Fan et al. *Res Math Sci* (2019) 6:21
https://doi.org/10.1007/s40687-019-0183-3

Research in the Mathematical Sciences

RESEARCH

A multiscale neural network based on hierarchical nested bases

Yuwei Fan[1*], Jordi Feliu-Fabà[2], Lin Lin[3,4], Lexing Ying[1,2] and Leonardo Zepeda-Núñez[4]

*Correspondence:
ywfan@stanford.edu
[1]Department of Mathematics, Stanford University, Stanford, CA 94305, USA
Full list of author information is available at the end of the article

Abstract

In recent years, deep learning has led to impressive results in many fields. In this paper, we introduce a multiscale artificial neural network for high-dimensional nonlinear maps based on the idea of hierarchical nested bases in the fast multipole method and the $\mathcal{H}^2$-matrices. This approach allows us to efficiently approximate discretized nonlinear maps arising from partial differential equations or integral equations. It also naturally extends our recent work based on the generalization of hierarchical matrices (Fan et al. arXiv:1807.01883), but with a reduced number of parameters. In particular, the number of parameters of the neural network grows linearly with the dimension of the parameter space of the discretized PDE. We demonstrate the properties of the architecture by approximating the solution maps of nonlinear Schrödinger equation, the radiative transfer equation and the Kohn–Sham map.

Keywords: Hierarchical nested bases, Fast multipole method, $\mathcal{H}^2$-matrix, Nonlinear mappings, Artificial neural network, Locally connected neural network, Convolutional neural network

1 Introduction

In recent years, deep learning and more specifically deep artificial neural networks have received ever-increasing attention from the scientific community. Coupled with a significant increase in the computer power and the availability of massive datasets, artificial neural networks have fueled several breakthroughs across many fields, ranging from classical machine learning applications such as object recognition [32,51,55,63], speech recognition [24], natural language processing [48,52] or text classification [60] to more modern domains such as language translation [54], drug discovery [38], genomics [34,62], game playing [50], among many others. For a more extensive review of deep learning, we point the reader to [18,33,49].

Recently, neural networks have also been employed to solve challenging problems in numerical analysis and scientific computing [3,6,7,10,11,15,27,41,44,47,53]. While a fully connected neural network can be theoretically used to approximate very general mappings [14,26,28,40], it may also lead to a prohibitively large number of parameters, resulting in extremely long training stages and overwhelming memory footprints. Therefore, it is often necessary to incorporate existing knowledge of the underlying structure of the problem into the design of the network architecture. One promising and general strategy is to build neural networks based on a multiscale decomposition [17,35,61]. The

general idea, often used in image processing [4,9,12,37,46,59], is to learn increasingly coarse-grained features of a complex problem across different layers of the network structure, so that the number of parameters in each layer can be effectively controlled.

In this paper, we aim at employing neural networks to effectively approximate nonlinear maps of the form

$$u = \mathcal{M}(v), \quad u, v \in \Omega \subset \mathbb{R}^d, \tag{1.1}$$

which can be viewed as a nonlinear generalization of pseudo-differential operators. This type of maps may arise from parameterized and discretized partial differential equations (PDE) or integral equations (IE), with u being the quantity of interest and v the parameter that serves to identify a particular configuration of the system.

We propose a neural network architecture based on the idea of *hierarchical nested bases* used in the fast multipole method (FMM) [19] and the $\mathcal{H}^2$-matrix [22] to represent nonlinear maps arising in computational physics, motivated by the favorable complexity of the FMM/$\mathcal{H}^2$-matrices in the linear setting. The proposed neural network, which we call MNN-$\mathcal{H}^2$, is able to efficiently represent the nonlinear maps benchmarked in the sequel; in such cases, the number of parameters required to approximate the maps can grow linearly with respect to N, the dimension of the parameter space of the discretized PDE. Our presentation will mostly follow the notation of the $\mathcal{H}^2$-matrix framework due to its algebraic nature.

The proposed architecture, MNN-$\mathcal{H}^2$, is a direct extension of the framework used to build a multiscale neural networks based on $\mathcal{H}$-matrices (MNN-$\mathcal{H}$) [17] to $\mathcal{H}^2$-matrices. We demonstrate the capabilities of MNN-$\mathcal{H}^2$ with three classical yet challenging examples in computational physics: the nonlinear Schrödinger equation [2,42], the radiative transfer equation [29,30,39,43] and the Kohn–Sham map [25,31]. We find that MNN-$\mathcal{H}^2$ can yield comparable results to those obtained from MNN-$\mathcal{H}$, but with a reduced number of parameters, thanks to the use of hierarchical nested bases.

The outline of the paper is as follows. Section 2 reviews the $\mathcal{H}^2$-matrices and interprets them within the framework of neural networks. Section 3 extends the neural network representation of $\mathcal{H}^2$-matrices to the nonlinear case. Section 4 discusses the implementation details and demonstrates the accuracy of the architecture in representing nonlinear maps, followed by the conclusion and future directions in Sect. 5.

2 Neural network architecture for $\mathcal{H}^2$-matrices

In this section, we reinterpret the matrix–vector multiplication of $\mathcal{H}^2$-matrices within the framework of neural networks. In Sect. 2.1, we briefly review $\mathcal{H}^2$-matrices for the 1D case and propose the neural network architecture for the matrix–vector multiplication of $\mathcal{H}^2$-matrices in Sect. 2.2. An extension to the multidimensional setting is presented in Sect. 2.3.

2.1 $\mathcal{H}^2$-matrices

The concept of hierarchical matrices ($\mathcal{H}$-matrices) was first introduced by Tyrtyshnikov [58], Hackbusch [20] and Hackbusch and Khoromskij [21] as an algebraic formulation of algorithms for hierarchical off-diagonal low-rank matrices. This framework provides efficient numerical methods for solving linear systems arising from integral equations and partial differential equations [8] and it enjoys an $O(N \log(N))$ arithmetic com-

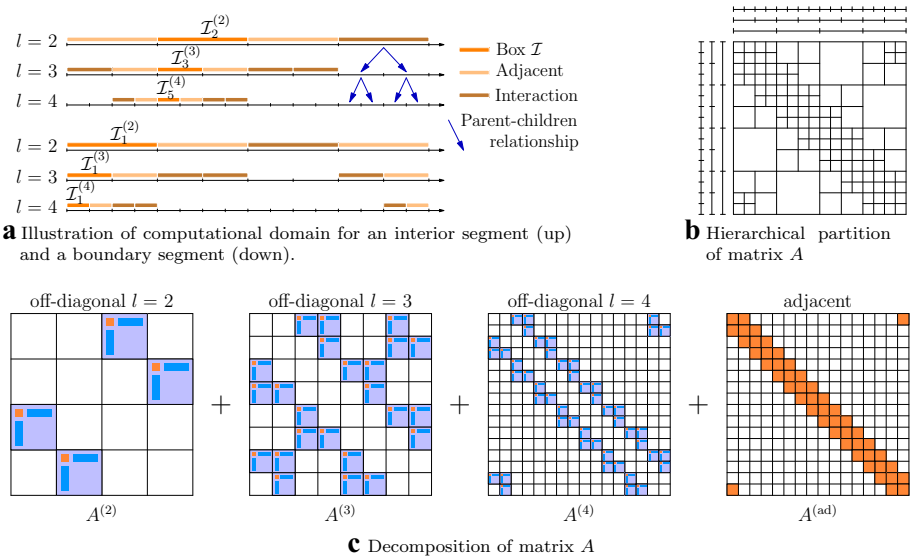


Fig. 1 Hierarchical partition of computational domain, its corresponding partition of matrix A and the decomposition of matrix A

plexity for the matrix–vector multiplication. By incorporating the idea of hierarchical nested bases from the fast multipole method [19], the $\mathcal{H}^2$-matrices were introduced in [22] to further reduce the logarithmic factor in the complexity, provided that a so-called consistency condition is fulfilled. In the sequel, we follow the notation introduced in [17] to provide a brief introduction to the framework of $\mathcal{H}^2$-matrices in a simple uniform Cartesian setting. We refer readers to [8,22,36] for further details.

Consider the integral equation

$$u(x) = \int_{\Omega} g(x, y)v(y)\,\mathrm{d}y, \quad \Omega = [0, 1), \tag{2.1}$$

where u and v are periodic in Ω and $g(x, y)$ is smooth and numerically low rank away from the diagonal. A discretization with an uniform grid with $N = 2^L m$ discretization points yields the linear system given by

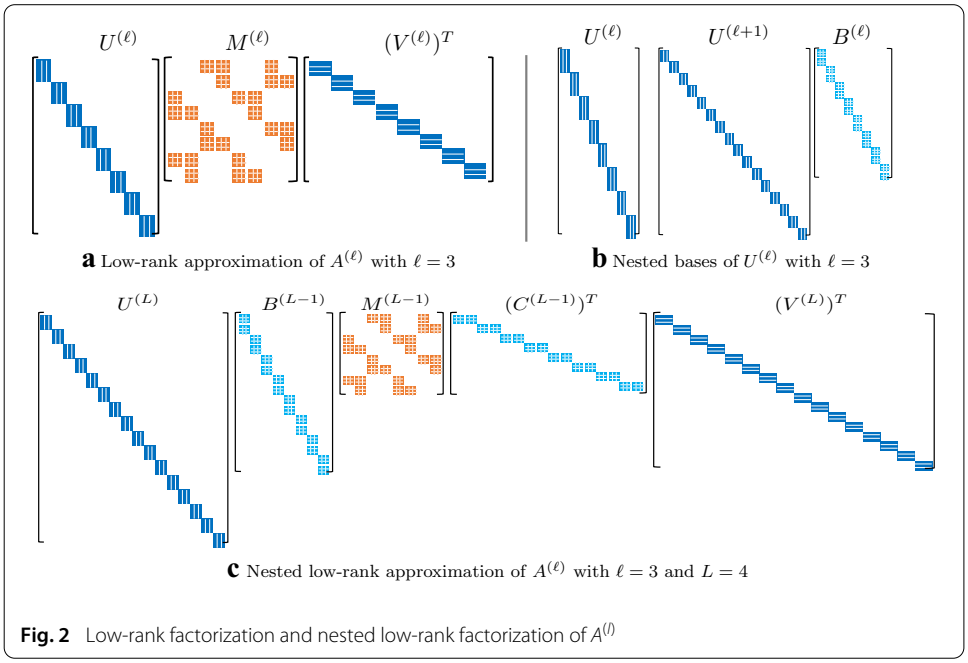
$$u = Av, \tag{2.2}$$

where $A \in \mathbb{R}^{N\times N}$, and $u, v \in \mathbb{R}^N$ are the discrete analogs of $u(x)$ and $v(x)$, respectively.

A hierarchical dyadic decomposition of the grid in $L + 1$ levels can be introduced as follows. Let $\mathcal{I}^{(0)}$, the 0th level of the decomposition, be the set of all grid points defined as

$$\mathcal{I}^{(0)} = \{k/N : k = 0, \ldots, N - 1\}. \tag{2.3}$$

At each level ℓ ($1 \le \ell \le L$), the grid is decomposed in 2^ℓ disjoint segments. Each segment is defined by $\mathcal{I}_i^{(\ell)} = \mathcal{I}^{(0)} \cap [(i - 1)/2^\ell, i/2^\ell)$ for $i = 1, \ldots, 2^\ell$. Throughout this manuscript, $\mathcal{I}^{(\ell)}$(or $\mathcal{J}^{(\ell)}$) will denote a generic segment of a given level ℓ, and the superscript ℓ will be omitted when the level is clear from the context.



a Low-rank approximation of $A^{(\ell)}$ with $\ell = 3$

b Nested bases of $U^{(\ell)}$ with $\ell = 3$

c Nested low-rank approximation of $A^{(\ell)}$ with $\ell = 3$ and $L = 4$

Fig. 2 Low-rank factorization and nested low-rank factorization of $A^{(l)}$

Given a vector $v \in \mathbb{R}^N$, we denote $v_{\mathcal{I}}$ the elements of v indexed by $\mathcal{I}$, and given a matrix $A \in \mathbb{R}^{N\times N}$, we denote $A_{\mathcal{I},\mathcal{J}}$ the submatrix of A indexed by $\mathcal{I} \times \mathcal{J}$. Following the usual nomenclature in $\mathcal{H}$-matrices, we define the following relationships between segments:

- $\mathcal{C}(\mathcal{I})$ children list of $\mathcal{I}$ for $\ell < L$: list of the segments on level $\ell + 1$ that are subset of $\mathcal{I}$;
- $\mathcal{P}(\mathcal{I})$ parent of $\mathcal{I}$ for $\ell > 0$: set of segments $\mathcal{J}$ such that $\mathcal{I} \in \mathcal{C}(\mathcal{J})$;
- $\mathsf{NL}(\mathcal{I})$ neighbor list of $\mathcal{I}$: list of the segments on level ℓ that are adjacent to $\mathcal{I}$ including $\mathcal{I}$ itself;
- $\mathsf{IL}(\mathcal{I})$ interaction list of $\mathcal{I}$ for $\ell \geq 2$: set that contains all the segments on level ℓ that are children of segments in $\mathsf{NL}(\mathcal{P}(\mathcal{I}))$ minus $\mathsf{NL}(\mathcal{I})$, i.e., $\mathsf{IL}(\mathcal{I}) = \mathcal{C}(\mathsf{NL}(\mathcal{P}(\mathcal{I}))) - \mathsf{NL}(\mathcal{I})$.

