MULTISCALE MODEL. SIMUL.
Vol. 17, No. 4, pp. 1189–1213

A MULTISCALE NEURAL NETWORK BASED ON HIERARCHICAL MATRICES*

YUWEI FAN†, LIN LIN‡, LEXING YING§, AND LEONARDO ZEPEDA-NÚÑEZ¶

Abstract. In this work we introduce a new multiscale artificial neural network based on the structure of $\mathcal{H}$-matrices. This network generalizes the latter to the nonlinear case by introducing a local deep neural network at each spatial scale. Numerical results indicate that the network is able to efficiently approximate discrete nonlinear maps obtained from discretized nonlinear partial differential equations, such as those arising from nonlinear Schrödinger equations and the Kohn–Sham density functional theory.

Key words. $\mathcal{H}$-matrix, multiscale neural network, locally connected neural network, convolutional neural network

AMS subject classifications. 65M75, 60H35, 65F30

DOI. 10.1137/18M1203602

1. Introduction.

In the past decades, there has been a great combined effort in developing efficient algorithms to solve linear problems issued from discretization of integral equations (IEs), and partial differential equations (PDEs). In particular, multiscale methods such as multigrid methods [10], the fast multipole method [22], wavelets [46], and hierarchical matrices [9, 24] have been strikingly successful in reducing the complexity for solving such systems. In several cases, for operators of pseudo-differential type, these algorithms can achieve linear or quasi-linear complexity. In a nutshell, these methods aim to use the inherent multiscale structure of the underlying physical problem to build efficient representations at each scale, thus compressing the information contained in the system. The gains in complexity stem mainly from processing information at each scale, and merging it in a hierarchical fashion.

Even though these techniques have been extensively applied to linear problems with outstanding success, their application to nonlinear problems has been, to the best of our knowledge, very limited. This is due to the high complexity of the solution maps. In particular, building a global approximation of such maps would normally require an extremely large amount of parameters, which, in return, is often translated to

*Received by the editors July 27, 2018; accepted for publication (in revised form) August 26, 2019; published electronically November 21, 2019.
https://doi.org/10.1137/18M1203602
Funding: The work of the first and third authors was 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, by the National Science Foundation under award DMS-1818449, and by the GCP Research Credits Program from Google. The work of the first author was also partially supported by the AWS Cloud Credits for Research program from Amazon. The work of the second and fourth authors was partially supported by the U.S. Department of Energy under grant DE-SC0017867, by the U.S. Department of Energy under the CAMERA project, and by the Air Force Office of Scientific Research under award FA9550-18-1-0095.
†Department of Mathematics, Stanford University, Stanford, CA 94305 (ywfan@stanford.edu).
‡Department of Mathematics, University of California, Berkeley, and Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (linlin@math.berkeley.edu).
§Department of Mathematics and Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305 (lexing@stanford.edu).
¶Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (lzepeda@lbl.gov).

algorithms with a prohibitive computational cost. The development of algorithms and heuristics to reduce the cost is an area of active research [6, 19, 18, 23, 45]. However, in general, each method is application-dependent and requires a deep understanding of the underlying physics.

On the other hand, the surge of interest in machine learning methods, in particular, deep neural networks, has dramatically improved speech recognition [31], visual object recognition [38], object detection, etc. This has fueled breakthroughs in many domains such as drug discovery [44], genomics [40], and automatic translation [58], among many others [39, 56]. Deep neural networks have empirically shown that it is possible to obtain efficient representations of very high-dimensional functions. Even though the universality theorem holds for neural networks [15, 33, 35, 48], i.e., they can approximate arbitrarily well any function with mild regularity conditions, how to efficiently build such approximations remains an open question. In particular, the degree of approximation depends dramatically on the architecture of the neural network, i.e., how the different layers are connected. In addition, there is a fine balance between the number of parameters, the architecture, and the degree of approximation [28, 29, 48].

This paper aims to combine the tools developed in deep neural networks with ideas from multiscale methods. In particular, we extend hierarchical matrices ($\mathcal{H}$-matrices) to nonlinear problems within the framework of neural networks. Let

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

be a nonlinear generalization of pseudo-differential operators, issued from an underlying physical problem, described by either an IE or a PDE, where v can be considered as a parameter in the equation, u is either the solution of the equation or a function of it, and n is the number of variables.

We build a neural network with a novel multiscale structure inspired by hierarchical matrices. We interpret the application of an $\mathcal{H}$-matrix to a vector using a neural network structure as follows. We first reduce the dimensionality of the vector, or *restrict* it, by multiplying it by a short and wide structured matrix. Then we *process* the encoded vector by multiplying it by a structured square matrix. Then we return the vector to its original size, or *interpolate* it, by multiplying it by a structured tall and skinny matrix. Such operations are performed separately at different spatial scales. The final approximation to the matrix-vector multiplication is obtained by adding the contributions from all spatial scales, including the near-field contribution, which is represented by a near-diagonal matrix. Since every step is linear, the overall operation is also a linear mapping. This interpretation allows us to directly generalize the $\mathcal{H}$-matrix to nonlinear problems by replacing the structured square matrix in the processing stage by a structured nonlinear network with several layers. The resulting artificial neural network, which we call *multiscale neural network*, only requires a relatively modest number of parameters even for large problems.

We demonstrate the performance of the multiscale neural network by approximating the solution to the nonlinear Schrödinger equation [2, 51], as well as the Kohn–Sham map [32, 37]. These mappings are highly nonlinear, and are still well approximated by the proposed neural network, with a relative accuracy on the order of $10^{-4} \sim 10^{-3}$. Furthermore, we do not observe overfitting even in the presence of a relatively small set of training samples.

1.1. Related work. Although machine learning and deep learning literature is vast, the application of deep learning to numerical analysis problems is relatively

new, though that is rapidly changing. Research using deep neural networks with multiscale architectures has primarily focused on image [4, 11, 13, 43, 54, 60] and video [47] processing.

Deep neural networks have been recently used to solve PDEs [7, 8, 12, 17, 27, 41, 52, 55, 57] and classical inverse problems [3, 49]. For general applications of machine learning to nonlinear numerical analysis problems, the work of Raissi and Karnidiakis used machine learning, in particular, Gaussian processes, to find parameters in nonlinear equations [53]; Chan and Elsheikh predicted the basis function on the coarse grid in a multiscale finite volume method by neural networks (NNs); Khoo, Lu, and Ying used neural network (NN) in the context of uncertainty quantification [34]; and Zhang et al. used NN in the context of generating high-quality interatomic potentials for molecular dynamics [64, 65]. Wang et al. applied nonlocal multicontinuum NN on time-dependent nonlinear problems [61]. Khrulkov, Novikov, and Oseledets [35] and Cohen, Sharir, and Shashua [15] developed deep NN architectures based on tensor-train decomposition. In addition, we note that deep NNs with related multiscale structures [54, 50] have been proposed mainly for applications such as image processing. However, we are not aware of any applications of such architectures to solving nonlinear differential or integral equations.

1.2. Organization. The reminder of the paper is organized as follows. Section 2 reviews the $\mathcal{H}$-matrices and interprets the $\mathcal{H}$-matrices using the framework of NNs. Section 3 extends the NN representation of $\mathcal{H}$-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 section 5.

2. Neural network architecture for $\mathcal{H}$-matrices. In this section, we aim to represent the matrix-vector multiplication of $\mathcal{H}$-matrices within the framework of NNs. For the sake of clarity, we succinctly review the structure of $\mathcal{H}$-matrices for the one-dimensional (1D) case in subsection 2.1. We interpret $\mathcal{H}$-matrices using the framework of NNs in subsection 2.2, and then extend it to the multidimensional case in subsection 2.3.

2.1. $\mathcal{H}$-matrices. Hierarchical matrices ($\mathcal{H}$-matrices) were first introduced by Hackbusch et al. in a series of papers [24, 26, 25] as an algebraic framework for representing matrices with a hierarchically off-diagonal low-rank structure. This framework provides efficient numerical methods for solving linear systems arising from IEs and PDEs [9]. In what follows, we follow the notation in [42] to provide a brief introduction to the framework of $\mathcal{H}$-matrices in a simple uniform and Cartesian setting. The interested reader is referred to [24, 9, 42] for further details.

Consider the IE

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

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 a 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 analogues of $u(x)$ and $v(x)$, respectively.

We introduce a hierarchical dyadic decomposition of the grid in $L+1$ levels as follows. We start by the 0th level of the decomposition, which corresponds to the set of all grid points defined as

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

At each level ℓ ($0 \le \ell \le L$), we decompose the grid 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 $i = 1, \dots, 2^\ell$. Throughout this manuscript, $\mathcal{I}^{(\ell)}$ (or $\mathcal{J}^{(\ell)}$) denotes a generic segment of a given level ℓ, and the superscript ℓ will be omitted when the level is clear from the context. Moreover, following the usual terminology in $\mathcal{H}$-matrices, we say that a segment $\mathcal{J}^{(l)}$ ($\ell \ge 1$) is the *parent* of a segment $\mathcal{I}^{(l-1)}$ if $\mathcal{I}^{(l-1)} \subset \mathcal{J}^{(l)}$. Symmetrically, $\mathcal{I}^{(l-1)}$ is called a *child* of $\mathcal{J}^{(l)}$. Clearly, each segment, except those on level L, has two child segments.

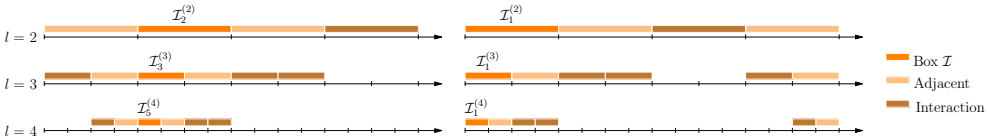


FIG. 1. *Illustration of the computational domain at level $\ell = 2, 3, 4$. The left and right figures represent an interior segment and a boundary segment and their adjacent and interaction lists at different levels.*

In addition, for a segment $\mathcal{I}$ on level $\ell \ge 2$, we define the following lists:

- $\mathsf{NL}(\mathcal{I})$ *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 $\mathcal{I}$. If $\ell = 2$, $\mathsf{IL}(\mathcal{I})$ contains all the segments on level 2 minus $\mathsf{NL}(\mathcal{I})$. If $\ell > 2$, $\mathsf{IL}(\mathcal{I})$ contains all the segments on level ℓ that are children of segments in $\mathsf{NL}(\mathcal{P})$ with $\mathcal{P}$ being $\mathcal{I}$'s parent minus $\mathsf{NL}(\mathcal{I})$.

