Journal of Computational Mathematics and Data Science 1 (2021) 100008

Contents lists available at ScienceDirect

Journal of Computational Mathematics and Data Science

journal homepage: www.elsevier.com/locate/jcmds

Neural networks as smooth priors for inverse problems for PDEs

Jens Berg, Kaj Nyström [*]

Department of Mathematics, Uppsala University, SE-751 06 Uppsala, Sweden

ARTICLE INFO

Keywords:
Inverse problems
Ill-posed problems
Partial differential equations
Neural networks
The finite element method

ABSTRACT

In this paper we discuss the potential of using artificial neural networks as smooth priors in classical methods for inverse problems for PDEs. Exploring that neural networks are global and smooth function approximators, the idea is that neural networks could act as attractive priors for the coefficients to be estimated from noisy data. We illustrate the capabilities of neural networks in the context of the Poisson equation and we show that the neural network approach show robustness with respect to noisy, incomplete data and with respect to mesh and geometry.

1. Introduction

Inverse problems are well motivated and challenging problems in the sciences and engineering. In this paper we discuss the classical coefficient approximation problem for partial differential equations (PDEs). This inverse problem consists of determining the coefficient(s) of a PDE given more or less noisy measurements of its solution. A typical example is the heat distribution in a material with unknown thermal conductivity. Given measurements of the temperature at certain locations, we are to estimate the thermal conductivity of the material by solving the inverse problem for the underlying equation.

This type of inverse problem is of both practical and theoretical interest. From a practical point of view the governing equation for some physical process is often known, but the material and its properties are not. From a theoretical point of view, the inverse problem is a challenging often ill-posed problem in the sense of Hadamard. Inverse problems have been studied for a long time with important contributions due to [1,2] and [3,4] and [5]. Today there is a vast literature devoted to inverse problems and PDE constrained optimization problems but here we have no ambition to give a detailed bibliography. Instead, for overviews of these topics, and as examples of relevant survey papers and textbooks, we refer the reader to [6–9] and the references therein.

As the underlying PDE cannot in general be solved analytically, a numerical method is required. A common choice is the finite element method (FEM) which have been frequently used with success for large classes of inverse problems, see for example [10–13]. Recently, there has also been a lot of activity in a branch of numerical analysis known as probabilistic numerics, with successful applications in inverse problems and in more general PDE constrained optimization problems, see for example [14–21].

In this paper we, for the sake of illustration, work with the scalar Poisson equation

$$-\nabla \cdot (q(x)\nabla u(x)) = f(x), \tag{1.1}$$

in a bounded Lipschitz domain $\Omega \subset \mathbb{R}^N$ and subject to Dirichlet boundary data defined by a function g. $q : \Omega \to \mathbb{R}$ is a scalar real-valued function. The problem in (1.1) is considered in the weak sense and we let $L^2 = L^2(\Omega)$ denote the standard Hilbert space of square integrable real-valued function on Ω and we let $H^1 := H^1(\Omega)$ be the first order Sobolev space of functions u such that u and its first order weak derivatives exist and belong to L^2. We let $B^{2,2}_{1/2} := B^{2,2}_{1/2}(\partial\Omega)$ denote the appropriate Besov space defined as the trace space of H^1 to $\partial\Omega$ (this space is often denoted $H^{1/2}(\partial\Omega)$ in the literature). It is well known, see [22] for example, that if Ω is a bounded Lipschitz domain, then there exists a bounded continuous operator $T : H^1 \to B^{2,2}_{1/2}$, called the trace operator,

[*] Corresponding author.
E-mail addresses: jens.berg@math.uu.se (J. Berg), kaj.nystrom@math.uu.se (K. Nyström).

https://doi.org/10.1016/j.jcmds.2021.100008
Received 21 May 2021; Received in revised form 8 August 2021; Accepted 18 September 2021

2772-4158/© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

and a bounded continuous operator $E : B^{2,2}_{1/2} \to H^1$ called the extension operator. We let $H^{-1} := H^{-1}(\Omega)$ denote the space dual to $H^1_0 := H^1_0(\Omega)$ where the latter is the set of all $u \in H^1$ with trace 0 ($Tu = 0$) on $\partial\Omega$. In particular, given $g \in B^{2,2}_{1/2}$, $u \in H^1$ is a weak solution to (1.1) if and only of $(u - Eg) \in H^1_0$ and if $A(q, u) = 0$ is the sense that

$$A(q,u)(v) := b(q,u,v) - L(v) = 0, \tag{1.2}$$

for all $v \in H^1_0$, where

$$b(q,u,v) := \int_\Omega q(x)\nabla u(x) \cdot \nabla v(x) dx, \;\; L(v) := \langle f, v\rangle, \tag{1.3}$$

and where $\langle \cdot, \cdot\rangle$ denotes the duality coupling in H^{-1}. To ensure solvability of the forward problem, ellipticity of the operator on Ω is assumed by assuming that q is uniformly elliptic on Ω. Under these assumptions the weak formulation of the forward problem in (1.1), (1.2), has a unique solution in $H^1 = H^1(\Omega)$. Given q this is the solution $u = u(q)$ considered throughout the paper. Well-posedness for the Dirichlet problem is derived from the Lax–Milgram theorem and we refer to [22] for details.

For the model problem in (1.1), (1.2) a class of inverse problems can be defined as

$$q^* := \operatorname*{argmin}_{q\in\mathcal{D}} \{ I(d_0, \mathcal{Q}(u(q))) : A(q,u) = 0\}, \tag{1.4}$$

where $\mathcal{D}$ is the admissible set of scalar real-valued functions. In this formulation

$$\mathcal{Q} : H^1 \to \mathcal{O} \tag{1.5}$$

is an observation operator which maps the direct solution to an observable quantity, a quantity of interest. $d_0 \in \mathcal{O}$ denotes actual observation data belonging to observation data space. $I : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$ is an error/misfit functional (sometimes referred to as the misfit functional). For example, in practice it is often important to compute the coefficient q based on an appropriate error/misfit functional and given a series of more or less noisy measurements $\hat{u}_i$ of u, for $i = 0, \dots, M$. It is clear that the inverse problem is ill-posed whenever $\nabla u = 0$ on an open set, as in this case there is no information available about the coefficient q on that set.

In general well-posedness of the inverse problem depends on the norm under consideration and, as discussed in the bulk of the paper, in this paper we will consistently use and discuss error/misfit functionals of the form

$$J(u,q) := \frac{1}{2}\|u - \hat{u}\|^2 + R := \frac{1}{2}\int_\Omega |u - \hat{u}|^2 dx + R(q), \tag{1.6}$$

where $u = u(q)$. $R = R(q)$ is a real-valued non-negative functional representing regularization. The goal is then to compute q^* such that

$$q^* = \operatorname*{argmin}_{q} J(u,q) \tag{1.7}$$

subject to u satisfying (1.1), (1.2). In this formulation $\mathcal{Q}$ is the identity operator and it is assumed that we have measurements of u, denoted $\hat{u}$, at all points in Ω. In real applications the error/misfit functional is discretized. One can argue for different error/misfit functionals but as PDE constrained optimization in the H^1-norm in general is not practically applicable, since one does usually not have measurements of $\nabla\hat{u}$, we will in this paper consistently use the error/misfit functional defined in (1.6). For results concerning the existence of Frechet differentials of error/misfit functionals, applicable in our context, we refer to [23] for a recent account.

To solve the underlying PDE, i.e. (1.1), (1.2), we will use FEM. To effectively use FEM, a finite element space V for the solution (u) is selected, and another space Q is selected for the coefficient (q). Given q we solve for $u = u(q)$ and we iteratively search for q minimizing the error/misfit functional while solving the underlying PDE. One choice for Q is the space of piecewise constants but many other specifications are feasible. Depending on the regularity of the local basis functions used for V, regularization of the error/misfit functional is often needed to reconstruct a smooth coefficient. Frequently Tikhonov regularization is used in the error/misfit functional and one considers the functional

$$\frac{1}{2}\int_\Omega |u - \hat{u}|^2 dx + \frac{\alpha}{2}\int_\Omega |q|^2 dx, \tag{1.8}$$

where $\alpha > 0$ is a regularization parameter. If, for some reason, an estimate of the coefficient and noise level is known a priori, one can further improve the results by considering a generalized Tikhonov regularization of the form

$$\frac{1}{2}\int_\Omega |u - \hat{u}|^2 dx + \frac{\alpha}{2}\int_\Omega |q - q_*|^2 dx, \tag{1.9}$$

where q_* is an a priori estimate of the coefficient. The actual value of the regularization parameter α depends on the problem at hand, and finding an optimal value is a non-trivial task. The optimal generalized Tikhonov regularization parameter α can be computed by using, for example, the Morozov discrepancy principle or heuristic L-curves. These methods are often of limited practical use in a PDE constrained optimization context as they become extremely expensive. However, see Appendix for a discuss of the optimal regularization of the 1D Poisson problem.

The purpose of this paper is to present neural networks as a possible ansatz for q, and in particular as potential priors (q_*) to classical methods for inverse problems. We will, to convey the idea, only use the most basic settings: we will use a tiny network with only three neurons in the hidden layer and we will avoid any fine tuning of hyperparameters, or any other parameters, with the ambition to get as much of a black box solution strategy as possible. In particular, this means that when implementing the neural networks approach we will not use any regularization in the error/misfit functional (1.6). Still, it is well known that large neural networks are, in general, at the risk of overfitting. In particular with noisy data, a network with too high capacity could potentially fit the noise and this causes a severe increase in the number of optimization iterations, and a lack of generalization. Therefore larger networks require weight regularization, dropout and/or any other method which reduces overfitting and we refer to [24–26] for more on the topic of regularization of neural networks. In particular, similar to the case of Tikhonov regularization, optimal regularization of deep neural networks is a non-trivial problem.

While FEM is a vehicle to solve the underlying PDE, the representation of q can be seen as a separate question. However, in the context of FEM it is reasonable to make use of the discretization of the domain (the spaces V and Q) when representing q. The neural network ansatz makes no particular reference to the discretization/elements used in FEM. In this sense two approaches emerge: one where u and q are tied to the discretization of the domain used in FEM and one where q is represented by a neural network while u is found using the discretization of the domain used in FEM. In both cases, given q we solve for $u = u(q)$ and we iteratively search for q minimizing the error/misfit functional while solving the underlying PDE. The two approaches discussed are the methods compared in this paper. However, let us be clear once and for all: the purpose of this paper is not to say that the second approach can necessarily serve as a replacement for the first approach or any other technique used to solve the type of inverse problem considered in this paper. Instead, the idea is to illustrate some of the contributions/capabilities that neural networks, and potentially deep neural networks, can give/bring to the important field of inverse problems for PDEs. For example, we believe that the idea to use a hybrid approach where one in a first step calculates a prior based on an ansatz with neural networks, and then in a second step resolve the inverse problem, with a generalized Tikhonov regularization based on the neural network prior constructed, with established FEM or other methods, is worth further exploration. In particular, the comparisons we make in our numerical illustrations, between the first approach (u and q are tied to the discretization of the domain used in FEM) and the second approach (q is represented by a neural network while u is found using the discretization of the domain used in FEM), should be seen as illustrations of the capabilities of the second approach, and not necessarily as the lack of capabilities of the first approach. In particular, if not seen in this way our illustrations and comparisons are not fair to the first approach, but finding the optimal function spaces and regularizations are beyond the scope of this paper.

In this paper we will use the finite element method as supplied by the software package `FEniCS` [27,28] to solve the underlying PDE. To compute the gradient we need for the PDE constrained optimization, we use the software `dolfin-adjoint` [29] which works together with `FEniCS` to automatically derive and solve the adjoint PDE. Finally, we combine `FEniCS` and `dolfin-adjoint` with an artificial neural network to represent the unknown coefficient(s) in the PDE. The neural networks we consider are simple feed-forward neural networks with sigmoid activation functions in the hidden layers, and linear activations in the output layer. Such a neural network defines a smooth mapping $\mathbb{R}^N \to \mathbb{R}$ which can approximate, in theory and at the expense of potentially having to use a very wide neural network, any continuous function to arbitrary accuracy [30].

After this paper,[1] and its companions [32,33], were written a few new contributions to the study of inverse problems for PDEs using neural networks have been posted. While we believe that the study of the use of neural networks to solve inverse problems of the nature considered in this paper still is at an early stage we note a more recent contributions in [34] which seems to contain ideas similar to what we propose in this paper. More generally we refer to [18,19,33–37] and the references therein, for more on solving PDEs (not only related to the type of inverse problems studied in this paper) using neural network techniques.

The rest of paper is organized as follows. Section 2 is of preliminary nature and we here discuss adjoint equations and differentiation at some length as we believe this could be of value for the reader. Note that parts of our derivations here are formal but all calculations can be made rigorous using the weak formulation of the problems and the functional setting outlined above. Concerning FEM our discussion is quite brief, one reason being that we here simply use the FEM software `FEniCS` and that the method is well documented. An other reason is, as discussed above, that our ambition is not to evaluate FEM per se in our context, rather the idea is to illustrate some of the important features of neural networks for the field of inverse problems for PDEs. However, it is important to emphasize that FEM is used to solve the forward problem, while neural networks are used as the ansatz for the unknown and sought for coefficient q. We end Section 2 with a discussion of the more elementary neural networks used in this paper and we have here also include some material concerning the fundamental ANN approximation problem. In Section 3 and Section 4 we illustrate the use of the neural network approach to recover q in (1.1). All our illustrations are based on the same idea: we first fix q and u and we then let $f(x) := -\nabla \cdot (q(x)\nabla u(x))$. This defines a PDE as in (1.1). We then generate a dataset representing measurements of u (denoted $\hat{u}$) by sampling u, with noise added, on the grid used in the discretization for FEM. Once this is done we try to reconstruct q by producing approximations $\hat{q}$ of q using the two approaches outlined above. In our illustrations the error/misfit functional is implemented as the integral of a high-order interpolation onto the finite element space as provided by the `errornorm` function in `FEniCS`, and $\hat{u}$ is the unperturbed exact solution. In particular, in our illustrations we consistently produce our own data. In Section 3 we discuss what we call moderately ill-posed examples and in these examples the coefficients to be estimated are smooth and the measurement data is available in the whole domain. In contrast, in Section 4 we discuss some examples of severely ill-posed problems where we face discontinuous coefficients or incomplete measurements. To mention a few, what we believe, particularly interesting things discussed in these sections we mention the discussion concerning mesh independent convergence, the inverse problem for the three-dimensional Poisson problem in a complex geometry from [38], and the problem with incomplete data where we illustrate that neural networks can reconstruct a coefficient in the whole domain despite that data is being known only close to the boundaries. Finally, in Section 5 we provide a summary, state some conclusions and indicate directions for future research.

[1] A first version of the paper [31] was posted on ArXiv in December 2017.

2. Preliminaries

The problem in (1.1), (1.2), can formally be stated

$$\begin{aligned} A(q,u) &= 0, \quad \text{in } \Omega, \\ Tu &= g, \quad \text{on } \partial\Omega, \end{aligned} \tag{2.1}$$

where T is the trace operator, g the boundary data, and $q = q(x)$ is the a priori unknown and sought for scalar function. The weak formulation of the problem and the functional setting was outlined in detailed in the introduction.