Figure 1a illustrates this dyadic hierarchical partition of the computational domain, the parent–children relationship, the neighbor list and interaction list on levels $\ell = 2, 3, 4$. The matrix A can be hierarchically partitioned as illustrated in Fig. 1b. The partition leads to a multilevel decomposition of A shown in Fig. 1c, which can be written as

$$A = \sum_{\ell=2}^{L} A^{(\ell)} + A^{(\mathrm{ad})}, \qquad A^{(\ell)}_{\mathcal{I},\mathcal{J}} = \begin{cases} A_{\mathcal{I},\mathcal{J}}, & \mathcal{I} \in \mathsf{IL}(\mathcal{J}); \\ 0, & \text{otherwise}, \end{cases} \quad \mathcal{I}, \mathcal{J} \text{ at level } l, \quad 2 \leq l \leq L,$$
$$A^{(\mathrm{ad})}_{\mathcal{I},\mathcal{J}} = \begin{cases} A_{\mathcal{I},\mathcal{J}}, & \mathcal{I} \in \mathsf{NL}(\mathcal{J}); \\ 0, & \text{otherwise}, \end{cases} \quad \mathcal{I}, \mathcal{J} \text{ at level } L. \tag{2.4}$$

For simplicity, we suppose that each block has a fixed numerical rank at most r, i.e.,

$$A^{(\ell)}_{\mathcal{I},\mathcal{J}} \approx U^{(\ell)}_{\mathcal{I}} M^{(\ell)}_{\mathcal{I},\mathcal{J}} (V^{(\ell)}_{\mathcal{J}})^T, \quad U^{(\ell)}_{\mathcal{I}}, V^{(\ell)}_{\mathcal{J}} \in \mathbb{R}^{N/2^\ell \times r}, \quad M^{(\ell)}_{\mathcal{I},\mathcal{J}} \in \mathbb{R}^{r\times r}, \tag{2.5}$$

where $\mathcal{I}$ and $\mathcal{J}$ are any interacting segments at level ℓ. We can approximate $A^{(\ell)}$ as $A^{(\ell)} \approx U^{(\ell)} M^{(\ell)} (V^{(\ell)})^T$ as depicted in Fig. 2a. Here $U^{(\ell)}$, $V^{(\ell)}$ are block diagonal matrices

with diagonal blocks $U_{\mathcal{I}}^{(\ell)}$ and $V_{\mathcal{I}}^{(\ell)}$ for $\mathcal{I}$ at level ℓ, respectively, and $M^{(\ell)}$ aggregates all the blocks $M_{\mathcal{I},\mathcal{J}}^{(\ell)}$ for all interacting segments $\mathcal{I}$, $\mathcal{J}$ at level ℓ.

The key feature of $\mathcal{H}^2$-matrices is that the bases matrices $U_{\mathcal{I}}^{(\ell)}$ and $V_{\mathcal{I}}^{(\ell)}$ between parent and children segments enjoy a *nested* low-rank approximation. More precisely, if $\mathcal{I}$ is at level $2 \le l < L$ and $\mathcal{J}_1, \mathcal{J}_2 \in \mathcal{C}(\mathcal{I})$ are at level $l+1$, then $U_{\mathcal{I}}^{(l)}$ and $V_{\mathcal{I}}^{(l)}$ satisfy the following approximation:

$$U_{\mathcal{I}}^{(\ell)} \approx \begin{pmatrix} U_{\mathcal{J}_1}^{(\ell+1)} & \\ & U_{\mathcal{J}_2}^{(\ell+1)} \end{pmatrix} \begin{pmatrix} B_{\mathcal{J}_1}^{(\ell)} \\ B_{\mathcal{J}_2}^{(\ell)} \end{pmatrix}, \quad V_{\mathcal{I}}^{(\ell)} \approx \begin{pmatrix} V_{\mathcal{J}_1}^{(\ell+1)} & \\ & V_{\mathcal{J}_2}^{(\ell+1)} \end{pmatrix} \begin{pmatrix} C_{\mathcal{J}_1}^{(\ell)} \\ C_{\mathcal{J}_2}^{(\ell)} \end{pmatrix}, \tag{2.6}$$

where $B_{\mathcal{J}_i}^{(\ell)}, C_{\mathcal{J}_i}^{(\ell)} \in \mathbb{R}^{r\times r}$, $i = 1, 2$. As depicted in Fig. 2b, if we introduce the matrix $B^{(l)}$ ($C^{(l)}$) that aggregates all the blocks $B_{\mathcal{J}_i}^{(\ell)}$ ($C_{\mathcal{J}_i}^{(\ell)}$) for all the parent–children pairs ($\mathcal{I}$, $\mathcal{J}_i$), (2.6) can be compactly written as $U^{(l)} \approx U^{(l+1)}B^{(l)}$ and $V^{(l)} \approx V^{(l+1)}C^{(l)}$. Thus, the decomposition (2.4) can be further factorized as

$$A = \sum_{\ell=2}^{L} A^{(\ell)} + A^{(\mathrm{ad})} \approx \sum_{\ell=2}^{L} U^{(L)}B^{(L-1)}\cdots B^{(\ell)}M^{(\ell)}(C^{(\ell)})^T\cdots(C^{(L-1)})^T(V^{(L)})^T + A^{(\mathrm{ad})}. \tag{2.7}$$

The matrix–vector multiplication of A with an arbitrary vector v can be approximated by

$$Av \approx \sum_{\ell=2}^{L} U^{(L)}B^{(L-1)}\cdots B^{(\ell)}M^{(\ell)}(C^{(\ell)})^T\cdots(C^{(L-1)})^T(V^{(L)})^T v + A^{(\mathrm{ad})}v. \tag{2.8}$$

Algorithm 1 Application of $\mathcal{H}^2$-matrices on a vector $v \in \mathbb{R}^N$.

1: $u^{(\mathrm{ad})} = A^{(\mathrm{ad})}v$;
2: $\xi^{(L)} = (V^{(L)})^T v$;
3: **for** ℓ from $L-1$ to 2 by -1 **do**
4: $\quad \xi^{(\ell)} = (C^{(\ell)})^T \xi^{(\ell+1)}$;
5: **end for**
6: **for** ℓ from 2 to L **do**
7: $\quad \zeta^{(\ell)} = M^{(\ell)}\xi^{(\ell)}$;
8: **end for**
9: $\chi = 0$;
10: **for** ℓ from 2 to $L-1$ **do**
11: $\quad \chi = \chi + \zeta^{(\ell)}$;
12: $\quad \chi = B^{(\ell)}\chi$;
13: **end for**
14: $\chi = \chi + \zeta^{(L)}$;
15: $\chi = U^{(L)}\chi$;
16: $u = \chi + u^{(\mathrm{ad})}$;

Algorithm 1 provides the implementation of the matrix–vector multiplication of $\mathcal{H}^2$-matrices. The key properties of the matrices $U^{(L)}$, $V^{(L)}$, $B^{(\ell)}$, $C^{(\ell)}$, $M^{(\ell)}$ and $A^{(\mathrm{ad})}$ are summarized as follows:

Property 1 *The matrices*

1. $U^{(L)}$ *and* $V^{(L)}$ *are block diagonal matrices with block size* $N/2^L \times r$;
2. $B^{(\ell)}$ *and* $C^{(\ell)}$, $\ell = 2, \cdots, L-1$ *are block diagonal matrices with block size* $2r \times r$;
3. $M^{(\ell)}$, $\ell = 2, \cdots, L$ *are block cyclic band matrices with block size* $r \times r$ *and band size* $n_b^{(\ell)}$, *which is* 2 *for* $\ell = 2$ *and* 3 *for* $\ell > 2$;
4. $A^{(ad)}$ *is a block cyclic band matrix with block size* $m \times m$ *with band size* $n_b^{(\mathrm{ad})} = 1$.

2.2 Matrix–vector multiplication as a neural network

We represent the matrix–vector multiplication (2.8) using the framework of neural networks. We first introduce our main tool—locally connected network—in Sect. 2.2.1 and then present the neural network representation of (2.8) in Sect. 2.2.2.

2.2.1 Locally connected network

In order to simplify the notation, let us present the 1D case as an example. In this setup, an NN layer can be represented by a 2-tensor with size $\alpha \times N_x$, where α is called the channel dimension and N_x is usually called the spatial dimension. A locally connected network is a type of mapping between two adjacent layers, where the output of each neuron depends only locally on the input. If a layer ξ with size $\alpha \times N_x$ is connected to a layer ζ with size $\alpha' \times N_x'$ by a locally connected (LC) network, then

$$\zeta_{c',i} = \phi\left(\sum_{j=(i-1)s+1}^{(i-1)s+w} \sum_{c=1}^{\alpha} W_{c',c;i,j}\xi_{c,j} + b_{c',i}\right), \quad i = 1, \ldots, N_x',\ c' = 1, \ldots, \alpha', \tag{2.9}$$

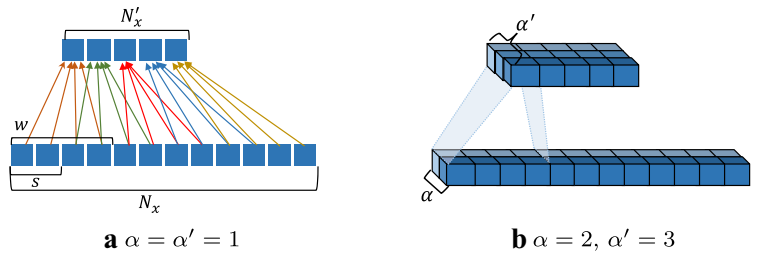
where ϕ is a pre-specified function, called activation, usually chosen to be, e.g., a linear function, a rectified linear unit (ReLU) function or a sigmoid function. The parameters w and s are called the kernel window size and stride, respectively. Figure 3 presents a sample of the LC network. Furthermore, we call the layer ζ locally connected layer (LC layer) hereafter.

In (2.9), the LC network is represented using tensor notation; however, we can reshape ζ and ξ to a vector by column major indexing and W to a matrix and write (2.9) into a matrix–vector form as

$$\zeta = \phi(W\xi + b). \tag{2.10}$$

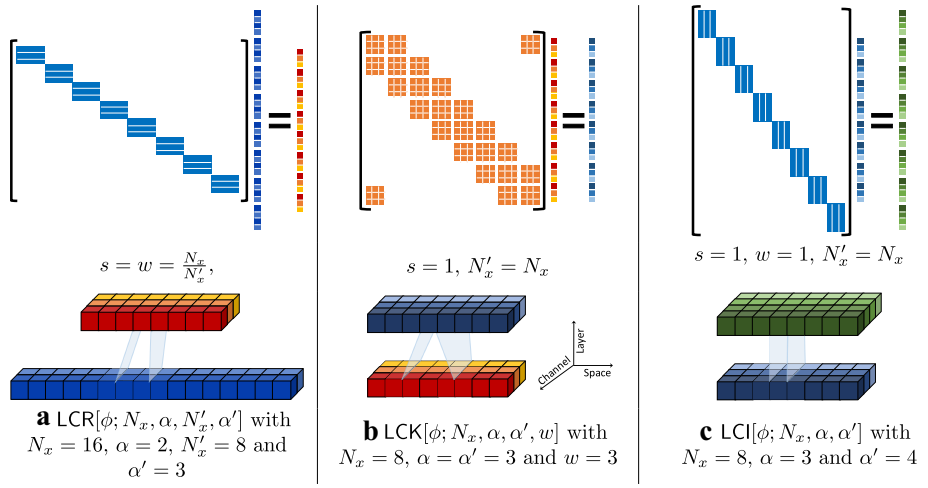
For later usage, we define $\mathsf{Reshape}[n_1, n_2]$ to be the map that reshapes a tensor with size $n_1' \times n_2'$ to a 2-tensor of size $n_1 \times n_2$ such that $n_1 n_2 = n_1' n_2'$ by column major indexing. Here, we implicitly regard a vector with size n as a 2-tensor with size $1 \times n$.

Each LC network has six parameters: N_x, α, N_x', α', w and s. We define three types of LC networks by specifying some of their parameters. The upper figures in Fig. 4 depict its corresponding formula in matrix–vector form (2.10), and the lower figures show a diagram of the map.



a $\alpha = \alpha' = 1$ **b** $\alpha = 2,\ \alpha' = 3$

Fig. 3 Sample of LC network with $N_x = 12$, $s = 2$, $w = 4$ and $N_x' = 5$



a LCR$[\phi; N_x, \alpha, N_x', \alpha']$ with $N_x = 16,\ \alpha = 2,\ N_x' = 8$ and $\alpha' = 3$

b LCK$[\phi; N_x, \alpha, \alpha', w]$ with $N_x = 8,\ \alpha = \alpha' = 3$ and $w = 3$

c LCI$[\phi; N_x, \alpha, \alpha']$ with $N_x = 8,\ \alpha = 3$ and $\alpha' = 4$

Fig. 4 Three instances of locally connected networks used to represent the matrix–vector multiplication. The upper portions of each column depict the patterns of the matrices, and the lower portions are their respective analogs using locally connect networks

LCR Restriction map: set $s = w = \frac{N_x}{N_x'}$ in LC. This map represents the multiplication of a block diagonal matrix with block sizes $\alpha' \times s\alpha$ and a vector with size $N_x\alpha$. We denote this map by LCR$[\phi; N_x, \alpha, N_x', \alpha']$. The application of LCR[linear; 16, 2, 8, 3] is depicted in Fig. 4a.

LCK Kernel map: set $s = 1$ and $N_x' = N_x$. This map represents the multiplication of a periodically banded block matrix (with block size $\alpha' \times \alpha$ and band size $\frac{w-1}{2}$) with a vector of size $N_x\alpha$. To account for the periodicity, we periodic pad the input layer $\xi_{c,j}$ on the spatial dimension to the size $(N_x + w - 1) \times \alpha$. We denote this map by LCK$[\phi; N_x, \alpha, \alpha', w]$, which contains two steps: the periodic padding of $\xi_{c,j}$ on the spatial dimension and the application of (2.9). The application of LCK[linear; 8, 3, 3, 3] is depicted in Fig. 4b.

LCI Interpolation map: set $s = w = 1$ and $N_x' = N_x$ in LC. This map represents the multiplication of a block diagonal matrix with block size $\alpha' \times \alpha$, times a vector of size $N_x\alpha$. We denote the map by LCI$[\phi; N_x, \alpha, \alpha']$. The application of LCI[linear; 8, 3, 4] is depicted in Fig. 4c.

