comparable levels of accuracy with respect to the mean squared error. In both cases, however, the predictive uncertainties fail to capture the empirical distribution of the network's errors. Indeed, after inspecting the empirical distribution of the network errors, it is quite evident that the errors remain normally distributed despite the change to the loss function.

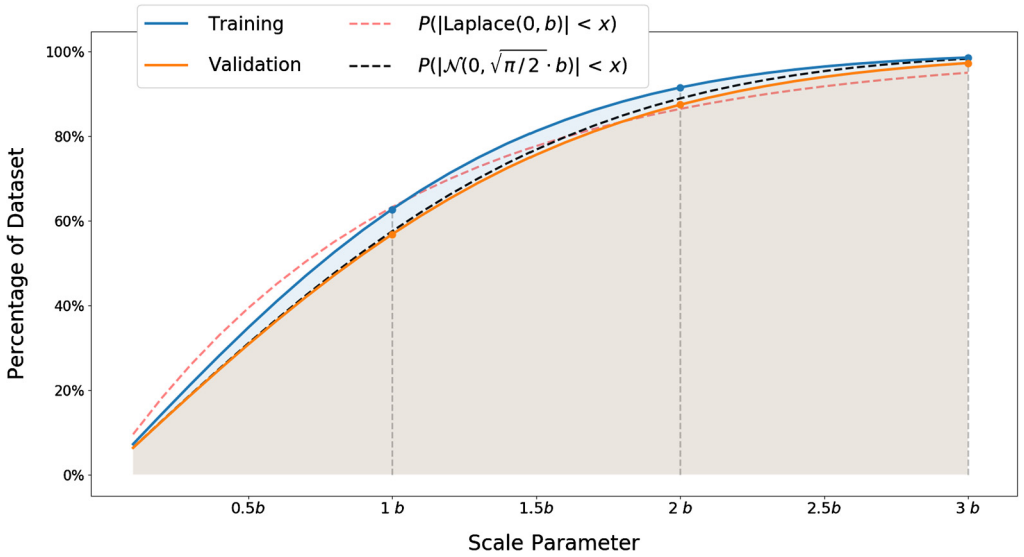


Fig. B.11. Quantitative analysis of the network's predicted uncertainties for the "Varying Domain Poisson Equation" using the Laplace distribution for uncertainty quantification. The network's errors are observed to be better fit by normal distributions with standard deviations rescaled by the factor of $\sqrt{\pi/2}$ derived in the MLE calculation for the Laplace scale parameter b.

In this section, we provide a detailed analysis of these alternative uncertainty models in order to provide insight into how the network's uncertainty schema works. In particular, we show that the network does, in fact, succeed in fitting the respective distributions to the data. However, since the network errors are observed to be normally distributed in practice, the fit of the Laplace and Cauchy distributions do not provide a faithful representation of the network's uncertainty. The true uncertainty can be easily recovered, however, by considering the relationship between the optimal scale parameters of the distributions and the standard deviations of the normal data to which they are fit.

B.1. Laplace uncertainty model

To understand how the use of the Laplace loss function affects the uncertainty schema, we consider the result of fitting a Laplace distribution to normally distributed data. Following the approach of Kundu [57], we consider the maximum likelihood estimator (MLE) for the scale parameter b of a mean-zero Laplace distribution corresponding to independent identically distributed observations $\{x_i\}_{i=1}^N$:

$$\text{MLE}(b) \;=\; \frac{1}{N}\sum_{i=1}^{N} |x_i| \tag{B.4}$$

Converting to a continuous setting, and assuming that the data is distributed as $x \sim \mathcal{N}(0,\sigma)$, we have:

$$\text{MLE}(b) \;=\; \int_{\mathbb{R}} \frac{|x|}{\sqrt{2\pi\sigma^2}}\, e^{-x^2/2\sigma^2}\, dx \;=\; \sqrt{\frac{2}{\pi}} \int_0^\infty \frac{x}{\sigma}\cdot e^{-x^2/2\sigma^2}\, dx \;=\; \sqrt{\frac{2}{\pi}} \int_0^\infty \sigma\, e^{-u}\, du \;=\; \sqrt{\frac{2}{\pi}}\,\sigma \tag{B.5}$$