It is important to recall that well-posedness of the type of inverse problems considered in this paper depends on the norm under consideration. For example, consider the Poisson equation in (1.1) in one space dimension, $x \in \Omega := (0,\pi)$, $q = 1/2$, $u = x^2$, and a sequence of coefficients and observations $\hat{q}_N = (2+\cos(Nx))^{-1}$, $\hat{u}_N = u + \delta_N$ with

$$\delta_N = \frac{x}{N}\sin(Nx) + \frac{1}{N^2}\cos(Nx). \tag{2.2}$$

In the L^∞-norm given by

$$\|u\|_\infty = \max_{x\in\Omega} |u(x)|, \tag{2.3}$$

we then have

$$\|\hat{u}_N - u\|_\infty \le \frac{c}{N} \to 0, \quad \|\hat{q}_N - q\|_\infty = \frac{1}{2} \not\to 0, \tag{2.4}$$

and we can clearly see that $\|\hat{q}_N - q\|_\infty \not\to 0$ though $\|\hat{u}_N - u\|_\infty \to 0$ and hence the problem is not well-posed in the sense of Hadamard. Still, as shown in [39] this problem is well-posed in (H^1, H^{-1}) in the sense that

$$\|q_1 - q_2\|_{H^{-1}} \le c\|u(q_1) - u(q_2)\|_{H^1}. \tag{2.5}$$

As previously mentioned, as PDE constrained optimization in the H^1-norm in general is not practically applicable, since one does usually not have measurements of $\nabla\hat{u}$, we in this paper consistently use the error/misfit functional defined in (1.6).

To solve the minimization problem (1.7), we will use the gradient of the error/misfit functional (1.6) with respect to q. To compute the gradient, we will use the adjoint approach as we next discuss in some detail. We here simply convey the ideas and our derivations are to some extent formal but all calculations can be made rigorous using the weak formulation of the problems and the functional setting outlined in the introduction. For rigorous justifications of the existence of Frechet differentials of error/misfit functionals, applicable in our context, we refer the reader for example to [23] and the references therein.

2.1. *The adjoint equations*

q defines a mapping

$$q \mapsto u(q), \tag{2.6}$$

which can be computed by solving (2.1) by analytical or numerical means. As a result, $A(q,u) \mapsto A(q,u(q))$, $J(u,q) \mapsto J(u(q),q)$, and the PDE constrained problem (1.7) becomes an unconstrained problem as the constraint is built into the solution of (2.1). The total derivative of the error/misfit functional can formally be computed as

$$\frac{\mathrm{d}J(u(q),q)}{\mathrm{d}q} = \frac{\partial J(u(q),q)}{\partial u(q)}\frac{\partial u(q)}{\partial q} + \frac{\partial J(u(q),q)}{\partial q}, \tag{2.7}$$

where $\partial J(u(q),q)/\partial u(q)$ and $\partial J(u(q),q)/\partial q$ are in general easily computed. To compute the Jacobian $\partial u(q)/\partial q$, we formally differentiate the operator $A(q,u(q))$ defining the PDE itself to obtain

$$\frac{\mathrm{d}A(q,u(q))}{\mathrm{d}q} = \frac{\partial A(q,u(q))}{\partial u(q)}\frac{\partial u(q)}{\partial q} + \frac{\partial A(q,u(q))}{\partial q}, \tag{2.8}$$

or equivalently

$$-\frac{\partial A(q,u(q))}{\partial u(q)}\frac{\partial u(q)}{\partial q} = \frac{\partial A(q,u(q))}{\partial q}. \tag{2.9}$$

Obviously the statements in the last two display have to be interpreted with care but in the case of (1.1), (1.2), the rigorous formulation of (2.9) is

$$-b(q, \frac{\partial u(q)}{\partial q}, v) = b(1,u,v) \tag{2.10}$$

for all $v \in H_0^1$ where b is introduced in (1.3). In general the relation in (2.9) is known as the tangent linear system related to the functional (1.6). In the rest of the derivations, we will write $A(q,u(q)) = A$, $J(u(q),q) = J$ and $u(q) = u$ to ease the notation. The tangent linear operator $\partial A/\partial u$ is the linearization of the differential operator around the solution u which acts on the solution Jacobian $\partial u/\partial q$. In the case of a linear PDE, for example (1.1), (1.2), the differential operator underlying (2.1) and the tangent linear operator coincide as we have seen. Stressing generalities, assuming that the tangent linear system is invertible, we can use (2.1) and (2.9) to solve

$$\frac{\partial u}{\partial q} = -\left(\frac{\partial A}{\partial u}\right)^{-1} \frac{\partial A}{\partial q}, \tag{2.11}$$

and we note that in the case of (1.1), (1.2), the equation in (2.11) is given explicit through (2.10). Sticking to generalities, substituting (2.11) into (2.7) and taking the transpose gives

$$\frac{\mathrm{d}J^*}{\mathrm{d}q} = -\frac{\partial A^*}{\partial q}\left(\frac{\partial A^*}{\partial u}\right)^{-1} \frac{\partial J^*}{\partial u} + \frac{\partial J^*}{\partial q}. \tag{2.12}$$

We define the *adjoint* variable λ by

$$\lambda = -\left(\frac{\partial A^*}{\partial u}\right)^{-1} \frac{\partial J^*}{\partial u}, \tag{2.13}$$

or equivalently

$$-\frac{\partial A^*}{\partial u}\lambda = \frac{\partial J^*}{\partial u}. \tag{2.14}$$

Eq. (2.14) is the *adjoint*, or *dual*, equation associated with the *forward* problem (2.1) and the error/misfit functional (1.6). By solving the adjoint equation (2.14) and substituting into (2.7), we can compute the total derivative of the error/misfit functional as

$$\frac{\mathrm{d}J}{\mathrm{d}q} = \lambda^* \frac{\partial A}{\partial q} + \frac{\partial J}{\partial q}. \tag{2.15}$$

With the total derivative of the error/misfit functional given by (2.15), any gradient based optimization method can be used to solve the minimization problem (1.7). In the case of (1.1), (1.2) we have explicitly, due to symmetry of the underlying form, that $\lambda^* = \lambda$ and

$$\frac{\mathrm{d}J}{\mathrm{d}q} = \lambda \frac{\partial A}{\partial q} + \frac{\partial J}{\partial q}. \tag{2.16}$$

where

$$\lambda = -\left(\frac{\partial A}{\partial u}\right)^{-1} \frac{\partial J}{\partial u}. \tag{2.17}$$

In particular,

$$-b(q, \lambda, v) = \langle \frac{\partial J}{\partial u}, v \rangle \tag{2.18}$$

for all $v \in H_0^1$, where b is introduced in (1.3), and the adjoint equation is of the same form as the forward problem.

2.2. The finite element method and automatic differentiation

As mentioned in the introduction, in this paper we have used the FEM software `FEniCS` to solve the PDE (2.1), together with `dolfin-adjoint` to automatically derive and solve the adjoint equation (2.14), (2.18). The automatic differentiation in `dolfin-adjoint` works by overloading the solver functions in `FEniCS` and recording a graph of solver operations. Each node in the graph has an associated gradient operation and the total derivative can be computed by reversed mode differentiation, i.e., backpropagation. The reversed mode differentiation has the benefit that the gradient computation is exact with respect to the FEM discretization, as long as it is smooth enough. The only source of error in the computation of (2.15), (2.16) is thus in the numerical solution of the forward problem. At least this is true as long as the adjoint equation can be solved by a direct method. For large enough systems, an iterative method must be used which introduces additional errors. These are, however, usually small compared to the discretization errors of the forward problem.

2.3. Fully connected feed-forward artificial neural network

In this paper, we consider deep, fully connected feed-forward ANNs. The ANN consists of $L+1$ layers, where layer 0 is the input layer and layer L is the output layer. The layers $0 < l < L$ are the hidden layers. The activation functions in the hidden layers can be any activation function such as for example sigmoids, rectified linear units, or hyperbolic tangents. Unless otherwise stated, we will use sigmoids in the hidden layers. The output activation will be the linear activation. In general, the ANN defines a mapping $\mathbb{R}^N \to \mathbb{R}^M$.

Each neuron in the ANN is supplied with a bias, including the output neurons but excluding the input neurons, and the connections between neurons in subsequent layers are represented by matrices of weights. We let b_j^l denote the bias of neuron j in layer l. The weight between neuron k in layer $l-1$, and neuron j in layer l is denoted by w_{jk}^l. The activation function in layer l will be denoted by σ_l regardless of the type of activation. We assume for simplicity that a single activation function is used for each layer. The output from neuron j in layer l will be denoted by y_j^l. See Fig. 1 for a schematic representation of a fully connected feed-forward ANN.

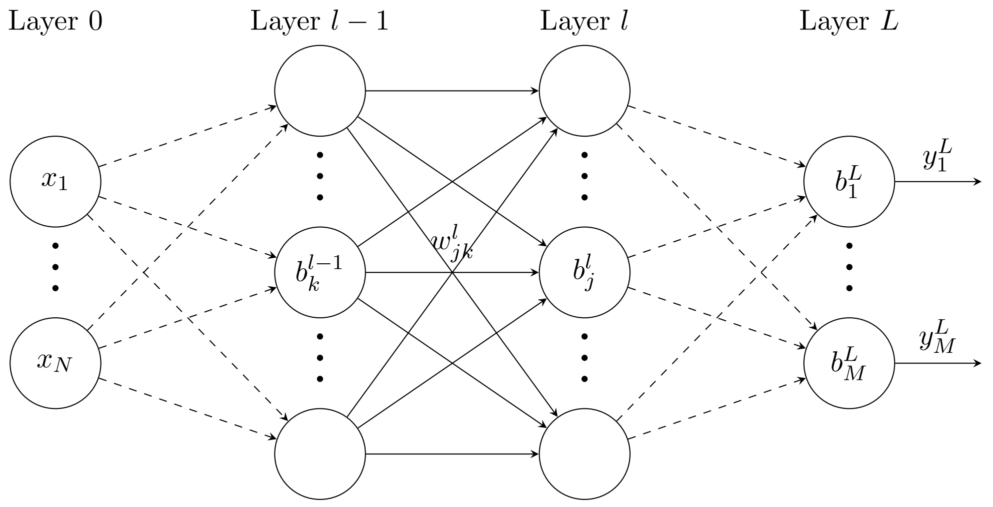


Fig. 1. Schematic representation of a fully connected feed-forward ANN.

The so-called weighted input which is defined as

$$z_j^l = \sum_k w_{jk}^l \sigma_{l-1}(z_k^{l-1}) + b_j^l, \tag{2.19}$$

where the sum is taken over all inputs to neuron j in layer l. That is, the number of neurons in layer $l-1$. The weighted input (2.19) can of course also be written in terms of the output from the previous layer as

$$z_j^l = \sum_k w_{jk}^l y_k^{l-1} + b_j^l, \tag{2.20}$$

where the output $y_k^{l-1} = \sigma_{l-1}(z_k^{l-1})$ is the activation of the weighted input. As we will be working with deep ANNs, we will prefer formula (2.19) as it naturally defines a recursion in terms of previous weighted inputs through the ANN. By definition we have

$$\sigma_0(z_j^0) = y_j^0 = x_j, \tag{2.21}$$

which terminates any recursion. By dropping the subscripts we can write (2.19) in the convenient vectorial form

$$z^l = W^l \sigma_{l-1}(z^{l-1}) + b^l = W^l y^{l-1} + b^l, \tag{2.22}$$

where each element in the z^l and y^l vectors are given by z_j^l and y_j^l, respectively, and the activation function is applied elementwise. The elements of the matrix W^l are given by $W_{jk}^l = w_{jk}^l$.

With the above definitions, the feed-forward algorithm for computing the output y^L, given the input x, is given by

$$\begin{aligned}
y^L &= \sigma_L(z^L) \\
z^L &= W^L \sigma_{L-1}(z^{L-1}) + b^L \\
z^{L-1} &= W^{L-1}\sigma_{L-2}(z^{L-2}) + b^{L-1} \\
&\vdots \\
z^2 &= W^2 \sigma_1(z^1) + b^2 \\
z^1 &= W^1 x + b^1.
\end{aligned} \tag{2.23}$$

For the calibration of feed-forward artificial neural network and more general deep neural networks using back propagation we refer to [26,40–42]. For more references and for a modern and modestly technical overview of several aspects of deep learning we refer to the popular book [43].

2.4. Neural network representation of the coefficient

Rather than representing the unknown coefficient in a finite element space, we can represent the coefficient by a feed-forward artificial neural network. That is, we let $q = q(x; W, b)$ be parameterized by the weights and biases of a neural network, here denoted by W and b. This approach yields some immediate benefits. A neural network is a global, smooth, function approximator which can approximate any continuous function to arbitrary accuracy by having a sufficient amount of parameters, see [30,44], and Section 2.5 below for a brief discussion of the ANN approximation problem. Since neural networks are global functions, they can be cheaply evaluated anywhere in the domain without first searching for the correct mesh element and performing interpolation, the latter being an expensive operation.

Following the discussion of the previous subsection, with $M = 1$, the coefficient q is computed by using (2.23)

$$\begin{aligned} q(x) &= \sigma(z^L) \\ z^L &= W^L \sigma_{L-1}(z^{L-1}) + b^L \\ z^{L-1} &= W^{L-1}\sigma_{L-2}(z^{L-2}) + b^{L-1} \\ &\vdots \\ z^2 &= W^2\sigma_1(z^1) + b^2 \\ z^1 &= W^1 x + b^1, \end{aligned} \tag{2.24}$$

where $W^l_{ij} = w^l_{ij}$ are the weight matrices connecting layers l and $l-1$ and b^l are the vectors of biases for each layer, and σ_l is the activation function of layer l. To determine weights and biases, we seek W^*, b^* such that

$$W^*, b^* = \underset{W,b}{\operatorname{argmin}} J. \tag{2.25}$$

As there is one extra layer of parameters, we need to apply the chain rule once more to compute the total derivative of the error/misfit functional. Let p denote any of the weight or biases parameters. Then

$$\frac{\mathrm{d}J}{\mathrm{d}p} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial q}\frac{\partial q}{\partial p} + \frac{\partial J}{\partial q}\frac{\partial q}{\partial p} = \left(\frac{\partial J}{\partial u}\frac{\partial u}{\partial q} + \frac{\partial J}{\partial q}\right)\frac{\partial q}{\partial p}. \tag{2.26}$$

The first factor on the right hand side of (2.26) is computed as before by using `FEniCS` and `dolfin-adjoint`. The last factor is the gradient of the network output with respect to the network parameters, which can be computed exactly by using backpropagation or automatic differentiation.