2.2.2 Neural network representation

We need to find a neural network representation of the following six operations in order to perform the matrix–vector multiplication (2.8) for $\mathcal{H}^2$-matrices:

$$\xi^{(L)} = (V^{(L)})^T v, \tag{2.11a}$$

$$\xi^{(\ell)} = (C^{(\ell)})^T \xi^{(\ell+1)}, \quad 2 \le \ell < L, \tag{2.11b}$$

$$\zeta^{(\ell)} = M^{(\ell)} \xi^{(\ell)}, \quad 2 \le \ell \le L, \tag{2.11c}$$

$$\chi^{(\ell)} = B^{(\ell)} \zeta^{(\ell)}, \quad 2 \le \ell < L, \tag{2.11d}$$

$$\chi^{(L)} = U^{(L)} \zeta^{(L)}, \tag{2.11e}$$

$$u^{(\mathrm{ad})} = A^{(\mathrm{ad})} v. \tag{2.11f}$$

Following Property 1.1 and the definition of LCR, we can directly represent (2.11a) as

$$(2.11a) \Rightarrow \xi^{(L)} = \mathsf{LCR}[\mathsf{linear}; N, 1, 2^L, r](v). \tag{2.12}$$

Here we note that the output of LCR is a 2-tensor, so we should reshape it to a vector. In the next step, when applying other operations, it is reshaped back to a 2-tensor with same size. These operations usually do not produce any effect on the whole pipeline, so they are omitted in the following discussion. Similarly, since all of $V^{(L)}$, $B^{(\ell)}$ and $C^{(\ell)}$ are block diagonal matrices (Properties 1.1 and 1.2),

$$\begin{aligned}
&(2.11b) \Rightarrow \xi^{(\ell)} = \mathsf{LCR}[\mathsf{linear}; 2^{\ell+1}, r, 2^{\ell}, r](\xi^{(\ell+1)}),\\
&(2.11d) \Rightarrow \chi^{(\ell)} = \mathsf{LCI}[\mathsf{linear}; 2^{\ell}, r, 2r](\zeta^{(\ell)}),\\
&(2.11e) \Rightarrow \chi^{(L)} = \mathsf{LCI}[\mathsf{linear}; 2^{L}, r, m](\zeta^{(\ell)}).
\end{aligned} \tag{2.13}$$

Analogously, using Properties 1.3, 1.4 and the definition of LCK,

$$\begin{aligned}
&(2.11c) \Rightarrow \zeta^{(\ell)} = \mathsf{LCK}[\mathsf{linear}; 2^{\ell}, r, r, 2n_b^{(\ell)} + 1](\xi^{(\ell)}),\\
&(2.11f) \Rightarrow u^{(\mathrm{ad})} = \mathsf{LCK}[\mathsf{linear}; 2^{L}, m, m, 2n_b^{(\mathrm{ad})} + 1](v).
\end{aligned} \tag{2.14}$$

Combining (2.12), (2.13) and (2.14) and adding necessary Reshape, we can now translate Algorithm 1 to a neural network representation of the matrix–vector multiplication of $\mathcal{H}^2$-matrices in Algorithm 2, which is illustrated in Fig. 5.

Let us now calculate the number of parameters used in the network in Algorithm 2. For simplicity, we ignore the number of parameters in the bias terms b and only consider the

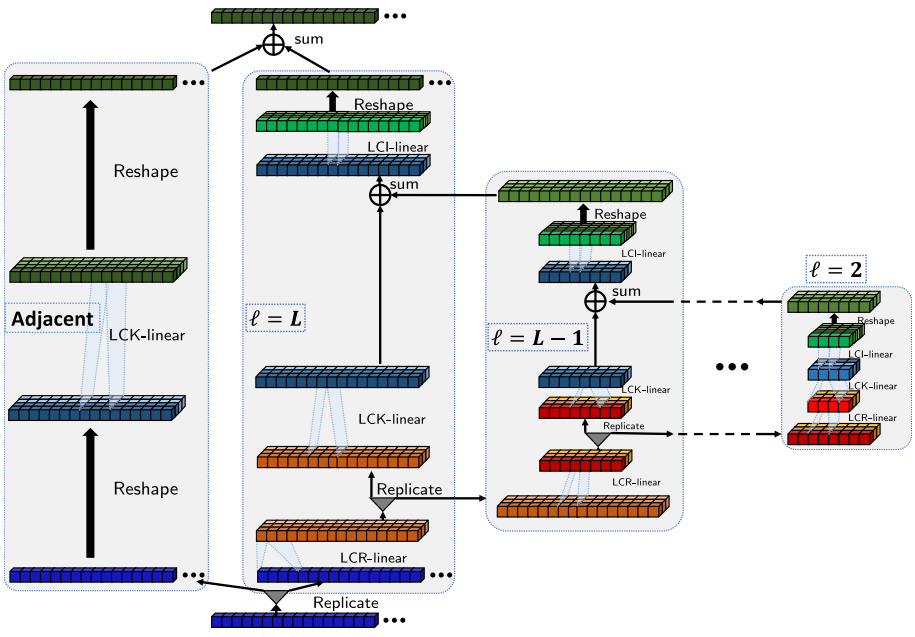


Fig. 5 Neural network architecture for the matrix–vector multiplication of $\mathcal{H}^2$-matrices

Algorithm 2 Application of NN architecture for $\mathcal{H}^2$-matrices on a vector $v \in \mathbb{R}^N$.

$\tilde{v} = \mathsf{Reshape}[m, 2^L](v)$;
$\tilde{u}^{(\mathrm{ad})} = \mathsf{LCK}[\mathsf{linear}; 2^L, m, m, 2n_b^{(\mathrm{ad})} + 1](\tilde{v})$;
$u^{(\mathrm{ad})} = \mathsf{Reshape}[1, N](\tilde{u}^{(\mathrm{ad})})$;
$\xi^{(L)} = \mathsf{LCR}[\mathsf{linear}; N, 1, 2^L, r](v)$;
for ℓ from $L-1$ to 2 by -1 **do**
    $\xi^{(\ell)} = \mathsf{LCR}[\mathsf{linear}; 2^{\ell+1}, r, 2^\ell, r](\xi^{(\ell+1)})$;
end for
for ℓ from 2 to L **do**
    $\zeta^{(\ell)} = \mathsf{LCK}[\mathsf{linear}; 2^\ell, r, r, 2n_b^{(\ell)} + 1](\xi^{(\ell)})$;
end for
$\chi = 0$;
for ℓ from 2 to $L-1$ **do**
    $\chi = \chi + \zeta^{(\ell)}$;
    $\chi = \mathsf{LCI}[\mathsf{linear}; 2^\ell, r, 2r](\chi)$;
    $\chi = \mathsf{Reshape}[r, 2^{\ell+1}](\chi)$;
end for
$\chi = \chi + \zeta^{(L)}$;
$\chi = \mathsf{LCI}[\mathsf{linear}; 2^L, r, m](\chi)$;
$\chi = \mathsf{Reshape}[1, N](\chi)$;
$u = \chi + u^{(\mathrm{ad})}$;

ones in the weight matrices W. Given that the number of parameters in an LC layer is $N_x' \alpha \alpha' w$, the number of parameters for each type of network is:

$$N_p^{\mathsf{LCR}} = N_x \alpha \alpha', \quad N_p^{\mathsf{LCK}} = N_x \alpha \alpha' w, \quad N_p^{\mathsf{LCI}} = N_x \alpha \alpha'. \tag{2.15}$$

Then the total number of parameters in Algorithm 2 is

$$\begin{aligned} N_p^{\mathcal{H}^2} &= 2^L m^2 (2n_b^{(\mathrm{ad})} + 1) + Nr + 2 \sum_{\ell=2}^{L-1} 2^{\ell+1} r^2 + \sum_{\ell=2}^{L} 2^\ell r^2 (2n_b^{(\ell)} + 1) + 2^L rm \\ &\le Nm(2n_b + 1) + 2Nr + 2Nr(2n_b + 3) \\ &\le 3Nm(2n_b + 3) = O(N), \end{aligned} \tag{2.16}$$

where $n_b = \max(n_b^{(\mathrm{ad})}, n_b^{(\ell)})$, $r \le m$ and $2^L m = N$ are used. The calculation shows that the number of parameters in the neural network scales linearly in N and is therefore of the same order as the memory storage in $\mathcal{H}^2$-matrices. This is lower than the quasilinear order $O(N \log(N))$ of $\mathcal{H}$-matrices and its neural network generalization.

2.3 Multidimensional case

Following the discussion in the previous section, Algorithm 2 can be easily extended to the d-dimensional case by performing a tensor product of the one-dimensional case. In this subsection, we consider $d = 2$, for instance, and the generalization to the d-dimensional case becomes straightforward. For the integral equation

$$u(x) = \int_\Omega g(x, y) v(y) \, \mathrm{d}y, \quad \Omega = [0, 1) \times [0, 1), \tag{2.17}$$

we discretize it with an uniform grid with $N \times N$, $N = 2^L m$, grid points and denote the resulting matrix obtained from the discretization of (2.17) by A. Conceptually Algorithm 2 required the following three components:

1. multiscale decomposition of the matrix A, given by (2.4);
2. nested low-rank approximation of the far-field blocks of A, given by (2.6) and Property 1 for the resulting matrices;
3. definition of LC layers and theirs relationship (2.12),(2.13) and (2.14) with the matrices in Property 1.

We briefly explain how each step can be seamlessly extended to the higher dimension in what follows.

Multiscale decomposition The grid is hierarchically partitioned into $L+1$ levels, in which each box is defined by $\mathcal{I}_i^{(d,\ell)} = \mathcal{I}_{i_1}^{(\ell)} \otimes \mathcal{I}_{i_2}^{(\ell)}$, where $i = (i_1, i_2)$ is a multidimensional index, $\mathcal{I}_{i_1}^{(\ell)}$ identifies the segments for 1D case and $\otimes$ is the tensor product. The definitions of the children list, parent, neighbor list and interaction list can be easily extended. Each box $\mathcal{I}$ with $\ell < L$ has four children. Similarly, the decomposition (2.4) on A can also be extended.

Nested low-rank approximation Following the structure of $\mathcal{H}^2$-matrices, the nonzero blocks of $A^{(\ell)}$ can be approximated by

$$A_{\mathcal{I},\mathcal{J}}^{(\ell)} \approx U_{\mathcal{I}}^{(\ell)} M_{\mathcal{I},\mathcal{J}}^{(\ell)} (V_{\mathcal{J}}^{(\ell)})^T, \quad U_{\mathcal{I}}^{(\ell)}, V_{\mathcal{J}}^{(\ell)} \in \mathbb{R}^{(N/2^\ell)^2 \times r}, \quad M_{\mathcal{I},\mathcal{J}}^{(\ell)} \in \mathbb{R}^{r \times r}, \tag{2.18}$$

and the matrices $U^{(\ell)}$ satisfy the consistency condition, i.e.,

$$U_{\mathcal{I}}^{(\ell)} \approx \begin{pmatrix} U_{\mathcal{J}_1}^{(\ell+1)} & & & \\ & U_{\mathcal{J}_2}^{(\ell+1)} & & \\ & & U_{\mathcal{J}_3}^{(\ell+1)} & \\ & & & U_{\mathcal{J}_4}^{(\ell+1)} \end{pmatrix} \begin{pmatrix} B_{\mathcal{J}_1}^{(\ell)} \\ B_{\mathcal{J}_2}^{(\ell)} \\ B_{\mathcal{J}_3}^{(\ell)} \\ B_{\mathcal{J}_4}^{(\ell)} \end{pmatrix}, \tag{2.19}$$

where $\mathcal{J}_j$ are children of $\mathcal{I}$, and $B_{\mathcal{J}_j}^{(\ell)} \in \mathbb{R}^{r \times r}$, $j = 1, \ldots, 4$. Similarly, the matrices $V^{(\ell)}$ also have the same nested relationship.

We denote an entry of a tensor T by $T_{i,j}$, where i is two-dimensional index $i = (i_1, i_2)$. Using the tensor notations, $U^{(L)}$ and $V^{(L)}$ in (2.8) can be treated as 4-tensors of dimension $N \times N \times 2^L r \times 2^L$, while $B^{(\ell)}$ and $C^{(\ell)}$ in (2.8) can be treated as 4-tensors of dimension $2^{\ell+1} r \times 2^{\ell+1} \times 2^\ell r \times 2^\ell$. We generalize the notion of band matrix A to *band tensors* T by satisfying

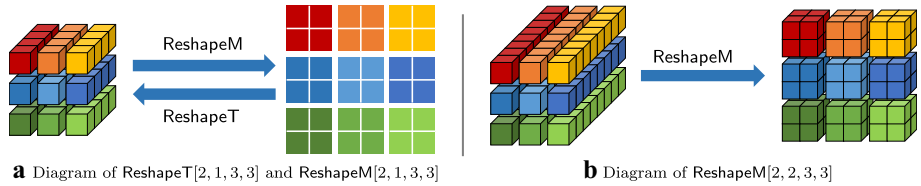
$$T_{i,j} = 0, \quad \text{if } |i_1 - j_1| > n_{b,1} \text{ or } |i_2 - j_2| > n_{b,2}, \tag{2.20}$$

where $n_b = (n_{b,1}, n_{b,2})$ is called the band size for tensor. Thus, Property 1 can be extended to

Property 2 *The 4-tensors*

1. $U^{(L)}$ *and* $V^{(L)}$ *are block diagonal tensors with block size* $N/2^L \times N/2^L \times r \times 1$.
2. $B^{(\ell)}$ *and* $C^{(\ell)}$, $\ell = 2, \cdots, L-1$, *are block diagonal tensors with block size* $2r \times 2 \times r \times 1$
3. $M^{(\ell)}$, $\ell = 2, \cdots, L$, *are block cyclic band tensors with block size* $r \times 1 \times r \times 1$ *and band size* $n_b^{(\ell)}$, *which is* $(2, 2)$ *for* $\ell = 2$ *and* $(3, 3)$ *for* $\ell > 2$;
4. $A^{(ad)}$ *is a block cyclic band matrix with block size* $m \times m \times m \times m$ *and band size* $n_b^{(\mathrm{ad})} = (1, 1)$.