Figure 1 illustrates the dyadic partition of the computational domain and the lists on levels $\ell = 2, 3, 4$. Clearly, $\mathcal{J} \in \mathsf{NL}(\mathcal{I})$ if and only if $\mathcal{I} \in \mathsf{NL}(\mathcal{J})$, and $\mathcal{J} \in \mathsf{IL}(\mathcal{I})$ if and only if $\mathcal{I} \in \mathsf{IL}(\mathcal{J})$.

For a vector $v \in \mathbb{R}^N$, $v_\mathcal{I}$ denotes the elements of v indexed by $\mathcal{I}$, and for a matrix $A \in \mathbb{R}^{N\times N}$, $A_{\mathcal{I},\mathcal{J}}$ represents the submatrix given by the elements of A indexed by $\mathcal{I} \times \mathcal{J}$. The dyadic partition of the grid and the interaction lists induce a multilevel decomposition of A as follows:

$$A = \sum_{\ell=2}^{L} A^{(\ell)} + A^{(\mathrm{ad})}, \tag{2.4}$$

$$A^{(\ell)}_{\mathcal{I},\mathcal{J}} = \begin{cases} A_{\mathcal{I},\mathcal{J}}, & \mathcal{I} \in \mathsf{IL}(\mathcal{J}); \\ 0 & \text{otherwise,} \end{cases} \quad \mathcal{I}, \mathcal{J} \text{ at level } \ell,\ 2 \le \ell \le L,$$

$$A^{(\mathrm{ad})}_{\mathcal{I},\mathcal{J}} = \begin{cases} A_{\mathcal{I},\mathcal{J}}, & \mathcal{I} \in \mathsf{NL}(\mathcal{J}); \\ 0 & \text{otherwise,} \end{cases} \quad \mathcal{I}, \mathcal{J} \text{ at level } L.$$

In a nutshell, $A^{(\ell)}$ considers the interaction at level ℓ between a segment and its interaction list, and $A^{(\mathrm{ad})}$ considers all of the interactions between adjacent segments at level L.

Figure 2 illustrates the block partition of A induced by the dyadic partition, and the decomposition induced by the different interaction lists at each level that follows (2.4).

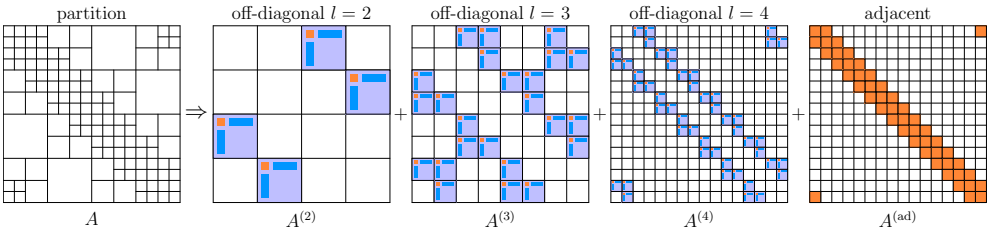
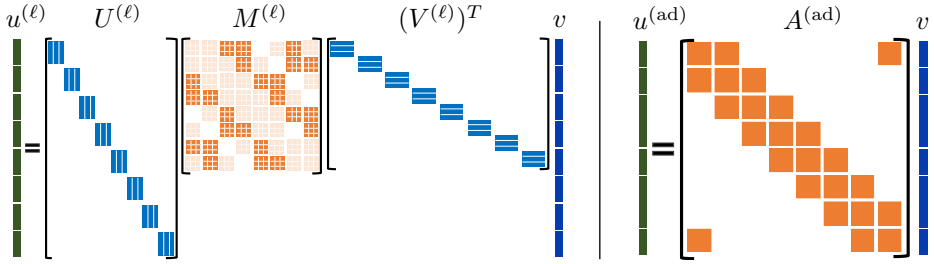


FIG. 2. *Partition of the matrix A for $L = 4$ and nonzero blocks of $A^{(\ell)}$, $\ell = 2, 3, 4$ (colored blue), and $A^{(\mathrm{ad})}$ (colored orange). Nonzero blocks of $A^{(\ell)}$ are approximated by low-rank approximation, and nonzero blocks of $A^{(\mathrm{ad})}$ are retained. (Figure in color online.)*



(a) $u^{(\ell)} = A^{(\ell)}v \approx U^{(\ell)}M^{(\ell)}(V^{(\ell)})^T v$ (b) $u^{(\mathrm{ad})} = A^{(\mathrm{ad})}v$

FIG. 3. *Diagram of matrix-vector multiplication* (2.7) *of the low-rank part and the adjacent part of $\mathcal{H}$-matrices. The blocks of $M^{(\ell)}$ colored by pale orange are zero blocks, and if we treat these blocks as nonzero blocks, the matrices $M^{(\ell)}$ are block cyclic band matrices. (Figure in color online.)*

The key idea behind $\mathcal{H}$-matrices is to approximate the nonzero blocks $A^{(\ell)}$ by a low-rank approximation (see [36] for a thorough review). This idea is depicted in Figure 2, in which each nonzero block is approximated by a tall and skinny matrix, a small squared one, and a short and wide one, respectively. In this work, we focus on the uniform $\mathcal{H}$-matrices [21], and, for simplicity, we suppose that each block has a fixed 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.

The main observation is that it is possible to factorize each $A^{(\ell)}$ as $A^{(\ell)} \approx U^{(\ell)}M^{(\ell)}(V^{(\ell)})^T$ depicted in Figure 3. $U^{(\ell)}$ is a block diagonal matrix with diagonal blocks $U^{(\ell)}_{\mathcal{I}}$ for $\mathcal{I}$ at level ℓ, $V^{(\ell)}$ is a block diagonal matrix with diagonal blocks $V^{(\ell)}_{\mathcal{J}}$ for $\mathcal{J}$ at level ℓ, and finally, $M^{(\ell)}$ aggregates all of the blocks $M^{(\ell)}_{\mathcal{I},\mathcal{J}}$ for all interacting segments $\mathcal{I}, \mathcal{J}$ at level ℓ. This factorization induces a decomposition of A given by

$$A = \sum_{\ell=2}^{L} A^{(\ell)} + A^{(\mathrm{ad})} \approx \sum_{\ell=2}^{L} U^{(\ell)}M^{(\ell)}(V^{(\ell)})^T + A^{(\mathrm{ad})}. \tag{2.6}$$

Thus the matrix-vector multiplication (2.2) can be expressed as

$$u \approx \sum_{\ell=2}^{L} u^{(\ell)} + u^{(\mathrm{ad})} = \sum_{\ell=2}^{L} U^{(\ell)} M^{(\ell)} (V^{(\ell)})^T v + A^{(\mathrm{ad})} v, \tag{2.7}$$

as illustrated in Figure 3, which constitutes the basis for writing $\mathcal{H}$-matrices as a NN.

In addition, the matrices $\{U^{(\ell)}, V^{(\ell)}, M^{(\ell)}\}_{\ell=2}^{L}$ and $A^{(\mathrm{ad})}$ have the following properties.

PROPERTY 1. *The matrices are as follows:*

1. $U^{(\ell)}$ *and* $V^{(\ell)}$, $\ell = 2, \ldots, L$, *are block diagonal matrices with block size* $N/2^{\ell} \times r$,
2. $A^{(\mathrm{ad})}$ *is a block cyclic tridiagonal matrix with block size* $m \times m$, *and*
3. $M^{(\ell)}$, $\ell = 2, \ldots, L$, *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* $\ell \geq 3$, *if we treat all of the pale orange colored blocks of* $M^{(\ell)}$ *in Figure* 3 (a) *as nonzero blocks.*

We point out that the band sizes $n_b^{(\ell)}$ and $n_b^{(\mathrm{ad})}$ depend on the definitions of NL and IL. In this case, the lists were defined using the *strong admissible condition* in $\mathcal{H}$-matrices. Other conditions can certainly be used, such as the *weak admissibility condition*, leading to similar structures.

2.2. Matrix-vector multiplication as an NN. An artificial NN, in particular, a feed-forward network, can be thought of as the composition of several simple functions, usually called *propagation functions*, in which the intermediate 1D variables are called *neurons*, which, in return, are organized in vector, or tensor, variables called *layers*. For example, an artificial feed-forward NN

$$u = \mathcal{F}(v), \qquad u, v \in \mathbb{R}^n, \tag{2.8}$$

with $K + 1$ layer can be written using the following recursive formula:

$$\begin{aligned} \xi^{(0)} &= v, \\ \xi^{(k)} &= \phi(W^{(k)} \xi^{(k-1)} + b^{(k)}), \\ u &= \xi^{(K)}, \end{aligned} \tag{2.9}$$

where for each $k = 1, \ldots, K$ we have that $\xi^{(k)}, b^{(k)} \in \mathbb{R}^{n_k}, W^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}$. Following the terminology of machine learning, ϕ is called the *activation function* that is applied entrywise, $W^{(k)}$ are the *weights*, $b^{(k)}$ are the *biases*, and $\xi^{(k)}$ is the kth layer containing n_k number of *neurons*. Typical choices for the activation function are linear function, the rectified-linear unit (ReLU), or sigmoid function. In addition, (2.9) can easily be rewritten using tensors by replacing the matrix-vector multiplication by the more general tensor contraction. We point out that representing layers with tensors or vectors is equivalent up to reordering and reshaping. The main advantage of using the former is that layers, and their propagating functions, can be represented in a more compact fashion. Therefore, in what follows we predominantly use a tensor representation.

Within this context, training a network refers to finding the weights and biases, whose entries are collectively called *parameters*, in order to approximate a given map. This is usually done by minimizing a loss function using a stochastic optimization algorithm.

