Meta-learning Pseudo-differential Operators with Deep Neural Networks

Jordi Feliu-Fabà∗, Yuwei Fan†, Lexing Ying‡

Abstract
This paper introduces a meta-learning approach for parameterized pseudo-differential operators with deep neural networks. With the help of the nonstandard wavelet form, the pseudo-differential operators can be approximated in a compressed form with a collection of vectors. The nonlinear map from the parameter to this collection of vectors and the wavelet transform are learned together from a small number of matrix-vector multiplications of the pseudo-differential operator. Numerical results for Green’s functions of elliptic partial differential equations and the radiative transfer equations demonstrate the efficiency and accuracy of the proposed approach.

Keywords: Deep neural networks; Convolutional neural networks; Nonstandard wavelet form; Meta-learning; Green’s functions; Radiative transfer equation.

1 Introduction
Many physical models for scientific and engineering applications can be written in a general form
\[ L_\eta u(x) = f(x), \quad x \in \Omega \subset \mathbb{R}^d \quad (1.1) \]
for a domain \( \Omega \) with appropriate boundary conditions, where \( L_\eta \) is often a partial differential or integral operator parameterized by a parameter function \( \eta(x) \). Solving for \( u(x) \) for a given \( f(x) \) amounts to representing the inverse operator (sometimes also known as the Green’s function) \( G_\eta = L_\eta^{-1} \) either explicitly or implicitly via an efficient algorithm. Representing \( G_\eta \), even if implicitly, can be computationally challenging, especially for multidimensional problems. The past few decades have witnessed steady progresses in developing efficient algorithms for this.

Problem statement. This paper is concerned with a more ambitious task: representing the nonlinear map from \( \eta \) to \( G_\eta \)
\[ \mathcal{M} : \eta \to G_\eta = L_\eta^{-1}, \quad (1.2) \]
when the operator \( G_\eta \) is a pseudo-differential operator (PDO) [57]. Although \( L_\eta \) and \( G_\eta \) can be linear operators, this map \( \mathcal{M} \) from \( \eta \) to \( G_\eta \) is highly nonlinear.

Background. In the recent years, deep learning has become the most versatile and effective tool in artificial intelligence and machine learning, witnessed by impressive achievements in computer vision [35, 58, 28], speech and natural language processing [29, 53, 48, 54, 11], drug discovery [41] or game playing [51, 15, 55]. Recent reviews on deep learning and its impacts on other fields can be found in for example [37, 50]. At the center of deep learning, the model of deep neural networks (NNs) provides a flexible framework for approximating high-dimensional functions, while allowing for efficient training and good generalization properties in practice [40, 44].

More recently, several groups have started applying NNs to partial differential equations (PDEs) and integral equations (IEs) arising from physical systems. In one direction, the NN model has been used to
approximate high-dimensional solutions of high-dimensional PDEs \[36, 52, 14, 46, 4, 6, 13, 25, 31, 39\]. In a somewhat orthogonal direction, the NNs have been utilized to approximate the high-dimensional parameter-to-solution of various PDEs and IEs \[30, 26, 18, 17, 19, 32, 25, 2, 38, 20\].

**Main idea.** Following these recent advances in applying NNs to physical models, this paper takes a deep learning approach for representing the map in Eq. (1.2). The most straightforward solution would be to take a supervised learning approach, i.e., trying to learn the map 
\[M : \eta \rightarrow G_\eta\]
from a large set of training data \[
\{(\eta_i, G_\eta_i)\}_i.
\] However, since it is often difficult or even impossible to compute and store \[G_\eta_i\] due to the enormous discretization size, this straightforward supervised learning approach is not practical for Eq. (1.2). Without explicit access to the matrices \[
\{G_\eta_i\},
\] one is faced with two key difficulties.

- How should we represent the output \[G_\eta\] for an arbitrary test input \[\eta\]?
- How should we represent the training data?

To address the first question, \[G_\eta\] should be represented in a compressed form. For pseudo-differential operators, several compressed representations exist, including hierarchical matrices \[22, 23, 24\], discrete symbol calculus \[10\], etc. In this paper, we choose to represent \[G_\eta\] with the nonstandard wavelet form introduced in \[5\]. The main advantage of the nonstandard wavelet form is that the nonzero entries of this compressed representation are simply organized into a small number of vectors. More precisely,
\[G_\eta \approx WS[C_\eta]W^T,\]
where \(W\) is a redundant form of a wavelet transform, \(C_\eta\) stands for the collection of vectors that contain the nonzero entries of the compressed form, and \(S\) is a certain operator that generates a sparse matrix from the vector collection \(C_\eta\). Compared to \[5\], a key difference is that the current approach allows for \(W\) to be fine-tuned for the map \(M\).

To address the second question, this paper takes a meta-learning or learning-to-learn approach \[49, 3, 27, 21, 56\]. More specifically, instead of explicitly representing \(G_\eta\), the training data contains samples of form \[
(\eta_i, \{f_{ij}, u_{ij}\}),
\]
where \(u_{ij} = G_\eta_i f_{ij}\). For a fixed \(\eta_i\), such data can be obtained by solving the equation \(L_\eta_i u_{ij} = f_{ij}\) for each \(f_{ij}\), possibly with a fast algorithm.

Putting these two pieces together, the meta-learning approach of this paper learns two following key objects from the training data of form \[
\{(\eta_i, \{f_{ij}, u_{ij}\})_j\}_i.
\]
- a map from \(\eta\) to the vector collection \(C_\eta\),
- the \(M\)-dependent wavelet transform \(W\).

Once trained, for a given test input \(\eta\) the architecture calculates \(C_\eta\) and returns a linear NN that implements \[WS[C_\eta]W^T \approx G_\eta,\]

**Organization.** The rest of this paper is organized as follows. Section 2 briefly reviews the nonstandard wavelet form, used for representing \(G_\eta\). In Section 3, the NN architecture of the meta-learning approach is discussed in detail. Section 4 applies the proposed NN to the Green’s function of elliptic PDEs, in both the Schrödinger form and the divergence form. The application to the radiative transfer equation is presented in Section 5.

### 2 Nonstandard wavelet form

This section summarizes the nonstandard wavelet form proposed in \[5\]. To make things concrete, compactly supported orthonormal Daubechies wavelets \[8\] are used as the basis functions as an example.
2.1 Wavelet transform

In the one-dimensional multiresolution analysis, one starts by defining a scaling function \( \phi(x) \) that generates, through dyadic translations and dilations, a family of functions

\[
\phi_k^{(\ell)}(x) = 2^{\ell/2} \phi(2^{\ell} x - k), \quad \ell = 0, 1, 2, \ldots, \quad k \in \mathbb{Z}. \tag{2.1}
\]

For each scale \( \ell \), the functions \( \{ \phi_k^{(\ell)} \} \) form a Ritz basis for a space \( V_\ell \), which satisfies a nested relationship \( V_\ell \subset V_{\ell+1} \). This nested property of \( \{ V_\ell \} \) implies the following *dilation relation* of the scaling function

\[
\phi(x) = \sqrt{2} \sum_{i \in \mathbb{Z}} h_i \phi(2x - i). \tag{2.2}
\]