LC layers An NN layer for 2D can be represented by a 3-tensor of size $\alpha \times N_{x,1} \times N_{x,2}$, where α is the channel dimension and $N_{x,1}, N_{x,2}$ are the spatial dimensions. If a layer ξ with



a Diagram of ReshapeT$[2,1,3,3]$ and ReshapeM$[2,1,3,3]$ **b** Diagram of ReshapeM$[2,2,3,3]$

Fig. 6 Diagram of ReshapeT and ReshapeM in Algorithm 3

size $\alpha \times N_{x,1} \times N_{x,2}$ is connected to a locally connected layer ζ with size $\alpha' \times N'_{x,1} \times N'_{x,2}$, then

$$\zeta_{c',i} = \phi\left(\sum_{j=(i-1)s+1}^{(i-1)s+w} \sum_{c=1}^{\alpha} W_{c',c;i,j}\xi_{c,j} + b_{c',i}\right),$$
$$i_1 = 1,\ldots,N'_{x,1}, \quad i_2 = 1,\ldots,N'_{x,2}, \quad c' = 1,\ldots,\alpha', \tag{2.21}$$

where $(i-1)s = ((i_1-1)s_1, (i_2-1)s_2)$. As in the 1D case, the channel dimension corresponds to the rank r, and the spatial dimensions correspond to the grid points of the discretized domain. Analogously to the 1D case, we define the LC networks LCR, LCK and LCI and use them to express the six operations in (2.11) that constitute the building blocks of the neural network. The parameters N_x, s and w in the one-dimensional LC networks are replaced by their two-dimensional counterpart $N_x = (N_{x,1}, N_{x,2})$, $s = (s_1, s_2)$ and $w = (w_1, w_2)$, respectively. We point out that $s = w = \frac{N_x}{N'_x}$ for the 1D case is replaced by $s_j = w_j = \frac{N_{x,j}}{N'_{x,j}}$, $j = 1, 2$ for the 2D case in the definition of LC.

Using the notations above, we extend Algorithm 2 to the 2D case in Algorithm 3. It is crucial to note that the ReshapeT and ReshapeM functions in Algorithm 3 are *not* the usual column major-based reshaping operations. ReshapeM$[a, r, n_1, n_2]$ reshapes a 3-tensor T with size $a^2 r \times n_1 \times n_2$ to a 3-tensor with size $r \times an_1 \times an_2$ by reshaping each row $T_{\cdot,j,k}$ to a 3-tensor with size $r \times a \times a$ and joining them to a large 3-tensor. ReshapeT$[a, r, n_1, n_2]$ is the inverse of ReshapeM$[a, r, n_1, n_2]$. Figure 6 diagrams these two reshape functions.

Algorithm 3 Application of NN architecture for $\mathcal{H}^2$-matrices on a vector $v \in \mathbb{R}^{N^2}$.

$\tilde{v} =$ ReshapeT$[m, 1, 2^L, 2^L](v)$;
$\tilde{u}^{(\mathrm{ad})} =$ LCK[linear; $(2^L, 2^L), m^2, m^2, 2n_b^{(\mathrm{ad})} + 1](\tilde{v})$;
$u^{(\mathrm{ad})} =$ ReshapeM$[m, 1, 2^L, 2^L](\tilde{u}^{(\mathrm{ad})})$;
$\xi^{(L)} =$ LCR[linear; $(N, N), 1, (2^L, 2^L), r](v)$;
for ℓ from $L-1$ to 2 by -1 **do**
  $\xi^{(\ell)} =$ LCR[linear; $(2^{\ell+1}, 2^{\ell+1}), r, (2^\ell, 2^\ell), r](\xi^{(\ell+1)})$;
end for
for ℓ from 2 to L **do**
  $\zeta^{(\ell)} =$ LCK[linear; $(2^\ell, 2^\ell), r, r, 2n_b^{(\ell)} + 1](\xi^{(\ell)})$;
end for
$\chi = 0$;
for ℓ from 2 to $L-1$ **do**
  $\chi = \chi + \zeta^{(\ell)}$;
  $\chi =$ LCI[linear; $(2^\ell, 2^\ell), r, 2r](\chi)$;
  $\chi =$ ReshapeM$[2, r, 2^\ell, 2^\ell](\chi)$;
end for
$\chi = \chi + \zeta^{(L)}$;
$\chi =$ LCI[linear; $(2^L, 2^L), r, m^2](\chi)$;
$\chi =$ ReshapeM$[m, 1, 2^L, 2^L](\chi)$;
$u = \chi + u^{(\mathrm{ad})}$;

3 Multiscale neural network

The nonlinear map in the form $u = \mathcal{M}(v)$ with $u, v \in \mathbb{R}^{N^d}$, which can be viewed as a nonlinear generalization of pseudo-differential operators, is ubiquitous from integral equations and partial differential equations in practical applications. In general, to evaluate

such nonlinear maps, one needs to use iterative methods that may require a large number of iterations, and at each iteration, one may need to solve the underlying equation several times, resulting in computational expensive algorithms. Instead, we propose to bypass this endeavor by leveraging the ability of NNs to represent high-dimensional nonlinear maps. In this section, we construct a hierarchical approximation of such a nonlinear map by extending the architectures provided in Algorithms 2 and 3 to the nonlinear case. We refer to the resulting NN architecture as *multiscale neural network-*$\mathcal{H}^2$ (MNN-$\mathcal{H}^2$) due to its multiscale structure inspired by $\mathcal{H}^2$-matrices.

To simplify the notation, we focus on the 1D case in this section. The following presentation can be readily extended to the multidimensional case by following the discussion in Sect. 2.3.

3.1 Algorithm and architecture

Similar to [17], we extend Algorithm 2 to the nonlinear case by replacing the linear activation function by a nonlinear one and extend one LCK layer to $K \in \mathbb{N}$ nonlinear LCK layers. Algorithm 2 is then revised to Algorithm 4. Following [17], the last layer corresponding to the adjacent part, the layer corresponding to $(V^{(L)})^T v$ and $U^{(L)}\zeta$ are set to linear layers. In addition, the layer in line 3 of Algorithm 4 is a linear layer when $k = K$, and the activation ϕ in Algorithm 4 can be any nonlinear or linear activation function depending of the target application. Figure 7 illustrates the architecture of MNN-$\mathcal{H}^2$.

Algorithm 4 Application of MNN-$\mathcal{H}^2$ to a vector $v \in \mathbb{R}^N$.

1: $\xi_0 = \mathsf{Reshape}[m, 2^L](v)$;
2: **for** k from 1 to K do **do**
3: $\quad \xi_k = \mathsf{LCK}[\phi; 2^L, m, m, 2n_b^{(\mathrm{ad})} + 1](\xi_{k-1})$;
4: **end for**
5: $u^{(\mathrm{ad})} = \mathsf{Reshape}[1, N](\xi_K)$;
6: $\zeta_0^{(L)} = \mathsf{LCR}[\mathsf{linear}; N, 1, 2^L, r](v)$;
7: **for** ℓ from $L-1$ to 2 by -1 **do**
8: $\quad \zeta_0^{(\ell)} = \mathsf{LCR}[\phi; 2^{\ell+1}, r, 2^\ell, r](\zeta_0^{(\ell+1)})$;
9: **end for**
10: **for** ℓ from 2 to L **do**
11: $\quad$ **for** k from 1 to K **do**
12: $\quad\quad \zeta_k^{(\ell)} = \mathsf{LCK}[\phi; 2^\ell, r, r, 2n_b^{(\ell)} + 1](\zeta_{k-1}^{(\ell)})$;
13: $\quad$ **end for**
14: **end for**
15: $\chi = 0$;
16: **for** ℓ from 2 to $L-1$ **do**
17: $\quad \chi = \chi + \zeta_K^{(\ell)}$;
18: $\quad \chi = \mathsf{LCI}[\phi; 2^\ell, r, 2r](\chi)$;
19: $\quad \chi = \mathsf{Reshape}[r, 2^{\ell+1}](\chi)$;
20: **end for**
21: $\chi = \chi + \zeta_K^{(L)}$;
22: $\chi = \mathsf{LCI}[\mathsf{linear}; 2^L, r, m](\chi)$;
23: $\chi = \mathsf{Reshape}[1, N](\chi)$;
24: $u = \chi + u^{(\mathrm{ad})}$;

Similarly to the linear case, we compute the number of parameters of MNN-$\mathcal{H}^2$ to obtain

$$\begin{aligned} N_{p,LC} &= 2^L m^2 K(2n_b^{(\mathrm{ad})} + 1) + Nr + 2\sum_{\ell=2}^{L-1} 2^{\ell+1} r^2 + K\sum_{\ell=2}^{L} 2^\ell r^2 (2n_b^{(\ell)} + 1) + 2^L rm \\ &\le 2Nr + 2Nr(2 + K(2n_b + 1)) + NmK(2n_b + 1) \\ &\le 3NmK(2n_b + 3) \sim O(N). \end{aligned} \tag{3.1}$$

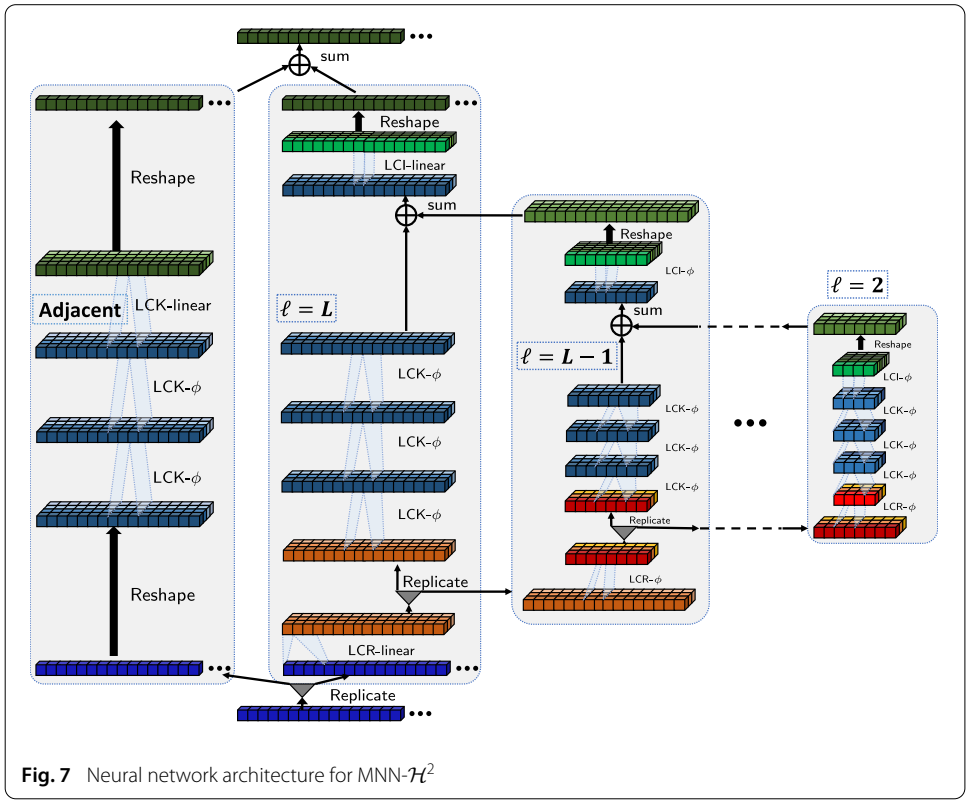


Fig. 7 Neural network architecture for MNN-$\mathcal{H}^2$

Here the number of parameters in b from (2.9) is also ignored. Compared to $\mathcal{H}$-matrices, the main saving of the arithmetic complexity of $\mathcal{H}^2$-matrices is its nested structure of $U^{(\ell)}$ and $V^{(\ell)}$. Therefore, accordingly compared to MNN-$\mathcal{H}$ in [17], the main saving on the number of parameters of MNN-$\mathcal{H}^2$ comes from the nested structure of LCR and LCI layers in Algorithm 4. For a system with large N, MNN-$\mathcal{H}^2$ has fewer parameters; thus, it can reduce the computational cost and storage cost.

3.2 Translation invariant case

For the linear system (2.1), if the kernel is of convolution type, i.e., $g(x, y) = g(x - y)$, then the matrix A is a Toeplitz matrix. As a result, the matrices $M^{(\ell)}$, $A^{(\mathrm{ad})}$, $U^{(L)}$, $V^{(L)}$, $B^{(\ell)}$ and $C^{(\ell)}$ are all block cyclic matrices. In the more general nonlinear case, the operator $\mathcal{M}$ is *translation invariant* (or more accurately translation equivariant) if

$$\mathcal{T}\mathcal{M}(v) = \mathcal{M}(\mathcal{T}v) \tag{3.2}$$

holds for any translation operator $\mathcal{T}$. This indicates that the weights $W_{c',c;i,j}$ and bias $b_{c,i}$ in (2.9) can be independent of index i. This is the case of a *convolutional neural network* (CNN):

$$\zeta_{c',i} = \phi\left(\sum_{j=(i-1)s+1}^{(i-1)s+w}\sum_{c=1}^{\alpha} W_{c',c;j}\xi_{c,j} + b_{c'}\right), \quad i = 1, \ldots, N'_x,\ c' = 1, \ldots, \alpha'. \tag{3.3}$$

Note that the difference between this and an LC network is that here W and b are independent of i. In this convolutional setting, we shall instead refer to the LC layers LCR, LCK

and LCI as CR, CK and CI, respectively. By replacing the LC layers in Algorithm 4 with the corresponding CNN layers, we obtain the neural network architecture for the translation invariant kernel. It is easy to calculate that the number of parameters of CR, CK and CI are

$$N_p^{\mathsf{CR}} = \frac{N_x}{N_x'}\alpha', \quad N_p^{\mathsf{CK}} = \alpha\alpha' w, \quad N_p^{\mathsf{CI}} = \alpha\alpha'. \tag{3.4}$$

Thus, the number of parameters in Algorithm 4 implemented by CNN is $O(\log(N))$ as shown below:

$$\begin{aligned} N_{p,\mathrm{CNN}} &= \frac{N}{2^L} r + 2\sum_{\ell=2}^{L-1} 2r^2 + K\sum_{\ell=2}^{L} r^2(2n_b^{(\ell)}+1) + rm + m^2K(2n_b^{(\mathrm{ad})}+1) \\ &\le 2mr + 4(L-3)r^2 + Kr^2(2n_b+1)(L-2) + Km^2(2n_b+1) \\ &\le m^2(4L + K(2n_b+1)(L-1)) = O(\log(N)). \end{aligned} \tag{3.5}$$

Mixed model for the non-translation invariant case Note that the number of parameters in the translation invariant case is much lower compared to the non-invariant case. In addition, the constant $3mK(2n_b+3)$ in (3.1) is usually a large number for practical applications. For example, if $m = 5$, $K = 5$, $n_b = 3$, the constant is 675. To reduce the number of parameters in MNN-$\mathcal{H}^2$, we propose a mixed model to replace some of the LC layers by CNN layers even in the non-translation invariant setting. For example, in one of the numerical applications in Sect. 4, we use LC layers for the LCR and LCI layers and for the last layer of the adjacent part, while using CK for the remaining layers. We will verify the effectiveness of this heuristic mixed model in Sect. 4.2.

