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.

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 multi-grid methods [7], the fast multipole method [17], wavelets [38], and hierarchical matrices [19, 6], 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 algorithms with a prohibitive computational cost. The development of algorithms and heuristics to reduce the cost is an area of active research [4, 18, 37, 15, 14]. However, in general, each method is application-

*Submitted to the editors July 4, 2018.

Funding: The work of Y.F. and L.Y. is partially supported by the U.S. Department of Energys Advanced Scientific Computing Research program under award DE-FC02-13ER26134/DE-SC0009409 and the National Science Foundation under award DMS-1521830. 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.

†Department of Mathematics, Stanford University, Stanford, CA 94305 (ywfan@stanford.edu, http://web.stanford.edu/~ywfan/)

‡Department of Mathematics, University of California, Berkeley, and Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (linlin@math.berkeley.edu, https://math.berkeley.edu/~linlin/)

§Department of Mathematics and Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305 (lexing@stanford.edu, https://web.stanford.edu/~lexing/)

¶Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (lzepeda@lbl.gov, http://math.mit.edu/~lzepeda/)

dependent, and requires a deep understanding of the physics behind.

On the other hand, the surge of interest in machine learning methods, in particular, deep neural networks, has dramatically improved speech recognition [25], visual object recognition [31], object detection, etc. This has fueled breakthroughs in many domains such as drug discovery [36], genomics [33], and automatic translation [45], among many others. Several recent reviews include [32] and [43]. 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 [27, 39, 11], *i.e.*, they can approximate arbitrarily well any function with mild regularity conditions, how to build such approximation efficiently 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 [22, 23, 39].

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 map, issued from an underlying physical problem, described by either an integral equation or a partial differential equation, 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. To this end, 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 structured short and wide matrix. Then we *process* the encoded vector by multiplying it by a structured square matrix. Then we bring the vector to its original size, or *interpolate* it, by multiplying it with a structured tall and skinny matrix. Such operations are performed at different spatial scales separately. The contributions from all spatial scales, together with the near-field contribution represented by a near-diagonal matrix, are added together to obtain the final approximation to the matrix-vector multiplication. Since every step is linear, the overall operation is also a linear mapping. For nonlinear problems, such an interpretation allows us to directly generalize the $\mathcal{H}$-matrix 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*, maintains a relatively modest amount of parameters even for large problems.

We demonstrate the performance of the multiscale neural network by approximating the solution to the nonlinear Schrödinger equation [40, 2], as well as the Kohn-Sham map [26, 30]. These mappings are highly nonlinear, and yet can be 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 extremely vast, the application of deep learning to numerical analysis problems is relatively new, though

that seems to be rapidly changing. Several works have used deep neural networks to solve PDEs [5, 42, 9, 13, 44, 34]. A combination of machine learning and multiscale methods was applied by Chan, and Elsheikh using a multiscale finite elements with a response trained using a neural network [8]. For general applications of machine learning to nonlinear numerical analysis problems the work of Raissi and Karnidiakis uses machine learning, in particular, Gaussian processes, to find parameters in nonlinear equations [41]; Khoo, Lu and Ying used neural network in the context of uncertainty quantification [28]; Zhang et al used neural network in the context of generating high-quality interatomic potentials for molecular dynamics [51, 50]. Wang et al. used multiscale model reduction methods coupled with deep neural networks [47].

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 neural networks. Section 3 extends the neural network 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 neural networks. For the sake of clarity, we succinctly review the structure of $\mathcal{H}$-matrices for the one dimensional case in subsection 2.1. We interpret $\mathcal{H}$-matrices using the framework of neural networks in subsection 2.2, and then extend it to the multi-dimensional 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 [19, 21, 20] 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 integral equations (IE) and partial differential equations (PDE) [6]. In the sequel, we follow the notation in [35] to provide a brief introduction to the framework of $\mathcal{H}$-matrices in a simple uniform and Cartesian setting. The interested readers are referred to [19, 6, 35] 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 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 0-th level of the decomposition, which corresponds to 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 ($0 \le \ell \le L$), 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 $i = 1, \ldots, 2^\ell$. Throughout this manuscript, $\mathcal{I}^{(\ell)}$(or $\mathcal{J}^{(\ell)}$) denote 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, have two child segments.