2.2.1. Locally connected networks. We interpret the structure of $\mathcal{H}$-matrices (2.6) using the framework of NNs. The different factors in (2.7) possess a distinctive structure, which we aim to exploit by using locally connected (LC) networks. LC networks are propagating functions whose weights have a block-banded constraint. For the 1D example, we also treat ξ as a 2-tensor of dimensions $\alpha \times N_x$, where α is the *channel dimension*, N_x is the *spatial dimension*, and ζ is a 2-tensor of dimensions $\alpha' \times N_x'$. We say that ξ is connected to ζ by an LC network if

(2.10)
$$\zeta_{c',i} = \phi\left(\sum_{j=(i-1)s+1}^{(i-1)s+w}\sum_{c=1}^{\alpha} W_{c',c;i,j}\xi_{c,j} + b_{c',i}\right), \quad i = 1,\dots,N_x', \quad c' = 1,\dots,\alpha',$$

where w and $s \in \mathbb{N}$ are the *kernel window size* and *stride*, respectively. In addition, we say that ζ is an LC layer if it satisfies (2.10).

Each LC network requires six parameters, N_x, α, N_x', α', w, and s, to be characterized. Next, we define three types of LC networks by specifying some of their parameters.

- LCR *Restriction* network: We set $s = w = \frac{N_x}{N_x'}$ and $\alpha = 1$ in an LC network. This network represents the multiplication of a block diagonal matrix with block sizes $\alpha' \times s$ and a vector with size $N_x\alpha$, as illustrated by Figure 4 (a). We denote this network using $\mathsf{LCR}[\phi; N_x, N_x', \alpha']$. The application of $\mathsf{LCR}[\mathsf{linear}; 32, 8, 3]$ is depicted in Figure 4 (a).
- LCK *Kernel* network: We set $s = 1$ and $N_x' = N_x$. This network represents the multiplication of a cyclic block band matrix of block size $\alpha' \times \alpha$ and band size $\frac{w-1}{2}$ times a vector of size $N_x\alpha$, as illustrated by the upper portion of Figure 4 (b). To account for the periodicity we pad the input layer $\xi_{c,j}$ on the spatial dimension to the size $(N_x + w - 1) \times \alpha$. We denote this network by $\mathsf{LCK}[\phi; N_x, \alpha, \alpha', w]$. This network has two steps: the periodic padding of $\xi_{c,j}$ on the spatial dimension, and the application of (2.10). The application of $\mathsf{LCK}[\mathsf{linear}; 8, 3, 3, 3]$ is depicted in Figure 4 (b).
- LCI *Interpolation* network: We set $s = w = 1$ and $N_x' = N_x$ in an LC network. This network represents the multiplication of a block diagonal matrix with block size $\alpha' \times \alpha$, times a vector of size $N_x\alpha$, as illustrated by the upper figure in Figure 4 (c). We denote the network $\mathsf{LCI}[\phi; N_x, \alpha, \alpha']$, which has two steps: the application of (2.10), and the reshaping of the output to a vector by column major indexing. The application of $\mathsf{LCI}[\mathsf{linear}; 8, 3, 4]$ is depicted in Figure 4 (c).

2.2.2. NN representation. Following (2.7), in order to construct the NN architecture for (2.7), we need to represent the following four operations:

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

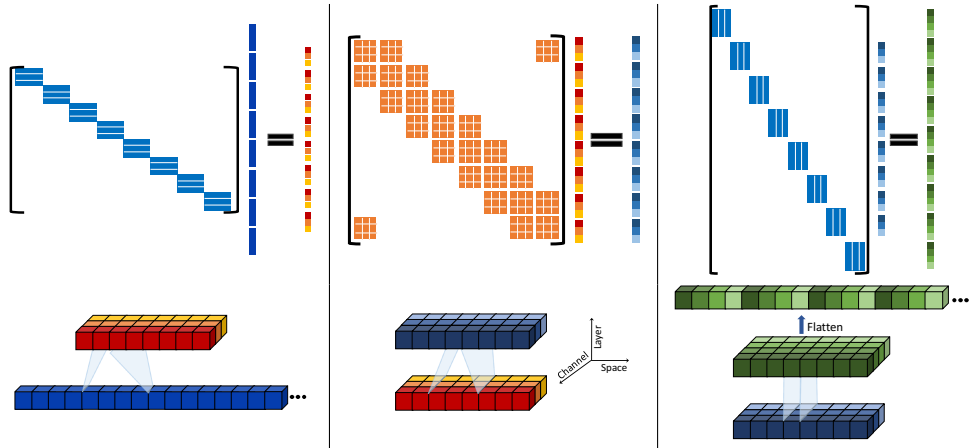
$$\zeta^{(\ell)} = M^{(\ell)}\xi^{(\ell)}, \tag{2.11b}$$

$$u^{(\ell)} = U^{(\ell)}\zeta^{(\ell)}, \tag{2.11c}$$

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

From Property 1.1 and the definition of LCR and LCI, the operations (2.11a) and (2.11c) are equivalent to

$$\xi^{(\ell)} = \mathsf{LCR}[\mathsf{linear}; N, 2^\ell, r](v), \quad u^{(\ell)} = \mathsf{LCI}\left[\mathsf{linear}; 2^\ell, r, \frac{N}{2^\ell}\right](\zeta^{(\ell)}), \tag{2.12}$$



$s = w = \frac{N_x}{N'_x}$, $\alpha = 1$
(a) $\mathsf{LCR}[\phi; N_x, N'_x, \alpha']$ with $N_x = 32$, $N'_x = 8$, and $\alpha' = 3$

$s = 1$, $N'_x = N_x$
(b) $\mathsf{LCK}[\phi; N_x, \alpha, \alpha', w]$ with $N_x = 8$, $\alpha = \alpha' = 3$. and $w = 3$

$s = 1$, $w = 1$, $N'_x = N_x$
(c) $\mathsf{LCI}[\phi; N_x, \alpha, \alpha']$ with $N_x = 8$, $\alpha = 3$, and $\alpha' = 4$

FIG. 4. *Three instances of LC networks used to represent the matrix-vector multiplication in* (2.7). *The upper portions of each column depict the patterns of the matrices, and the lower portions are their respective analogues using LC networks.*

respectively. Analogously, Property 1.3 indicates that (2.11b) is equivalent to

$$\zeta^{(\ell)} = \mathsf{LCK}[\mathsf{linear}; 2^\ell, r, r, 2n_b^{(\ell)} + 1](\xi^{(\ell)}). \tag{2.13}$$

We point out that $\xi^{(\ell)}$ is a vector in (2.11c) but a 2-tensor in (2.12) and (2.13). In principle, we need to flatten $\xi^{(\ell)}$ in (2.12) into a vector and reshape it back into a 2-tensor before (2.13). These operations do not alter the algorithmic pipeline, so they are omitted.

Given that $v, u^{(\mathrm{ad})}$ are vectors, but LCK is defined for 2-tensors, we explicitly write the reshape and flatten operations. We denote as $\mathsf{Reshape}[n_1, n_2]$ the map that reshapes a vector of size $n_1 n_2$ into a 2-tensor of size $n_1 \times n_2$ by column major indexing, and $\mathsf{Flatten}$ is defined as the inverse of $\mathsf{Reshape}$. Using Property 1.2, we can write (2.11d) as

$$\tilde{v} = \mathsf{Reshape}[m, 2^L](v), \quad \tilde{u}^{(\mathrm{ad})} = \mathsf{LCK}\left[\mathsf{linear}; 2^L, m, m, 2n_b^{(\mathrm{ad})} + 1\right](\tilde{v}), \tag{2.14}$$
$$u^{(\mathrm{ad})} = \mathsf{Flatten}(\tilde{u}^{(\mathrm{ad})}).$$

Combining (2.12), (2.13), and (2.14), we obtain Algorithm 1, whose architecture is illustrated in Figure 5. In particular, Figure 5 is the translation to the NN framework of (2.7) (see Figure 3) using the building blocks depicted in Figure 4.

Moreover, the memory footprints of the NN architecture and $\mathcal{H}$-matrices are asymptotically the same with respect to the spatial dimension of u and v. This can be readily shown by computing the total number of parameters. For the sake of simplicity, we only count the parameters in the weights, ignoring those in the biases. A direct calculation yields the number of parameters in LCR, LCK, and LCI:

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

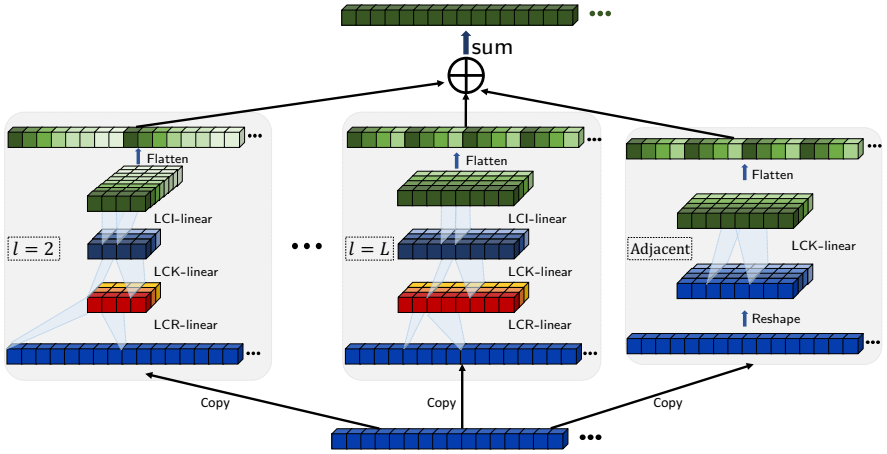


FIG. 5. *NN architecture for $\mathcal{H}$-matrices.*

Algorithm 1 Application of the NN representation of an $\mathcal{H}$-matrix to a vector $v \in \mathbb{R}^N$.

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

respectively. Hence, the number of parameters in Algorithm 1 is

$$
\begin{aligned}
N_p^{\mathcal{H}} &= \sum_{\ell=2}^{L} \left(Nr + 2^\ell r^2 (2n_b^{(\ell)} + 1) + 2^\ell r \frac{N}{2^\ell} \right) + 2^L m^2 (2n_b^{(\mathrm{ad})} + 1) \\
&\le 2LNr + 2^{L+1} r^2 \left(2 \max_{\ell=2}^{L} n_b^{(\ell)} + 1 \right) + Nm(2n_b^{(\mathrm{ad})} + 1) \\
&\le 2N \log(N) r + 3Nm(2n_b + 1) \sim O(N \log(N)),
\end{aligned} \tag{2.16}
$$

where $n_b = \max(n_b^{(\mathrm{ad})}, n_b^{(\ell)}, \ell = 2, \ldots, L)$, and $r \le m$ is used.