Note that the neural network augmentation is not restricted to FEM. Any numerical method which efficiently can solve the forward problem (2.1), and the adjoint problem (2.14), can be used in combination. The neural network is simply a prior for the coefficient with good function approximation properties, and efficient means for computing its gradient. The application of the chain rule in (2.26) factors the discretization of the forward and backward problems from the computation of the gradient of the network. A positive effect of having a neural network representation of the coefficient is that the number of optimization parameters is independent of the number of degrees of freedom in the discretization. In the simplest FEM case, the coefficient is represented as a constant on each mesh element. The number of optimization parameters is then equal to the number of mesh elements, which quickly become infeasible for large scale problems.

2.5. The ANN approximation problem

Let u_1 and u_2 solve (1.1), (1.2), with unknown scalar functions q_1 and q_2 respectively. Then

$$A(q_i, u_i)(v) = b(q_i, u_i, v) - L(v) = 0, \tag{2.27}$$

for all $v \in H^1_0$ and for $i \in \{1, 2\}$. In particular, $v = u_1 - u_2$ is a valid test function for the equation in (2.27). Using this one deduces, using the Cauchy Schwarz inequality and elementary estimates, that

$$\int_\Omega q_1(x)|\nabla(u_1 - u_2)(x)|^2 dx \le c\big(\sup_{x\in\Omega} |q_1(x) - q_2(x)|\big)^2 \int_\Omega |\nabla u_2(x)|^2 dx, \tag{2.28}$$

where c is a non-negative constant independent of q_i and u_i. Furthermore, using apriori estimates we have

$$\int_\Omega |\nabla u_2(x)|^2 dx \le c(\|g\|_{B^{2,2}_{1/2}}, \|f\|_{H^{-1}}). \tag{2.29}$$

Assuming that $q_i \ge \lambda > 0$, and using the Poincare inequality for functions in H^1_0, (2.28) and (2.29) imply

$$\int_\Omega |(u_1 - u_2)(x)|^2 dx \le \hat{c}\big(\sup_{x\in\Omega} |q_1(x) - q_2(x)|\big)^2, \tag{2.30}$$

where $\hat{c}$ is a non-negative constant depending only on the dimension of space, the diameter of Ω, $\|g\|_{B^{2,2}_{1/2}}$ and $\|f\|_{H^{-1}}$. Using (2.30) we see that we can control $\|u_1 - u_2\|$ if we can control

$$\sup_{x\in\Omega} |q_1(x) - q_2(x)|. \tag{2.31}$$

We next estimate the difference $J(u_1, q_1) - J(u_2, q_2)$ and we first note, for $\hat{u}$ and R given, that

$$|J(u_1, q_1) - J(u_2, q_2)| \le \frac{1}{2}\int_\Omega |(u_1 - u_2)|(|u_1 - \hat{u}| + |u_2 - \hat{u}|)dx + |R(q_1) - R(q_2)|. \tag{2.32}$$

Hence

$$\begin{aligned}|J(u_1,q_1)-J(u_2,q_2)|^2 &\leq c\bigg(\int_\Omega |(u_1-u_2)|^2dx\bigg)\bigg(\int_\Omega |u_1-\hat{u}|^2dx+\int_\Omega |u_2-\hat{u}|^2dx\bigg)+c|R(q_1)-R(q_2)|^2\\ &= c\bigg(\int_\Omega |(u_1-u_2)|^2dx\bigg)\bigg(J(u_1,q_1)-R(q_1)+J(u_2,q_2)-R(q_2)\bigg)+c|R(q_1)-R(q_2)|^2. \end{aligned} \tag{2.33}$$

In particular, making the reasonable assumption that

$$\bigg(J(u_1,q_1)-R(q_1)+J(u_2,q_2)-R(q_2)\bigg)\leq \Lambda, \tag{2.34}$$

for some fixed and potentially large constant Λ, we can conclude that

$$|J(u_1,q_1)-J(u_2,q_2)|^2 \leq c\Lambda\int_\Omega |(u_1-u_2)|^2dx+c|R(q_1)-R(q_2)|^2. \tag{2.35}$$

Hence we deduce that

$$|J(u_1,q_1)-J(u_2,q_2)|^2 \leq c\hat{c}\Lambda\big(\sup_{x\in\Omega}|q_1(x)-q_2(x)|\big)^2+c|R(q_1)-R(q_2)|^2. \tag{2.36}$$

Given a fixed and potentially large constant Λ' we let D be the set of points x such that

$$|q_1(x)|\leq\Lambda',\ \ |q_2(x)|\leq\Lambda'. \tag{2.37}$$

Using this notation and assuming that

$$|R(q_1)-R(q_2)|\leq M\big(\sup_{x\in D}|q_1(x)-q_2(x)|\big), \tag{2.38}$$

we can combine the above to conclude that

$$|J(u_1,q_1)-J(u_2,q_2)|^2 \leq c\hat{c}\Lambda\big(\sup_{x\in\Omega}|q_1(x)-q_2(x)|\big)^2+cM^2\big(\sup_{x\in D}|q_1(x)-q_2(x)|\big)^2. \tag{2.39}$$

Let $q_1=q$, and let $q_2=\hat{q}$ where $\hat{q}$ is a ANN. Then

$$|J(u_1,q_1)-J(u_2,q_2)|^2 \leq c\hat{c}\Lambda\big(\sup_{x\in\Omega}|q(x)-\hat{q}(x)|\big)^2+cM^2\big(\sup_{x\in D}|q(x)-\hat{q}(x)|\big)^2. \tag{2.40}$$

In particular, if $J(u_1,q_1)$ realizes the infimum in the underlying minimization problem, then under the above restrictions $J(u_2,q_2)$ can be made arbitrary close to $J(u_1,q_1)$ if the unknown q can be approximated closely, in the norms defined and subject to the restrictions stated, by a ANN $\hat{q}$. While this argument can be made more rigorously at the expense of additional formalism, the essence is that the argument gives rational, and in fact even proves in the case of the Poisson equation, that in theory our ansatz for q using neural network is effective and can achieve the infimum in the minimization problem. In general though it is unclear what the size of $J(u,q)$ is at its lowest as this depends on $\hat{u}$. Note that if J takes the particular form in (1.9), then (2.40) becomes

$$|J(u_1,q_1)-J(u_2,q_2)|^2 \leq (c\hat{c}\Lambda+cM^2\Lambda')\big(\sup_{x\in\Omega}|q(x)-\hat{q}(x)|\big)^2, \tag{2.41}$$

if

$$|q_1(x)|\leq\Lambda',\ \ |q_2(x)|\leq\Lambda',\ \ |q_*(x)|\leq\Lambda', \tag{2.42}$$

for all $x\in\Omega$. In this case, if $q_*=\hat{q}$ then in theory our the method could render the size of $J(u,q)$ to be zero at its lowest.

The considerations outlined above lead us to discuss the approximation capabilities of (deep) neural networks. While, as stated, we in our numerical illustrations have mainly used small feed forward networks with one hidden layer and with sigmoid activation functions, there is an abundance of different neural network representations one could attempt to use as an ansatz for q. Recall that $q:\Omega\subset\mathbb{R}^N\to\mathbb{R}$. Several of the results stated in the literature concern the approximation by ANN considering $(0,1)^N$ as the underlying domain. These results are applicable in more general domains Ω whenever a (regular) map $(0,1)^N\to\Omega$ exists. We first recall a version of the Kolmogorov Superposition Theorem, which states that any continuous function defined on $[0,1]^N$, can be represented (exactly), for an appropriate activation function, as a neural network. For a recent and informative account of the Kolmogorov Superposition Theorem we refer to [45].

Theorem 1. *There exist N constants $\{\lambda_j\}$, $\lambda_j>0$, $\sum_{j=1}^N\lambda_j\leq 1$ and $2N+1$ strictly increasing continuous functions $\{\phi_i\}$, each of which mapping $[0,1]$ to itself, such that if $q\in C([0,1]^N)$ then q can be represented in the form*

$$q(x)=q(x_1,\dots,x_N)=\sum_{i=1}^{2N+1}h\Bigg(\sum_{j=1}^N\lambda_j\phi_i(x_j)\Bigg) \tag{2.43}$$

for some $h\in C([0,1])$ depending on q.

While Theorem 1 stresses the power of (general) neural networks to represent functions, it adds little in practice as we usually want to design the neural network without a priori knowledge of q. In the following we first discuss the approximation capabilities of networks with one or two hidden layers before giving some references to more recent results concerning the approximation capabilities of (deep) neural networks.

The approximation properties of neural network with one hidden layer is rather well understood in literature. As previously mentioned, neural networks with one hidden layer can approximate any continuous function to arbitrary accuracy by having a sufficient amount of parameters [30]. We here formulate an other result which can found in [46]. By definition, given $1 \leq p \leq \infty$ and $r \geq 1$, $q \in W^p_{r,N} := W^p_r((-1,1)^N)$, if

$$\|q\|_{W^p_{r,N}} := \sum_{0\leq k\leq r} \|\nabla^k q\|_p < \infty, \tag{2.44}$$

where $\|\cdot\|_p$ denotes the L^p norm on $(-1,1)^N$.

Theorem 2. *Let $1 \leq N' \leq N$ and let $\phi : \mathbb{R}^{N'} \to \mathbb{R}$ be a function which is infinitely differentiable in some open sphere in $\mathbb{R}^{N'}$. Assume also that there exists b in this sphere such that*

$$\nabla^k \phi(b) \neq 0 \text{ for all } k \geq 0. \tag{2.45}$$

Then there exist $N' \times N$ matrices $\{W_j\}_{j=1}^n$ with the property that there exist, given $q \in W^p_{r,N}$, coefficients $a_j(q)$ such that

$$\|q(\cdot) - \sum_{j=1}^{n} a_j(q)\phi(W_j(\cdot) + b)\|_p \leq c n^{-r/N} \|q\|_{W^p_{r,N}}. \tag{2.46}$$

Theorem 2 applies when $N' = 1$, and with $\phi = \sigma$ being the sigmoid activation function. Note that in the case $r = 1$, then Theorem 2 produces a neural network with one hidden layer and with hidden size equal to n. To achieve a precision in the approximation of $q \in W^p_{1,N}$ with a predetermined error $\varepsilon > 0$, we need

$$n^{-1/N} = \mathcal{O}(\varepsilon) \Rightarrow n = \mathcal{O}(\varepsilon^{-N}). \tag{2.47}$$

This observation indicates that neural networks with one hidden layer suffer from the curse of dimensionality in the sense that number of required neurons in the hidden layer grows exponentially with the input dimension N. A possible way to overcome this is to consider, motivated by the Kolmogorov Superposition Theorem referred to above, multi-layer neural networks. Considering neural networks with two hidden layers, in [47] it is proved that if the number of units in the hidden layers are $6N + 3$ and $3N$, respectively, then neural networks with two hidden layers can approximate any function to arbitrary precision.

Theorem 3. *There exists an analytical, strictly increasing activation function σ such that if $q \in C([0,1]^N)$ and $\varepsilon > 0$, then there exist constants d_i, c_{ij}, θ_{ij}, γ_i and vectors $\mathbf{w}^{ij} \in \mathbb{R}^N$ for which*

$$\left| q(x) - \sum_{i=1}^{6N+3} d_i \sigma\left(\sum_{j=1}^{3N} c_{ij} \sigma(\mathbf{w}^{ij} \cdot x - \theta_{ij}) - \gamma_i \right) \right| < \varepsilon$$

for all $x \in [0,1]^N$.

Note that as in the Kolmogorov Superposition Theorem, Theorem 3 only implies the approximation result for a neural network with two hidden layers for appropriate activation functions. As it turns out, these activation functions can be quite pathological to achieve the desired accuracy and its is unclear if the activation functions used in practice, for example the sigmoid activation function, can be used to achieve the approximation result.

Theorem 2 sheds some light on the relevance and potential advantage of using neural networks with two hidden layers compared to neural networks with one hidden layer to approximate functions. In general rigorous and still practically applicable theorems concerning the approximation with deep neural networks are still lacking. However, there is an emerging literature on the topic and for interesting surveys and introductions to the topic and its context, we refer the interested reader to [48–50] and the reference therein.

3. Moderately ill-posed examples

In our numerical examples we have used an exact solution and coefficient to compute a forcing function in (1.1). The exact solution is used to generate the data in the error/misfit functional. In the case of added noise, we add a normally distributed value δr, $r \in \mathcal{N}(0,1)$, for some noise level δ, to each of the interior data points. The data on the boundary is always exact.

The results differ somewhat depending on the noise and random initialization of the network's weights. In the figures and tables, we use `Numpy`'s random number generators with `numpy.random.seed(2)`, for reproducibility, to add noise and initialize the weights. We use the same initial guess for the coefficient and settings for the optimizer in both the FEM and network cases.

As discussed, the equation we use in the examples is the Poisson equation and to ensure that the coefficient is initially positive, we initialize the network's weights from a uniform distribution $\mathcal{U}[0,1)$ and all the biases are initialized to zero. Initialization from a uniform distribution is usually bad practice as it considerably slows down the rate of convergence for traditional machine learning tasks. While we have no theoretical guarantees that the coefficient remain positive along the iteration, in our experiments this turns out not to be an issue. Still, if we want to be sure the coefficient remain positive along the iteration we could simply modify the neural network structured by applying a map to the positive reals in the final (output) layer of the neural networks. Note that the endpoint is excluded by default in `numpy`'s random number generators, but this is of no practical importance.

We call the following examples moderately ill-posed as the coefficients to be estimated are smooth and the measurement data is available in the whole domain. The purpose of the examples is to show the applicability and simplicity (easy to use) of the method due to the implicit regularization by the neural network.

3.1. One-dimensional Poisson equation

The one-dimensional Poisson equation with Dirichlet boundary conditions is given by

$$\begin{aligned} -(qu_x)_x &= f, \quad x \in (0,1), \\ u(0) = g_0, &\quad u(1) = g_1, \end{aligned} \tag{3.1}$$

where f, g_0, g_1 are known, and $q = q(x) : \mathbb{R} \to \mathbb{R}_+$ is the coefficient to be estimated. We discretize the domain Ω into $M = 101$ continuous piecewise linear elements where the solutions are represented. We solve the minimization problem, using standard FEM, with the coefficient represented in the solution space and represented by a neural network, respectively. Here, we use the BFGS [51] optimizer from `SciPy` [52] as the number of optimization parameters is rather small. We iterate until the norm of the gradient of the error/misfit functional is less than 10^{-6}.

The neural network is a tiny feed forward network with one input neuron, one linear output neuron, and one hidden sigmoid layer with 3 neurons. We choose the solution u to be

$$u(x) = \sin^2(2\pi x) \tag{3.2}$$

and we will compute the inverse problem for a few different exact coefficients $\hat{q}$ and noise levels. The neural network has 10 parameters which will be optimized, compared to the FEM problem which has 101 (same as the number of elements). We consider the exact coefficients $\hat{q}(x) = 1$, $\hat{q}(x) = 1 + x$, $\hat{q}(x) = 1 + x^2$, and $\hat{q}(x) = 1 + 0.5\sin(2\pi x)$ with noise level $\delta = 0$ and $\delta = 5 * 10^{-2}$. The results can be seen in Figs. 2–5 and Table 1. The network representation is insensitive to noise and always produces smooth solutions and coefficients. FEM does not converge to smooth solutions and coefficients due to the lack of regularization in the error/misfit functional. We can also see that the number of iterations increase in the network case as the coefficient becomes more oscillatory. A deeper network with higher capacity might help to mitigate the increase as was seen in [32].