For the Daubechies’ wavelets \([8]\), the scaling function \( \phi(x) \) has a compact support \([0, 2p - 1]\) for a given positive integer \( p \) and therefore the coefficients \( \{ h_i \} \) are only nonzero for \( i = 0, \ldots, 2p - 1 \). The scaling function also satisfies the orthonormal condition

\[
\int_{\mathbb{R}} \phi(x - a) \phi(x - b) \, dx = \delta_{a,b}, \quad \forall a, b \in \mathbb{Z}, \tag{2.3}
\]

which leads to an orthonormal condition for the coefficients \( \{ h_i \} \)

\[
\sum_{i \in \mathbb{Z}} h_i^2 = 1, \quad \sum_{i \in \mathbb{Z}} h_i h_{i+2m} = 0, \quad m \in \mathbb{Z} \setminus \{0\}. \tag{2.4}
\]

Given the scaling function \( \phi(x) \), another important component of the multiresolution analysis is the wavelet function \( \psi(x) \), defined by

\[
\psi(x) = \sqrt{2} \sum_{i \in \mathbb{Z}} g_i \phi(2x - i), \tag{2.5}
\]

where \( g_i = (-1)^{1-i} h_{1-i} \) for \( i \in \mathbb{Z} \). A simple calculation shows that the support of \( \psi(x) \) is \([-p+1,p]\) and \( \{ g_i \} \) is nonzero only for \( i = -2p + 2, \ldots, 1 \), based on the support of the \( \phi \) and the nonzero entries pattern of \( \{ h_i \} \). The Daubechies wavelets are then defined as

\[
\psi_k^{(\ell)}(x) = 2^{\ell/2} \psi(2^{\ell} x - k), \quad \ell = 0, 1, 2, \ldots, \quad k \in \mathbb{Z}. \tag{2.6}
\]

For a function \( v(x) \in L^2(\mathbb{R}) \), its scaling and wavelet coefficients \( d_k^{(\ell)} \) and \( v_k^{(\ell)} \) are defined as the inner product with the scaling functions and the wavelets

\[
v_k^{(\ell)} := \int v(x) \phi_k^{(\ell)}(x) \, dx, \quad d_k^{(\ell)} := \int v(x) \psi_k^{(\ell)}(x) \, dx. \tag{2.7}
\]

Using the recursive relationships of the scaling function \([\text{Eq. (2.2)}]\) and the wavelet function \([\text{Eq. (2.5)}]\), one obtains a recursive relationship of the scaling and wavelet coefficients

\[
v_k^{(\ell)} = \sum_{i \in \mathbb{Z}} h_i v_{2k+1}^{(\ell+1)}, \quad d_k^{(\ell)} = \sum_{i \in \mathbb{Z}} g_i v_{2k+1}^{(\ell+1)}. \tag{2.8}
\]

By defining \( \mathbf{v}^{(\ell)} = \left( v_k^{(\ell)} \right)_{k \in \mathbb{Z}} \) and \( \mathbf{d}^{(\ell)} = \left( d_k^{(\ell)} \right)_{k \in \mathbb{Z}} \), \( \text{Eq. (2.8)} \) can be written in a matrix form

\[
\mathbf{v}^{(\ell)} = \left( W_s^{(\ell)} \right)^T \mathbf{v}^{(\ell+1)}, \quad \mathbf{d}^{(\ell)} = \left( W_w^{(\ell)} \right)^T \mathbf{v}^{(\ell+1)}, \tag{2.9}
\]

where the operators \( W_s^{(\ell)} \) and \( W_w^{(\ell)} : \ell^2(\mathbb{Z}) \rightarrow \ell^2(\mathbb{Z}) \) are banded with a bandwidth \( 2p \) due to the support of \( \{ h_i \} \) and \( \{ g_i \} \). By introducing the orthogonal operator \( W^{(\ell)} = \left( W_s^{(\ell)} \ W_w^{(\ell)} \right) \), \( \text{Eq. (2.9)} \) can be rewritten as

\[
\begin{pmatrix}
\mathbf{d}^{(\ell)} \\
\mathbf{v}^{(\ell)}
\end{pmatrix} = \begin{pmatrix}
W^{(\ell)}
\end{pmatrix}^T \begin{pmatrix}
\mathbf{v}^{(\ell+1)}
\end{pmatrix}, \quad \mathbf{v}^{(\ell+1)} = \begin{pmatrix}
\mathbf{d}^{(\ell)} \\
\mathbf{v}^{(\ell)}
\end{pmatrix}. \tag{2.10}
\]
The procedure for computing the wavelet and scaling coefficients can be illustrated in the following diagram

\[\begin{align*}
\ldots \rightarrow v^{(\ell)} \rightarrow v^{(\ell-1)} \rightarrow v^{(\ell-2)} \rightarrow \ldots \rightarrow v^{(2)} \rightarrow v^{(1)} \rightarrow v^{(0)} \\
\downarrow d^{(\ell)} \downarrow d^{(\ell-1)} \downarrow d^{(\ell-2)} \ldots \downarrow d^{(2)} \downarrow d^{(1)} \downarrow d^{(0)}.
\end{align*}\]

(2.11)

The discussion until now is concerned with the wavelets on ℝ. It is straightforward to extend it the functions defined on a finite domain with periodic boundary condition. If the function \(v(x)\) is periodic on a finite domain, for instance, \([0, 1]\), then the only modification is that all the shifts and scaling in the \(x\) variable are done modulo the integer. When working with periodic functions, the procedure in Eq. (2.11) usually stops at a coarse level \(L_0 = O(\log_2(p))\) before the wavelet and scaling functions start to overlap itself.

\[\begin{align*}
\ldots \rightarrow v^{(\ell+1)} \rightarrow v^{(\ell)} \rightarrow v^{(\ell-1)} \rightarrow \ldots \rightarrow v^{(L_0)} \\
\downarrow d^{(\ell+1)} \downarrow d^{(\ell)} \downarrow d^{(\ell-1)} \ldots \downarrow d^{(L_0)}.
\end{align*}\]

(2.12)

### 2.2 Nonstandard wavelet form for integral operator

Let \(A\) be an integral operator with kernel \(a(x, y)\), applied to periodic functions defined on \([0, 1]\), i.e.,

\[u = Av, \quad \text{equivalently} \quad u(x) = \int a(x, y)v(y)\, dy.\]

(2.13)

Denote by \(A^{(L)} = (A^{(L)}_{k_1, k_2}) \in \mathbb{R}^{2^L \times 2^L}\) the Galerkin projection of \(A\) to the space \(V_L\), for a sufficiently deep level \(L\), i.e.

\[A^{(L)}_{k_1, k_2} = \int \int \varphi_{k_1}(x, y) a(x, y) \varphi_{k_2}(y) \, dx \, dy.
\]

The nonstandard form described in [5] is a remarkably efficient way to compress the matrix \(A^{(L)}\).