4 Applications

In this section, we study the performance of the MNN-$\mathcal{H}^2$ structure using three examples: the nonlinear Schrödinger equation (NLSE) in Sect. 4.1, the steady-state radiative transfer equation (RTE) in Sect. 4.2 and the Kohn–Sham map in Sect. 4.3.

The MNN-$\mathcal{H}^2$ structure was implemented in Keras [13], a high-level neural network application programming interface (API) running on top of TensorFlow [1], which is an open-source software library for high-performance numerical computation. The loss function is chosen as the mean squared error. The optimization is performed using the Nadam optimizer [56]. The weights in MNN-$\mathcal{H}^2$ are initialized randomly from the normal distribution and the batch size is always set between 1/100th and 1/50th of the number of training samples. As discussed in Sect. 3.2, if the operator $\mathcal{M}$ is translation invariant, all the layers are implemented using CNN layers; otherwise, we use LC layers or a mixture of LC and CNN layers.

In all the tests, the band size is chosen as $n_{b,\mathrm{ad}} = 1$ and $n_b^{(\ell)}$ is 2 for $\ell = 2$ and 3 otherwise. The activation function in LCR and LCI is chosen to be linear, while ReLU is used in LCK. All the tests are run on GPU with data type `float32`. The selection of parameters r (number of channels), L ($N = 2^L m$) and K (number of layers in Algorithm 4) is problem dependent.

The training and test errors are measured by the relative error with respect to ℓ^2 norm

$$\epsilon = \frac{||u - u_{NN}||_{\ell^2}}{||u||_{\ell^2}}, \tag{4.1}$$

where u is the target solution generated by numerical discretization of PDEs and u_{NN} is the prediction solution by the neural network. We denote by ϵ_{train} and ϵ_{test} the average training error and average test error within a given set of samples, respectively. Similarly, we denote by σ_{train} and σ_{test} the estimated standard deviation of the training and test errors within the given set of samples. The numerical results presented in this section are obtained by repeating the training a few times, using different random seeds.

4.1 NLSE with inhomogeneous background potential

The nonlinear Schrödinger equation (NLSE) is a widely used model in quantum physics to study phenomenon such as the Bose–Einstein condensation [2,42]. It has been studied in [17] using the MNN-$\mathcal{H}$ structure. In this work, we use the same example to compare the results from MNN-$\mathcal{H}^2$ with those from MNN-$\mathcal{H}$. Here we study the NLSE with inhomogeneous background potential $V(x)$

$$\begin{aligned} &-\Delta u(x) + V(x)u(x) + \beta u(x)^3 = Eu(x), \quad x \in [0,1]^d, \\ &\text{s.t.} \int_{[0,1]^d} u(x)^2 \,\mathrm{d}x = 1, \text{ and } \int_{[0,1]^d} u(x)\,\mathrm{d}x > 0, \end{aligned} \tag{4.2}$$

with periodic boundary conditions, to find its ground state $u_G(x)$. We consider a defocusing cubic Schrödinger equation with a strong nonlinear term $\beta = 10$. The normalized gradient flow method in [5] is employed for the numerical solution of NLSE.

In this work, we use neural networks to learn the map from the background potential to the ground state

$$V(x) \to u_G(x). \tag{4.3}$$

Clearly, this map is translation invariant, and thus, MNN-$\mathcal{H}^2$ is implemented using CNN rather than LC network. In the following, we study MNN-$\mathcal{H}^2$ on 1D and 2D cases, respectively.

In order to compare with MNN-$\mathcal{H}$ in [17], we choose the same potential V as in [17]

$$V(x) = -\sum_{i=1}^{n_g} \sum_{j_1,\ldots,j_d=-\infty}^{\infty} \frac{\rho^{(i)}}{(2\pi T)^{d/2}} \exp\left(-\frac{|x - j - c^{(i)}|^2}{2T}\right), \tag{4.4}$$

where the periodic summation imposes periodicity on the potential and the parameters $\rho^{(i)} \sim \mathcal{U}(1,4)$, $c^{(i)} \sim \mathcal{U}(0,1)^d$, $i = 1,\ldots,n_g$ and $T \sim \mathcal{U}(2,4) \times 10^{-3}$.

4.1.1 One-dimensional case

For the one-dimensional case, we choose the number of discretization points $N = 320$ and set $L = 7$ and $m = 5$. The numerical experiments performed in this section use the same datasets as those in [17]. In that context, we study how the performance of MNN-$\mathcal{H}^2$ depends on the number of training samples $N^{\text{train}}_{\text{samples}}$ (Fig. 8), the number of channels r (Fig. 9a), the number of CK layers K (Fig. 9b) and the number of Gaussians n_g (Fig. 9c).

Figure 8 shows that MNN-$\mathcal{H}^2$ can achieve small training error with as few as 200 training samples, which is much smaller than the number of parameters used in the example ($N_{\text{params}} = 7209$). To see why this is possible, let us consider first the linear system $u = Av$ with $A \in \mathbb{R}^{N\times N}$. In order to determine the matrix A using matrix–vector

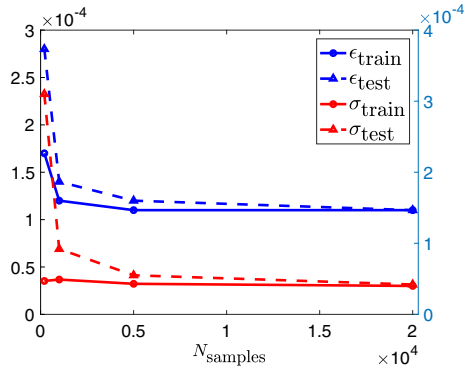


Fig. 8 Mean (ϵ with respect to the left y-axis) and standard deviation (σ with respect to the right y-axis) of the relative error in approximating the ground state of NLSE for different number of samples $N_{\text{samples}}^{\text{train}}$ for 1D case with $r = 6$, $K = 5$ and $n_g = 2$. In this case, $N_{\text{params}} = 7209$

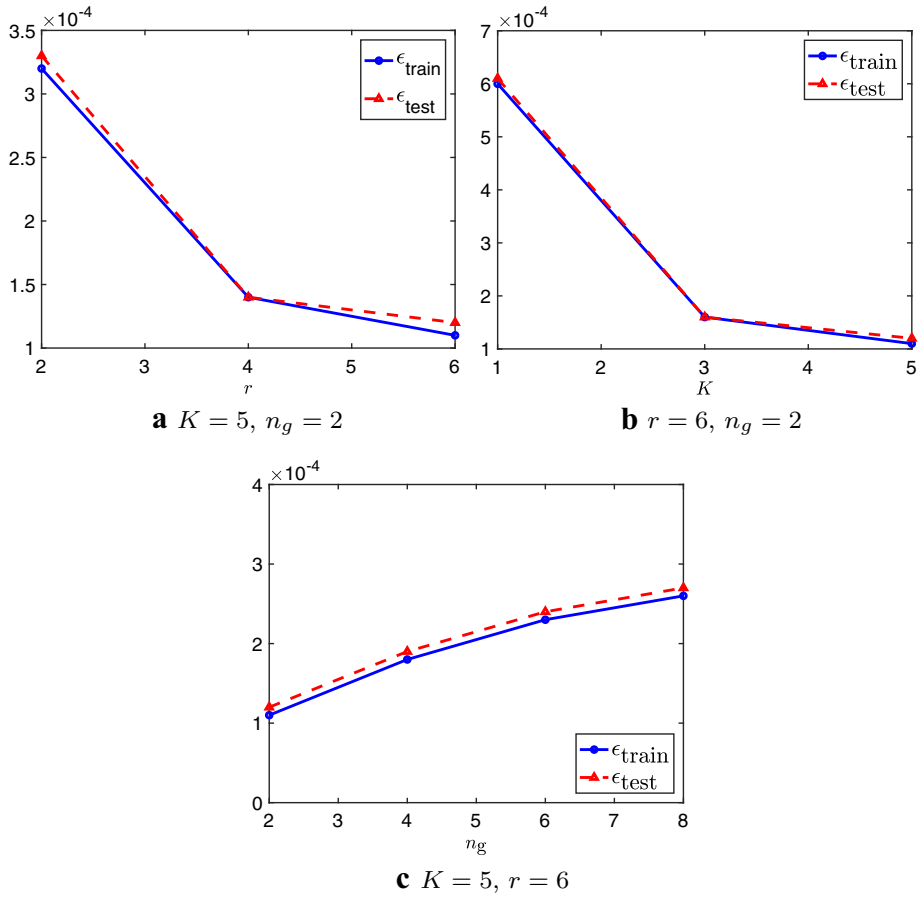


Fig. 9 Relative error in approximating the ground state of 1D NLSE for different number of channels r, different number of **CK** layers K and different number of Gaussians n_g with $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 5000$

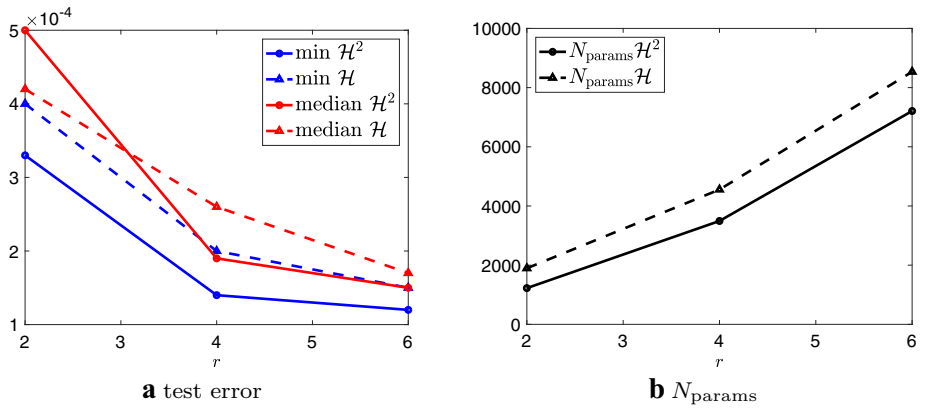


Fig. 10 Numerical results of MNN-$\mathcal{H}$/MNN-$\mathcal{H}^2$ for the minimum and median ϵ_{train} for 1D NLSE with random initial seed. The "min" and "median" stand for the test error corresponding to the minimum and median training data cases, respectively, and H and H^2 stand for MNN-$\mathcal{H}$ and MNN-$\mathcal{H}^2$, respectively. The setup of MNN-$\mathcal{H}^2$ is $K = 5$, $n_g = 2$ and $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 5000$

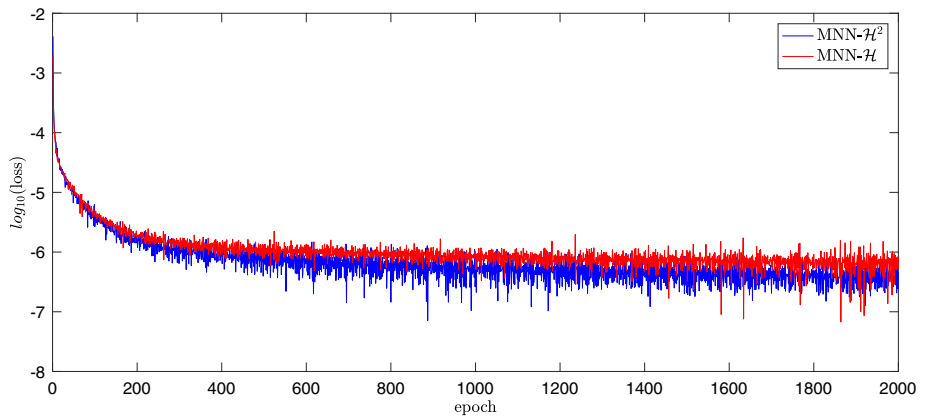


Fig. 11 Behavior of the loss function of MNN-$\mathcal{H}^2$ and MNN-$\mathcal{H}$ with $r = 6$ and $K = 5$ in the training procedure

products, we need at most $O(N)$ independent samples of the form (u, v). Furthermore, if A is an $\mathcal{H}$-matrix (resp. $\mathcal{H}^2$-matrix), the number of parameters in A is reduced to $O(N \log N)$ (resp. $O(N)$). Hence, only $O(\log(N))$ (resp. $O(1)$) samples of the form (u, v) are sufficient to determine A [20,21,36]. We expect that similar results can be generalized to the MNN-$\mathcal{H}^2$ network, i.e., the number of samples of the form (u, v) should also be proportional to the ratio of the number of degrees of freedom in the network and N. For instance, the neural network used in Fig. 8, $\frac{N_{\text{params}}}{N} = \frac{7209}{320} \approx 22.5$, which is much smaller than the number of training samples used in the simulation.