2.3. Multidimensional case. Following the previous section, the extension of Algorithm 1 to the d-dimensional case can be easily deduced using the tensor product of 1D cases. Consider $d = 2$ below, for instance, and the generalization to the higher-dimensional case will be straightforward.

Suppose that we have an IE in two dimensions given by

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

We discretize the domain Ω with a uniform grid with $n = N^2$ ($N = 2^L m$) discretization points and let A be the resulting matrix obtained from discretizing (2.17). We denote the set of all grid points as

$$
\mathcal{I}^{(d,0)} = \{(k_1/N, k_2/N) : k_1, k_2 = 0, \ldots, N - 1\}. \tag{2.18}
$$

Clearly, $\mathcal{I}^{(d,0)} = \mathcal{I}^{(0)} \otimes \mathcal{I}^{(0)}$, where $\mathcal{I}^{(0)}$ is defined in (2.3), and $\otimes$ is tensor product. At each level ℓ ($0 \le \ell \le L$), we decompose the grid into 4^l disjoint boxes as $\mathcal{I}_i^{(d,\ell)} = \mathcal{I}_{i_1}^{(\ell)} \otimes \mathcal{I}_{i_2}^{(\ell)}$ for $i_1, i_2 = 1, \ldots, 2^l$.

The definition of the lists NL and IL can be easily extended. For each box $\mathcal{I}$, $\mathsf{NL}(\mathcal{I})$ contains three boxes for the 1D case, and 3^2 boxes for the two-dimensional (2D) case. Similarly, the decomposition (2.4) on the matrix A can be easily extended for this case.

Following the structure of $\mathcal{H}$-matrices, the off-diagonal blocks of $A^{(\ell)}$ can be approximated as

$$A^{(\ell)}_{\mathcal{I},\mathcal{J}} \approx U^{(\ell)}_{\mathcal{I}} M^{(\ell)}_{\mathcal{I},\mathcal{J}} (V^{(\ell)}_{\mathcal{J}})^T, \quad \mathcal{I}, \mathcal{J} \in \mathcal{I}^{(\ell)}, \quad U^{(\ell)}_{\mathcal{I}}, V^{(\ell)}_{\mathcal{J}} \in \mathbb{R}^{(N/2^\ell)^2 \times r}, \quad M^{(\ell)}_{\mathcal{I},\mathcal{J}} \in \mathbb{R}^{r \times r}. \tag{2.19}$$

As mentioned before, we can describe the network using tensors or vectors. In what follows, we will switch between representations in order to illustrate the concepts in a compact fashion. We denote an entry of a tensor T by $T_{i,j}$, where i is the 2D index $i = (i_1, i_2)$. Using the tensor notation, $U^{(\ell)}$, $V^{(\ell)}$ in (2.7) can be treated as 4-tensors of dimension $N \times N \times 2^\ell r \times 2^\ell$. We generalize the notion of band matrix to *band tensors*. A band tensor T satisfies that

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

where $n_b = (n_{b,1}, n_{b,2})$ is called the band size for tensors. Thus Property 1 can be generalized to tensors yielding the following properties.

PROPERTY 2. *The 4-tensors are as follows:*

1. $U^{(\ell)}$ *and* $V^{(\ell)}$, $\ell = 2, \ldots, L$, *are block diagonal tensors with block size* $N/2^\ell \times N/2^\ell \times r \times 1$,
2. $A^{(\mathrm{ad})}$ *is a block band cyclic tensor with block size* $m \times m \times m \times m$ *and band size* $n_b^{(\mathrm{ad})} = (1,1)$, *and*
3. $M^{(\ell)}$, $\ell = 2, \ldots, L$, *are block band cyclic 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* $\ell \ge 3$.

Next, we characterize LC networks for the 2D case. An NN layer for the 2D case can be represented by a 3-tensor size $\alpha \times N_{x,1} \times N_{x,2}$, in which α is the channel dimension and $N_{x,1}$, $N_{x,2}$ are the spatial dimensions. If a layer ξ with size $\alpha \times N_{x,1} \times N_{x,2}$ is connected to an LC layer ζ with size $\alpha' \times N'_{x,1} \times N'_{x,2}$, then

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

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 four operations in (2.11) which constitute the building blocks of the NN. The extension is trivial. The parameters N_x, s, and w in the 1D LC networks are replaced by their 2D counterparts $N_x = (N_{x,1}, N_{x,2})$, $s = (s_1, s_2)$, 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.

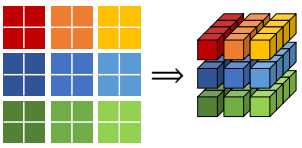


FIG. 6. *Diagram of* Reshape$[2^2, 3, 3]$ *in Algorithm* 2.

Using the notation above we extend Algorithm 1 to the 2D case in Algorithm 2. We crucially remark that the Reshape$[r^2, n_1, n_2]$ function in Algorithm 2 is not the usual major column-based reshaping. It reshapes a 2-tensor T with size $rn_1 \times rn_2$ into a 3-tensor S with size $r^2 \times n_1 \times n_2$ by treating the former as a block tensor with block size $r \times r$, and reshaping each block as a vector following the formula $S(k,i,j) = T((i-1)r+k_1, (j-1)r+k_2)$ with $k = (k_1-1)r+k_2$, for $k_1, k_2 = 1, \ldots, r$, $i = 1, \ldots, n_1$, and $j = 1, \ldots, n_2$. Figure 6 provides an example for the case Reshape$[2^2, 3, 3]$. The Flatten is its inverse.

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

u $= 0$;
for $\ell = 2$ to L **do**
  $\xi = \mathsf{LCR}[\mathsf{linear}; (N,N), (2^\ell, 2^\ell), r](v)$;
  $\zeta = \mathsf{LCK}[\mathsf{linear}; (2^\ell, 2^\ell), r, r, (2n_{b,1}^{(\ell)} + 1, 2n_{b,2}^{(\ell)} + 1)](\xi)$;
  $u = u + \mathsf{LCI}[\mathsf{linear}; (2^\ell, 2^\ell), r, \left(\frac{N}{2^\ell}\right)^2](\zeta)$;
end for
$\tilde{v} = \mathsf{Reshape}[m^2, 2^L, 2^L](v)$;
$\tilde{u}^{(\mathrm{ad})} = \mathsf{LCK}\left[\mathsf{linear}; (2^L, 2^L), m^2, m^2, (2n_{b,1}^{(\mathrm{ad})} + 1, 2n_{b,2}^{(\mathrm{ad})} + 1)\right](\tilde{v})$;
$u^{(\mathrm{ad})} = \mathsf{Flatten}(\tilde{u}^{(\mathrm{ad})})$;
$u = u + u^{(\mathrm{ad})}$.

3. Multiscale NN. In this section, we extend the aforementioned NN architecture to represent a nonlinear generalization of pseudo-differential operators of the form

$$u = \mathcal{M}(v), \quad u, v \in \mathbb{R}^{N^d}. \tag{3.1}$$

Due to its multiscale structure, we refer to the resulting NN architecture as the *multiscale neural network* (MNN). We consider the 1D case below for simplicity, and that the generalization to higher dimensions follows directly as in subsection 2.3.

Algorithm 3 Application of MNN to a vector $v \in \mathbb{R}^N$.

u $= 0$;
for $\ell = 2$ to L **do**
  $\xi_0 = \mathsf{LCR}[\mathsf{linear}; N, 2^\ell, r](v)$;
  for $k = 1$ to K **do**
    $\xi_k = \mathsf{LCK}[\phi; 2^\ell, r, r, 2n_b^{(\ell)} + 1](\xi_{k-1})$;
  end for
  $u = u + \mathsf{LCI}[\mathsf{linear}; 2^\ell, r, \frac{N}{2^\ell}](\xi_K)$;
end for
$\xi_0 = \mathsf{Reshape}[m, 2^L](v)$;
for $k = 1$ to $K-1$ **do**
  $\xi_k = \mathsf{LCK}[\phi; 2^L, m, m, 2n_b^{(\mathrm{ad})} + 1](\xi_{k-1})$;
end for
$\xi_K = \mathsf{LCK}[\mathsf{linear}; 2^L, m, m, 2n_b^{(\mathrm{ad})} + 1](\xi_{K-1})$;
$u^{(\mathrm{ad})} = \mathsf{Flatten}(\xi_K)$;
$u = u + u^{(\mathrm{ad})}$.

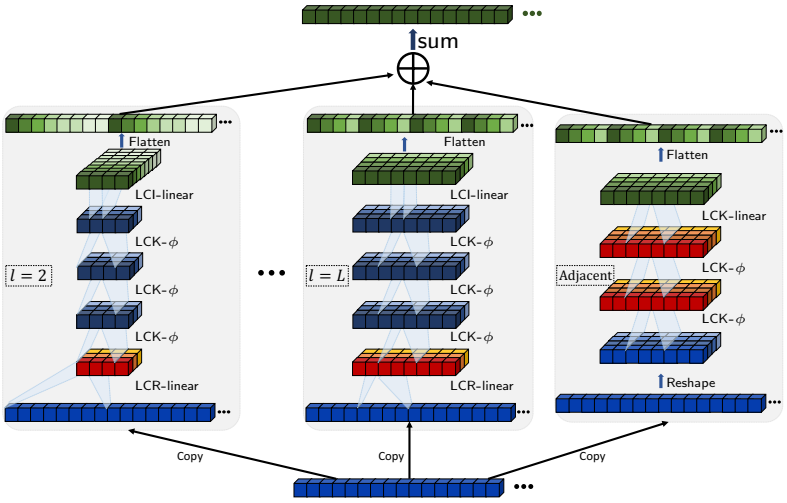


FIG. 7. *MNN architecture for nonlinear mappings, which is an extension of the NN architecture for $\mathcal{H}$-matrices Figure 5. ϕ is an activation function.*

3.1. Algorithm and architecture. NN can represent nonlinearities by choosing the activation function, ϕ, to be nonlinear, such as ReLU or sigmoid. The range of the activation function also imposes constraints on the output of the NN. For example, the range of "ReLU" in $[0, \infty)$ and the range of the sigmoid function is $[0, 1]$. Thus, the last layer is often chosen to be a linear layer to relax such constraint. Algorithm 1 is then revised to Algorithm 3, and the architecture is illustrated in Figure 7. We remark that the nonlinear activation function is only used in the LCK network. The LCR and LCI networks in Algorithm 1 are still treated as restriction and interpolation operations between coarse grid and fine grid, respectively, so we use the linear activation functions in these layers. Particularly, we also use the linear activation function for the last layer of the adjacent part, which is marked in line 13 in Algorithm 3.