The main step for the nonstandard form is to treat \(A^{(L)}\) as an image and use the 2D multiresolution analysis

\[\begin{align*}
D_{1, k_1, k_2}^{(\ell)} := & \int \int \psi^{(\ell)}_{k_1}(x, y) a(x, y) \psi^{(\ell)}_{k_2}(y) \, dx \, dy, & D_{2, k_1, k_2}^{(\ell)} := & \int \int \psi^{(\ell)}_{k_1}(x, y) a(x, y) \varphi^{(\ell)}_{k_2}(y) \, dx \, dy, \\
D_{3, k_1, k_2}^{(\ell)} := & \int \int \varphi^{(\ell)}_{k_1}(x, y) a(x, y) \psi^{(\ell)}_{k_2}(y) \, dx \, dy, & A^{(L)}_{k_1, k_2} := & \int \int \varphi^{(L)}_{k_1}(x, y) a(x, y) \varphi^{(L)}_{k_2}(y) \, dx \, dy,
\end{align*}\]

for \(\ell = L_0, \ldots, L-1\), and \(k_1, k_2 = 0, \ldots, 2^\ell - 1\). For convenience, these coefficients are organized into the matrix form as

\[A^{(\ell)} = (A^{(\ell)}_{k_1, k_2})_{k_1, k_2=0,\ldots, 2^\ell-1}, \quad D^{(\ell)}_j = (D^{(\ell)}_{j, k_1, k_2})_{k_1, k_2=0,\ldots, 2^{\ell-1}, j=1, 2, 3}.
\]

(2.15)

In this setting, a similar recursive relation to Eq. (2.10) can be obtained

\[\begin{pmatrix}
D_1^{(\ell)} & D_2^{(\ell)} \\
D_3^{(\ell)} & A^{(\ell)}
\end{pmatrix} = (W^{(\ell)})^T A^{(\ell+1)} W^{(\ell)}, \quad \ell = L_0, \ldots, L-1.
\]

(2.16)

If \(A\) is a Calderon-Zygmund operator, the entries of the matrices \(D_j^{(\ell)}\) with \(j = 1, 2, 3\) decay rapidly away from the diagonal. For a prescribed relative accuracy \(\epsilon\), each matrix \(D_j^{(\ell)}\) can be approximated by a band matrix by truncating at a band of width \(O(\log(1/\epsilon))\). Since the bandwidth is independent of the specific choices of \(\ell, j, \) or the mesh size \(N = 2^L\), the nonstandard form of \(A\) stores only \(O(N)\) nonzero entries. The readers are referred to [5] for more details. With a slight abuse of notation, the matrices \(D_j^{(\ell)}\) are assumed to be pre-truncated in what follows.

One can assemble all the matrices \(D_j^{(\ell)}\) and \(A^{(L_0)}\) together, by defining the matrix \(S^{(L)}\) in a recursive way as

\[S^{(L_0)} = \begin{pmatrix}
D_1^{(L_0)} & D_2^{(L_0)} \\
D_3^{(L_0)} & A^{(L_0)}
\end{pmatrix}, \quad S^{(\ell+1)} = \begin{pmatrix}
D_1^{(\ell+1)} & D_2^{(\ell+1)} & 0 \\
D_3^{(\ell+1)} & 0 & 0 \\
0 & 0 & S^{(\ell)}
\end{pmatrix}, \quad \ell = L_0, \ldots, L-1.
\]

(2.17)
The matrix $S := S^{(L)}$ is the nonstandard form of the matrix $A = A^{(L)}$ satisfying

$$A = WSW^T.$$  

(2.18)

Here $W$ is the extended wavelet transform matrix, defined in the recursive form as

$$T^{(L_0)} = W^{(L_0)}, \quad T^{(\ell + 1)} = \left( \begin{array}{cc} W^{(\ell + 1)} & W_s^{(\ell + 1)}T^{(L_0)} \end{array} \right), \quad \ell = L_0, \ldots, L - 1, \quad W := T^{(L)}.$$  

(2.19)

Figure 1 illustrates the matrices $W$ and $S$ along with the formulation Eq. (2.18).

![Figure 1: Illustration of $A = WSW^T$ with $L_0 = 0$ and $L = 3$. The sparsity pattern of $S$ is marked in gray.](image)

To clarify the notations, we denote by $W_s^{(\ell)}$ and $W_w^{(\ell)}$ the transform matrices defined in Eq. (2.9) for the scaling and wavelet parts on level $\ell$, respectively. $W^{(\ell)} = (W_w^{(\ell)} \quad W_s^{(\ell)})$ is the wavelet transform matrix at the level $\ell$.

### 2.3 Matrix-vector multiplication in the nonstandard form

With a Galerkin discretization of Eq. (2.13) at level $L$, the matrix-vector multiplication takes the form

$$u^{(L)} = A^{(L)}v^{(L)}.$$  

(2.20)

The nonstandard form allows for accelerating the evaluation of Eq. (2.20) Using the nonstandard form $A^{(L)} = WSW^T$ obtained above, the matrix-vector multiplication

$$u^{(L)} = WSW^Tv^{(L)}$$  

(2.21)

can be split into four steps:

1. $A^{(L)} \rightarrow S$: generate the nonstandard form $S$ from the matrix $A^{(L)}$ or the kernel $a(x,y)$;
2. $v^{(L)} \rightarrow \hat{v} := W^Tv^{(L)}$: apply (forward) wavelet transform on $v^{(L)}$ to get $\hat{v}$;
3. $\hat{u} := S\hat{v}$: evaluate the matrix-vector multiplication in the nonstandard form;
4. $\hat{u} \rightarrow u^{(L)} := W\hat{u}$: apply inverse wavelet transform on $\hat{u}$ to obtain $u^{(L)}$.

The first step is computed using Eq. (2.16) if the matrix $A^{(L)}$ is given. The second step follows Eq. (2.10)

The third step can be written as

$$\begin{pmatrix} w^{(\ell)} \\ s^{(\ell)} \end{pmatrix} = \begin{pmatrix} D_1^{(\ell)} & D_2^{(\ell)} \\ D_3^{(\ell)} & D_4^{(\ell)} \end{pmatrix} \begin{pmatrix} d^{(\ell)} \\ v^{(\ell)} \end{pmatrix},$$  

(2.22)

where $D_4^{(\ell)} = 0$ for $\ell = L_0 + 1, \ldots, L - 1$ and $D_4^{(L_0)} = A^{(L_0)}$. The fourth step is essentially an inverse wavelet transform, implemented as

$$u^{(L_0)} = 0, \quad u^{(\ell + 1)} = W^{(\ell)}\begin{pmatrix} w^{(\ell)} \\ s^{(\ell)} + u^{(\ell)} \end{pmatrix}, \quad \ell = L_0, \ldots, L - 1.$$  

(2.23)

A step-by-step description of these four steps are summarized in Algorithm 1.
Similarly, the recursive relation Eq. (2.16) can be extended as well three types of wavelet coefficients Eq. (2.10) is redefined to be \( W^{(\ell)} \) \( \in \mathbb{R}^{2^\ell \times 2^\ell} \) as well two-dimensional setting, one defines at each scale \( \ell \) three different types of wavelets of the form

\[
\psi_{1,k}(x,y) = \varphi_{k_1,x}(x)\varphi_{k_2,y}(y), \quad \psi_{2,k}(x,y) = \varphi_{k_1,x}(x)\varphi_{k_2,y}(y), \quad \psi_{3,k}(x,y) = \varphi_{k_1,x}(x)\varphi_{k_2,y}(y),
\]

with \( k = (k_1, k_2) \in \mathbb{Z}^2 \). Using these three types of wavelets, the transform matrix at each scale \( \ell \) used in Eq. (2.10) is redefined to be \( W^{(\ell)} = \begin{pmatrix} W^{(\ell)}_{w,1} & W^{(\ell)}_{w,2} & W^{(\ell)}_{w,3} & W^{(\ell)}_{s} \end{pmatrix} \). The 2D analog of Eq. (2.10) contains three types of wavelet coefficients

\[
\begin{pmatrix} d^{(\ell)}_1 \\ d^{(\ell)}_2 \\ d^{(\ell)}_3 \\ v^{(\ell)} \end{pmatrix} = \begin{pmatrix} W^{(\ell)} \end{pmatrix}^T \begin{pmatrix} u^{(\ell+1)} \\ v^{(\ell+1)} \end{pmatrix} = \begin{pmatrix} d^{(\ell)}_1 \\ d^{(\ell)}_2 \\ d^{(\ell)}_3 \\ v^{(\ell)} \end{pmatrix}.
\]