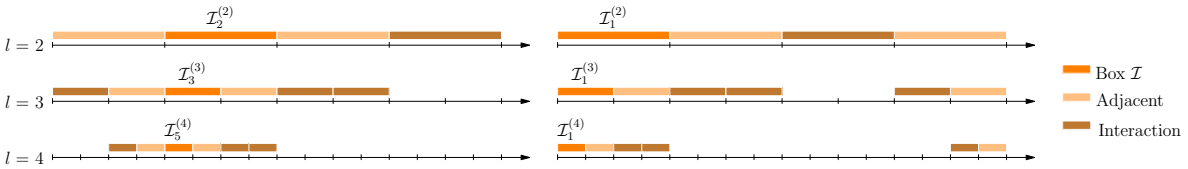


Figure 1: Illustration of the computational domain at level $\ell = 2, 3, 4$. The left and right figures represent an interior segment and a boundary segment and their adjacent and interaction list at different levels.

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 $\mathcal{I} \in \mathsf{IL}(\mathcal{J})$.

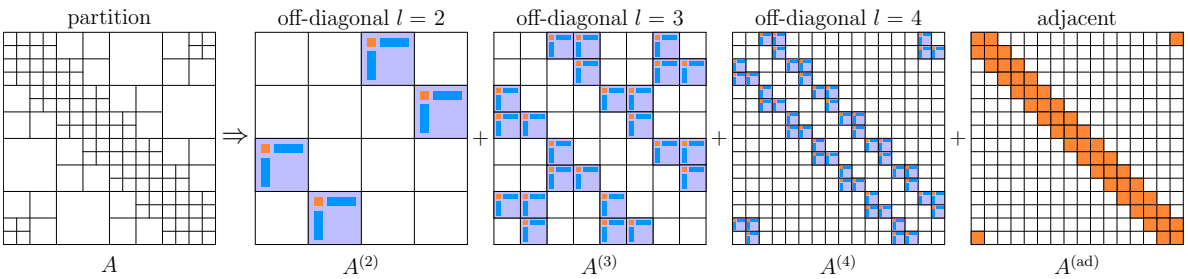


Figure 2: Partition of the matrix A for $L = 4$ and nonzero blocks of $A^{(\ell)}$, $\ell = 2, 3, 4$ (colored blue) and $A^{(\mathrm{ad})}$ (colored orange). Nonzero blocks of $A^{(\ell)}$ are approximated by low rank approximation and nonzero blocks of $A^{(\mathrm{ad})}$ are retained.

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

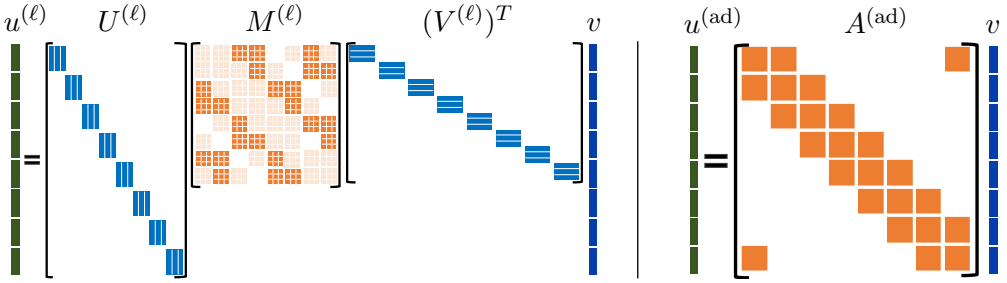
$$A = \sum_{\ell=2}^{L} A^{(\ell)} + A^{(\mathrm{ad})}, \tag{2.4}$$

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

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

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



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

Figure 3: Diagram of matrix-vector multiplication (2.7) of the low-rank part and 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 non-zero blocks, the matrices $M^{(\ell)}$ are block cyclic band matrices.

The key idea behind $\mathcal{H}$-matrices is to approximate the nonzero blocks $A^{(\ell)}$ by a low rank approximation (see [29] for a thorough review). This idea is depicted in Figure 2, in which each non-zero 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 [16], 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 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 neural network.