Thus, the scale parameter b of a Laplace distribution which is inappropriately fit to normally distributed data should converge to the true standard deviation σ scaled by a factor of $\sqrt{2/\pi}$. In particular, we can recover the true standard deviation σ of the normally distributed data from the MLE calculation for b by simply rescaling by a factor of $\sqrt{\pi/2}$. Indeed, this is precisely the result of training the network in the "Varying Domain Poisson Equation" setup using the Laplace loss function, as illustrated in Fig. B.11.

B.2. Cauchy uncertainty model

An analogous analysis can be carried out for the Cauchy uncertainty model by considering the maximum likelihood estimator (MLE) for the scale parameter γ of a mean-zero Cauchy distribution associated with independent identically distributed observations $\{x_i\}_{i=1}^N$. The MLE criterion for the scale parameter γ has been derived in Equation 2 of [58] and is as follows:

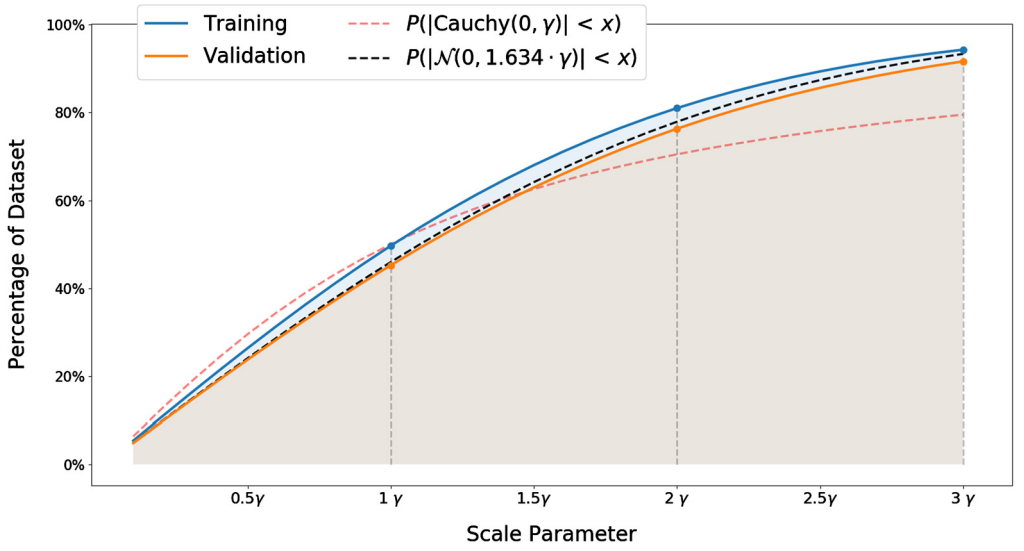


Fig. B.12. Quantitative analysis of the network's predicted uncertainties for the "Varying Domain Poisson Equation" using the Cauchy distribution for uncertainty quantification. The network's errors are seen to be better fit by normal distributions with standard deviations rescaled by the factor of 1.634 derived in the MLE calculation for the Cauchy scale parameter γ.

$$\text{MLE}(\gamma) = \gamma \quad \text{such that} \quad \frac{1}{N}\sum_{i=1}^{N}\frac{\gamma^2}{x_i^2+\gamma^2} = \frac{1}{2} \tag{B.6}$$

Converting to a continuous setting, and assuming that the data is distributed as $x \sim \mathcal{N}(0,\sigma)$, we have:

$$\text{MLE}(\gamma) = \gamma \quad \text{such that} \quad \int_{\mathbb{R}} \frac{\gamma^2}{x^2+\gamma^2}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-x^2/2\sigma^2}\,dx = \frac{1}{2} \tag{B.7}$$