Similarly, the recursive relation Eq. (2.16) can be extended as well

\[
\begin{pmatrix} D^{(\ell)}_1 & D^{(\ell)}_2 & D^{(\ell)}_3 & D^{(\ell)}_4 \\ D^{(\ell)}_5 & D^{(\ell)}_6 & D^{(\ell)}_7 & D^{(\ell)}_8 \\ D^{(\ell)}_9 & D^{(\ell)}_{10} & D^{(\ell)}_{11} & D^{(\ell)}_{12} \\ D^{(\ell)}_{13} & D^{(\ell)}_{14} & D^{(\ell)}_{15} & A^{(\ell)} \end{pmatrix} = \begin{pmatrix} W^{(\ell)} \end{pmatrix}^T A^{(\ell+1)} W^{(\ell)}, \quad \ell = L_0, \ldots, L - 1,
\]

where \( D^{(\ell)}_j, j = 1, \ldots, 15 \) are all sparse matrices with only \( O(4^\ell) \) non-negligible entries in each. The matrix-vector multiplication follows the steps of Algorithm 1 with these necessary changes.

### 2.4 The multidimensional case

The matrix-vector multiplication in the nonstandard form can be easily extended to the multidimensional case with the help of multidimensional orthogonal wavelets (see [42] for more details). For instance, in the two-dimensional setting, one defines at each scale \( \ell \) three different types of wavelets of the form

\[
\psi_{1,k}(x,y) = \varphi_{k_1,x}(x)\varphi_{k_2,y}(y), \quad \psi_{2,k}(x,y) = \varphi_{k_1,x}(x)\varphi_{k_2,y}(y), \quad \psi_{3,k}(x,y) = \varphi_{k_1,x}(x)\varphi_{k_2,y}(y),
\]

with \( k = (k_1, k_2, k_3) \in \mathbb{Z}^3 \). Using these three types of wavelets, the transform matrix at each scale \( \ell \) used in Eq. (2.10) is redefined to be \( W^{(\ell)} = \begin{pmatrix} W^{(\ell)}_{w,1} & W^{(\ell)}_{w,2} & W^{(\ell)}_{w,3} & W^{(\ell)}_{s} \end{pmatrix} \). The 2D analog of Eq. (2.10) contains three types of wavelet coefficients

\[
\begin{pmatrix} d^{(\ell)}_1 \\ d^{(\ell)}_2 \\ d^{(\ell)}_3 \\ v^{(\ell)} \end{pmatrix} = \begin{pmatrix} W^{(\ell)} \end{pmatrix}^T \begin{pmatrix} u^{(\ell+1)} \\ v^{(\ell+1)} \end{pmatrix} = \begin{pmatrix} d^{(\ell)}_1 \\ d^{(\ell)}_2 \\ d^{(\ell)}_3 \\ v^{(\ell)} \end{pmatrix}.
\]

Similarly, the recursive relation Eq. (2.16) can be extended as well

\[
\begin{pmatrix} D^{(\ell)}_1 & D^{(\ell)}_2 & D^{(\ell)}_3 & D^{(\ell)}_4 \\ D^{(\ell)}_5 & D^{(\ell)}_6 & D^{(\ell)}_7 & D^{(\ell)}_8 \\ D^{(\ell)}_9 & D^{(\ell)}_{10} & D^{(\ell)}_{11} & D^{(\ell)}_{12} \\ D^{(\ell)}_{13} & D^{(\ell)}_{14} & D^{(\ell)}_{15} & A^{(\ell)} \end{pmatrix} = \begin{pmatrix} W^{(\ell)} \end{pmatrix}^T A^{(\ell+1)} W^{(\ell)}, \quad \ell = L_0, \ldots, L - 1,
\]

where \( D^{(\ell)}_j, j = 1, \ldots, 15 \) are all sparse matrices with only \( O(4^\ell) \) non-negligible entries in each. The matrix-vector multiplication follows the steps of Algorithm 1 with these necessary changes.

### 3 Meta-learning approach

The plan is to apply the nonstandard form to the operator \( G_\eta \) in Eq. (1.2)

\[
u = G_\eta f, \quad u(x) = \int g_\eta(x,y)f(y) \, dy.
\]

(3.1)

With a slight abuse of notations, the same letters are used to denote the discretizations. The discrete version of Eq. (3.1) takes the form

\[
u = G_\eta f, \quad u, f, \eta \in \mathbb{R}^N \quad \text{and} \quad G_\eta \in \mathbb{R}^{N \times N},
\]

(3.2)
with \( N = 2^L \). The main goal of this paper is to construct a neural network to learn the map \( \eta \rightarrow G_\eta \).

Following Eq. (2.18) and applying the wavelet transform to the matrix \( G_\eta \) leads to

\[
G_\eta \approx W S_\eta W^T,
\]

(3.3)

where \( W \) is the extended wavelet transform matrix, independent of the parameter \( \eta \). Since each block of matrix \( S_\eta \) is a band matrix, the nonzero entries of each block can be represented by a set of vectors. We define \( C_\eta^{(l)} \), of size \( 2^l \times N \), to be the collection of these vectors of \( S_\eta \) at level \( l \), with \( N \) dependent on the bandwidth and \( l \). By denoting \( C_\eta := \{ C_\eta^{(l)} \}_{l=L_0,\ldots,L-1} \), the map \( M : \eta \rightarrow G_\eta \) can be recovered once the map \( \eta \rightarrow C_\eta \) and the wavelet transform matrix \( W \) are determined.

As mentioned in Section 1, it is computationally expensive or even impractical to take a supervised learning approach. Instead, we adopt a learning-to-learn approach. More precisely, given a set of training samples of the form

\[
(\eta_i, \{ f_{ij}, u_{ij} \}),
\]

(3.4)

where \( u_{ij} = G_\eta f_{ij} \) can be obtained by solving \( L_\eta u_{ij} = f_{ij} \) with right hand side \( f_{ij} \), the meta-learning approach learns the map from \( \eta \rightarrow C_\eta \) and the wavelet transform \( W \) together.

### 3.1 Neural network architecture

Using the factorization of \( G_\eta \) in Eq. (3.3) one can factorize \( u_{ij} = G_\eta f_{ij} \) as

\[
u_{ij} \approx W S_\eta W^T f_{ij}.
\]

(3.5)

Similar to the matrix-vector multiplication in Section 2.3 of the nonstandard form, we propose a neural network for meta-learning Eq. (3.5) with four modules:

1. \( \eta \rightarrow S_\eta \): a module learns the map \( \eta \rightarrow C_\eta \) and then generates the banded sparse matrix \( S_\eta \) from \( C_\eta \) (denoted as \( S_\eta = \mathcal{S}[C_\eta] \));

2. \( v^{(l)} \rightarrow \hat{v} := W^T v^{(l)} \): a module applies the forward wavelet transform to \( v^{(l)} \) to generate \( \hat{v} \);

3. \( \hat{u} := S_\eta \hat{v} \): a module evaluates the matrix-vector multiplication in the nonstandard form;

4. \( \hat{u} \rightarrow u^{(l)} := W \hat{u} \): a module applies the inverse wavelet transform on \( \hat{u} \) to generate \( u^{(l)} \).