As in the linear case, we calculate the number of parameters of MNN and obtain (neglecting the number of parameters in b in (2.10))

$$
\begin{aligned}
N_p^{\mathrm{MNN}} &= \sum_{\ell=2}^{L} \left(Nr + K2^{\ell} r^2 (2n_b^{(\ell)} + 1) + 2^{\ell} r \frac{N}{2^{\ell}} \right) + K2^{L} m^2 (2n_b^{(\mathrm{ad})} + 1) \\
&\leq 2LNr + K2^{L+1} r^2 \left(2 \max_{\ell=2}^{L} n_b^{(\ell)} + 1 \right) + NKm(2n_b^{(\mathrm{ad})} + 1) \\
&\leq 2N \log(N) r + 3NKm(2n_b + 1).
\end{aligned} \tag{3.2}
$$

3.2. Translation invariant case. For the linear case (2.1), if the kernel is *translation invariant*, i.e., $g(x, y) = g(x - y)$, then the matrix A is a Toeplitz matrix. Then the matrices $M^{(\ell)}$ and $A^{(\mathrm{ad})}$ are Toeplitz matrices and all matrix blocks of $U^{(\ell)}$ (resp., $V^{(\ell)}$) can be represented by the same matrix.

This leads to the *convolutional neural network* (CNN) as

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

Compared to the LC network, the only difference is that the parameters W and b are independent of i. Hence, inheriting the definition of LCR, LCK, and LCI, we define the layers CR, CK, and CI, respectively. By replacing the LC layers in Algorithm 1 by the corresponding CNN layers, we obtain the NN architecture for the translation invariant kernel.

For the nonlinear case, the translation invariant kernel for the linear case can be extended to kernels that are *equivariant to translation*, i.e., for any translation $\mathcal{T}$,

$$\mathcal{T}\mathcal{M}(v) = \mathcal{M}(\mathcal{T}v). \tag{3.4}$$

For this case, all of the LC layers in Algorithm 3 can be replaced by its corresponding CNN layers. The number of parameters of CR, CK, and CI are

$$N_p^{\mathsf{CR}} = \frac{N_x}{N_x'}\alpha', \quad N_p^{\mathsf{CK}} = \alpha\alpha' w, \quad N_p^{\mathsf{CI}} = \alpha\alpha'. \tag{3.5}$$

Thus, the number of parameters in Algorithm 3 using CNN is

$$\begin{aligned} N_{p,CNN}^{\mathrm{MNN}} &= \sum_{\ell=2}^{L}\left(r\frac{N}{2^\ell} + Kr^2(2n_b^{(\ell)}+1) + r\frac{N}{2^\ell}\right) + Km^2(2n_b^{(\mathrm{ad})}+1) \\ &\le rN + (r^2\log(N)+m^2)(2n_b+1)K. \end{aligned} \tag{3.6}$$

4. Numerical results. In this section we discuss the implementation details of MNN. We demonstrate the accuracy of the MNN architecture using two nonlinear problems: the nonlinear Schrödinger equation (NLSE), and the Kohn–Sham map (KS map) in the Kohn–Sham density functional theory (KSDFT).

4.1. Implementation. Our implementation of MNN uses Keras [14], a high-level application programming interface (API) running, in this case, on top of TensorFlow [1] (a library of tools for training NNs). The loss function is chosen as the mean squared relative error, in which the relative error is defined with respect to ℓ^2 norm as

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

where u is the target solution generated by a numerical discretization of the PDEs and u_{NN} is the predicted solution by MNN. The optimization is performed using the NAdam optimizer [16]. The weights and biases in MNN are initialized randomly from the normal distribution, and the batch size is always set between 1/100th and 1/50th of the number of train samples.

In all of the tests, the band size is chosen as $n_{b,\mathrm{ad}} = 1$, and $n_b^{(l)}$ is 2 for $l = 2$ and 3 otherwise. The nonlinear activation function is chosen as ReLU. All of the tests are run on GPU with data type `float32`. All of the numerical results are the best results by repeating the training a few times, using different random seeds. The selection of parameters r (number of channels), L ($N = 2^L m$), and K (number of layers in Algorithm 3) is problem dependent.

4.2. NLSE with inhomogeneous background potential. The NLSE is widely used in quantum physics to describe the single particle properties of the Bose–Einstein

condensation phenomenon [51, 2]. Here we study the NLSE with inhomogeneous background potential $V(x)$:

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

with period boundary condition. We aim to find its ground state denoted by $u_G(x)$. We take a strongly nonlinear case $\beta = 10$ in this work and thus consider a defocusing cubic Schrödinger equation. Due to the cubic term, an iterative method is required to solve (4.2) numerically. We employ the method in [5] for the numerical solution, which solves a time-dependent NLSE by a normalized gradient flow. The MNN is used to learn the map from the background potential to the ground state

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

This map is equivariant to translation, and thus MNN is implemented using the CNN layers. The constraints in (4.2) can be guaranteed by adding a post-correction in the network as

$$u = \text{const} \times \frac{\hat{u}}{\|\hat{u}\|_2}, \tag{4.4}$$

where $\hat{u}$ is the prediction of the NN. In the following, we study the performance of MNN on 1D and 2D cases.

4.2.1. 1D case. For the 1D case, the number of discretization points is $N = 320$, and we set $L = 6$ and $m = \frac{N}{2^L} = 5$. The potential V is chosen as

$$V = -20 \exp(I_F(v)), \tag{4.5}$$

where $v \in \mathcal{N}(0,1)^{40}$ and $I_F : \mathbb{R}^{40} \to \mathbb{R}^{320}$ is the Fourier interpolation operator. In all of the tests, the number of test samples is the same as that the number of train samples if not properly specified. We perform numerical experiments to study the behavior of MNN for a different number of channels r, a different number of CK layers K, and a different number of training samples $N_{\text{samples}}^{\text{train}}$. All of the networks are trained using 5000 epochs.

TABLE 1
Relative error in approximating the ground state of NLSE for a different number of samples $N_{\text{samples}}^{\text{train}}$ for the 1D case with $r = 8$ and $K = 7$.

$N_{\text{samples}}^{\text{train}}$	$N_{\text{samples}}^{\text{test}}$	Training error	Validation error
500	5000	8.9e-3	1.7e-3
1000	5000	9.4e-4	1.2e-3
5000	5000	8.1e-4	8.4e-4
20000	20000	8.0e-4	8.1e-4

Usually, the number of samples should be greater than that of parameters to avoid overfitting. However, in NNs it has been consistently found that the number of samples can be less than that of parameters [62, 63]. We present the numerical results for different $N_{\text{samples}}^{\text{train}}$ with $K = 7$ and $r = 8$ in Table 1. In this case, the number of parameters is $N_{\text{params}} = 18299$. The validation error is slightly larger than

TABLE 2

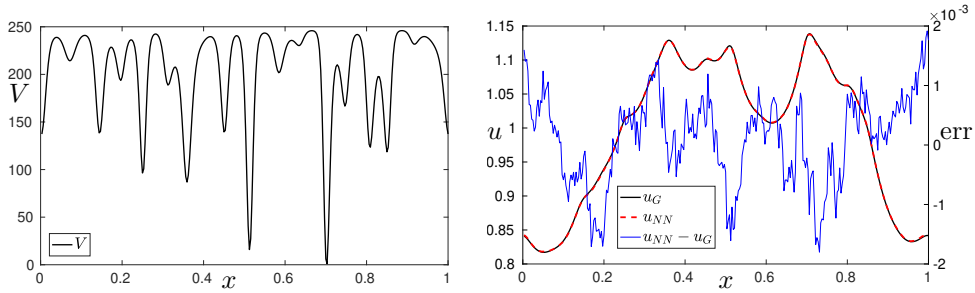
Relative error in approximating the ground state of NLSE for a different number of channels r for 1D case with $K = 7$ and $N^{\text{train}}_{\text{samples}} = 5000$.

r	N_{params}	Training error	Validation error
2	2339	2.6e-3	2.6e-3
4	5811	1.2e-3	1.2e-3
6	11131	9.9e-4	1.0e-3
8	18299	8.1e-4	8.4e-4

TABLE 3

Relative error in approximating the ground state of NLSE for a different number of CK *layers K for 1D case with $r = 8$ and $N^{\text{train}}_{\text{samples}} = 5000$.*

K	N_{params}	Training error	Validation error
1	4907	8.2e-3	8.2e-3
3	9371	1.6e-3	1.7e-3
5	13835	1.1e-3	1.1e-3
7	18299	8.1e-4	8.4e-4



(a) V (b) u_{NN}

FIG. 8. *A sample of the potential V (4.5) in the test set and its corresponding solution u_G, predicted solution u_{NN}, and its error with respect to the u_G by MNN with $r = 8$ and $K = 7$ for the 1D NLSE.*

the training error even for the case $N^{\text{train}}_{\text{samples}} = 500$ and is approaching the training error as the $N^{\text{train}}_{\text{samples}}$increases. For the case $N^{\text{train}}_{\text{samples}} = 5000$, the validation error is close to the training error; thus there is no overfitting, and the errors are only slightly larger than that when $N^{\text{train}}_{\text{samples}} = 20000$. This allows us to train MNN with $N^{\text{train}}_{\text{samples}} < N_{\text{params}}$. This feature is particularly useful for high-dimensional cases, given that in such cases $N_{\text{params}}$is usually very large and generating a large amount of samples can be prohibitively expensive.