For the case $N_{\text{samples}}^{\text{train}} = 200$, the test error is slightly larger than the training error, and the standard deviation within the set of test samples σ_{test} is relatively large. As $N_{\text{samples}}^{\text{train}}$ increases to 1000, the test error is reduced by a factor of 2, and σ_{test} is reduced by a factor of 3. When $N_{\text{samples}}^{\text{train}}$ increases to 5000 and 20,000, the test error remains nearly unchanged, while σ_{test} continues to decrease. For the nonlinear map $u = \mathcal{M}(v)$, $v \in \Omega \subset \mathbb{R}^N$, a large

number of samples are required to obtain an accurate approximation. Furthermore, we do not observe overfitting in Fig. 8.

Figure 9 presents the numerical results for different choices of channels r, CK layers K and Gaussians n_g. As r or K increases, Fig. 9a, b show that the error decreases and then stagnates. The choice of $r = 6$ and $K = 5$ is used for the 1D NLSE below as a balance of efficiency and accuracy. In Fig. 9c, we find that increasing the number of wells and hence the complexity of the input field only leads to marginal increase in the training and test errors.

As demonstrated in the complexity analysis earlier, because of the hierarchical nested bases used in the restriction and interpolation layers, MNN-$\mathcal{H}^2$ should use a fewer number of parameters than MNN-$\mathcal{H}$ for the same parameter setup, which is shown in Fig. 10b.

Figure 10a compares MNN-$\mathcal{H}^2$ and MNN-$\mathcal{H}$ in terms of the minimum error and median error of the networks by performing the training procedure for a few times with different random seeds. These results are reported for different number of channels r ranging from 2 to 6. The setup of the training is the same for both MNN-$\mathcal{H}$ and MNN-$\mathcal{H}^2$. The learning rate is 10^{-3}, the number of epochs is 6000, $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 5000$, and the batch size is 50. We find that the errors of both networks are comparable for all values of r, in terms of both the minimum and the median. Thus, the reduction in the number of parameters in MNN-$\mathcal{H}^2$ does not sacrifice accuracy as compared with MNN-$\mathcal{H}$. Concerning the training and the test procedures, the training and test times of MNN-$\mathcal{H}^2$ are a bit smaller than those of MNN-$\mathcal{H}$. For example, for the setup with $r = 6$ and $K = 5$, the training time for MNN-$\mathcal{H}^2$ on a GPU Tesla P100 is 4.5 h, while that for MNN-$\mathcal{H}$ with same setup is 5.5 h. The prediction time on 10,000 samples is 0.26 s for MNN-$\mathcal{H}^2$ compared to 0.29 s for MNN-$\mathcal{H}$. The behavior of the loss function for MNN-$\mathcal{H}^2$ and MNN-$\mathcal{H}$ during training is depicted in Fig. 11. Here we only present the results for the first 2000 epochs because the loss function hardly decreases in the remaining epochs. One can see that the loss functions for both networks exhibit similar behavior and the loss of the MNN-$\mathcal{H}^2$ is relatively smaller. In comparison with MNN-$\mathcal{H}$, MNN-$\mathcal{H}^2$ has fewer parameters, trains faster and yields smaller prediction time. This behavior is also consistently observed in other examples in this section.

4.1.2 Two-dimensional case

For the two-dimensional example, we choose the number of discretization N in each dimension to be 80 and set $L = 4, m = 5$. The datasets in [17] were used for the 2D experiments. We study the behavior of MNN for: different number of channels, r (see Fig. 12a for the best results and Fig. 13 for the median error); different number of CK layers, K (Fig. 12b); and different number of Gaussians, n_g (Fig. 12c).

Due to the increase in the number of parameters in the 2D networks, we set $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 20{,}000$. From Figs. 12 and 13, we arrive at similar conclusions as the 1D case: (a) no overfitting is observed for all the tests; (b) the error first decreases and then stagnates as r or K increases; (c) MNN-$\mathcal{H}^2$ is not sensitive to the complexity of the input; and (d) MNN-$\mathcal{H}^2$ uses fewer number of parameters and obtains a comparable error as MNN-$\mathcal{H}$.

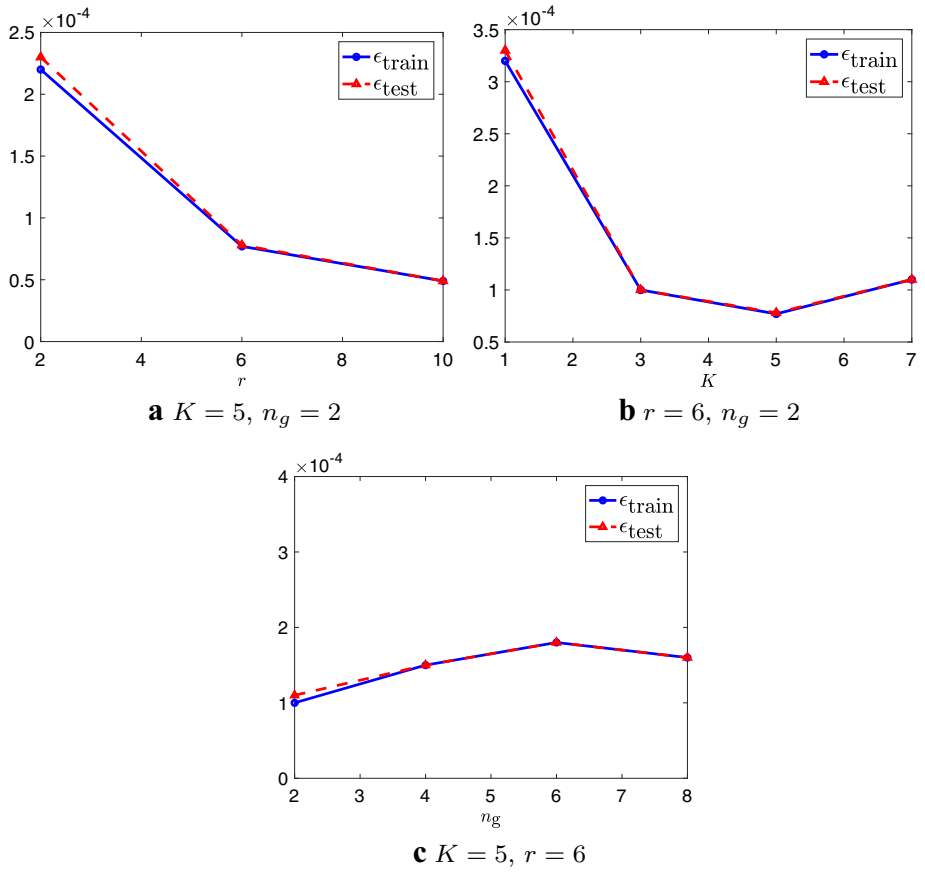


Fig. 12 Relative error in approximating the ground state of 2D NLSE for different number of channels r, different number of CK layers and different number of Gaussians n_g for the 2D case with $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 20{,}000$

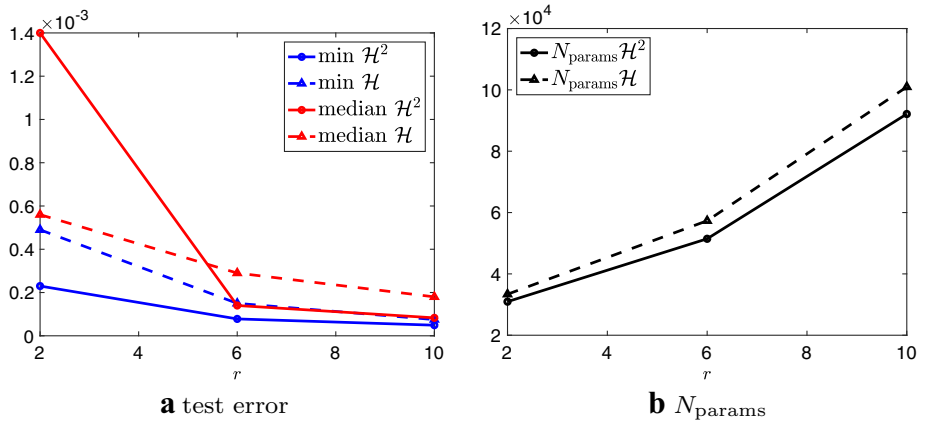


Fig. 13 Numerical results of MNN-$\mathcal{H}$/MNN-$\mathcal{H}^2$ for the minimum and median ϵ_{train} for 1D NLSE with random initial seed. The "min" and "median" stand for the test error corresponding to the minimum and median training data cases, respectively, and H and H^2 stand for MNN-$\mathcal{H}$ and MNN-$\mathcal{H}^2$, respectively. The setup of MNN-$\mathcal{H}^2$ is $K = 5$, $n_g = 2$ and $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 20{,}000$

4.2 Radiative transfer equation

Radiative transport equation (RTE) is the widely used tool for describing particle propagation in many different fields, such as neutron transport in reactor physics [43], light transport in atmospheric radiative transfer [39], heat transfer [30] and optical imaging [29]. Here we consider the steady-state RTE in the homogeneous scattering regime

$$\begin{aligned} v \cdot \nabla_x \varphi(x,v) + \mu_t(x)\varphi(x,v) &= \mu_s(x)u(x) + f(x), \quad \text{in } \Omega \times \mathbb{S}^{d-1}, \quad \Omega \subset \mathbb{R}^d, \\ \varphi(x,v) &= 0, \quad \text{on } \{(x,v) \in \partial\Omega \times \mathbb{S}^{d-1} : n(x) \cdot v < 0\}, \\ u(x) &= \frac{1}{4\pi} \int_{\mathbb{S}^{d-1}} \varphi(x,v)\, \mathrm{d}v, \end{aligned} \tag{4.5}$$

where d is the dimension, $\varphi(x,v)$ denotes the photon flux that depends on both space x and angle v, $f(x)$ is the light source, $\mu_s(x)$ is the scattering coefficient and $\mu_t(x)$ is the total absorption coefficient. In most applications, one can assume that $\mu_t(x)$ is equal to $\mu_s(x)$ plus a constant background. The mean density $u(x)$ is uniquely determined by μ_s, μ_t and f [16]. In this homogeneous regime, by eliminating $\varphi(x,v)$ from the equation and keeping only $u(x)$ as unknown, one can rewrite RTE as an integral equation

$$u = (\mathcal{I} - \mathcal{K}\mu_s)^{-1}\mathcal{K}f, \tag{4.6}$$

with the operator $\mathcal{K}$ defined as

$$\mathcal{K}f = \int_{y\in\Omega} K(x,y)f(y)\,\mathrm{d}y, \quad K(x,y) = \frac{\exp\left(-|x-y|\int_0^1 \mu_t(x - s(x-y))\,\mathrm{d}s\right)}{4\pi|x-y|^{d-1}}. \tag{4.7}$$

In practical applications such as inverse problems, either (4.5) or (4.6) is often solved repetitively, which can be quite expensive even if the fast algorithms, for example, in [16,45] are used. Here, we use MNN-$\mathcal{H}^2$ to learn the map

$$\mu_s(x) \to u(x) \tag{4.8}$$

from the scattering coefficient μ_s to the mean density $u(x)$.

4.2.1 One-dimensional slab geometry case

We first study the one-dimensional slab geometry case for $d = 3$, i.e., the parameters are homogeneous on the direction x_2 and x_3. With slight abuse of notations, we denote x_1 by x in this subsection. Then, (4.6) turns to

$$u(x) = (\mathcal{I} - \mathcal{K}_1\mu_s)^{-1}\mathcal{K}_1 f(x), \tag{4.9}$$

where the operator $\mathcal{K}_1$ is defined as

$$\begin{aligned} \mathcal{K}_1 f(x) &= \int_{y\in\Omega} K_1(x,y)f(x)\,\mathrm{d}y, \\ K_1(x,y) &= \frac{1}{2}\mathrm{Ei}\left(-|x-y|\int_0^1 \mu_t(x - s(x-y))\,\mathrm{d}s\right), \end{aligned} \tag{4.10}$$

and $\mathrm{Ei}(\cdot)$ is the exponential integral.

Here we set $f(x) = 1$ and $\mu_a(x) = \mu_t(x) - \mu_s(x) = 0.2$, $x \in \Omega$, and the scattering coefficient has the form

$$\mu_s(x) = \sum_{i=1}^{n_g} \frac{\rho^{(i)}}{\sqrt{2\pi T}} \exp\left(-\frac{|x - c^{(i)}|^2}{2T}\right), \tag{4.11}$$

where the parameters $\rho^{(i)} \sim \mathcal{U}(0.1, 0.3)$, $c^{(i)} \sim \mathcal{U}(0.2, 0.8)$, $i = 1, \ldots, n_g$, and $T \sim \mathcal{U}(2, 4) \times 10^{-3}$. The numerical samples are generated by solving (4.9).

Because the map $\mu_s \to u$ is not translation invariant, MNN-$\mathcal{H}^2$ cannot be implemented using CNNs as before. As discussed at the end of Sect. 3.2, we can combine LC layers and CNN layers together to reduce the number of parameters. The resulting neural network is denoted by MNN-$\mathcal{H}^2$-Mix. As a reference, we implement MNN-$\mathcal{H}^2$ by LC network and it is denoted by MNN-$\mathcal{H}^2$-LC. Note that since both μ_s and u are not periodic the periodic padding in LCK/CK should be replaced by zero padding.