Instead of computing \( C_\eta \) from the full operator \( G_\eta \) as described in Section 2.3, the first module forms \( C_\eta \) directly from the parameter \( \eta \) using a deep NN. This module can be split into two steps: (1) carrying out the map \( \eta \rightarrow C_\eta^{(l)} \) for each scale \( l \); (2) constructing the nonstandard form \( S_\eta \) from \( C_\eta := \{ C_\eta^{(l)} \}_{l=L_0,\ldots,L-1} \). The NN architecture for the map \( \eta \rightarrow C_\eta^{(l)} \) is often problem-dependent. For many applications, including the applications to be considered in Sections 4 and 5, the problem is often translation-invariant, i.e., for any translation operator \( T \), if \( u = G_\eta f \), then \( (Tu) = G(T\eta)(Tf) \). For such problems, a convolutional NN is often used for its efficiency and robustness.

The second and fourth modules perform the forward and inverse wavelet transforms (as in Section 2.3), respectively, for a specific wavelet basis. The selection of an effective wavelet basis is often problem-dependent. The capability of learning a problem-dependent wavelet transform from data is essential for the accuracy of the NN architecture.

Combining these four modules results in the architecture summarized in Algorithm 2. An illustration is given in Fig. 2. Below we describe details of the layers and parameters used in this architecture.

**Implementation details.** The input, output, and intermediate data of the NN architecture are all represented with 2-tensors. For a tensor of size \( N \times \alpha \), we refer to \( N \) as the spatial dimensions and \( \alpha \) as the channel dimension. The main tool is the convolutional layer. Given an input tensor \( \xi \) of size \( N \times \alpha \), the convolutional layers outputs a tensor \( \zeta \) of size \( N' \times \alpha' \) obtained via

\[
\zeta_{i,c'} = \phi \left( \sum_{j=i}^{i+w-1} \sum_{c=0}^{\alpha-1} W_{j,c'c} \xi_{j,c} + b_{c'} \right), \quad i = 0, \ldots, N' - 1, \quad c' = 0, \ldots, \alpha' - 1,
\]

(3.6)
Algorithm 2 Neural network architecture for meta-learning $u = G_n f$.

Require: $f(L) = f$, $η(L) = η$, $α_1, α_2 \in \mathbb{N}$;
Ensure: $u$;
▷ Module 1: Learn the map $η \rightarrow S_η$
1: for $ℓ$ from $L - 1$ to $L_0$ by $-1$ do
2: $C^η_ℓ = \text{ConvNet}[ℓ, α, n_{\text{cnn}}](η)$
3: end for
4: Generate $D^j_ℓ$, $j = 1, 2, 3$, $ℓ = L_0, \ldots, L - 1$ and $A^{(L_0)}$ from $C^η_ℓ$, $ℓ = L_0, \ldots, L - 1$
▷ Module 2: forward wavelet transform on $v(L)$
5: $u(L) = f(L)$
6: for $ℓ$ from $L - 1$ to $L_0$ by $-1$ do
7: $(d^{(ℓ)}, v^{(ℓ)}) = \text{FWT}[α_2](v^{(ℓ+1)})$
8: end for
▷ Module 3: matrix-vector multiplication in the nonstandard form
9: $D^j_ℓ = 0$ for $ℓ = L_0 + 1, \ldots, L - 1$ and $D^4_ℓ = A^{(L_0)}$
10: for $ℓ$ from $L_0$ to $L - 1$ do
11: $\begin{bmatrix} w^{(ℓ)} \\ s^{(ℓ)} \end{bmatrix} = \begin{bmatrix} D^{(ℓ)} \\ D^{4}_ℓ \end{bmatrix} \begin{bmatrix} d^{(ℓ)} \\ v^{(ℓ)} \end{bmatrix}$
12: end for
▷ Module 4: Inverse wavelet transform
13: $u^{(L_0)} = 0$
14: for $ℓ$ from $L_0$ to $L - 1$ do
15: $u^{(ℓ+1)} = \text{IWT}[α_2](\begin{bmatrix} w^{(ℓ)} \\ s^{(ℓ)} + u^{(ℓ)} \end{bmatrix})$
16: end for
17: Average over the channel direction of $u^{(L)}$ to give $u$;
18: return $u$

where $w$ is the window size, $s$ is the stride and $ϕ$ is the activation function, usually chosen to be a linear function, a rectified-linear unit (ReLU) function, or a sigmoid function. We denote this convolutional layer as

$$\zeta = \text{Conv1d}[α', w, s, ϕ](ξ).$$

The basic building blocks and layers used in Algorithm 2 are listed below.

- $η \rightarrow C^η_ℓ : C^η_ℓ = \text{ConvNet}[ℓ, α, n_{\text{cnn}}](η)$. As discussed above, it is often a convolutional NN if the system Eq. (3.1) is translation invariant. Since the spatial size of $η$ is greater than that of $C^η_ℓ$, ConvNet[ℓ, α, n_{\text{cnn}}] consists of $n_{\text{cnn}}$ convolutional layers and several downsampling or pooling layers.
- Forward wavelet transform at level $ℓ$: $(d^{(ℓ)}, v^{(ℓ)}) = \text{FWT}[α](v^{(ℓ+1)})$. This is the NN representation of the first equation in Eq. (2.10). It is implemented as $f^{(ℓ)} = \text{Conv1d}[2α, 2p, 2, \text{id}](v^{(ℓ)})$, where the first $α$ and the last $α$ channels of $f^{(ℓ)}$ are assigned to $d^{(ℓ)}$ and $v^{(ℓ)}$, respectively.
- Inverse wavelet transform at level $ℓ$: $u^{(ℓ+1)} = \text{IWT}[α](\begin{bmatrix} w^{(ℓ)} \\ s^{(ℓ)} \end{bmatrix})$. This is the NN representation of the second equation in Eq. (2.10). The expression $[w^{(ℓ)}, s^{(ℓ)}]$ stands for concatenating the 2-tensors $w^{(ℓ)}$ and $s^{(ℓ)}$ of size $2^{ℓ} \times α$ to a 2-tensor of size $2^{ℓ} \times 2α$ along the channel dimension. This layer first applies the inverse transform, implemented by Conv1d[2α, p, 1, id], and then reshapes the output of size $2^{ℓ} \times 2α$ to a 2-tensor of size $2^{ℓ+1} \times α$ by a column-first ordering.