Using Equation 7 in Section 3.2 of [59], with $a = 1/\sigma\sqrt{2}$ and $z = \gamma$, the integral in consideration can be expressed in terms of the complementary error function $\text{erfc}(x) = 1 - \text{erf}(x) = 1 - 2/\sqrt{\pi}\int_0^x e^{-t^2}\,dt$:

$$\text{erfc}\left(\frac{\gamma}{\sigma\sqrt{2}}\right) = \frac{2\gamma}{\pi}e^{-\gamma^2/2\sigma^2}\int_0^\infty \frac{e^{-x^2/2\sigma^2}}{x^2+\gamma^2}\,dx = \frac{\gamma}{\pi}e^{-\gamma^2/2\sigma^2}\int_{\mathbb{R}}\frac{e^{-x^2/2\sigma^2}}{x^2+\gamma^2}\,dx \tag{B.8}$$

Solving for the integral on the right hand side and rescaling by $\gamma^2/\sqrt{2\pi\sigma^2}$ yields:

$$\int_{\mathbb{R}} \frac{\gamma^2}{x^2+\gamma^2}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-x^2/2\sigma^2}\,dx = \sqrt{\frac{\pi}{2}}\cdot\frac{\gamma}{\sigma}\cdot e^{\gamma^2/2\sigma^2}\cdot\text{erfc}\left(\frac{\gamma}{\sigma\sqrt{2}}\right) \tag{B.9}$$

The optimal value of γ can now be approximated numerically by minimizing the difference between the expression on the right and the target value of $1/2$. In particular, we find that $\gamma \approx 0.612$ in the case of a unit standard deviation $\sigma = 1$ and that the scale parameter γ scales linearly with the standard deviation σ.

Thus, the MLE calculation for the scale parameter γ can be approximated by $\gamma \approx 0.612\cdot\sigma$, and the true standard deviation of the normally distributed data can be recovered via $\sigma \approx 1.634\cdot\gamma$. This again aligns perfectly with the results of training the network in the "Varying Domain Poisson Equation" setup using the Cauchy loss function, as illustrated in Fig. B.12.

These results strongly suggest that the network has in fact accurately estimated the optimal scale parameters b and γ for the Laplace and Cauchy distributions, respectively. The mismatch between the predicted uncertainties and observed errors is simply a consequence of fitting a non-normal distribution to normally distributed data. As detailed above, the correct uncertainty can be recovered by observing that the errors are in fact normally distributed; the correction factors from the MLE calculations can then be used to derive the true standard deviations from the predicted scale parameter values.

References

[1] H. Lee, I.S. Kang, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1) (1990) 110–131.