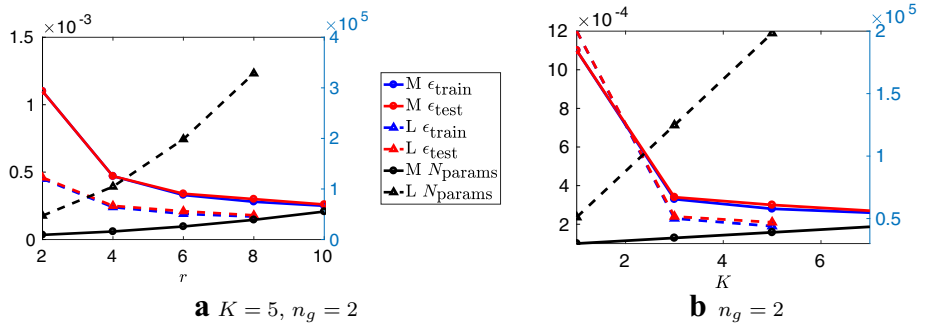
The number of discretization points is $N = 320$, and $L = 6$ and $m = 5$. We perform numerical experiments to study the numerical behavior for different number of channels (Fig. 14a) and different number of CK/LCK layers K (Fig. 14b). For both MNN-$\mathcal{H}^2$-Mix and MNN-$\mathcal{H}^2$-LC, as r or K increase, the errors first decrease and then stagnate. We use $r = 8$ and $K = 5$ for MNN-$\mathcal{H}^2$-Mix in the following. For the same setup, the error of MNN-$\mathcal{H}^2$-LC is somewhat smaller and the number of parameters is quite larger than that of MNN-$\mathcal{H}^2$-Mix. Thus, MNN-$\mathcal{H}^2$-Mix serves as a good balance between the number of parameters and the accuracy.

Figure 15 summarizes the results of MNN-$\mathcal{H}^2$-Mix for different n_g with $K = 5$ and $r = 8$. Numerical results show that MNN-$\mathcal{H}^2$-Mix is not sensitive to the complexity of the input.

4.2.2 Two-dimensional case

Here we set $f(x) = 1$ and $\mu_a(x) = \mu_t(x) - \mu_s(x) = 0.2$ for $x \in \Omega$. The scattering coefficient takes the form

$$\mu_s(x) = \sum_{i=1}^{2} \frac{\rho^{(i)}}{2\pi T} \exp\left(-\frac{|x - c^{(i)}|^2}{2T}\right), \tag{4.12}$$



a $K = 5$, $n_g = 2$ **b** $n_g = 2$

Fig. 14 Relative error in approximating the density of RTE for 1D case for different number of channels r and different number of CK/LCK layers K with $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 20{,}000$. "M" and "L" stand for MNN-$\mathcal{H}^2$-Mix and MNN-$\mathcal{H}^2$-LC, respectively. b The number of channel r is 8 for MNN-$\mathcal{H}^2$-Mix and is 6 for MNN-$\mathcal{H}^2$-LC

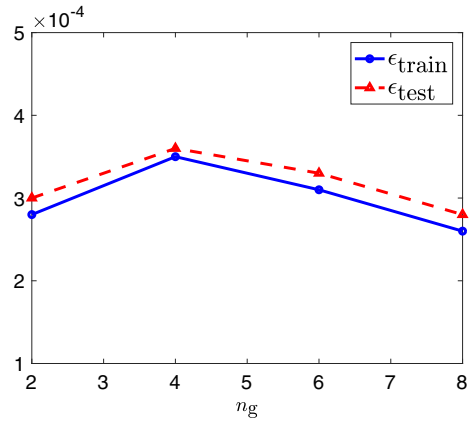


Fig. 15 Relative error in approximating the density of RTE for 1D case for different number of Gaussians n_g for MNN-$\mathcal{H}^2$-Mix with $K = 5$, $r = 8$ and $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 20{,}000$

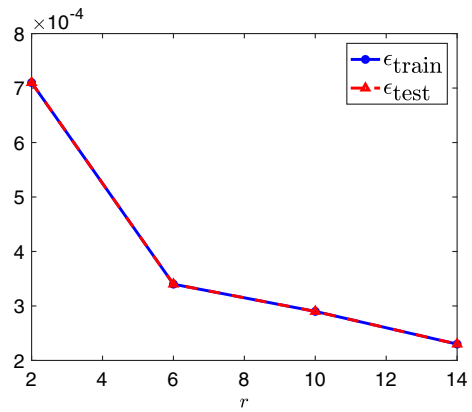


Fig. 16 Relative error in approximating the density of RTE for the 2D case for different number of channels for MNN-$\mathcal{H}^2$-Mix with $K = 5$ and $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 20{,}000$

where $x = (x_1, x_2)$ and the parameters $\rho^{(i)} \sim \mathcal{U}(0.01, 0.03)$, $c^{(i)} \sim \mathcal{U}(0.2, 0.8)^2$, $i = 1, 2$, and $T \sim \mathcal{U}(2, 4) \times 10^{-3}$. The numerical samples are generated by solving (4.6).

Because the map $\mu_s \to u$ is not translation invariant, we implement the MNN-$\mathcal{H}^2$-Mix architecture as the 1D case. Considering that the adjacent part takes a large number of parameters for the 2D case, we implement the adjacent part by the CK layers. Figure 16 gathers the results for different number of channels r. Note that, similar to the 1D case, there is no overfitting for all the tests and the relative error decreases as r increases.

4.3 Kohn–Sham map

In the Kohn–Sham density functional theory [25,31], one needs to solve the following nonlinear eigenvalue equations (spin degeneracy omitted):

$$\begin{aligned} &\left(-\frac{1}{2}\Delta + V(x)\right)\psi_i(x) = \varepsilon_i\psi_i(x), \ \ x \in \Omega = [-1, 1)^d \\ &\int_\Omega \psi_i(x)\psi_j(x)dx = \delta_{ij}, \quad \rho(x) = \sum_{i=1}^{n_e} |\psi_i(x)|^2, \end{aligned} \tag{4.13}$$

where n_e is the number of electrons, d is the spatial dimension and δ_{ij} stands for the Kronecker delta. All eigenvalues $\{\varepsilon_i\}$ are real and ordered non-decreasingly. The electron density $\rho(x)$ satisfies the constraint

$$\rho(x) \geq 0, \quad \int_\Omega \rho(x)\,\mathrm{d}x = n_e. \tag{4.14}$$

In this subsection, we employ the multiscale neural networks to approximate the Kohn–Sham map

$$\mathcal{F}_{\mathrm{KS}} : V \to \rho. \tag{4.15}$$

The potential function V is given by

$$V(x) = -\sum_{i=1}^{n_e} \sum_{j\in\mathbb{Z}^d} \rho^{(i)} \exp\left(-\frac{(x - c^{(i)} - 2j)^2}{2\sigma^2}\right), \qquad x \in [-1, 1)^d, \tag{4.16}$$

where $c^{(i)} \in [-1, 1)^d$ and $\rho^{(i)} \in \mathcal{U}(0.8, 1.2)$. We set $\sigma = 0.05$ for 1D and $\sigma = 0.2$ for the 2D case. The centers of the Gaussian wells $c^{(i)}$ are chosen randomly under the constraint that $|c^{(i)} - c^{(j)}| > 2\sigma$. The Kohn–Sham map is discretized using a pseudo-spectral method [57] and solved by a standard eigensolver.

4.3.1 One-dimensional case

For the one-dimensional case, we choose $N = 320$, $L = 7$ and $m = 5$, and use the same datasets as in [17] to study the numerical behavior of MNN-$\mathcal{H}^2$ for different n_e, r and K (Fig. 17).

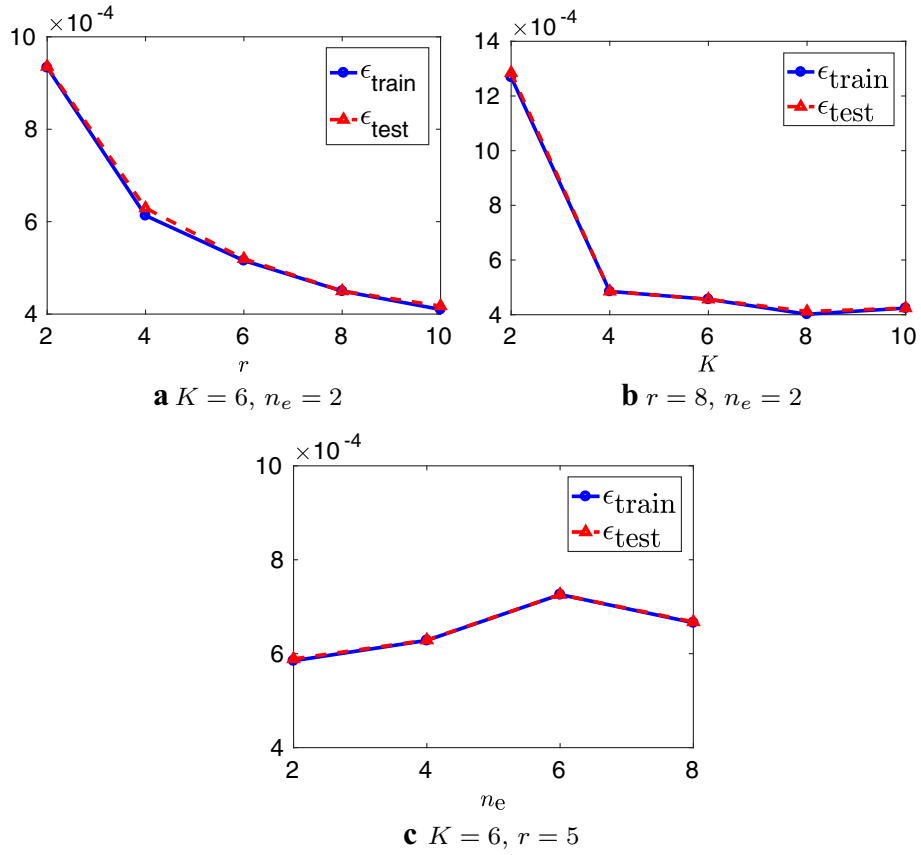
From Fig. 18, we observe that both architectures, MNN-$\mathcal{H}^2$ and MNN-$\mathcal{H}$, provide comparable results even as the MNN-$\mathcal{H}^2$ has fewer parameters to fit. Both architectures show the same trends. As the number of channels, r, increases, the error decreases sharply and then stagnates rapidly as shown in Fig. 17a. On the other hand, as the number of layers, K, increases, the error decreases sharply and then stagnates as K becomes large as shown in Fig. 17b. Finally, Fig. 17c shows that the accuracy of MNN-$\mathcal{H}^2$ is relatively insensitive to the number of wells. In addition, as shown before, we do not observe overfitting for this example.

4.3.2 Two-dimensional case

The discretization is the standard extension to 2D using tensor products, using a 64×64 grid. We consider $n_e = 2$ and follow the same number of training and test samples as that in the 1D case. We fixed $K = 6$, $L = 4$ and $m = 4$, and we trained both networks for different number of channels, r. The results are displayed in Fig. 19, which shows the same behavior as for the 1D case, comparable errors for both architectures with the error decreasing as r increases, with virtually no overfitting.

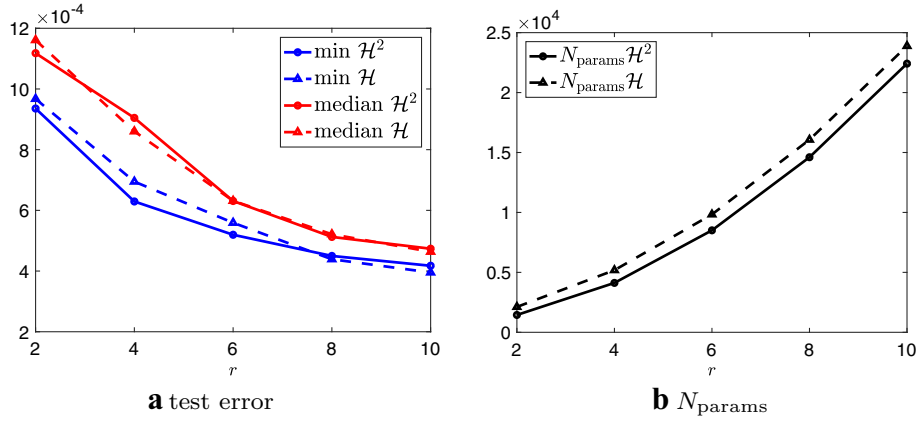
5 Conclusion

In this paper, motivated by the fast multipole method (FMM) and $\mathcal{H}^2$-matrices, we developed a multiscale neural network architecture (MNN-$\mathcal{H}^2$) to approximate nonlinear maps arising from integral equations and partial differential equations. Using the framework of



a $K = 6,\ n_e = 2$ **b** $r = 8,\ n_e = 2$ **c** $K = 6,\ r = 5$

Fig. 17 Relative error on the approximation of the Kohn–Sham map for different r, K and n_e $N_{\text{samples}}^{\text{train}} = 16{,}000$ and $N_{\text{samples}}^{\text{test}} = 4000$



a test error **b** N_{params}

Fig. 18 Numerical results of MNN-$\mathcal{H}$/MNN-$\mathcal{H}^2$ for the minimum and median ϵ_{train} for 1D Kohn–Sham map with random initial seed. The "min" and "median" stand for the test error corresponding to the minimum and median training data cases, respectively, and H and H^2 stand for MNN-$\mathcal{H}$ and MNN-$\mathcal{H}^2$, respectively. The setup of MNN-$\mathcal{H}^2$ is $K = 5$, $n_e = 2$ and $N_{\text{samples}}^{\text{train}} = N_{\text{samples}}^{\text{test}} = 5000$

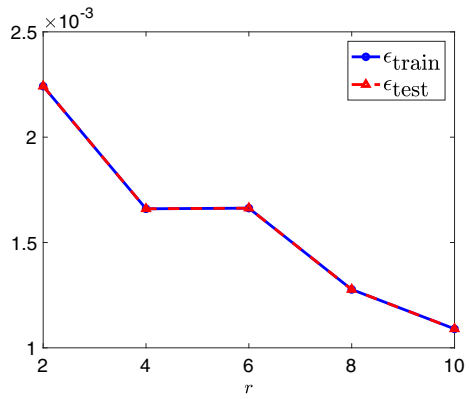


Fig. 19 Relative test error on the approximation of the 2D Kohn–Sham map for different number of channels r and $N_{\text{samples}}^{\text{train}} = 16{,}000$

neural networks, MNN-$\mathcal{H}^2$ naturally generalizes $\mathcal{H}^2$-matrices to the nonlinear setting. Compared to the multiscale neural network based on hierarchical matrices (MNN-$\mathcal{H}$), the distinguishing feature of MNN-$\mathcal{H}^2$ is that the interpolation and restriction layers are represented using a set of nested layers, which reduces the computational and storage cost for large systems. Numerical results indicate that MNN-$\mathcal{H}^2$ can effectively approximate complex nonlinear maps arising from the nonlinear Schrödinger equation, the steady-state radiative transfer equation and the Kohn–Sham density functional theory. The MNN-$\mathcal{H}^2$ architecture can be naturally extended. For instance, the LCR and LCI networks can involve nonlinear activation functions and can be extended to networks with more than one layer. The LCK network can also be altered to other network structures, such as the sum of two parallel subnetworks or the ResNet architecture [23].