The generation of $D^j_ℓ$ and $A^{(L_0)}$ from $C^η_ℓ$ in Line 4 and the matrix-vector multiplication in Line 11 of Algorithm 2 require some discussion. Figure 3 illustrates two approaches for evaluating the matrix-vector multiplication of a band matrix whose nonzero entries are stored in a set of vectors. The left figure corresponds to the case in Algorithm 2, while the right one is used in the actual implementation. To avoid the copying and shifting of $d^{(ℓ)}$ and $v^{(ℓ)}$, it is convenient to set $α_2 = α_1 = α$. Though there are slightly more NN parameters in this case, this implementation change allows for a more flexible NN that can learn faster.
3.2 The multidimensional case

Let us focus on the 2D case. The input, the output and the intermediate data are all 3-tensors of size \( N_1 \times N_2 \times \alpha \), where \( N = (N_1, N_2) \) is the spatial dimension and \( \alpha \) is the channel dimension. The convolutional layer takes the form

\[
\zeta_{i,c'} = \phi \left( \sum_{j_1=i_1,s}^{i_1+w-1} \sum_{j_2=i_2,s}^{i_2+w-1} \sum_{c=0}^{\alpha-1} W_{j_1',c',c} \xi_{j,c} + b_{c'} \right), \quad i_1 = 0, \ldots, N_1' - 1, i_2 = 0, \ldots, N_2' - 1, c' = 0, \ldots, \alpha - 1,
\]

(3.8)
where the input $\xi$ is of size $N_1 \times N_2 \times \alpha$ and the output $\zeta$ is of size $N_1' \times N_2' \times \alpha'$. Here, the same stride $s$ and window size $w$ are used in both dimensions. We denote this convolutional layer as

$$\zeta = \text{Conv2d}[\alpha, w, s, \phi](\xi).$$

Algorithm 2 can be easily extended to the 2D case, following the same way that Algorithm 1 was extended in Section 2.4. Since there are three different types of wavelets in 2D, the layers in Algorithm 2 are redefined as follows:

- $\eta \rightarrow C_\eta^{(\ell)}$ module: $C_\eta^{(\ell)} = \text{ConvNet}[\ell, \alpha, n_{\text{cnn}}](\eta)$. This module is often a two-dimensional convolutional NN with several downsampling or pooling layers.

- Wavelet transform at level $\ell$: $(d_1^{(\ell)}, d_2^{(\ell)}, d_3^{(\ell)}, v^{(\ell)}) = \text{FWT}(\alpha)(v^{(\ell+1)})$. This is implemented using $f^{(\ell)} = \text{Conv2d}[4\alpha, 2p, 2, \text{id}](v^{(\ell)})$. The first, second, third and last $\alpha$ channels of $f^{(\ell)}$ are assigned to $d_1^{(\ell)}$, $d_2^{(\ell)}$, $d_3^{(\ell)}$ and $v^{(\ell)}$, respectively.

- Inverse wavelet transform at level $\ell$: $u^{(\ell+1)} = \text{IWT}[\alpha](\omega^{(\ell)}).$ This is implemented by first computing $\text{Conv2d}[4\alpha, p, 1, \text{id}](\omega^{(\ell)})$, and then reshaping the output of size $2^\ell \times 2^\ell \times 2 \times 2 \times \alpha$ to a 3-tensor of size $2^{\ell+1} \times 2^{\ell+1} \times \alpha$. The reshape operation is performed as follows: (1) reshape the output to a 5-tensor of size $2^\ell \times 2^\ell \times 2 \times 2 \times \alpha$ by splitting the last dimension; (2) permute the second and third dimensions to obtain a 5-tensor of size $2^\ell \times 2^\ell \times 2 \times 2 \times \alpha$; (3) group the first and second dimensions, and the third and fourth dimensions, respectively, to obtain the resulting 3-tensor of size $2^{\ell+1} \times 2^{\ell+1} \times \alpha$.

4 Elliptic partial differential equations

This section applies the meta-learning approach described in Section 3 to the Green’s functions of elliptic PDEs, both in the Schrödinger form and in the divergence form.

4.1 Schrödinger form

Consider the equation

$$-\Delta u(x) + \eta(x)u(x) = f(x), \quad x \in \Omega = [0, 1]^d,$$

with a periodic boundary condition, where $\eta(x) > 0$ is the potential and $f(x)$ is the source term. Following the notations of Section 1

$$L_\eta = -\Delta + \eta(x), \quad G_\eta = L_\eta^{-1}, \quad u = G_\eta f.$$

Since the problem Eq. (4.1) is translation-invariant due to the periodic boundary condition, the map $\eta \rightarrow C_\eta^{(\ell)}$ can be represented with a convolutional NN. In what follows, we first derive the explicit dependence of $C_\eta^{(\ell)}$ on $\eta$ using a linear perturbative analysis and then report some numerical studies.

Mathematical analysis. When $\eta$ is close to a fixed homogeneous background $\eta_0 > 0$, it is convenient to write

$$L_\eta = L_0 - E_\eta, \quad L_0 = -\Delta + \eta_0, \quad E_\eta = \text{diag}(-\eta + \eta_0),$$

Let $G_0 = L_0^{-1}$ be the Green’s function of $L_0$ with the periodic boundary condition. Using the Neumann series for the resolvent $(I - G_0 E_\eta)^{-1}$ with $|\eta(x) - \eta_0|$ sufficiently small, one can write the Green’s function $G_\eta$ as a perturbative expansion

$$G_\eta = (L_0 - E_\eta)^{-1} = G_0 + G_0 E_\eta G_0 + G_0 E_\eta G_0 E_\eta G_0 + \ldots.$$

For sufficiently small $|\eta(x) - \eta_0|$, the operator $G_\eta$ can be approximated by its linear part as

$$G_\eta \approx G_0 + G_0 E_\eta G_0.$$

10
Let \( g_0(x) \) and \( g_\eta(x) \) be the kernel of \( G_0 \) and \( G_\eta \), respectively. Since \( G_0 \) is the Green’s function of \(-\Delta + \eta_0\) with the periodic boundary condition, the kernel \( g_0 \) is translation-invariant, i.e., \( g_0(x, y) = g_0(x - y) \). The wavelet-wavelet coefficients of \( G_\eta \) at level \( \ell \) take the form

\[
D_{1,k_1,k_2}^{(\ell)} = \iint \psi_{k_1}^{(\ell)}(x) g_\eta(x, y) \psi_{k_2}^{(\ell)}(y) \, dx \, dy \\
\approx \iint \psi_{k_1}^{(\ell)}(x) g_0(x - y) \psi_{k_2}^{(\ell)}(y) \, dx \, dy + \iint \psi_{k_1}^{(\ell)}(x) g_\eta(x - z) (\eta(z) - \eta_0) g_0(z - y) \psi_{k_2}^{(\ell)}(y) \, dx \, dy \, dz \\
= \iint \psi_{k_1}^{(\ell)}(x) g_0(x - y) \psi_{k_2}^{(\ell)}(y) \, dx \, dy + \int \tilde{\psi}_{k_1}^{(\ell)}(z) \tilde{\psi}_{k_2}^{(\ell)}(z) (\eta(z) - \eta_0) \, dz,
\]

where \( \tilde{\psi}_{k}^{(\ell)}(z) := (g_0 * \psi_{k}^{(\ell)})(z) \). For a fixed diagonal of \( D_{1,k_1,k_2}^{(\ell)} \) with \( k_2 = k_1 + c \) for a constant \( c \), Eq. (4.6) states that the map from \( \eta \) to \( D_{1,k_1,k_1+c}^{(\ell)} \) for all possible \( k_1 \) is simply a convolution with an addition of a term independent of \( \eta \), which can be simply represented by the \texttt{Conv1D} layer in Eq. (3.6). It is straightforward to extend the conclusion to \( D_{j,k_1,k_2}^{(\ell)} \), \( j = 2, 3 \) and \( A_{k_1,k_2}^{(L_0)} \).

When \( |\eta(x) - \eta_0| \) is not small, one can account for the nonlinearities neglected in the perturbative analysis by using multiple convolutional layers and making use of nonlinear activation functions. In other words, it is natural to approximate the map \( \eta \rightarrow C_\eta^{(\ell)} \) using a convolutional NN with enough layers and appropriate window size.

Moreover, since the matrix \( G_\eta \) is a symmetric matrix, \( D_1^{(\ell)} \) and \( A_{L_0}^{(L_0)} \) are symmetric and \( (D_2^{(\ell)})^T = D_3^{(\ell)} \).