[2] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[3] J. Takeuchi, Y. Kosugi, Neural network representation of finite element method, Neural Netw. 7 (2) (1994) 389–395.
[4] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[5] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049.
[6] A. Malek, R.S. Beidokhti, Numerical solution for high order differential equations using a hybrid neural network-optimization method, Appl. Math. Comput. 183 (1) (2006) 260–271.
[7] E. Weinan, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. 5 (4) (2017) 349–380.
[8] J. Sirignano, K. Spiliopoulos, Dgm: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[9] R. Tripathy, I. Bilionis, Deep uq: learning deep neural network surrogate models for high dimensional uncertainty quantification, J. Comput. Phys. 375 (2018) 565–588.
[10] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10561.
[11] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part ii): data-driven discovery of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10566.
[12] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[13] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. 3 (5) (1990) 551–560.
[14] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 303–314.
[15] P. Hennig, S. Hauberg, Probabilistic solutions to differential equations and their application to Riemannian statistics, in: Artificial Intelligence and Statistics, 2014, pp. 347–355.
[16] O.A. Chkrebtii, D.A. Campbell, B. Calderhead, M.A. Girolami, et al., Bayesian solution uncertainty quantification for differential equations, Bayesian Anal. 11 (4) (2016) 1239–1267.
[17] Y.P. Wang, Y. Cheng, Z.Y. Zhang, G. Lin, Calibration of reduced-order model for a coupled Burgers equations based on pc-enkf, Math. Model. Nat. Phenom. 13 (2018) 21.
[18] Y. Wang, K. Hu, L. Ren, G. Lin, Optimal observations-based retrieval of topography in 2d shallow water equations using pc-enkf, J. Comput. Phys. 382 (2019) 43–60.
[19] Y.P. Wang, L.L. Ren, Z.Y. Zhang, G. Lin, C. Xu, Sparsity-promoting elastic net method with rotation for high-dimensional nonlinear inverse problem, Comput. Methods Appl. Mech. Eng. 345 (2019) 263–282.
[20] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: representing model uncertainty in deep learning, in: International Conference on Machine Learning, 2016, pp. 1050–1059.
[21] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, in: Advances in Neural Information Processing Systems, 2017, pp. 6402–6413.
[22] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, in: Advances in Neural Information Processing Systems, 2017, pp. 5574–5584.
[23] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.
[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[25] O. Ronneberger, P. Fischer, T. Brox, U-net: convolutional networks for biomedical image segmentation, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2015, pp. 234–241.
[26] R.C. Kirby, Algorithm 839: Fiat, a new paradigm for computing finite element basis functions, ACM Trans. Math. Softw. 30 (4) (2004) 502–516, https://doi.org/10.1145/1039813.1039820.
[27] R.C. Kirby, M.G. Knepley, A. Logg, L.R. Scott, Optimizing the evaluation of finite element matrices, SIAM J. Sci. Comput. 27 (3) (2005) 741–758, https://doi.org/10.1137/040607824.
[28] R.C. Kirby, A. Logg, A compiler for variational forms, ACM Trans. Math. Softw. 32 (3) (2016) 417–444, https://doi.org/10.1145/1163641.1163644.
[29] R.C. Kirby, A. Logg, L.R. Scott, A.R. Terrel, Topological optimization of the evaluation of finite element matrices, SIAM J. Sci. Comput. 28 (1) (2006) 224–240, https://doi.org/10.1137/050635547.
[30] R.C. Kirby, L.R. Scott, Geometric optimization of the evaluation of finite element matrices, SIAM J. Sci. Comput. 29 (2) (2007) 827–841.
[31] R.C. Kirby, A. Logg, Efficient compilation of a class of variational forms, ACM Trans. Math. Softw. 33 (3) (2007) 17, https://doi.org/10.1145/1268769.1268771.
[32] A. Logg, Automating the finite element method, Arch. Comput. Methods Eng. 14 (2) (2007) 93–138, https://doi.org/10.1007/s11831-007-9003-9.
[33] K.B. Ølgaard, A. Logg, G.N. Wells, Automated code generation for discontinuous Galerkin methods, SIAM J. Sci. Comput. 31 (2) (2008) 849–864, https://doi.org/10.1137/070710032.
[34] R.C. Kirby, A. Logg, Benchmarking domain-specific compiler optimizations for variational forms, ACM Trans. Math. Softw. 35 (2) (2008) 1–18, https://doi.org/10.1145/1377612.1377614.
[35] A. Logg, Efficient representation of computational meshes, Int. J. Comput. Sci. Eng. 4 (4) (2009) 283–295, https://doi.org/10.1504/IJCSE.2009.029164.
[36] M.E. Rognes, R.C. Kirby, A. Logg, Efficient assembly of h(div) and h(curl) conforming finite elements, SIAM J. Sci. Comput. 31 (6) (2009) 4130–4151, https://doi.org/10.1137/08073901X.
[37] M.S. Alnæs, A. Logg, K.-A. Mardal, O. Skavhaug, H.P. Langtangen, Unified framework for finite element assembly, Int. J. Comput. Sci. Eng. 4 (4) (2009) 231–244, https://doi.org/10.1504/IJCSE.2009.029160.
[38] M.S. Alnæs, K.-A. Mardal, On the efficiency of symbolic computations combined with code generation for finite element methods, ACM Trans. Math. Softw. 37 (1) (2010) 6, https://doi.org/10.1145/1644001.1644007.
[39] A. Logg, G.N. Wells, Dolfin: automated finite element computing, ACM Trans. Math. Softw. 37 (2) (2010) 20, https://doi.org/10.1145/1731022.1731030.
[40] K.B. Ølgaard, G.N. Wells, Optimisations for quadrature representations of finite element tensors through automated code generation, ACM Trans. Math. Softw. 37 (1) (2010) 8, https://doi.org/10.1145/1644001.1644009.
[41] J. Hoffman, J. Jansson, R.V. de Abreu, C. Degirmenci, N. Jansson, K. Müller, M. Nazarov, J.H. Spühler, Unicorn: parallel adaptive finite element simulation of turbulent flow and fluid-structure interaction for deforming domains and complex geometry, Comput. Fluids 80 (2013) 310–319.
[42] N. Jansson, J. Jansson, J. Hoffman, Framework for massively parallel adaptive finite element computational fluid dynamics on tetrahedral meshes, SIAM J. Sci. Comput. 34 (1) (2012) C24–C41.
[43] M.E. Rognes, D.A. Ham, C.J. Cotter, A.T.T. McRae, Automating the solution of pdes on the sphere and other manifolds in fenics 1.2, Geosci. Model Dev. 6 (2013) 2099–2119, https://doi.org/10.5194/gmd-6-2099-2013.