For one-dimensional problems the number of mesh elements is usually rather limited. This means that the network is at a high risk of overfitting. In particular with noisy data, a network with too high capacity will fit the noise which causes a severe increase in the number of optimization iterations, and of course also lack of generalization. This is the reason we have used a tiny network with only three neurons in the hidden layer. A larger network would require weight regularization: see the introduction for more on this and for references.

3.2. Two-dimensional Poisson

The two-dimensional Poisson equation with Dirichlet boundary conditions is given by

$$\begin{aligned} -\nabla \cdot (q\nabla u) &= f, \quad x \in \Omega, \\ u &= g, \quad x \in \partial\Omega, \end{aligned} \tag{3.3}$$

where $\Omega \subset \mathbb{R}^2$, is the computational domain, $\partial\Omega$ its boundary. The functions f, g are known, and $q = q(x,y) : \mathbb{R}^2 \to \mathbb{R}_+$ is the a priori unknown coefficient to be estimated. Here, we choose the solution to be

$$u = \sin(\pi x)\sin(\pi y), \tag{3.4}$$

for the exact coefficients $\hat{q}(x,y) = 1$, $\hat{q}(x,y) = 1 + x + y$, $\hat{q}(x,y) = 1 + x^2 + y^2$, and $\hat{q}(x,y) = 1 + 0.5\sin(2\pi x)\sin(2\pi y)$. The network is a feed-forward network with one hidden layer and with 10 neurons in the hidden layer. The optimization is performed using BFGS and we iterate until the norm of the gradient of the error/misfit functional is less than 10^{-7}.

3.2.1. Unit square with uniform mesh

We choose $\Omega = [0,1]\times[0,1]$ to be the unit square discretized by 101×101 piecewise linear elements. The coefficient represented by the network has 41 parameters, while it has 10201 when represented in the finite element space. Due to the very different nature of the optimization problems, a direct comparison is not meaningful. The BFGS method scales quadratically in complexity with the number of optimization parameters, and is only useful for small-scale optimization problems, where it is very efficient. The FEM minimization problems can instead be solved by, for example, the memory limited BFGS method [53,54].

We plot the difference between the exact and computed coefficient in Figs. 6–9 and summarize the results in Table 2. We can see the same pattern as in the 1D case. The neural network representation is insensitive to noise and we always recover a smooth coefficient even without regularization. FEM does not converge to smooth solutions or coefficients with added noise due to the lack of regularization. Note that the errors are larger at the corners of the domain. This is probably due to the fact that the optimization is performed in the L^2 norm which allows larger errors in isolated points. The errors can probably be reduced by performing the optimization under the H^1 norm instead. See for example [55] and references therein.

3.2.2. *Mesh independent convergence*

In [56] it is shown that the number of optimization iterations grows polynomially with the ratio between the volumes of the smallest and largest mesh elements, $h_{\max}/h_{\min}$. The reason is that most gradient based optimization libraries do not take the underlying function space inner product into account when computing the step size and direction. As the neural network augmentation is mesh independent, we expect that the number of iterations remains constant when locally refining the mesh, as long as the mesh is fine enough to accurately compute the solution of the forward problem.

To test the mesh independent convergence, we consider a simple test case. We solve the inverse problem for the Poisson equation (3.3), on the unit square, with coefficient $\hat{q} = 1$ and the exact solution given by (3.4) without any added noise. We locally refine the mesh in concentric circles around the center of the domain with radius $r(n) = 0.5/n$, for $n = 1, \dots, 7$, as can be seen in Fig. 10. The results is summarized in Table 3. We can clearly see that as the mesh is refined, the number of iterations required until convergence remain stable and that a stable minimum (for the minimization problem underlying the inverse problem) is found.

3.3. *Three-dimensional Poisson*

As a final example, we consider the Poisson equation (3.3) in a three dimensional domain $\Omega \subset \mathbb{R}^3$ with complex geometry as supplied by [38]. We take the exact coefficient $\hat{q} = 1$ and the analytical solution

$$u = \sin(\pi x)\sin(\pi y)\sin(\pi z) \tag{3.5}$$

with no noise and compute the inverse problem. We use a network with 10 neurons in the hidden layer which gives a total of 51 optimization parameters. Convergence was achieved after 41 BFGS iterations with gradient norm tolerance 10^{-7}, which is consistent with the result in Table 3. This indicates that the neural network approach is also rather stable with respect to geometry. The domain and difference between the exact and computed network coefficient can be seen in Fig. 11.

Remark 4. The extra BFGS iteration required in the 3D case is because the forward problem could not be solved by a direct method due to memory requirements. We used the GMRES iterative method with an incomplete LU preconditioner to solve the forward problem. The solution to the forward problem is thus slightly less accurate due to the tolerance settings of the iterative method, which in turn means that the computed error/misfit functional gradients are also less accurate and more BFGS iterations are required.

4. Severely ill-posed examples

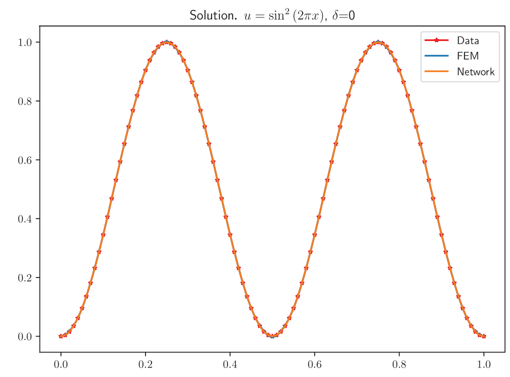
In the previous section we discussed what we called moderately ill-posed problems where we have measurements of the smooth coefficients in the whole domain. Here we will show some examples of severely ill-posed problems where we have discontinuous coefficients or incomplete measurements.

4.1. *1D discontinuous coefficient*

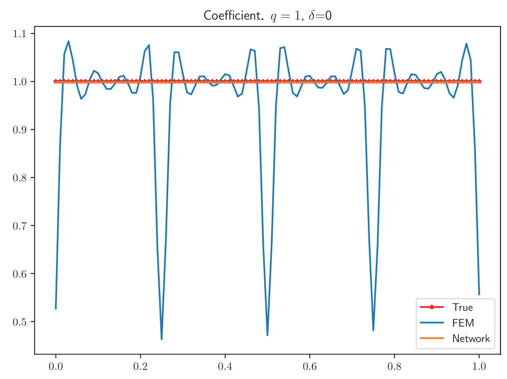
Here we repeat the examples in Section 3.1 for the Poisson problem (3.1) with

$$q = \begin{cases} 0.5, & 0 \le x < 0.5 \\ 1.5, & 0.5 \le x \le 1, \end{cases} \qquad f = 10, \qquad g_0 = g_1 = 0. \tag{4.1}$$

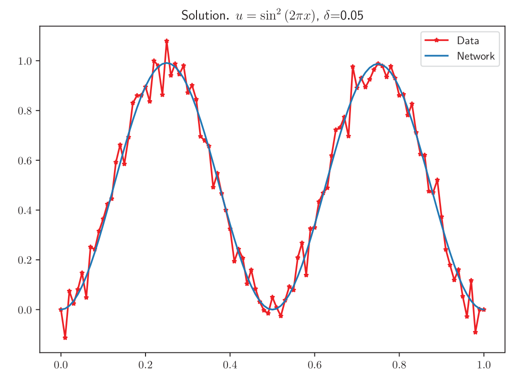
In this case, the analytical solution is not known and we compute the reference solution and data using FEM. We used the same small network with one hidden layer and with 3 neurons and the sigmoid activation function. The results can be seen in Fig. 12 and Table 4. The network captures the discontinuity perfectly for noiseless data, and the smoothing effect of the implicit regularization is clearly seen when noise is added.



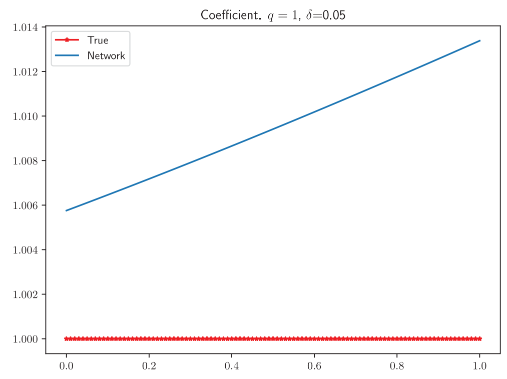
(a) Solutions with the optimized coefficients and no noise.



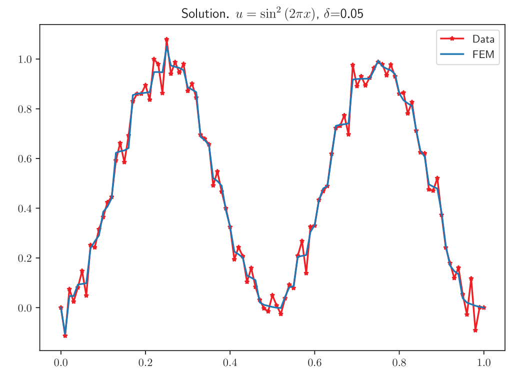
(b) The optimized coefficients without noise.



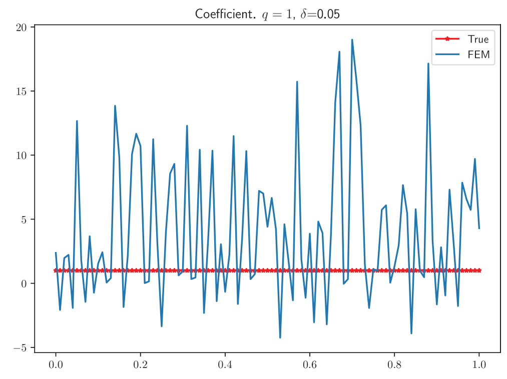
(c) Network solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient with some noise.

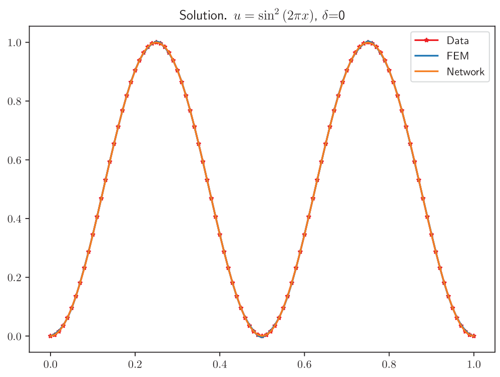


(e) FEM solution with the optimized coefficient and 5% noise level.

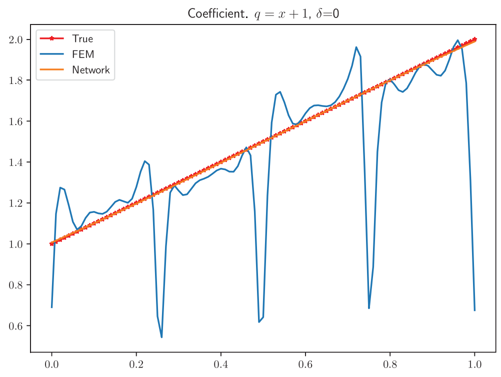


(f) The optimized FEM coefficient with some noise.

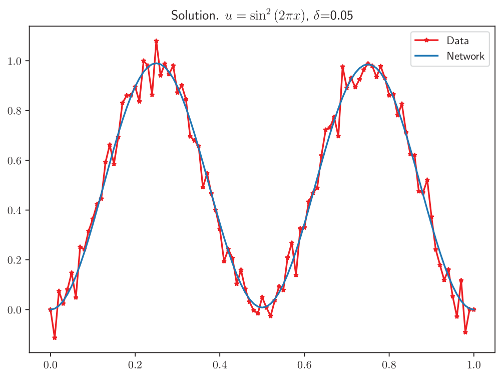
Fig. 2. Comparison between FEM and a neural network with constant coefficient $\hat{q} = 1$.



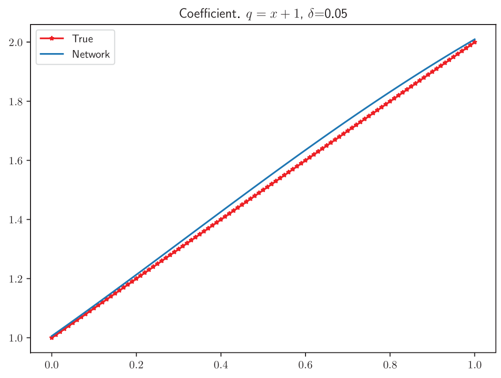
(a) Solutions with the optimized coefficients and no noise.



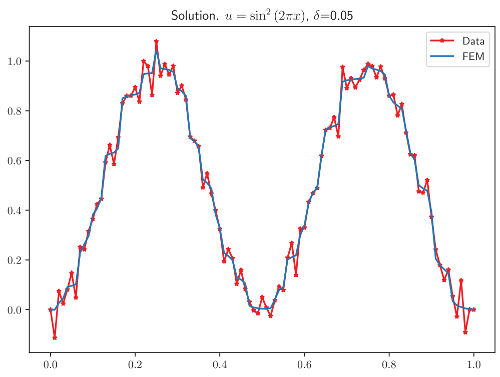
(b) The optimized coefficients without noise.



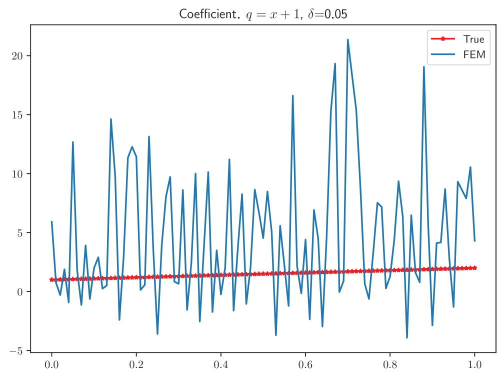
(c) Netrwork solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient with some noise.

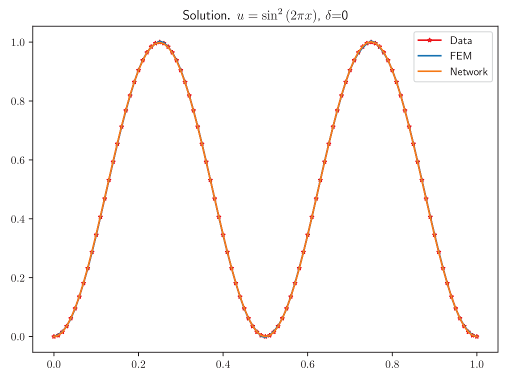


(e) FEM solution with the optimized coefficient and 5% noise level.

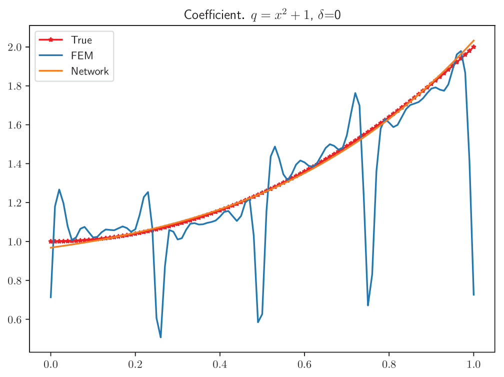


(f) The optimized FEM coefficient with some noise.