The implementation, the symmetry is enforced by generating \( D_1^{(\ell)} \) from \( D_2^{(\ell)} \), and replacing \( D_1^{(\ell)} \) (or \( A_{L_0}^{(L_0)} \)) by \( \frac{1}{2}(D_1^{(\ell)} + (D_1^{(\ell)})^T) \) (or \( \frac{1}{2}(A_{L_0}^{(L_0)} + (A_{L_0}^{(L_0)})^T) \), respectively. Since the Schrödinger form considered in this section includes the periodic boundary condition, the convolutional layers are all implemented with periodic padding.

**Numerical results.** The NN discussed above is implemented in Keras [17] (running on top of TensorFlow [1]). The parameters of the NN are initialized randomly from the normal distribution. The loss function is set to be the mean squared error. The NN is trained until convergence using the Nadam optimizer [12] with the learning rates equal to \( 10^{-3} \) for the 1D case and \( 10^{-4} \) for the 2D case. The batch size is set to be one percent of the number of training samples. The support of the scaling function \( \varphi \) is chosen to be \( 2p = 6 \).

The number of levels \( L - L_0 \) in the wavelet transform is \( 6 \) for the 1D case and \( 4 \) for the 2D case.

The data set contains 5,000 different \( \eta \) and for each \( \eta \) Eq. (4.2) is solved with 20 randomly generated \( f \) using the central difference scheme. Therefore, the number of training samples corresponds to the number of different \( \{f_{ij}\} \), rather than different \( \{\eta_i\} \). Half of the generated data is used for training data, while the other half is reserved for testing. The exact solution, obtained by solving Eq. (4.2) is denoted as \( u \) and the prediction of the NN as \( u^{NN} \). The accuracy of the NN is measured by the relative error in the \( \ell^2 \) norm

\[
\epsilon = \frac{\|u - u^{NN}\|_{\ell^2}}{\|u\|_{\ell^2}}.
\]

The training error \( \epsilon_{\text{train}} \) and test error \( \epsilon_{\text{test}} \) are calculated by averaging the relative error over all training and test samples, respectively. The number of parameters in the NN is denoted by \( N_{\text{params}} \).

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( n_{\text{train}} )</th>
<th>( N_{\text{params}} )</th>
<th>( \epsilon_{\text{train}} )</th>
<th>( \epsilon_{\text{test}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>30201</td>
<td>4.38e-3</td>
<td>4.57e-3</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>38061</td>
<td>4.69e-3</td>
<td>4.86e-3</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>58717</td>
<td>4.30e-3</td>
<td>4.47e-3</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>74089</td>
<td>3.68e-3</td>
<td>3.86e-3</td>
</tr>
</tbody>
</table>

Table 1: Relative error in approximating the solution to the Schrödinger form in the 1D case.

For the 1D case, the domain \( \Omega = [0, 1] \) is discretized by a uniform Cartesian grid with 320 points. The positive potential \( \eta(x) \) is generated by (1) sampling independently from \( N(0, 1) \) on a uniform grid with 40 points, (2) interpolating to the 320-point grid via a Fourier interpolation, and (3) point-wise exponentiating.
Figure 4: Two samples (one in red line and the other in blue dashed line) from the test set for the potentials \( \eta \), the source terms \( f \), the predictions \( u^{NN} \) with \( \alpha = 7 \) and \( n_{cnn} = 7 \), and their corresponding error for the Schrödinger form in the 1D case.

followed by a scaling. The source term \( f(x) \) is generated by sampling independently from \( \mathcal{N}(0, 1) \) with the same Fourier interpolation procedure. The results for different values of \( \alpha \) (channel number) and \( n_{cnn} \) (layer number) are reported in Table 1. The best approximation, obtained with \( \alpha = 5 \) and \( n_{cnn} = 5 \), results in a test error of \( 4.6 \times 10^{-3} \) with only \( 3 \times 10^4 \) parameters. Two random samples from the test data are illustrated in Fig. 4 along with the NN prediction.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( n_{cnn} )</th>
<th>( N_{params} )</th>
<th>( \epsilon_{\text{train}} )</th>
<th>( \epsilon_{\text{test}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>5</td>
<td>930447</td>
<td>5.54e-3</td>
<td>5.57e-3</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>1226071</td>
<td>5.51e-3</td>
<td>5.52e-3</td>
</tr>
</tbody>
</table>

Table 2: Relative error in approximating the solution to the Schrödinger form in the 2D case.

For the 2D case, the domain \( \Omega = [0, 1]^2 \) is discretized with a \( 80 \times 80 \) uniform Cartesian mesh. The potential \( \eta(x) \) is generated by (1) sampling independently from \( \mathcal{N}(0, 1) \) on a uniform mesh with \( 10 \times 10 \) points, (2) then interpolating to \( 80 \times 80 \) points via a Fourier interpolation, and (3) point-wise exponentiating followed by appropriate scaling. The source term is sampled point-wisely from a standard Gaussian distribution and interpolated via Fourier interpolation. When trained with \( \alpha = 11 \) and \( n_{cnn} = 5 \), the NN achieves a test error of \( 5.6 \times 10^{-3} \) with only \( 9.3 \times 10^5 \) parameters, as reported in Table 2. \( \eta \) and \( f \) of a representative sample are displayed in Fig. 5 along with the NN prediction and the error.
4.2 Divergence form

The same NN architecture is applied to the Green’s functions of the divergence form

\[- \nabla \cdot (\eta(x)\nabla u(x)) = f(x), \quad x \in [0, 1]^d,\]

\[\int_{[0,1]^d} u(x) \, dx = 0,\]

with \(\eta(x) \geq \eta_0 > 0\) along with the periodic boundary condition. Following the notations of Section 1,

\[L_\eta = -\nabla \cdot \text{diag}(\eta)\nabla, \quad G_\eta = L_\eta^{-1}\]

in the constraint of \(\int_{[0,1]^d} u(x) \, dx = 0.\) (4.9)

When \(\eta(x)\) is close to a fixed \(\eta_0 > 0\), the operator can be decomposed as

\[L_0 = -\eta_0 \Delta, \quad E_\eta = \nabla \cdot \text{diag}(\eta(x) - \eta_0)\nabla, \quad L_\eta = L_0 - E_\eta.\] (4.10)

Since the operator \(E_\eta\) is linearly dependent on \(\eta\), it is easy to check that the discussion for the Schrödinger form holds for the divergence form case as well.

Numerical results. The parameter field \(\eta(x)\) is generated in a way similar to the potential of the Schrödinger form, with the difference that the scaling factor is set to 1/5 and an additive term of 0.5 is applied point-wise to avoid the ill-conditioning of \(G_\eta\). The numerical results for different choices of \(\alpha\) (channel number) and \(n_{cnn}\) (layer number) are summarized in Table 3. For example, a test error of \(8.4 \times 10^{-3}\) is achieved at \(\alpha = 7\) and \(n_{cnn} = 5\) with \(5.9 \times 10^4\) parameters. Two random samples from the test data are illustrated in Fig. 6.
<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$n_{\text{cnn}}$</th>
<th>$N_{\text{params}}$</th>
<th>$\epsilon_{\text{train}}$</th>
<th>$\epsilon_{\text{test}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>58717</td>
<td>7.43e-3</td>
<td>8.41e-3</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>74089</td>
<td>9.34e-3</td>
<td>1.03e-2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>96625</td>
<td>6.79e-3</td>
<td>8.81e-3</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>122005</td>
<td>7.40e-3</td>
<td>1.09e-2</td>
</tr>
</tbody>
</table>

Table 3: Relative error in approximating the solution of divergence form in the 1D case.