[44] M.S. Alnæs, A. Logg, K.B. Ølgaard, M.E. Rognes, G.N. Wells, Unified form language: a domain-specific language for weak formulations of partial differential equations, ACM Trans. Math. Softw. 40 (2) (2012) 9, https://doi.org/10.1145/2566630.
[45] M.S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, The fenics project version 1.5, Arch. Numer. Softw. 3 (100) (2015), https://doi.org/10.11588/ans.2015.100.20553.
[46] A. Logg, K.-A. Mardal, G.N. Wells, et al., Automated Solution of Differential Equations by the Finite Element Method, Springer, 2012.
[47] A. Logg, G.N. Wells, J. Hake, DOLFIN: A C++/Python Finite Element Library, Springer, 2012, Ch. 10.
[48] A. Logg, K.B. Ølgaard, M.E. Rognes, G.N. Wells, FFC: The FEniCS Form Compiler, Springer, 2012, Ch. 11.
[49] R.C. Kirby, FIAT: Numerical Construction of Finite Element Basis Functions, Springer, 2012, Ch. 13.
[50] M.S. Alnæs, K.-A. Mardal, SyFi and SFC: Symbolic Finite Elements and Form Compilation, Springer, 2012, Ch. 15.
[51] M.S. Alnæs, A. Logg, K.-A. Mardal, UFC: A Finite Element Code Generation Interface, Springer, 2012, Ch. 16.
[52] M.S. Alnæs, UFL: A Finite Element Form Language, Springer, 2012, Ch. 17.
[53] J. Hoffman, J. Jansson, N. Jansson, C. Johnson, R.V. de Abreu, Turbulent Flow and Fluid-Structure Interaction, Springer, 2012, Ch. 28.
[54] D. Gilbarg, N.S. Trudinger, Elliptic Partial Differential Equations of Second Order, Springer, 2015.
[55] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, arXiv preprint, arXiv:1312.6114.
[56] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint, arXiv:1412.6980.
[57] D. Kundu, Discriminating between normal and Laplace distributions, in: Advances in Ranking and Selection, Multiple Comparisons, and Reliability, Springer, 2005, pp. 65–79.
[58] J. Copas, On the unimodality of the likelihood for the Cauchy distribution, Biometrika 62 (3) (1975) 701–704.
[59] E.W. Ng, M. Geller, A table of integrals of the error functions, J. Res. Natl. Bur. Stand. B 73 (1) (1969) 1–20.