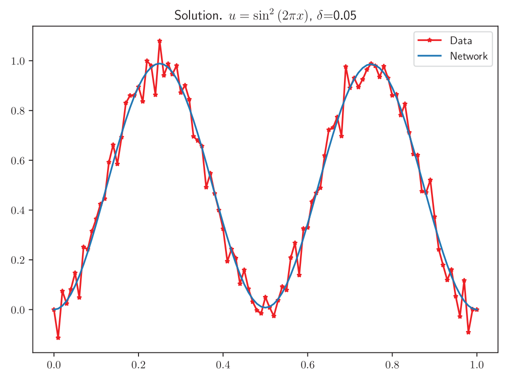
Fig. 3. Comparison between FEM and a neural network with linear coefficient $\hat{q} = 1 + x$.



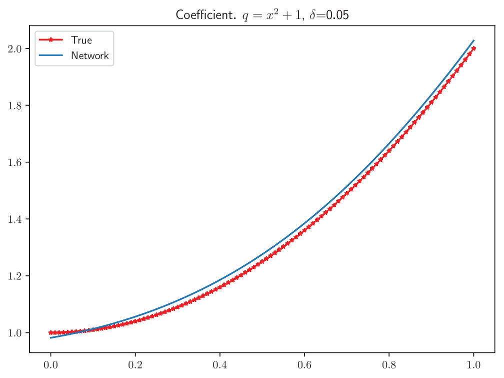
(a) Solutions with the optimized coefficients and no noise.



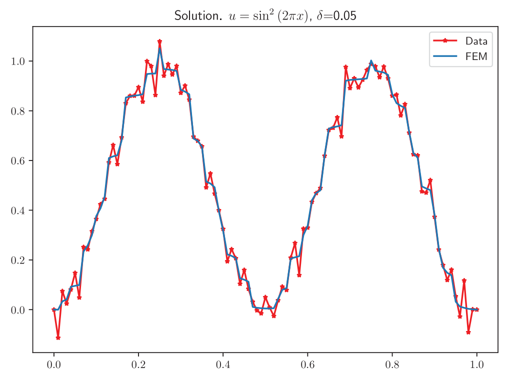
(b) The optimized coefficients without noise.



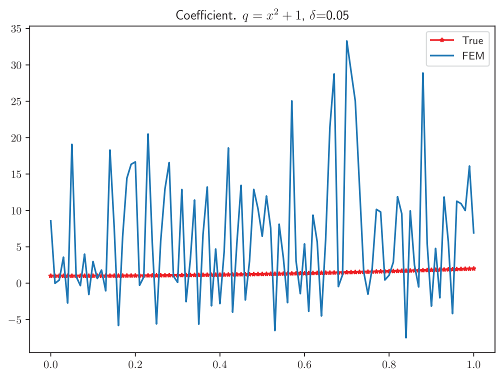
(c) Network solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient with some noise.

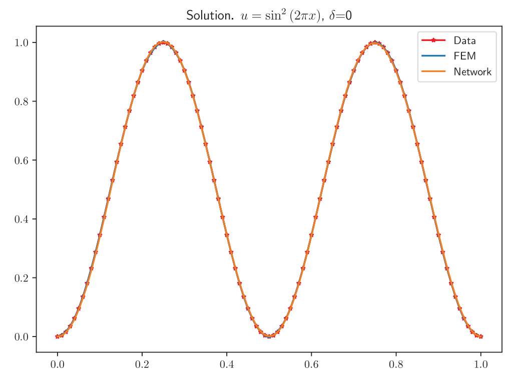


(e) FEM solution with the optimized coefficient and 5% noise level.

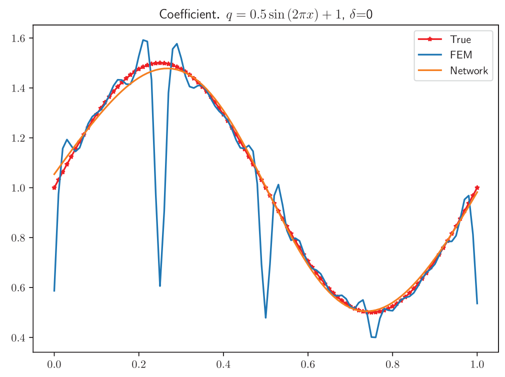


(f) The optimized FEM coefficient with some noise.

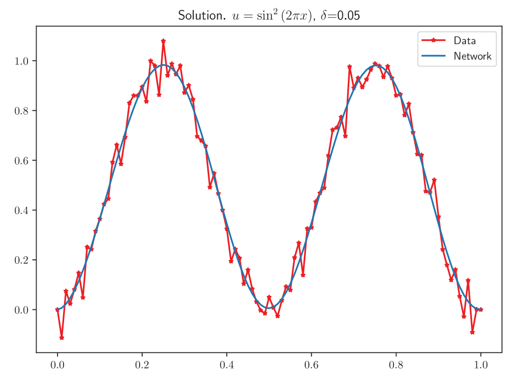
Fig. 4. Comparison between FEM and a neural network with quadratic coefficient $\hat{q} = 1 + x^2$.



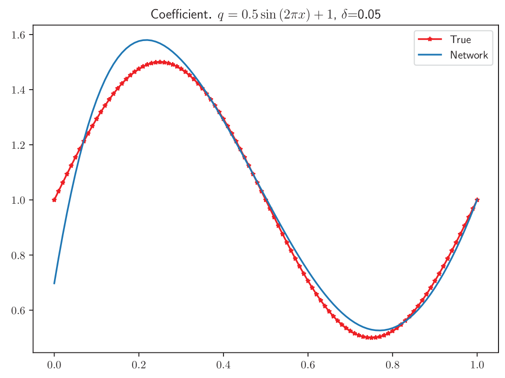
(a) Solutions with the optimized coefficients and no noise.



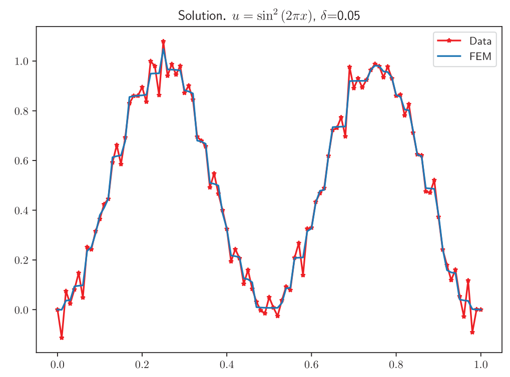
(b) The optimized coefficients without noise.



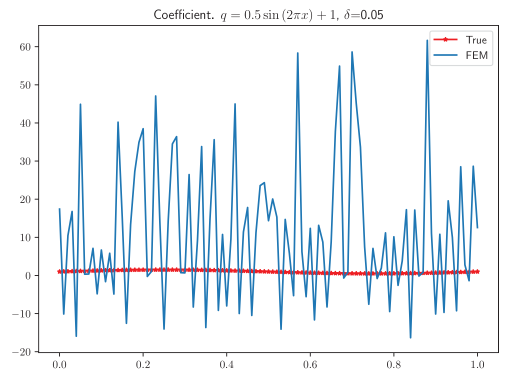
(c) Network solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient with some noise.

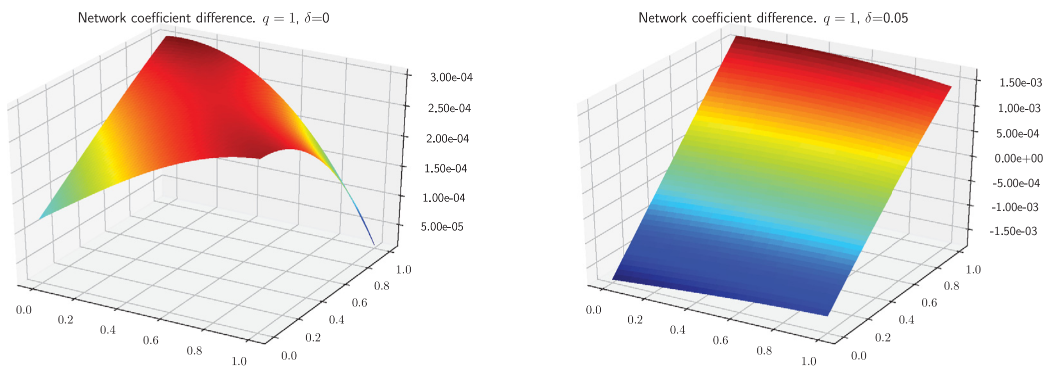


(e) FEM solution with the optimized coefficient and 5% noise level.



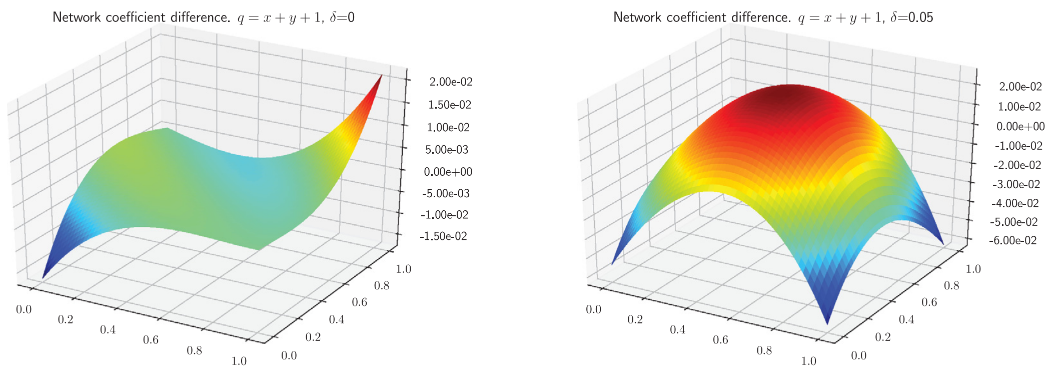
(f) The optimized FEM coefficient with some noise.

Fig. 5. Comparison between FEM and a neural network with coefficient $\hat{q} = 1 + 0.5\sin(2\pi x)$.



(a) The optimized network coefficient without noise. (b) The optimized network coefficient with some noise.

Fig. 6. Difference between the optimized network and exact coefficient when $\hat{q} = 1$.



(a) The optimized network coefficient without noise. (b) The optimized network coefficient with some noise.

Fig. 7. Difference between the optimized network and exact coefficient when $\hat{q} = 1 + x + y$.

4.2. 1D incomplete data

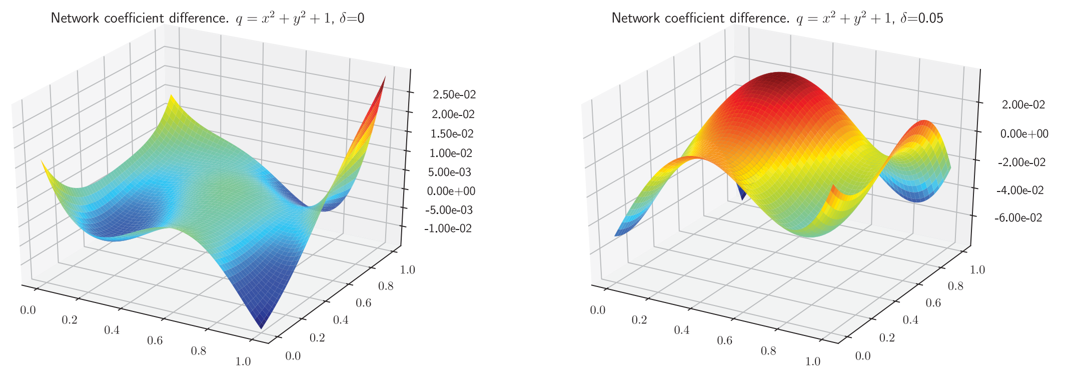
We repeat the examples in Section 3.1 for the constant coefficient case, $q = 1$, under the assumption that we only have access to measurement data at a distance d from the boundaries. In this case, the error/misfit functional we want to minimize is reduced to

$$J(u, q) = \frac{1}{2}\int_0^d |u - \hat{u}|^2 dx + \frac{1}{2}\int_{1-d}^1 |u - \hat{u}|^2 dx. \tag{4.2}$$

We let, as before, the neural network representing the unknown coefficient is a neural network with one hidden layer and with 3 neurons and the sigmoid activation function. The result can seen in Figs. 13–15 for $d = 0.4$, 0.2, and 0.1, respectively. We can clearly see that the neural network is able to reconstruct the coefficient and solution in the whole domain despite the data being known only close to the boundaries.

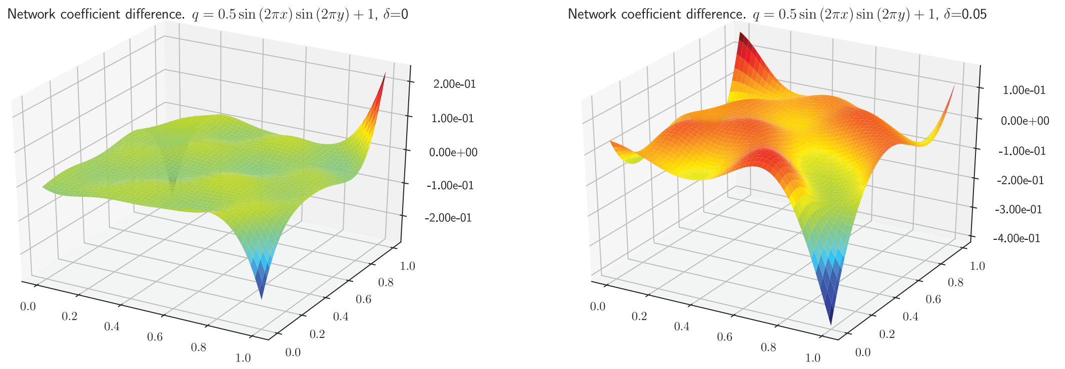
4.3. Remark on multiple discontinuities and multiscale coefficients

The neural network approach without explicit regularization works well when there are only a few discontinuities of the same size. As the number of discontinuities grow, or the scales of the coefficient start to differ too much, the low capacity networks are no



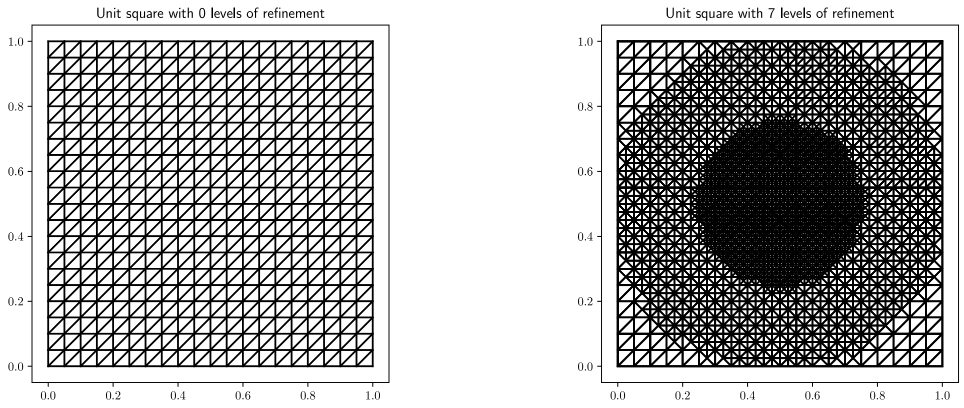
(a) The optimized network coefficient without noise. (b) The optimized network coefficient with some noise.