Table 2 presents the numerical results for a different number of channels, r (i.e., the rank of the $\mathcal{H}$-matrix), with $K = 7$ and $N^{\text{train}}_{\text{samples}} = 5000$. As r increases, we find that the error first consistently decreases and then stagnates. We use $r = 8$ for the 1D NLSE below to balance between efficiency and accuracy.

Similarly, Table 3 presents the numerical results for a different number of CK layers, K, with $r = 8$ and $N^{\text{train}}_{\text{samples}} = 5000$. The error consistently decreases with respect to the increase of K, as NN can represent increasingly more complex functions with respect to the depth of the network. However, after a certain threshold, adding more layers provides very marginal gains in accuracy. In practice, $K = 7$ is a good

TABLE 4

Relative error in approximating the ground state of NLSE for the CNN with a different number of channels α and a different number of layers K with window size to be 25 for the 1D case with $N_{\text{samples}}^{\text{train}} = 5000$.

α	K	N_{params}	Training error	Validation error
8	13	18097	4.6e-3	4.6e-3
10	13	28121	3.9e-3	3.9e-3
12	13	40345	3.5e-3	3.5e-3
14	13	54769	3.8e-3	3.8e-3
12	15	47569	4.0e-3	4.0e-3

choice for the NLSE for the 1D case.

We also compare the MNN with an instance of the classical CNN. The CNN architecture used in [20] is adopted as the reference architecture. Table 4 presents the setup of the networks and the training and validation errors for CNN. Clearly, by comparing the results of MNN in Tables 2 and 3 with the results of CNN in Table 4, one can observe that the MNN not only reduces the number of parameters, but also improve the accuracy. Throughout the results shown in Tables 2 and 3, the validation errors are very close to the corresponding training errors, and thus no overfitting is observed. Figure 8 presents a sample for the potential V and its corresponding solution and prediction solution by MNN. We can observe that the prediction solution agrees with the target solution very well.

4.2.2. 2D case. For the 2D case, the number of discretization points is $n = 80 \times 80$, and we set $L = 4$ and $m = 5$. The potential V is chosen as

$$V = -20 \exp(I_F(v)), \tag{4.6}$$

where $v \in \mathcal{N}(0,1)^{10\times 10}$ and $I_F : \mathbb{R}^{10\times 10} \to \mathbb{R}^{80\times 80}$ is the 2D Fourier interpolation operator. In all of the tests the number of test data is the same as that of the train data. We perform several simulations to study the behavior of MNN for a different number of channels, r, and a different number of CK layers, K. As discussed in the 1D case, MNN allows $N_{\text{samples}}^{\text{train}} < N_{\text{params}}$. In all of the tests, the number of samples is always 20000 and the networks are trained using 5000 epochs. A numerical test shows no overfitting for the test.

TABLE 5

Relative error in approximating the ground state of NLSE for a different number of channels r for the 2D case with $K = 7$ and $N_{\text{samples}}^{\text{train}} = 20000$.

r	N_{params}	Training error	Validation error
2	45667	6.1e-3	6.1e-3
6	77515	2.4e-3	2.4e-3
10	136915	2.9e-3	2.9e-3

Tables 5 and 6 present the numerical results for a different number of channels, r, and a different number of CK layers, K, respectively. Similar to the 1D case, the choice of parameters $r = 6$ and $K = 7$ also yields accurate results in the 2D case. Figure 9 presents a sample of the potential V in the test set and its corresponding solution, prediction solution, and its error with respect to the reference solution.

4.3. Kohn–Sham map. KSDFT [32, 37] is the most widely used electronic structure theory. It requires the solution of the following set of nonlinear eigenvalue

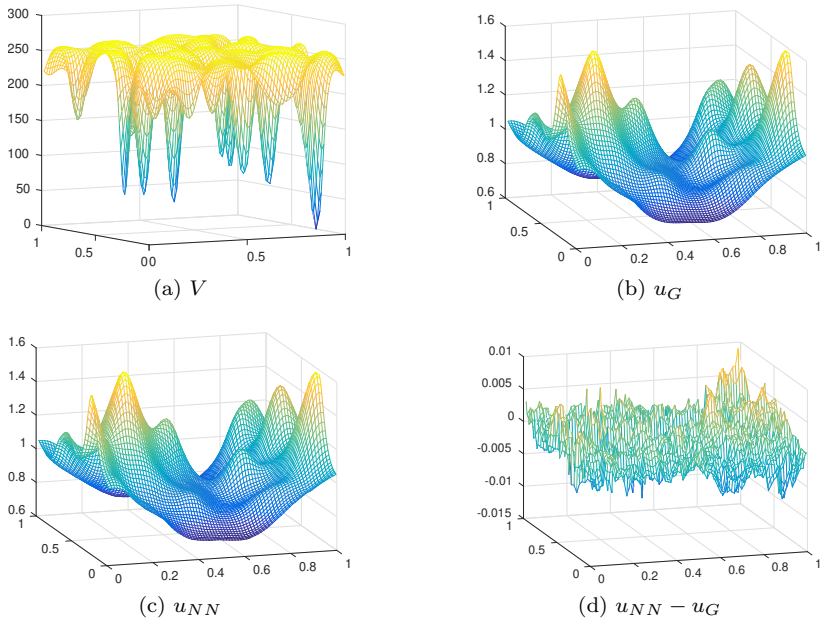


FIG. 9. *A sample of the potential V (4.6) in the test set and its corresponding solution u_G, predicted solution u_{NN} by MNN $K = 5$ and $r = 6$, and its error with respect to u_G for the 2D NLSE.*

TABLE 6
Relative error in approximating the ground state of NLSE for a different number of CK *layers K for the 2D case with $r = 6$ and $N_{\text{samples}}^{\text{train}} = 20000$.*

K	N_{params}	Training error	Validation error
3	37131	5.6e-3	5.6e-3
5	57323	3.8e-3	3.8e-3
7	77515	2.4e-3	2.4e-3

equations (real arithmetic assumed for all quantities):

$$\begin{aligned} &\left(-\frac{1}{2}\Delta + V[\rho](x)\right)\psi_i(x) = \varepsilon_i \psi_i(x), \quad x \in \Omega = [-1,1)^d, \\ &\int_\Omega \psi_i(x)\psi_j(x)\,\mathrm{d}x = \delta_{ij}, \quad \rho(x) = \sum_{i=1}^{n_e} |\psi_i(x)|^2. \end{aligned} \tag{4.7}$$

Here n_e is the number of electrons (spin degeneracy omitted), d is the spatial dimension, and δ_{ij} stands for the Kronecker delta. In addition, all eigenvalues $\{\varepsilon_i\}$ are real and ordered nondecreasingly, and $\rho(x)$ is the electron density, which satisfies the constraint

$$\rho(x) \geq 0, \quad \int_\Omega \rho(x)\,\mathrm{d}x = n_e. \tag{4.8}$$

The Kohn–Sham equations (4.7) need to be solved self-consistently, which can also be viewed as solving the following fixed point map:

$$\rho = \mathcal{F}_{\text{KS}}[V[\rho]]. \tag{4.9}$$

TABLE 7

Relative error on the approximation of the KS map for different r, with $K = 6$, $N^{\text{train}}_{\text{samples}} = 16000$, and $N^{\text{test}}_{\text{samples}} = 4000$.

r	N_{params}	Training error	Validation error
2	2117	6.7e-4	6.7e-4
4	5183	3.3e-4	3.4e-4
6	9833	2.8e-4	2.8e-4
8	16067	3.3e-4	3.3e-4
10	33013	1.8e-4	1.9e-4

Here the mapping $\mathcal{F}_{\text{KS}}[\cdot]$ from V to ρ is called the KS map, which for a fixed potential is reduced to a linear eigenvalue problem, and it constitues the most computationally intensive step for solving (4.7). We seek to approximate the KS map using an MNN, whose output was regularized so that it satisfies (4.8).

In the following numerical experiments the potential, V, is given by

$$V(x) = -\sum_{i=1}^{n_g} \sum_{j \in \mathbb{Z}^d} c_i \exp\left(-\frac{(x - r_i - 2j)^2}{2\sigma^2}\right), \qquad x \in [-1, 1)^d, \tag{4.10}$$

where d is the dimension and $r_i \in [-1, 1)^d$. We set $\sigma = 0.05$ or $\sigma = 0.15$ for one dimension and $\sigma = 0.2$ for two dimensions. The coefficients c_i are randomly chosen following the uniform distribution $\mathcal{U}([0.8, 1.2])$. The centers of the Gaussian wells, r_i, are chosen randomly under the constraint that $|r_i - r_{i'}| > 4\sigma$, unless explicitely specified. The KS map is discretized using a pseudo-spectral method [59], and solved by a standard eigensolver.

4.3.1. 1D case. We set $\sigma = 0.05$ and we generated four data sets using a different number of wells, n_g, which in this case is also equal to the number of electrons n_e, ranging from 2 to 8.

The number of discretization points is $N = 320$. We trained the architecture defined in section 3 for each n_g, setting the number of levels $L = 6$, using different values for r and K.

Table 7 shows that there is no overfitting, even at this level of accuracy and number of parameters. This behavior is found in all of the numerical examples; thus we only report the test error in what follows.

From Table 8 we can observe that as we increase r, the error decreases sharply. Figure 10 depicts this behavior. In Figure 10 we have that if $r = 2$, then the network output, ρ_{NN}, fails to approximate ρ accurately; however, by modestly increasing r, the network is able to accurately approximate ρ.

However, the accuracy of the network stagnates rapidly. In fact, increasing r beyond 10 does not provide any considerable gains. In addition, Table 8 shows that the accuracy of the network is agnostic to the number of Gaussian wells present in the system.

In addition, we studied the relation between the quality of the approximation and K. We fixed $r = 6$, and we trained several networks using different values of K, ranging from 2, i.e., a very shallow network, to 10. The results are summarized in Table 9. We can observe that the error decreases sharply as the depth of the network increases, but it rapidly stagnates as K becomes large.