Author details
[1]Department of Mathematics, Stanford University, Stanford, CA 94305, USA, [2]Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305, USA, [3]Department of Mathematics, University of California, Berkeley, Berkeley, CA, USA, [4]Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA.

Acknowlegements
The work of Y.F. and L.Y. is partially supported by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program, the National Science Foundation under Award DMS-1818449 and the GCP Research Credits Program from Google. The work of J.F. is partially supported by "la Caixa" Fellowship, sponsored by the "la Caixa" Banking Foundation of Spain. The work of L.L and L.Z. is partially supported by the Department of Energy under Grant No. DE-SC0017867 and the CAMERA Project.

Appendix: Comparing MNN-$\mathcal{H}^2$ with CNN

In this appendix, by comparing MNN-$\mathcal{H}^2$ with the classical convolutional neural networks (CNN), we show that multiscale neural networks not only reduce the number of parameters, but also improve the accuracy. Since the RTE example is not translation invariant, we perform the comparison using NLSE and Kohn–Sham map.

NLSE with inhomogeneous background potential Here we study the one-dimensional NLSE using the setup from Sect. 4.1.1 for different number of Gaussians in the potential V (4.2). The training and test errors for MNN-$\mathcal{H}^2$ and CNN are presented in Fig. 20. The channel number, layer number and window size of CNN are optimally tuned based on the

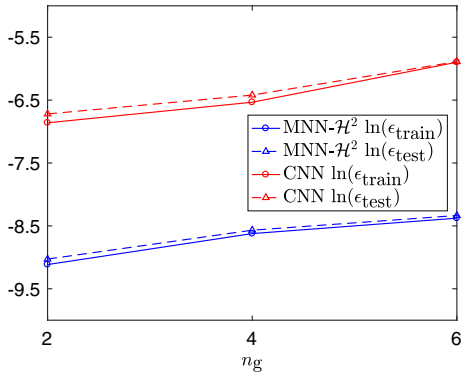


Fig. 20 Training and test errors of MNN-$\mathcal{H}^2$ with 7209 parameters ($r = 6$ and $K = 5$) and CNN with 38,161 parameters (15 layers, 10 channels and window size to be 25) for the one-dimensional NLSE

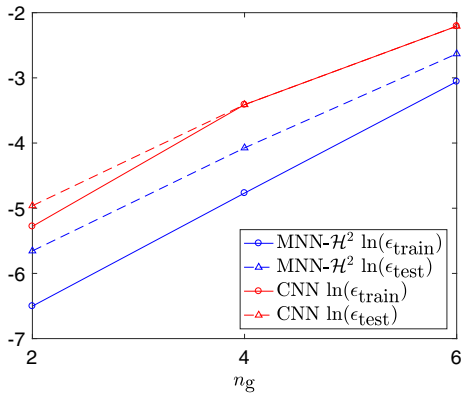


Fig. 21 Training and test errors of MNN-$\mathcal{H}^2$ with 18,985 parameters ($r = 10$ and $K = 5$) and CNN with 25,999 parameters (10 layers, 10 channels and window size to be 13) for the one-dimensional Kohn–Sham map

training error. The figure demonstrates that MNN-$\mathcal{H}^2$ has fewer parameters and gives a better approximation to the NLSE.

Kohn–Sham map For the Kohn–Sham map, we consider the one-dimensional setting in (4.16) with varying number of Gaussian wells. The width of the Gaussian well is set to be 6. In this case, the average size of the band gap is 0.01, and the electron density at point x can depend sensitively on the value of the potential at a point y that is far away. Figure 21 presents the training and test errors of MNN-$\mathcal{H}^2$ and CNN, where MNN-$\mathcal{H}^2$ outperforms a regular CNN with a comparable number of parameters.

Publisher's Note

Received: 5 August 2018 Accepted: 26 February 2019 Published online: 7 March 2019

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI, vol. 16, pp. 265–283. USENIX Association (2016)

2. Anglin, J.R., Ketterle, W.: Bose–Einstein condensation of atomic gases. Nature **416**(6877), 211 (2002)
3. Araya-Polo, M., Jennings, J., Adler, A., Dahlke, T.: Deep-learning tomography. Lead. Edge **37**(1), 58–66 (2018)
4. Badrinarayanan, V., Kendall, A., Cipolla, R.: SegNet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **39**, 2481–2495 (2017)
5. Bao, W., Du, Q.: Computing the ground state solution of Bose–Einstein condensates by a normalized gradient flow. SIAM J. Sci. Comput. **25**(5), 1674–1697 (2004)
6. Beck, C., E, W., Jentzen, A.: Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. arXiv:1709.05963 (2017)
7. Berg, J., Nyström, K.: A unified deep artificial neural network approach to partial differential equations in complex geometries. arXiv:1711.06464 (2017)
8. Börm, S., Grasedyck, L., Hackbusch, W.: Introduction to hierarchical matrices with applications. Eng. Anal. Bound. Elem. **27**(5), 405–422 (2003)
9. Bruna, J., Mallat, S.: Invariant scattering convolution networks. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1872–1886 (2013)
10. Chan, S., Elsheikh, A.H.: A machine learning approach for efficient uncertainty quantification using multiscale methods. J. Comput. Phys. **354**, 493–511 (2018)
11. Chaudhari, P., Oberman, A., Osher, S., Soatto, S., Carlier, G.: Partial differential equations for training deep neural networks. In: 2017 51st Asilomar Conference on Signals, Systems, and Computers, pp. 1627–1631 (2017)
12. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. IEEE Trans. Pattern Anal. Mach. Intell. **40**(4), 834–848 (2018)
13. Chollet, F., et al.: Keras. https://keras.io (2015). Accessed April 30, 2018
14. Cohen, N., Sharir, O., Shashua, A.: On the expressive power of deep learning: a tensor analysis. arXiv:1509.05009 (2018)
15. E, W., Han, J., Jentzen, A.: Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun. Math. Stat. **5**(4), 349–380 (2017)
16. Fan, Y., An, J., Ying, L.: Fast algorithms for integral formulations of steady-state radiative transfer equation. J. Comput. Phys. **380**, 191–211 (2019)
17. Fan, Y., Lin, L., Ying, L., Zepeda-Núñez, L.: A multiscale neural network based on hierarchical matrices. arXiv:1807.01883 (2018)
18. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
19. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comput. Phys. **73**(2), 325–348 (1987)
20. Hackbusch, W.: A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: introduction to $\mathcal{H}$-matrices. Computing **62**(2), 89–108 (1999)
21. Hackbusch, W., Khoromskij, B.N.: A sparse $\mathcal{H}$-matrix arithmetic: general complexity estimates. J. Comput. Appl. Math. **125**(1–2), 479–501 (2000)
22. Hackbusch, W., Khoromskij, B.N., Sauter, S.: On $\mathcal{H}^2$-Matrices. Lectures on Applied Mathematics. Springer, Berlin (2000)
23. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
24. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, Ar, Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
25. Hohenberg, P., Kohn, W.: Inhomogeneous electron gas. Phys. Rev. **136**(3B), B864 (1964)
26. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**(2), 251–257 (1991)
27. Khoo, Y., Lu, J., Ying, L.: Solving parametric PDE problems with artificial neural networks. arXiv:1707.03351 (2017)
28. Khrulkov, V., Novikov, A., Oseledets, I.: Expressive power of recurrent neural networks. arXiv:1711.00811 (2018)
29. Klose, A.D., Netz, U., Beuthan, J., Hielscher, A.H.: Optical tomography using the time-independent equation of radiative transfer—part 1: forward model. J. Quant. Spectrosc. Radiat. Transf. **72**(5), 691–713 (2002)
30. Koch, R., Becker, R.: Evaluation of quadrature schemes for the discrete ordinates method. J. Quant. Spectrosc. Radiat. Transf. **84**(4), 423–435 (2004)
31. Kohn, W., Sham, L.J.: Self-consistent equations including exchange and correlation effects. Phys. Rev. **140**(4A), A1133 (1965)
32. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 1, NIPS'12, pp. 1097–1105, USA, Curran Associates Inc (2012)
33. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
34. Leung, M.K.K., Xiong, H.Y., Lee, L.J., Frey, B.J.: Deep learning of the tissue-regulated splicing code. Bioinformatics **30**(12), i121–i129 (2014)
35. Li, Y., Cheng, X., Lu, J.: Butterfly-Net: Optimal function representation based on convolutional neural networks. arXiv:1805.07451 (2018)
36. Lin, L., Lu, J., Ying, L.: Fast construction of hierarchical matrix representation from matrix–vector multiplication. J. Comput. Phys. **230**(10), 4071–4087 (2011)
37. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., van der Laak, J.A.W.M., van Ginneken, B., Sánchez, C.I.: A survey on deep learning in medical image analysis. Med. Image Anal. **42**, 60–88 (2017)
38. Ma, J., Sheridan, R.P., Liaw, A., Dahl, G.E., Svetnik, V.: Deep neural nets as a method for quantitative structure–activity relationships. J. Chem. Inf. Model. **55**(2), 263–274 (2015)
39. Marshak, A., Davis, A.: 3D Radiative Transfer in Cloudy Atmospheres. Springer, Berlin (2005)
40. Mhaskar, H., Liao, Q., Poggio, T.: Learning functions: when is deep better than shallow. arXiv:1603.00988 (2018)
41. Paschalis, P., Giokaris, N.D., Karabarbounis, A., Loudos, G., Maintas, D., Papanicolas, C., Spanoudaki, V., Tsoumpas, C., Stiliaris, E.: Tomographic image reconstruction using artificial neural networks. Nucl. Instrum. Methods Phys. Res.

Sect. A Accel. Spectrom. Detect. Assoc. Equip. **527**(1), 211–215 (2004). (Proceedings of the 2nd International Conference on Imaging Technologies in Biomedical Sciences)
42. Pitaevskii, L.: Vortex lines in an imperfect Bose gas. Sov. Phys. JETP **13**(2), 451–454 (1961)
43. Pomraning, G.C.: The Equations of Radiation Hydrodynamics. Courier Corporation, Chelmsford (1973)
44. Raissi, M., Karniadakis, G.E.: Hidden physics models: machine learning of nonlinear partial differential equations. J. Comput. Phys. **357**, 125–141 (2018)
45. Ren, K., Zhang, R., Zhong, Y.: A fast algorithm for radiative transport in isotropic media. arXiv:1610.00835 (2016)
46. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015, pp. 234–241. Springer International Publishing, Cham (2015)
47. Rudd, K., Muro, G.D., Ferrari, S.: A constrained backpropagation approach for the adaptive solution of partial differential equations. IEEE Trans. Neural Netw. Learn. Syst. **25**(3), 571–584 (2014)
48. Sarikaya, R., Hinton, G.E., Deoras, A.: Application of deep belief networks for natural language understanding. IEEE/ACM Trans. Audio Speech Lang. Process. **22**(4), 778–784 (2014)
49. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
50. Silver, D., Huang, A., Maddison, C.J., Guez, L.S.A., Driessche, G.V.D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
51. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-sacle image recognition. Computing Research Repository (CoRR). arXiv:1409.1556 (2014)
52. Socher, R., Bengio, Y., Manning, C.D.: Deep learning for NLP (without magic). In: The 50th Annual Meeting of the Association for Computational Linguistics, Tutorial Abstracts, vol. 5 (2012)
53. Spiliopoulos, K., Sirignano, J.: DGM: A deep learning algorithm for solving partial differential equations. arXiv:1708.07469 (2018)
54. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 27, pp. 3104–3112. Curran Associates, Inc., New York (2014)
55. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. Computing Research Repository (CoRR). arXiv:1409.4842 (2014)
56. Timothy, D.: Incorporating Nesterov momentum into Adam. http://cs229.stanford.edu/proj2015/054_report.pdf (2015)
57. Trefethen, L.: Spectral Methods in MATLAB. Society for Industrial and Applied Mathematics, Philadelphia (2000)
58. Tyrtyshnikov, E.: Mosaic-skeleton approximations. Calcolo **33**(1–2), 47–57 (1998). (1996. Toeplitz matrices: structures, algorithms and applications (Cortona, 1996))
59. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Deep image prior. arXiv:1711.10925 (2018)
60. Wang, T., Wu, D.J., Coates, A., Ng, A.Y.: End-to-end text recognition with convolutional neural networks. In: Pattern Recognition (ICPR), 2012 21st International Conference on Pattern Recognition (ICPR2012), pp. 3304–3308 (2012)
61. Wang, Y., Siu, C.W., Chung, E.T., Efendiev, Y., Wang, M.: Deep multiscale model learning. arXiv:1806.04830 (2018)
62. Xiong, H.Y., et al.: The human splicing code reveals new insights into the genetic determinants of disease. Science **347**(6218), 1254806 (2015)
63. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision—ECCV 2014. Lecture Notes in Computer Science, vol. 8689, pp. 818–833. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10590-1_53