Fig. 8. Difference between the optimized network and exact coefficient when $\hat{q} = 1 + x^2 + y^2$.



(a) The optimized network coefficient without noise. (b) The optimized network coefficient with some noise.

Fig. 9. Difference between the optimized network and exact coefficient when $\hat{q} = 1 + 0.5\sin(2\pi x)\sin(2\pi y)$.


Unit square with 0 levels of refinement
Unit square with 7 levels of refinement
0.0
0.2
0.4
0.6
0.8
1.0

(a) Initial uniform mesh (b) Final refined mesh.

Fig. 10. Concentric mesh refinement.

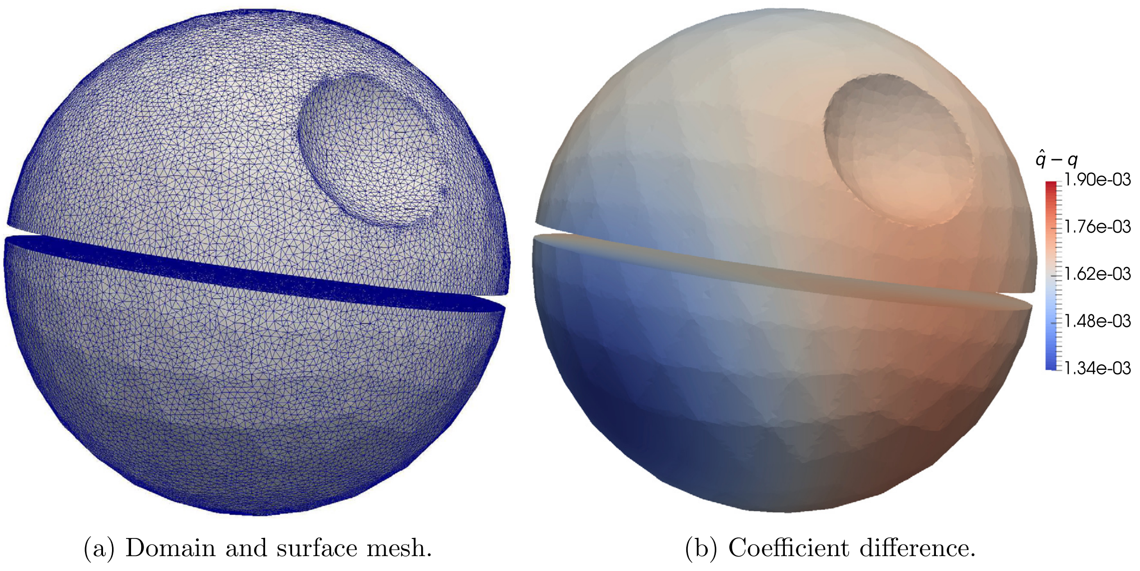


Fig. 11. The 3D Poisson equation on a complex geometry with 362172 degrees of freedom.

Table 1
Performance comparison between FEM and neural network representation of the coefficient for the 1D Poisson equation. #it is the number of iterations until the norm of the gradient of the error/misfit functional is less than 10^{-6}. The norm is measured as the integral of a high-order interpolation onto the finite element space as provided by the `errornorm` function in `FEniCS`, and $\hat{u}$ is the unperturbed exact solution in the case of added noise.

	$\hat{q} = 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	19	1	3.65e−5	1.38e−3	19	1	6.67e−3	9.72e−3
FEM	184	9	1.11e−3	1.30e−1	1748	103	3.25e−2	5.38e+00
	$\hat{q} = x + 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	29	2	2.26e−4	3.43e−3	37	2	7.47e−3	2.55e−2
FEM	195	10	1.44e−3	2.50e−1	1228	75	3.07e−2	5.70e+00
	$\hat{q} = x^2 + 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	61	3	1.20e−3	1.10e−2	56	3	7.45e−3	2.28e−2
FEM	218	13	1.05e−3	2.05e−1	1856	105	3.14e−2	8.84e+00
	$\hat{q} = 0.5\sin(2\pi x) + 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	324	14	2.56e−3	1.78e−2	240	11	1.07e−2	6.11e−2
FEM	218	11	9.55e−4	1.42e−1	2116	117	3.14e−2	1.79e+01

Table 2
Summary for the neural network representation of the coefficient for the 2D Poisson equation. #it is the number of iterations. The norm is measured as the integral of a high-order interpolation onto the finite element space as provided by the `errornorm` function in FEniCS, and $\hat{u}$ is the unperturbed exact solution in the case of added noise.

	$\hat{q} = 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	41	13	1.11e−5	2.51e−4	40	12	2.35e−4	9.28e−4
	$\hat{q} = x + y + 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	90	24	1.46e−4	2.99e−3	332	93	1.40e−3	1.74e−2
	$\hat{q} = x^2 + y^2 + 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	402	111	2.07e−4	4.22e−3	517	138	1.76e−3	2.04e−2
	$\hat{q} = 0.5\sin(2\pi x)\sin(2\pi y) + 1$							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	2112	560	6.32e−4	1.86e−2	1553	406	3.09e−3	4.48e−2

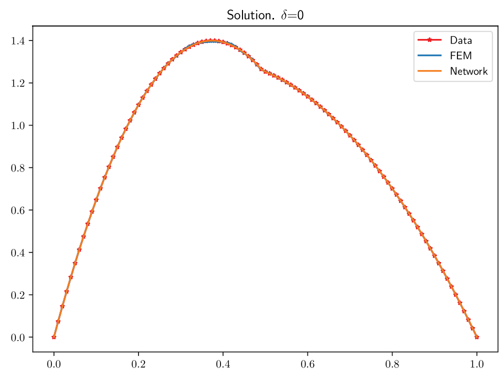
Table 3
Number of iterations for different levels of refinement.

n	0	1	2	3	4	5	6	7
#dofs	441	725	1017	1479	2482	5040	11922	32000
$h_{\max}/h_{\min}$	1	2	4	8	16	32	64	128
#it	32	35	40	40	40	40	40	40
$\|\|q - \hat{q}\|\|$	6.14e−3	4.91e−3	4.85e−3	4.84e−3	4.84e−3	4.84e−3	4.84e−3	4.84e−3

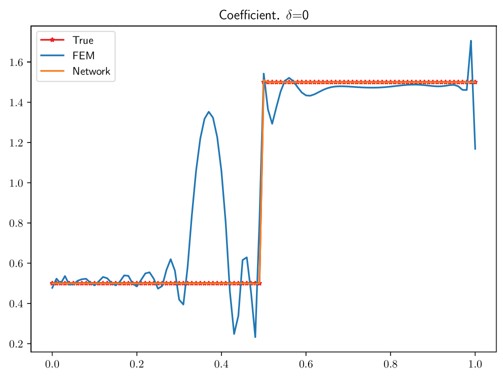
Table 4
Comparison between FEM and a neural network with a discontinuous coefficient.

	Discontinuous coefficient							
	$\delta = 0$				$\delta = 0.05$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	310	37	1.19e−05	9.33e−04	346	118	1.25e−02	1.64e−01
FEM	332	76	1.58e−03	2.07e−01	788	408	2.90e−02	5.34e+00

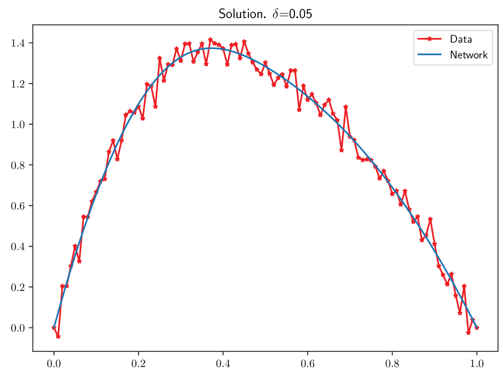
longer sufficient to approximate the coefficient. To approximate more complicated coefficients, the number of network parameters needs to be increased. However, without explicit regularization, we have been unable to obtain good results in these cases. Multiple discontinuities, highly oscillatory and multiscale coefficients require more advanced networks and/or explicit regularization and will be the topic of future research.



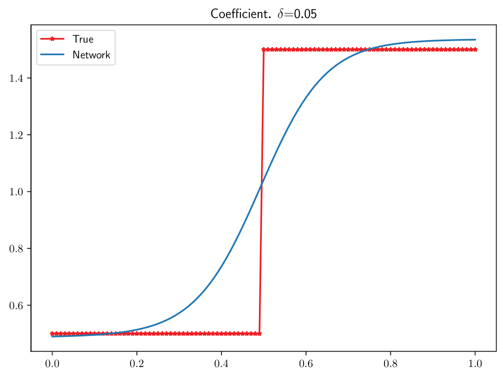
(a) Solutions with the optimized coefficients and no noise.



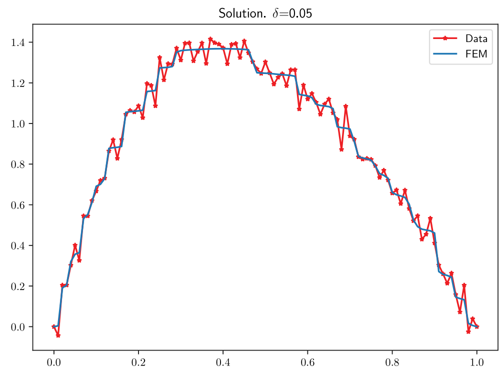
(b) The optimized coefficients without noise.



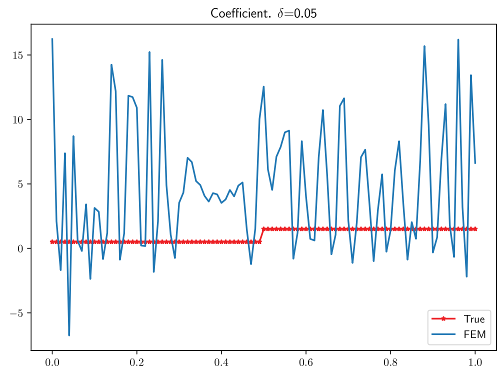
(c) Network solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient and 5% noise level.

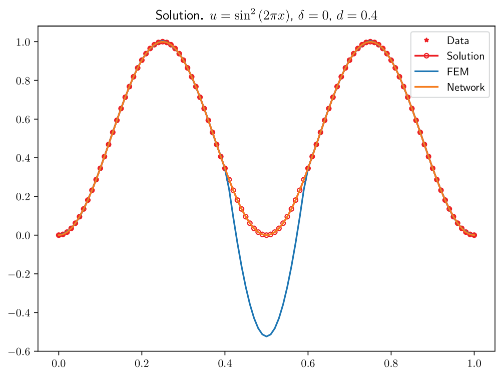


(e) FEM solution with the optimized coefficient and 5% noise level.

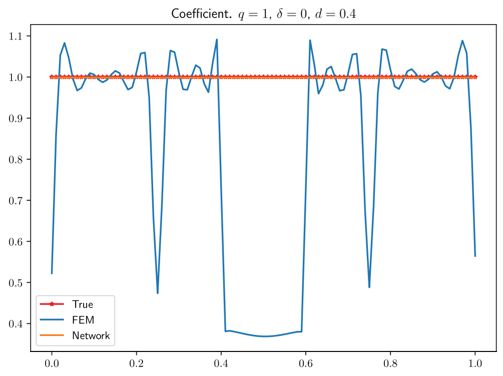


(f) The optimized FEM coefficient and 5% noise level.

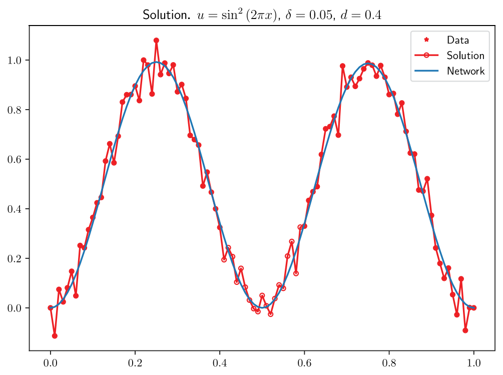
Fig. 12. Comparison between FEM and a neural network with a discontinuous coefficient.



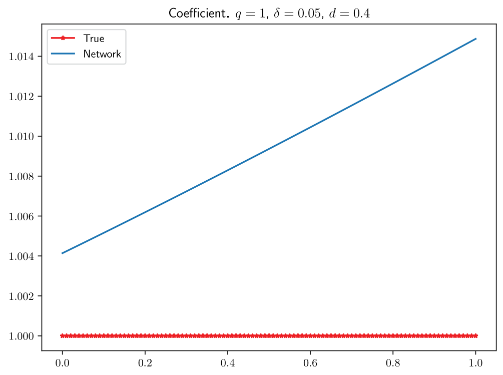
(a) Solutions with the optimized coefficients and no noise.



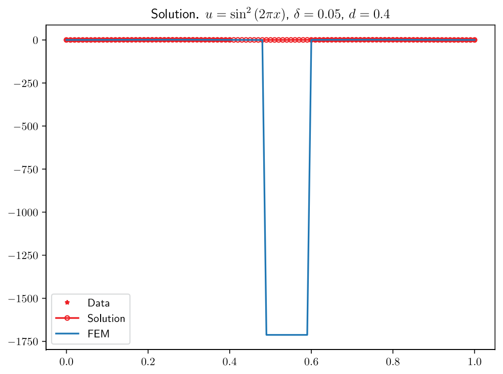
(b) The optimized coefficients without noise.



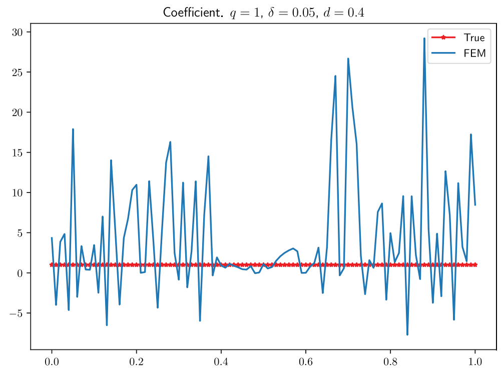
(c) Network solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient and 5% noise level.

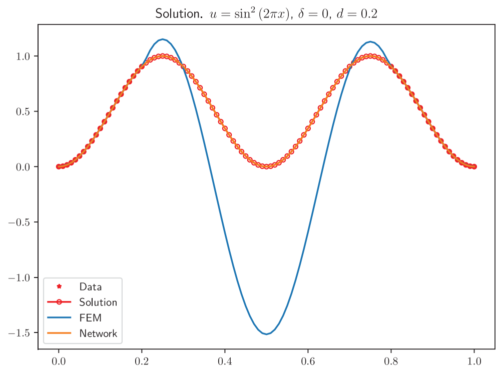


(e) FEM solution with the optimized coefficient and 5% noise level.