The KS map is a very nonlinear mapping, and we demonstrate below that the nonlinear activation functions in the network play a crucial role. Consider a linear

TABLE 8
Relative test error on the approximation of the KS map for different ranks r, with fixed $K = 6$ and $N^{\text{train}}_{\text{samples}} = 16000$.

$n_g \backslash r$	2	4	6	8	10
2	6.7e-4	3.3e-4	2.8e-4	3.3e-4	1.8e-4
4	8.5e-4	4.1e-4	2.9e-4	3.8e-4	2.4e-4
6	6.3e-4	4.8e-4	3.8e-4	4.0e-4	4.2e-4
8	1.2e-3	5.1e-4	3.7e-4	4.5e-4	3.7e-4

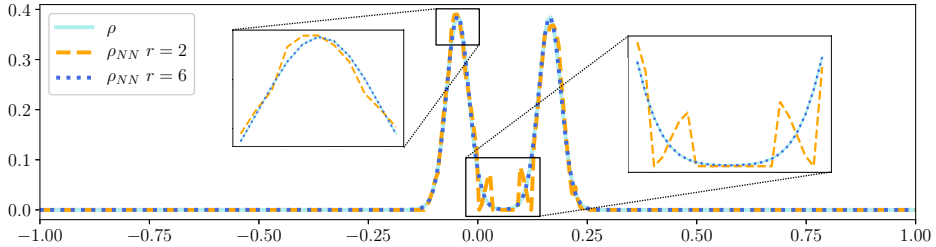


FIG. 10. *Estimation using two different multiscale networks with $r = 2$ and $r = 6$; with $K = 6$, and $L = 5$ fixed.*

network, in which we have completely eliminated the nonlinear activation functions. This linear network can be easily implemented by setting $K = 1$ and $\phi(x) = x$. We then train the resulting linear network using the same data as before and following the same optimization procedure. Table 10 shows that the approximation error can be very large, even for relatively small n_g.

In addition, we note that the favorable behavior of MNN with respect to n_g shown earlier is rooted in the fact that the band gap (here the band gap is equal to $\epsilon_{n_g+1} - \epsilon_{n_g}$) remains approximately the same as we increase n_g. According to the density functional perturbation theory (DFPT), the KS map is relatively insensitive to the change of the external potential. This setup mimics an insulating system. On the other hand, we may choose σ in (4.10) to be 0.15, and relax the constraint between Gaussian centers to $|r_i - r_{i'}| > 2\sigma$. The rest of the coefficients are randomly chosen using the same distributions as before. In this case, we generate a new data set, in which the average gap for the generated data sets is 0.2, 0.08, 0.05, and 0.03 for n_g equal to 2, 4, 6, and 8, respectively. The decrease of the band gap with respect to the increase of n_g resembles the behavior of a metallic system, in which the KS map becomes more sensitive to small perturbations of the potential. After generating the samples, we trained two different networks: the MNN with $K = 6$, $r = 10$ and a regular CNN with 15 layers, 10 channels, and window size 13. The results are shown in Tables 11 and 12. From Tables 11 and 12 we can observe that MNN outperforms CNN and that as the band gap decreases, the performance gap between MNN and CNN widens. We point out that it is possible to partially alleviate this adverse dependence on the band gap in the MNN by introducing a nested hierarchical structure to the interpolation and restriction operators as shown in [20].

TABLE 9

Relative test error on the approximation of the KS map for different K and fixed rank $r = 6$, and $N^{\text{train}}_{\text{samples}} = 16000$.

$n_g \backslash K$	2	4	6	8	10
2	1.4e-3	3.1e-4	2.8e-4	3.5e-4	2.3e-4
4	1.9e-3	5.8e-4	2.8e-4	6.0e-4	7.1e-4
6	2.1e-3	7.3e-4	3.8e-4	6.7e-4	6.7e-4
8	2.0e-3	8.8e-4	3.7e-4	6.7e-4	6.8e-4

TABLE 10

Relative error in approximating the KS map for a different number of Gaussians n_g for one dimension using a linear MNN network with $K = 1$, $r = 10$, and $N^{\text{train}}_{\text{samples}} = 16000$.

n_g	N_{params}	Training error	Validation error
2	8827	9.5e-2	9.5e-2
4	8827	9.7e-2	9.5e-2
6	8827	9.7e-2	9.7e-2
8	8827	8.7e-2	8.6e-2

TABLE 11

Relative error in approximating the KS map for a different number of Gaussians n_g for one dimension using an MNN network with $K = 6$, $r = 10$, and $N^{\text{train}}_{\text{samples}} = 16000$.

n_g	N_{params}	Training error	Validation error
2	20890	1.5e-3	1.9e-03
4	20890	7.1e-3	8.8e-03
6	20890	1.6e-2	2.3e-02
8	20890	2.9e-2	2.9e-02

TABLE 12

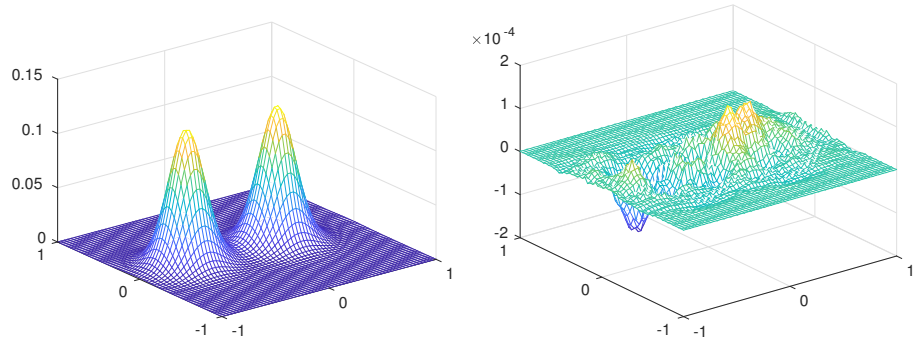
Relative error in approximating the KS map for a different number of Gaussians n_g for one dimension using a periodic CNN with 15 layers, 10 channels, windows size 13, and $N^{\text{train}}_{\text{samples}} = 16000$.

n_g	N_{params}	Training error	Validation error
2	19921	1.6e-3	1.9e-3
4	19921	1.7e-2	1.8e-2
6	19921	6.2e-2	6.5e-2
8	19921	9.0e-2	9.3e-2

4.3.2. 2D case. The discretization is the standard extension to the 2D case using tensor products, using a 64×64 grid. In this case we only used $n_g = 2$ and followed the same number of training and test samples as in the 1D case. We fixed $K = 6$, $L = 4$, and trained the network for a different number of channels, r. The results are displayed in Table 13, which shows the same behavior as for the 1D case, in which the error decays sharply and then stagnates, and there is no overfitting. In particular, the network is able to effectively approximate the KS map as shown in Figure 11. Figure 11a shows the output of the NN for a test sample, and Figure 11b shows the approximation error with respect to the reference.

TABLE 13
Relative errors on the approximation of the KS map for the 2D case for different r and $K = 6$, $N_{\text{samples}}^{\text{train}} = 16000$, and $N_{\text{samples}}^{\text{test}} = 4000$.

r	Training error	Validation error
4	5.2e-3	5.2e-3
6	1.6e-3	1.7e-3
8	1.2e-3	1.1e-3
10	9.1e-4	9.3e-4



(a) ρ_{NN} (b) $|\rho_{NN} - \rho|$

FIG. 11. (a) *Output of the trained network on a test sample for $K = 6$ and $\alpha = 10$;* (b) *error with respect to the reference solution.*

5. Conclusion. We have developed a multiscale neural network (MNN) architecture for approximating nonlinear mappings, such as those arising from the solution of integral equations (IEs) or partial differential equations (PDEs). In order to control the number of parameters, we first rewrite the widely used hierarchical matrix into the form of a NN, which mainly consists of three subnetworks: restriction network, kernel network, and interpolation network. The three subnetworks are all linear and correspond to the components of a singular value decomposition. We demonstrate that such structure can be directly generalized to nonlinear problems, simply by replacing the linear kernel network by a multilayer kernel network with nonlinear activation functions. Such "nonlinear singular value decomposition operation" is performed at different spatial scales, which can be efficiently implemented by a number of locally connected (LC) networks, or convolutional NNs (CNN) when the mapping is equivariant to translation. Using the parameterized nonlinear Schrödinger equation and the Kohn–Sham map as examples, we find that MNN can yield accurate approximation to such nonlinear mappings. When the mapping has N degrees of freedom, the complexity of MNN is only $O(N \log N)$. Thus the resulting MNN can be further used to accelerate the evaluation of the mapping, especially when a large number of evaluations are needed within a certain range of parameters.

In this work, we only provide one natural architecture of MNN based on hierarchical matrices. The architecture can be altered depending on the target application. Some of the possible modifications and extensions are listed as follows. (1) In this work, the NN is inspired by a hierarchical matrix with a special case of strong admissible condition. One can directly construct architectures for $\mathcal{H}$-matrices with the weak admissible condition, as well as other structures such as the fast multiple meth-

ods, $\mathcal{H}^2$-matrices, and wavelets. (2) The input, u, and output, v, in this work are periodic. The network can be directly extended to the nonperiodic case, by replacing the periodic padding in LCK by some other padding functions. One may also explore the mixed usage of LC networks and CNNs in different components of the architecture. (3) The matrices $A^{(\mathrm{ad})}$ and $M^{(\ell)}$ can be block-partitioned in different ways, which would result in different setups of parameters in the LCK layers. (4) The LCR and LCI networks in Algorithm 3 can involve nonlinear activation functions as well and can be extended to networks with more than one layer. (5) The LCK network in Algorithm 3 can be replaced by other architectures. In principle, for each scale, these LCK layers can be altered to any network, for example, the sum of two parallel subnetworks, or the ResNet structure [30]. (6) It is known that $\mathcal{H}$-matrices can well approximate smooth kernels but become less efficient for highly oscillatory kernels, such as those arising from the Helmholtz operator in the high frequency regime. The range of applicability of the MNN remains to be studied both theoretically and numerically.

Acknowledgment. The authors thank Yuehaw Khoo for constructive discussions.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: A system for large-scale machine learning*, in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, 2016, pp. 265–283.

[2] J. R. Anglin and W. Ketterle, *Bose–Einstein condensation of atomic gases*, Nature, 416 (2002), pp. 211–218.