Figure 6: Two samples (one in red line and the other in blue dashed line) from the test set for the distinct coefficients $\eta$, the source terms $f$, the predictions $u^{NN}$ with $\alpha = 10$ and $n_{\text{cnn}} = 7$, and their corresponding error for the divergence form in the 1D case.

5 Radiative transfer equation

The radiative transfer equation (RTE) is a fundamental model for describing particle propagation, with applications in many fields, such as neutron transport in reactor physics \[45\], light transport in atmospheric radiative transfer \[43\], heat transfer \[34\], and optical imaging \[33\]. The steady-state RTE in the homogeneous scattering regime is

\[
v \cdot \nabla_x \varphi(x, v) + (\eta(x) + \eta_a(x))\varphi(x, v) = \eta(x)u(x) + f(x), \quad \text{in } \Omega \times S^{d-1}, \quad \Omega \subset \mathbb{R}^d,
\]

\[
\varphi(x, v) = 0, \quad \text{on } \{(x, v) \in \partial\Omega \times S^{d-1} : n(x) \cdot v < 0\},
\]

\[
u(x) = \frac{1}{4\pi} \int_{S^{d-1}} \varphi(x, v) \, dv,
\]

where $\varphi(x, v)$ denotes the photon flux that depends on both space $x$ and angle $v$, $f(x)$ is the light source, $\eta(x)$ is the scattering coefficient, and $\eta_a(x)$ is the physical absorption coefficient. In many applications, it is reasonable to assume $\eta_a(x)$ to be constant. Below, we focus on the most challenging case $\eta_a(x) \equiv 0$.

The numerical solution to the RTE has been extensively studied using the Monte Carlo methods and
uniform Cartesian grid with 80 points, where \( y_0 > 0 \) is selected such that there are 300 points in \( \Omega \). The scattering coefficient \( \eta \) is generated in the same way for \( \eta \) and \( \alpha \) with \( \alpha = n_{\text{enn}} = 5 \). Two representative examples from the test set are shown in Fig. 7.

### Numerical results

The first test is concerned with the one-dimensional slab geometry, where the parameter \( \eta \) varies only in the \( x_1 \) direction (i.e., constant in the \( x_2 \) and \( x_3 \) directions). For this geometry, the integral equation Eq. (5.5) reduces to

\[
(I - K^{(1)}_\eta)u(x) = K^{(1)}_\eta f(x), \quad G_\eta = (I - K^{(1)}_\eta)^{-1}K^{(1)}_\eta,
\]

where \( x \) stands for only \( x_1 \) and the operator \( K^{(1)}_\eta \) is defined as

\[
K^{(1)}_\eta f(x) = \int_{y \in \Omega} k^{(1)}_\eta(x, y) f(y) \, dy,
\]

\[
k^{(1)}_\eta(x, y) = \frac{1}{2} \text{Ei}\left(-|x-y| \int_0^1 \eta(x-s(x-y)) \, ds\right),
\]

with the domain \( \Omega = [0, 1] \). In the implementation, \([-x_0, 1 + x_0]\) is discretized by a uniform Cartesian mesh with \( N = 320 \) points, where \( x_0 > 0 \) is selected such that there are 300 points in \( \Omega \). The scattering coefficient \( \eta \) is generated in the same way for \( \eta(x) \) in Section 4 followed by appropriate rescaling. The source term \( f(x) \), positive due to physical considerations, is generated by sampling independently from \( U(0, 1) \) instead of \( N'(0, 1) \). The values of \( \eta \) and \( f \) outside of \( \Omega \) are set to be 0. The results for different values of \( \alpha \) (channel number) and \( n_{\text{enn}} \) (layer number) are summarized in Table 4. A test error of \( 2.9 \times 10^{-3} \) is achieved with as few as \( 3.4 \times 10^4 \) parameters with \( \alpha = n_{\text{enn}} = 5 \). Two representative examples from the test set are shown in Fig. 7.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( K )</th>
<th>( N_{\text{params}} )</th>
<th>( \epsilon_{\text{train}} )</th>
<th>( \epsilon_{\text{test}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>34131</td>
<td>2.48e-3</td>
<td>2.93e-3</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>41991</td>
<td>2.46e-3</td>
<td>3.01e-3</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>66403</td>
<td>1.92e-3</td>
<td>2.45e-3</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>81775</td>
<td>2.05e-3</td>
<td>2.36e-3</td>
</tr>
</tbody>
</table>

Table 4: Relative error in approximating the solution to the 1D RTE.
scattering coefficient is generated following the same way of \( \eta \) in Section 4 for the 2D case, followed by an appropriate rescaling. The source term \( f(x) \) is generated by sampling independently from \( \mathcal{U}(0,1) \) instead of \( \mathcal{N}(0,1) \). The values of \( \eta \) and \( f \) outside of \( \Omega \) are set to be 0. Results reported in Table 5 show that by setting \( \alpha = 11 \) and \( n_{\text{cnn}} = 5 \), the NN can achieve a test error of \( 4.4 \times 10^{-3} \) with as few as \( 1.3 \times 10^6 \) parameters. A representative sample from the test set is illustrated in Fig. 8.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( K )</th>
<th>( N_{\text{params}} )</th>
<th>( \epsilon_{\text{train}} )</th>
<th>( \epsilon_{\text{test}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>5</td>
<td>1287903</td>
<td>4.39e-3</td>
<td>4.39e-3</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>1663831</td>
<td>5.16e-3</td>
<td>5.16e-3</td>
</tr>
</tbody>
</table>

Table 5: Relative error in approximating the solution to the 2D RTE.

6 Conclusions

This paper presented a meta-learning approach for learning the map from the equation parameter \( \eta \) to the pseudo-differential solution operator \( G_\eta \). Motivated by the nonstandard wavelet form [5], the pseudo-differential operator is compressed to a collection of vectors. The nonlinear map from the parameter to this collection of vectors and the wavelet transform are learned hand-in-hand in the meta-learning approach. Numerical studies are carried out for the Green’s functions of elliptic PDEs as well as the radiative transfer equation.

This approach can be extended in several directions. First, this paper is only concerned with linear operators \( G_\eta \). This work can be readily extended to nonlinear operators if a simple compressed representation
Figure 8: A sample from the test set for the scattering coefficient $\eta$, the source term $f$, the prediction $u^{NN}$ with $\alpha = 15$ and $n_{cnn} = 5$, and the corresponding error for the 2D RTE.

(such as the collection of vectors used here) can be identified. Second, the ConvNet module for the map $\eta \rightarrow C_\eta^{(\ell)}$ can be replaced with the recently proposed multiscale NNs [19, 17, 18], which are more effective for certain global-scale convolutions.

Acknowledgments

The work of Y.F. and L.Y. is partially supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program. The work of J.F. is partially supported by Stanford Graduate Fellowship in Science & Engineering and by “la Caixa” Fellowship, sponsored by the “la Caixa” Banking Foundation of Spain under Fellowship LCF/BQ/AA16/11580045. The work of L.Y. is also partially supported by the National Science Foundation under award DMS-1818449. This work is also supported by the GCP Research Credits Program from Google and AWS Cloud Credits for Research program from Amazon.

References