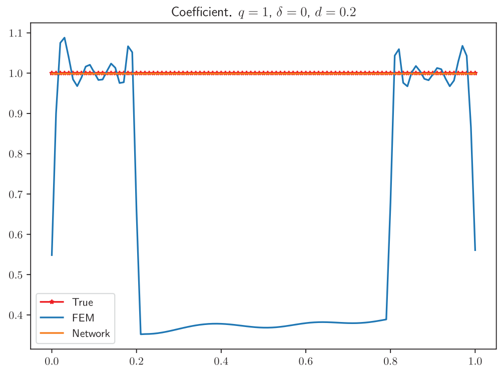


(f) The optimized FEM coefficient and 5% noise level.

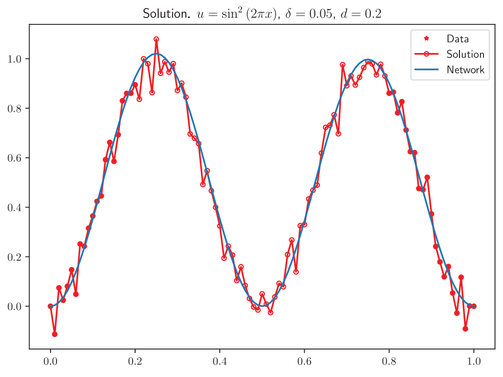
Fig. 13. Comparison between FEM and a neural network with a constant coefficient and $d = 0.4$.



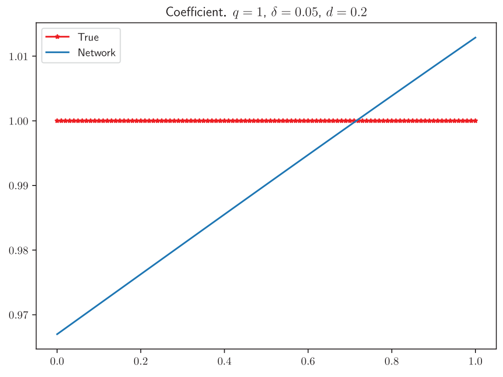
(a) Solutions with the optimized coefficients and no noise.



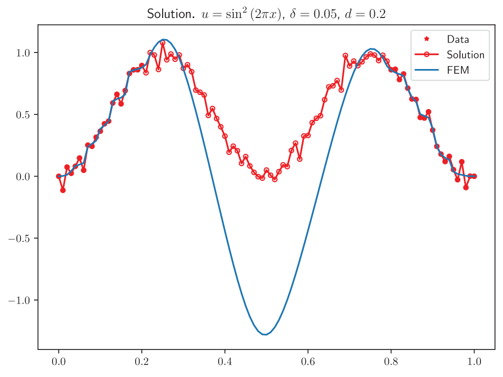
(b) The optimized coefficients without noise.



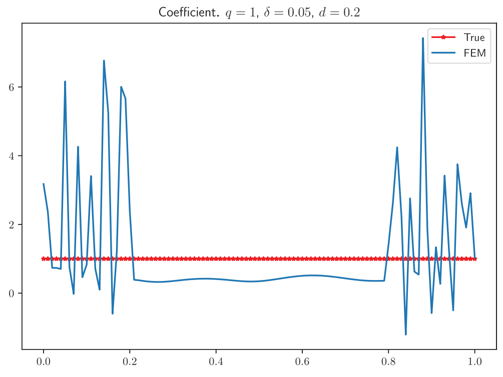
(c) Network solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient and 5% noise level.

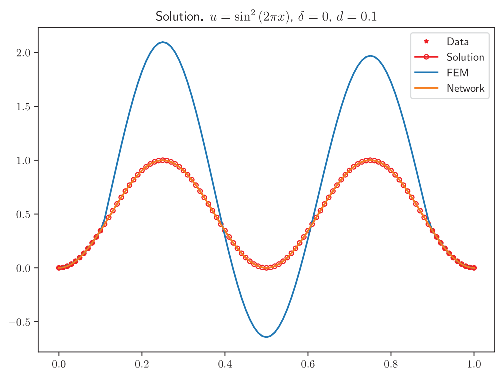


(e) FEM solution with the optimized coefficient and 5% noise level.

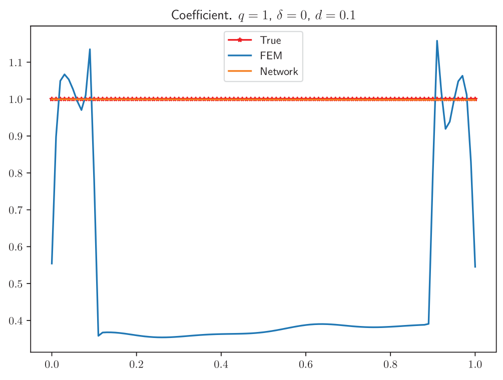


(f) The optimized FEM coefficient and 5% noise level.

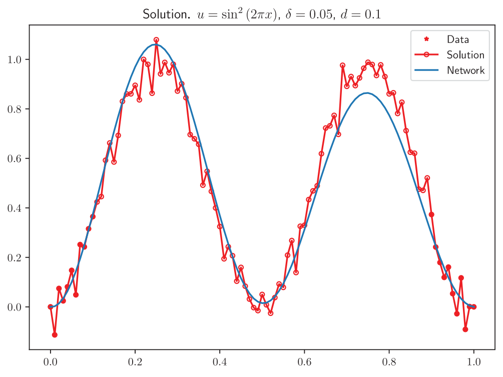
Fig. 14. Comparison between FEM and a neural network with a constant coefficient and $d = 0.2$.



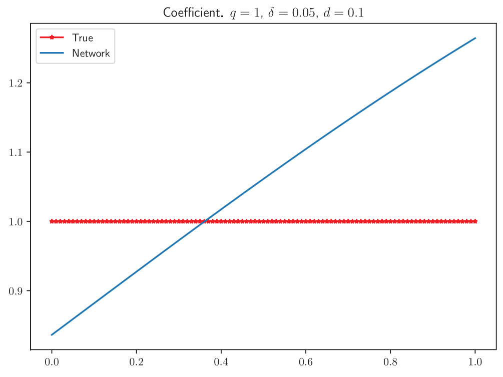
(a) Solutions with the optimized coefficients and no noise.



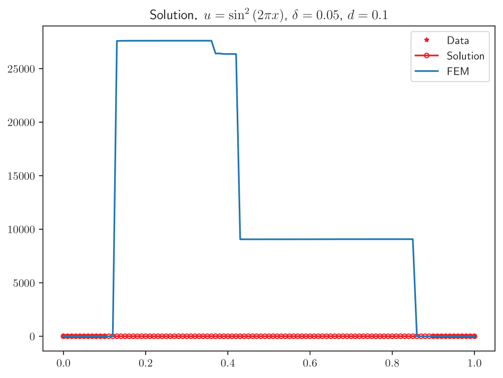
(b) The optimized coefficients without noise.



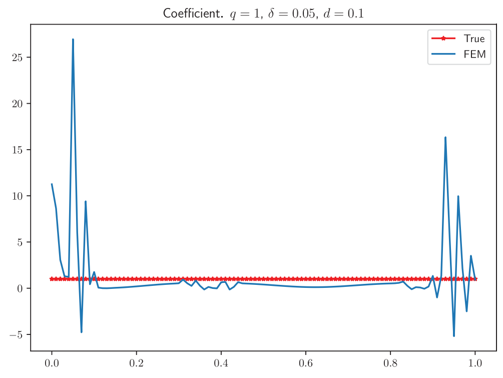
(c) Network solution with the optimized coefficient and 5% noise level.



(d) The optimized network coefficient and 5% noise level.



(e) FEM solution with the optimized coefficient and 5% noise level.



(f) The optimized FEM coefficient and 5% noise level.

Fig. 15. Comparison between FEM and a neural network with a constant coefficient and $d = 0.1$.

5. Summary and conclusions

We have discussed and shown the potential of using (artificial) neural networks to solve classical inverse problems. We used the finite element method to discretize and solve the forward problem, and automatic differentiation to derive and solve the adjoint problem related to the error/misfit functional. The total gradient of the error/misfit functional could then be computed exactly by combining the automatic adjoint for the finite element part, and backpropagation/automatic differentiation for the neural network part. By the chain rule, the discretization of the forward and backward problems are factored from the neural network prior in the computation of the gradient of the error/misfit functional. With the gradient of the error/misfit functional given, any gradient based optimization method can be used.

The idea proposed is that neural networks can potentially be used as a prior for the coefficient to be estimated from noisy data. As neural networks are global, smooth, universal approximators, they do not necessarily require explicit regularization of the error/misfit functional to recover both smooth solutions and coefficients. The convergence of classical optimization methods, with the neural network approach, is to some extent independent of both the mesh and geometry in contrast to standard finite elements. As the number of optimization parameters is independent of the mesh, large scale inverse problems can be computed with efficient optimization algorithms, such as BFGS, as long as the number of network parameters is rather small.

The implicit regularization by low capacity network priors allows a smooth reconstruction of discontinuous coefficients as long as there are not too many discontinuities. Since neural networks are globally defined, they allow the reconstruction of the coefficient in the whole domain even when the measurement data is available only close to the boundaries.

For multiple discontinuities and multiscale coefficients, and to in general explore the methodology discussed in this paper further, more research is needed on explicit regularization and the choice of neural network design. In general it is interesting to study strongly ill conditioned problems having error/misfit functionals which are multimodal using neural networks and to understand what conditions the inverse problem at hand has to satisfy for the applicability of this methodology discussed/proposed. An important direction for future research is also to compare the methodology discussed/proposed to other multi-phase methods for global optimization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors were partially supported by a grant from the Göran Gustafsson Foundation for Research in Natural Sciences and Medicine, Sweden. The authors also thank two anonymous referees for careful readings and for valuable suggestions and advice.

Appendix. Optimal regularization of the 1D Poisson equation

We return to the problem in Section 3.1 to discuss optimal regularization for a fair comparison with FEM. We are to minimize the generalized Tikhonov functional

$$\frac{1}{2}\int_{\Omega}|u-\hat{u}|^2dx+\frac{\alpha^{\delta}}{2}\int_{\Omega}|q-q_*|^2dx. \tag{A.1}$$

In this case we have a complete description of the noise which is uniformly distributed with a value of δr added to each measurement, where $r \in \mathcal{N}(0,1)$ and $\delta = 0.05$. Moreover, since the problem is artificially constructed we know the exact value of q_* in (A.1). This allows us to compute the optimal value of α^{δ} using the Morozov discrepancy principle for the various coefficients to be estimated. The Morozov discrepancy principle amounts to finding the unique zero of the monotone function

$$f(\alpha^{\delta}) = \int |u-\hat{u}|^2dx - \|\delta\|, \tag{A.2}$$

*where $\|\delta\|$ denotes the noise level. The computation of u requires that the inverse coefficient problem is solved by minimizing (A.1). Each evaluation of (A.2) thus require the solution of the forward problem and the backward problem until convergence in the coefficient q for a given α. If (A.2) is computed numerically by an iterative method, this procedure is repeated in each iteration making the computation of an optimal regularization extremely expensive. For the 1D Poisson problem, however, we can perform the computations to obtain the optimal regularization parameters for $\delta = 0.05$ as shown in Table 5.

The results in Section 3.1 are repeated here where we have used optimal regularization. It can clearly be seen that FEM outperforms neural networks this time. However, since optimal regularization is rare we propose that neural networks can be used to compute the estimated coefficient q_* in (A.1) which can then be used in a generalized Tikhonov regularization (see Figs. 16–19 and Table 6).

Table 5
Optimal regularization parameters by the Morozov discrepancy principle for 5% noise level and known coefficients q_*.

q_*	α^{δ} (FEM)	α^{δ} (Network)
1	10.3	0.29
$1+x$	18.6	0.29
$1+x^2$	17.3	0.29
$1+0.5\sin(2\pi x)$	12.0	0.29

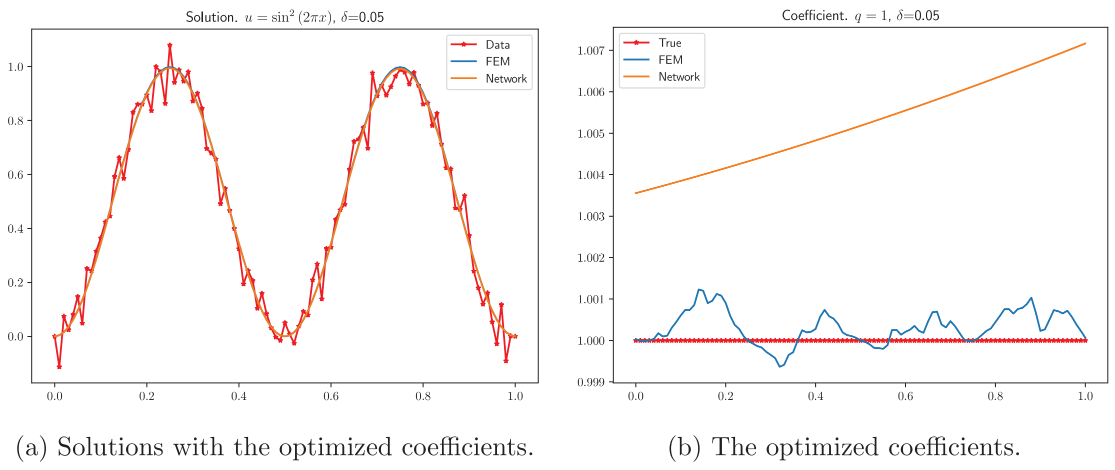


Fig. 16. Comparison between optimal FEM and a neural network with constant coefficient $\hat{q} = 1$ and 5% noise level.

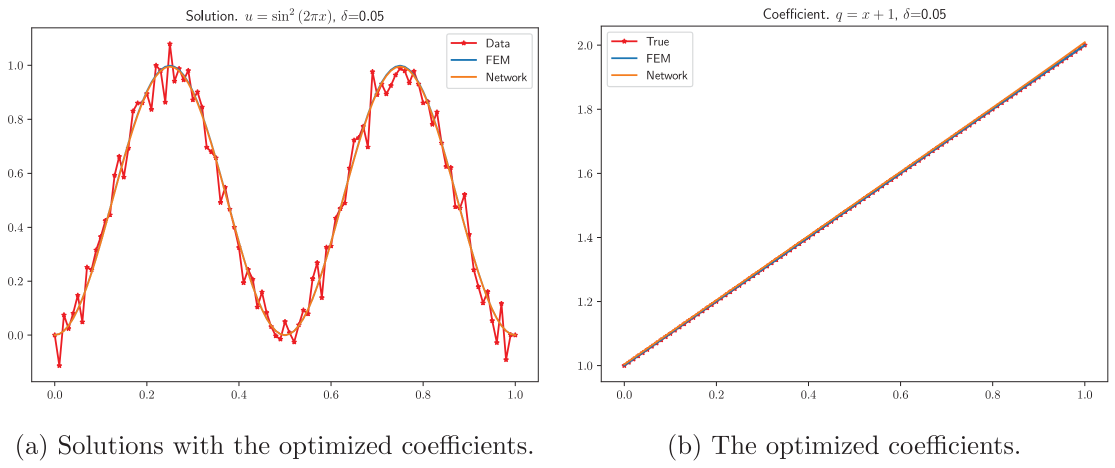


Fig. 17. Comparison between optimal FEM and a neural network with linear coefficient $\hat{q} = 1 + x$ and 5% noise level.