[3] M. Araya-Polo, J. Jennings, A. Adler, and T. Dahlke, *Deep-learning tomography*, The Leading Edge, 37 (2018), pp. 58–66, https://doi.org/10.1190/tle37010058.1.

[4] V. Badrinarayanan, A. Kendall, and R. Cipolla, *SegNet: A deep convolutional encoder-decoder architecture for image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 39 (2017), pp. 2481–2495.

[5] W. Bao and Q. Du, *Computing the ground state solution of Bose–Einstein condensates by a normalized gradient flow*, SIAM J. Sci. Comput., 25 (2004), pp. 1674–1697, https://doi.org/10.1137/S1064827503422956.

[6] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera, *An "empirical interpolation" method: Application to efficient reduced-basis discretization of partial differential equations*, C. R. Math. Acad. Sci. Paris, 339 (2004), pp. 667–672, https://doi.org/10.1016/j.crma.2004.08.006.

[7] C. Beck, W. E, and A. Jentzen, *Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations*, J. Nonlinear Sci., 29 (2017), pp. 1563–1619.

[8] J. Berg and K. Nyström, *A unified deep artificial neural network approach to partial differential equations in complex geometries*, Neurocomputing, 317 (2018), pp. 28–41.

[9] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem., 27 (2003), pp. 405–422.

[10] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

[11] J. Bruna and S. Mallat, *Invariant scattering convolution networks*, IEEE Trans. Pattern Anal. Mach. Intell., 35 (2013), pp. 1872–1886.

[12] P. Chaudhari, A. Oberman, S. Osher, S. Soatto, and G. Carlier, *Partial differential equations for training deep neural networks*, in Proceedings of the 2017 51st Asilomar Conference on Signals, Systems, and Computers, IEEE, 2017, pp. 1627–1631, https://doi.org/10.1109/ACSSC.2017.8335634.

[13] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully*

connected CRFs, IEEE Trans. Pattern Anal. Mach. Intell., 40 (2018), pp. 834–848, https://doi.org/10.1109/TPAMI.2017.2699184.

[14] F. Chollet et al., *Keras*, 2015, https://keras.io.

[15] N. Cohen, O. Sharir, and A. Shashua, *On the expressive power of deep learning: A tensor analysis*, in Proceedings of the 29th Annual Conference on Learning Theory, 2016, pp. 698–728.

[16] T. Dozat, *Incorporating Nesterov Momentum into Adam*, International Conference on Learning Representations, in Proceedings of the 4th International Conference on Learning Representations, Workshop Track, 2016.

[17] W. E, J. Han, and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, Commun. Math. Stat., 5 (2017), pp. 349–380, https://doi.org/10.1007/s40304-017-0117-6.

[18] Y. Efendiev, J. Galvis, G. Li, and M. Presho, *Generalized multiscale finite element methods. Nonlinear elliptic equations*, Commun. Comput. Phys., 15 (2014), pp. 733–755, https://doi.org/10.4208/cicp.020313.041013a.

[19] Y. Efendiev and T. Hou, *Multiscale finite element methods for porous media flows and their applications*, Appl. Numer. Math., 57 (2007), pp. 577–596, https://doi.org/10.1016/j.apnum.2006.07.009.

[20] Y. Fan, J. Feliu-Fabà, L. Lin, L. Ying, and L. Zepeda-Núñez, *A multiscale neural network based on hierarchical nested bases*, Res. Math. Sci., 6 (2019), 21.

[21] M. Fenn and G. Steidl, *FMM and $\mathcal{H}$-Matrices: A Short Introduction to the Basic Idea*, Tech. Report TR-2002-008, Department for Mathematics and Computer Science, University of Mannheim, Mannheim, Germany, 2002, https://ub-madoc.bib.uni-mannheim.de/744/1/TR-02-008.pdf.

[22] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.

[23] M. A. Grepl, Y. Maday, N. C. Nguyen, and A. T. Patera, *Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations*, M2AN Math. Model. Numer. Anal., 41 (2007), pp. 575–605, https://doi.org/10.1051/m2an:2007031.

[24] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part* I*: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[25] W. Hackbusch, L. Grasedyck, and S. Börm, *An introduction to hierarchical matrices*, Math. Bohem., 127 (2002), pp. 229–241.

[26] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic: General complexity estimates*, J. Comput. Appl. Math., 125 (2000), pp. 479–501.

[27] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 8505–8510.

[28] K. He and J. Sun, *Convolutional neural networks at constrained time cost*, in Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 5353–5360.

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 770–778.

[30] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2016, pp. 770–778.

[31] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*, IEEE Signal Processing Mag., 29 (2012), pp. 82–97, https://doi.org/10.1109/MSP.2012.2205597.

[32] P. Hohenberg and W. Kohn, *Inhomogeneous electron gas*, Phys. Rev. (2), 136 (1964), pp. B864–B871.

[33] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, 4 (1991), pp. 251–257, https://doi.org/10.1016/0893-6080(91)90009-T.

[34] Y. Khoo, J. Lu, and L. Ying, *Solving Parametric PDE Problems with Artificial Neural Networks*, preprint, https://arxiv.org/abs/1707.03351, 2017.

[35] V. Khrulkov, A. Novikov, and I. Oseledets, *Expressive Power of Recurrent Neural Networks*, preprint, https://arxiv.org/abs/1711.00811, 2017.

[36] N. Kishore Kumar and J. Schneider, *Literature survey on low rank approximation of matrices*, Linear Multilinear Algebra, 65 (2017), pp. 2212–2244.

[37] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Phys. Rev. (2), 140 (1965), pp. A1133–A1138.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, in Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12, Curran Associates, Red Hook, NY, 2012, pp. 1097–1105.
[39] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.
[40] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, *Deep learning of the tissue-regulated splicing code*, Bioinformatics, 30 (2014), pp. i121–i129, https://doi.org/10.1093/bioinformatics/btu277.
[41] Y. Li, X. Cheng, and J. Lu, *Butterfly-Net: Optimal Function Representation Based on Convolutional Neural Networks*, preprint, https://arxiv.org/abs/1805.07451, 2018.
[42] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix–vector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.
[43] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. W. M. van der Laak, B. van Ginneken, and C. I. Sánchez, *A survey on deep learning in medical image analysis*, Med. Image Anal., 42 (2017), pp. 60–88, https://doi.org/10.1016/j.media.2017.07.005.
[44] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, *Deep neural nets as a method for quantitative structure-activity relationships*, J. Chem. Inf. Model., 55 (2015), pp. 263–274, https://doi.org/10.1021/ci500747n.
[45] Y. Maday, O. Mula, and G. Turinici, *Convergence analysis of the generalized empirical interpolation method*, SIAM J. Numer. Anal., 54 (2016), pp. 1713–1731, https://doi.org/10.1137/140978843.
[46] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed., Academic Press, Boston, 2009, https://doi.org/10.1016/B978-0-12-374370-1.50001-9.
[47] M. Mathieu, C. Couprie, and Y. LeCun, *Deep Multi-Scale Video Prediction Beyond Mean Square Error*, preprint, http://arxiv.org/abs/1712.04741, 2016.
[48] H. Mhaskar, Q. Liao, and T. Poggio, *Learning Functions: When Is Deep Better Than Shallow*, preprint, https://arxiv.org/abs/1603.00988, 2016.
[49] P. Paschalis, N. D. Giokaris, A. Karabarbounis, G. Loudos, D. Maintas, C. Papanicolas, V. Spanoudaki, C. Tsoumpas, and E. Stiliaris, *Tomographic image reconstruction using artificial neural networks*, in Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 527 (2004), pp. 211–215, https://doi.org/10.1016/j.nima.2004.03.122.
[50] D. M. Pelt and J. A. Sethian, *A mixed-scale dense convolutional neural network for image analysis*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 254–259, https://doi.org/10.1073/pnas.1715832114.
[51] L. Pitaevskii, *Vortex lines in an imperfect Bose gas*, J. Exp. Theor. Phys., 13 (1961), pp. 451–454.
[52] M. Raissi, *Forward-Backward Stochastic Neural Networks: Deep Learning of High-Dimensional Partial Differential Equations*, preprint, https://arxiv.org/abs/1804.07010, 2018.
[53] M. Raissi and G. E. Karniadakis, *Hidden physics models: Machine learning of nonlinear partial differential equations*, J. Comput. Phys., 357 (2018), pp. 125–141, https://doi.org/10.1016/j.jcp.2017.11.039.
[54] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional networks for biomedical image segmentation*, in Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds., Springer, Cham, 2015, pp. 234–241.
[55] K. Rudd, G. D. Muro, and S. Ferrari, *A constrained backpropagation approach for the adaptive solution of partial differential equations*, IEEE Trans. Neural Netw. Learn. Syst., 25 (2014), pp. 571–584, https://doi.org/10.1109/TNNLS.2013.2277601.
[56] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks, 61 (2015), pp. 85–117, https://doi.org/10.1016/j.neunet.2014.09.003.
[57] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364.
[58] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., Curran Associates, Red Hook, NY, 2014, pp. 3104–3112.
[59] L. N. Trefethen, *Spectral Methods in MATLAB*, SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719598.
[60] D. Ulyanov, A. Vedaldi, and V. Lempitsky, *Deep image prior*, in Proceedings of the IEEE

Conference on Computer Vision and Pattern Recognition, IEEE, 2018, pp. 9446–9454.
[61] Y. Wang, C. W. Siu, E. T. Chung, Y. Efendiev, and M. Wang, *Deep Multiscale Model Learning*, preprint, https://arxiv.org/abs/1806.04830, 2018.
[62] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, *Understanding Deep Learning Requires Rethinking Generalization*, preprint, https://arxiv.org/abs/1611.03530, 2016.
[63] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, *Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising*, IEEE Trans. Image Process., 26 (2017), pp. 3142–3155.
[64] L. Zhang, J. Han, H. Wang, R. Car, and W. E, *DeePCG: Constructing coarse-grained models via deep neural networks*, J. Chem. Phys., 149 (2018), 034101.
[65] L. Zhang, H. Wang, and W. E, *Adaptive coupling of a deep neural network potential to a classical force field*, J. Chem. Phys., 149 (2018) 154107.