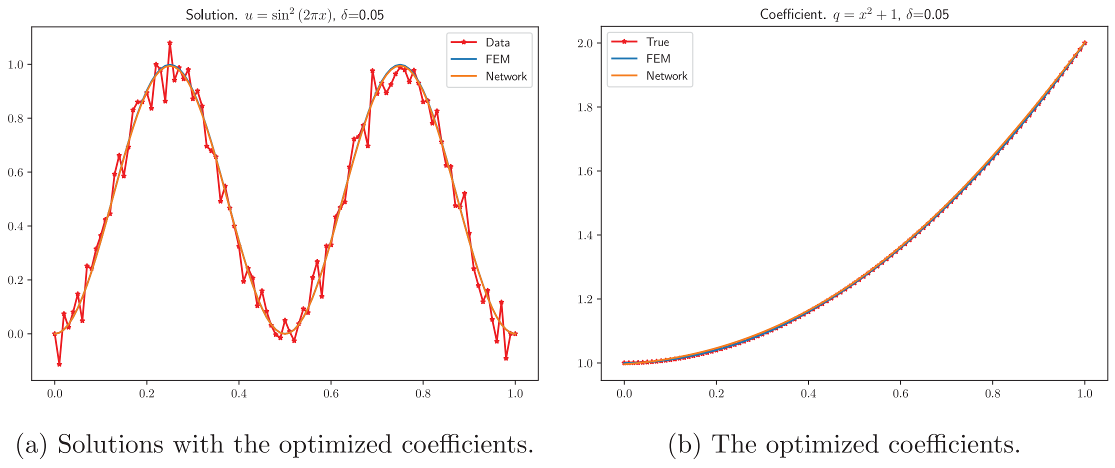


Fig. 18. Comparison between optimal FEM and a neural network with quadratic coefficient $\hat{q} = 1 + x^2$ and 5% noise level.

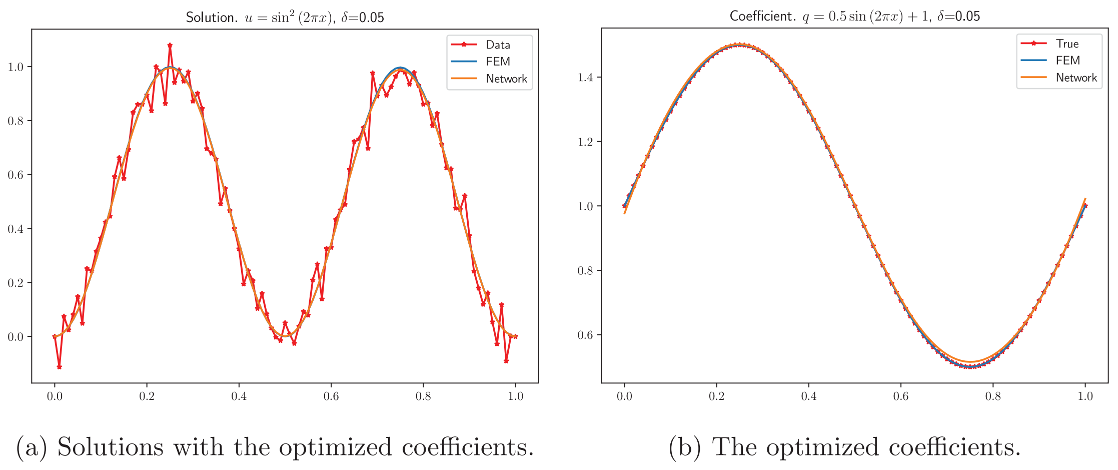


Fig. 19. Comparison between optimal FEM and a neural network with sine coefficient $\hat{q} = 1 + 0.5\sin(2\pi x)$ and 5% noise level.

Table 6
Performance comparison between neural networks and FEM with optimal regularization and 5% noise level.

	$q = 1$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	12	1	4.06e−03	5.34e−03
FEM	40	4	1.26e−03	5.00e−04
	$q = x + 1$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	354	46	2.90e−03	5.03e−03
FEM	36	8	9.30e−04	2.06e−04
	$q = x^2 + 1$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	281	78	3.26e−03	5.43e−03
FEM	37	12	9.99e−04	2.44e−04
	$q = 0.5\sin(2\pi x) + 1$			
	#it	Time (s)	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
Network	473	190	4.48e−03	8.09e−03
FEM	38	19	1.38e−03	4.82e−04

References

[1] Levenberg K. A method for the solution of certain non-linear problems in least squares. Quart Appl Math 1944;2(2):164–8, URL http://www.jstor.org/stable/43633451.
[2] Marquardt D. An algorithm for least-squares estimation of nonlinear parameters. J Soc Ind Appl Math 1963;11(2):431–41. http://dx.doi.org/10.1137/0111030.
[3] Kac M. Can one hear the shape of a drum? Amer Math Monthly 1966;73(4):1–23, URL http://www.jstor.org/stable/2313748.
[4] Tikhonov A. Solution of incorrectly formulated problems and the regularization method. Sov. Math. - Doklady 1963;4(3–4):1035–8.
[5] Tikhonov A, Arsenin V. Solutions of Ill-Posed Problems. Scripta series in mathematics, Winston; 1977, URL https://books.google.se/books?id=ECrvAAAAMAAJ.
[6] Kabanikhin S. Definitions and examples of inverse and ill-posed problems. J. Inverse Ill-Posed Probl. 1966;16(4):317–57. http://dx.doi.org/10.1515/JIIP.2008.019.
[7] Chavent G. Nonlinear Least Squares for Inverse Problems: Theoretical Foundations and Step-By-Step Guide for Applications. Scientific Computation, Springer Netherlands; 2010, http://dx.doi.org/10.1007/978-90-481-2785-6.
[8] Hinze M, Pinnau R, Ulbrich M, Ulbrich S. Optimization with PDE Constraints. Mathematical Modelling: Theory and Applications, Springer Netherlands; 2009, http://dx.doi.org/10.1007/978-1-4020-8839-1.
[9] Beilina L, Klibanov M. Approximate Global Convergence and Adaptivity for Coefficient Inverse Problems. Springer-Verlag New York; 2012, http://dx.doi.org/10.1007/978-1-4419-7805-9.
[10] Kärkkäinen T, Neittaanmäki P, Niemistö A. Numerical methods for nonlinear inverse problems. J Comput Appl Math 1996;74(1):231–44. http://dx.doi.org/10.1016/0377-0427(96)00026-X, URL http://www.sciencedirect.com/science/article/pii/037704279600026X.
[11] Zou J. Numerical methods for elliptic inverse problems. Int J Comput Math 1998;70(2):211–32. http://dx.doi.org/10.1080/00207169808804747.

[12] Hào D, Quyen T. Finite element methods for coefficient identification in an elliptic equation. Appl Anal 2014;93(7):1533–66. http://dx.doi.org/10.1080/00036811.2013.840365.

[13] Petra N, Stadler G. Model variational inverse problems governed by partial differential equations. The Institute for Computational Engineering and Sciences, The University of Texas at Austin; 2011.

[14] Cockayne J, Oates C, Sullivan T, Girolami M. BayesIan probabilistic numerical methods, arxiv e-prints. stat.me 1702.03673. 2017.

[15] Hennig P, Osborne M, Girolami M. Probabilistic numerics and uncertainty in computations. Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 2015;471(2179).

[16] Hennig P. Fast probabilistic optimization from noisy gradients. In: International Conference on Machine Learning (ICML). 2013.

[17] Hennig P, Kiefel M. Quasi-Newton methods – a new direction. J Mach Learn Res 2013;14:834–65.

[18] Raissi M, Perdikaris P, Karniadakis G. Machine learning of linear differential equations using Gaussian processes. J Comput Phys 2017;348:683–93. http://dx.doi.org/10.1016/j.jcp.2017.07.050, URL http://www.sciencedirect.com/science/article/pii/S0021999117305582.

[19] Raissi M, Perdikaris P, Karniadakis G. Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. SIAM J Sci Comput 2018;40(1):A172–98. http://dx.doi.org/10.1137/17M1120762.

[20] Cockayne J, Oates C, Sullivan T, Girolami M. Probabilistic meshless methods for partial differential equations and Bayesian inverse problems, arxiv. 2016.

[21] Bui-Thanh T, Girolami M. Solving large-scale PDE-constrained Bayesian inverse problems with Riemann manifold Hamiltonian Monte Carlo. 2014, ArXiv Pre-PrintarXiv:1407.1517.

[22] Grisvard P. Elliptic Problems in Nonsmooth Domains. John Wiley & Sons, Incorporated; 1986.

[23] Smołka M. Differentiability of the objective in a class of coefficient inverse problems. Comput Math Appl 2017;73:2375–87.

[24] LeCun Y, Bottou L, Orr G, Müller K-R. Efficient BackProp. In: Neural Networks: Tricks of the Trade. Springer; 1998, p. 9–50. http://dx.doi.org/10.1007/3-540-49430-8_2.

[25] Hinton G, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov R. Improving neural networks by preventing co-adaptation of feature detectors. 2012, ArXiv E-Prints arXiv:1207.0580.

[26] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting. J Mach Learn Res 2014;15:1929–58, URL http://jmlr.org/papers/v15/srivastava14a.html.

[27] Alnæs M, Blechta J, Hake J, Johansson A, Kehlet B, Logg A, Richardson C, Ring J, Rognes M, Wells G. The FEniCS project version 1.5. Arch. Numer. Softw. 2015;3(100). http://dx.doi.org/10.11588/ans.2015.100.20553.

[28] Logg A, Mardal K-A, Wells G, et al. Automated Solution of Differential Equations By the Finite Element Method. Springer; 2012, http://dx.doi.org/10.1007/978-3-642-23099-8.

[29] Farrell P, Ham D, Funke S, Rognes M. Automated derivation of the adjoint of high-level transient finite element programs. SIAM J Sci Comput 2013;35(4):C369–93. http://dx.doi.org/10.1137/120873558.

[30] Hornik K, Stinchcombe M, White H. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Netw 1990;3(5):551–60. http://dx.doi.org/10.1016/0893-6080(90)90005-6, URL http://www.sciencedirect.com/science/article/pii/0893608090900056.

[31] Berg J, Nyström K. Neural network augmented inverse problems for PDEs. 2017, arXiv:1712.09685,

[32] Berg J, Nyström K. A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 2018;317:28–41, arXiv:1711.06464.

[33] Berg J, Nyström K. Data-driven discovery of PDEs in complex datasets. J Comput Phys 2019;384:239–52, arXiv:1712.09685.

[34] Xu K, Darve E. The neural network approach to inverse problems in differential equations. 2019, ArXiv E-Prints arXiv:1901.07758.

[35] Raissi M, Perdikaris P, Karniadakis G. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. 2017, ArXiv E-Prints arXiv:1711.10561.

[36] Raissi M, Perdikaris P, Karniadakis G. Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations. 2017, ArXiv E-Prints arXiv:1711.10566.

[37] Raissi M, Karniadakis G. Hidden physics models: Machine learning of nonlinear partial differential equations. J Comput Phys 2018;357:125–41. http://dx.doi.org/10.1016/j.jcp.2017.11.039, URL http://www.sciencedirect.com/science/article/pii/S0021999117309014.

[38] Logg A. Mesh generation in FEniCS. 2016, http://www.logg.org/anders/2016/06/02/mesh-generation-in-fenics. (Accessed 12 December 2017).

[39] Kohn R, Lowe B. A variational method for parameter identification. ESAIM:Math. Modelling Numer. Anal. 1988;22(1):119–58. http://dx.doi.org/10.1051/m2an/1988220101191.

[40] Rumelhart D, Hinton G, Williams R. Learning representations by back-propagating errors. Nature 1986;323(6088):533–6. http://dx.doi.org/10.1038/323533a0.

[41] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. PMLR; 2010, p. 249–56, URL http://proceedings.mlr.press/v9/glorot10a.html.

[42] Hinton G, Osindero S, Teh Y-W. A fast learning algorithm for deep belief nets. Neural Comput 2006;18(7):1527–54. http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[43] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016, http://www.deeplearningbook.org.

[44] Li X. Simultaneous approximations of multivariate functions and their derivatives by neural networks with one hidden layer. Neurocomputing 1996;12(4):327–43. http://dx.doi.org/10.1016/0925-2312(95)00070-4, URL http://www.sciencedirect.com/science/article/pii/0925231295000704.

[45] Liu X. Kolmogorov Superposition Theorem and Its Applications. PhD thesis, Imperial College of London, 2015.

[46] Mhaska H. Neural networks for optimal approximation of smooth and analytic functions. Neural Comput 1996;8(1):164–77.

[47] Maiorov V, Pinkus A. Lower bounds for approximation by MLP neural networks. Neurocomputing 1999;25(1):81–91.

[48] Weinan E, Wojtowytsch S. On the Banach spaces associated with multi-layer ReLU networks: Function representation, approximation theory and gradient descent dynamics. 2020, ArXiv E-Prints, ArXiv:2007.15623.

[49] Weinan E. Machine learning and computational mathematics. 2020, ArXiv E-Prints, ArXiv:2009.14596.

[50] Weinan E, Ma C, Wojtowytsch S, Wu L. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't. 2020, ArXiv E-Prints, ArXiv:2009.10713.

[51] Fletcher R. Practical Methods of Optimization. second ed.. Wiley; 1987.

[52] Jones E, Oliphant T, Peterson P, et al. Scipy: Open source scientific tools for python. 2001, http://www.scipy.org. (Accessed 12 December 2017).

[53] Liu D, Nocedal J. On the limited memory BFGS method for large scale optimization. Math Program 1989;45(1):503–28. http://dx.doi.org/10.1007/BF01589116.

[54] Byrd R, Lu P, Nocedal J, Zhu C. A limited memory algorithm for bound constrained optimization. SIAM J Sci Comput 1995;16(5):1190–208. http://dx.doi.org/10.1137/0916069.

[55] Schwedes T, Ham D, Funke S, Piggott M. Mesh dependence in PDE-constrained optimisation. In: Mesh Dependence in PDE-Constrained Optimisation: An Application in Tidal Turbine Array Layouts. Springer International Publishing; 2017, p. 53–78. http://dx.doi.org/10.1007/978-3-319-59483-5_2.

[56] Schwedes T, Funke S, Ham D. An iteration count estimate for a mesh-dependent steepest descent method based on finite elements and Riesz inner product representation. 2016, ArXiv E-Prints arXiv:1606.08069.