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*

1. $U^{(\ell)}$ *and* $V^{(\ell)}$, $\ell = 2, \cdots, L$ *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* $m \times m$;
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* $\ell \geq 3$, *if we treat all the pale orange colored blocks of* $M^{(\ell)}$ *in Figure* 3a *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 list were defined using the *strong admissible condition* in $\mathcal{H}$-matrices. Other conditions can be certainly used, such as the *weak admissibility condition*, leading to similar structures.

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

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

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

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

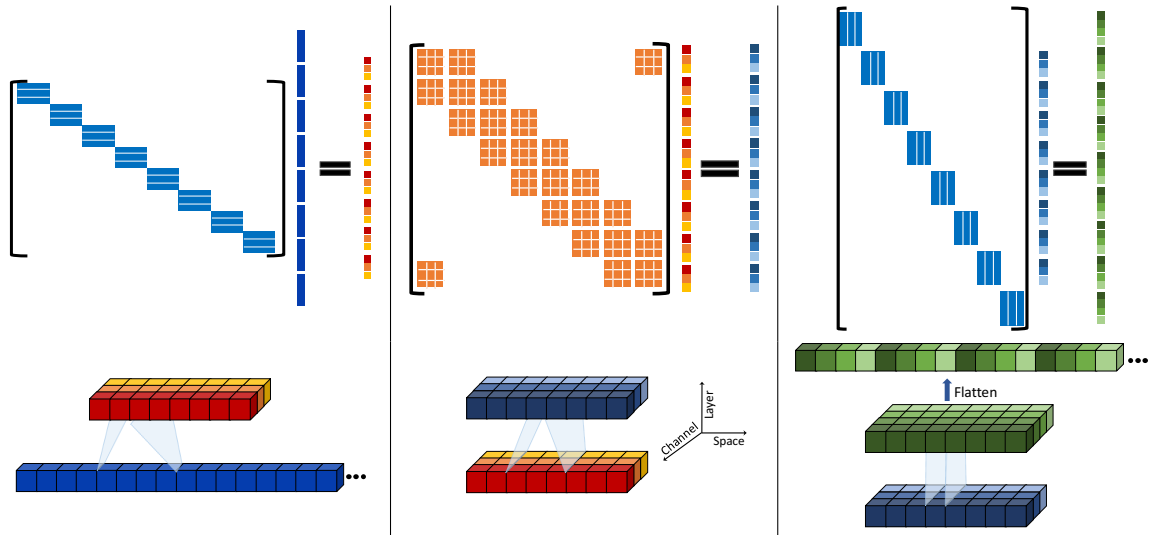
where for each $k = 1, ..., K$ we have that $\xi^{(k)}, b^{(k)} \in \mathbb{R}^{n_k}, W^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}$. Following the terminology of machine learning, ϕ is called the *activation function* that is applied entry-wise, $W^{(k)}$ are the *weights*, $b^{(k)}$ are the *biases*, and $\xi^{(k)}$ is the k-th 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 its 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 neural networks. The different factors in (2.7) possess a distinctive structure, which we aim to exploit by using locally connected (LC) network. LC networks are propagating functions whose weights have a block-banded constraint. For the one-dimensional example, we also treat ξ as a 2-tensor of dimensions $\alpha \times N_x$, where α is the *channel dimension* and N_x is the *spatial dimension*, and ζ be a 2-tensor of dimensions $\alpha' \times N'_x$. We say that ξ is connected to ζ by a LC networks if

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

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



$s = w = \frac{N_x}{N'_x}$, $\alpha = 1$

(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 $\alpha' = 4$

Figure 4: Three instances of locally connected networks used the represent the matrix-vector multiplication in (2.7). The upper portions of each column depicts the patterns of the matrices and the lower portions are their respective analogues using locally connect networks.

Each LC network requires 6 parameters, N_x, α, N'_x, α', w and s to be characterized. Next, we define three types of LC network by specifying some of their parameters,

LCR *Restriction* network: we set $s = w = \frac{N_x}{N'_x}$ and $\alpha = 1$ in LC. 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 4b. 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 LC. 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. Neural network representation. Following (2.7), in order to construct the neural network (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

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

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) to a vector and reshape it back to 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. 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 Flatten is defined as the inverse of Reshape. Using Property 1.2, we can write (2.11d) as

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

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 neural network framework of (2.7) (see Figure 3) using the building blocks depicted in Figure 4.

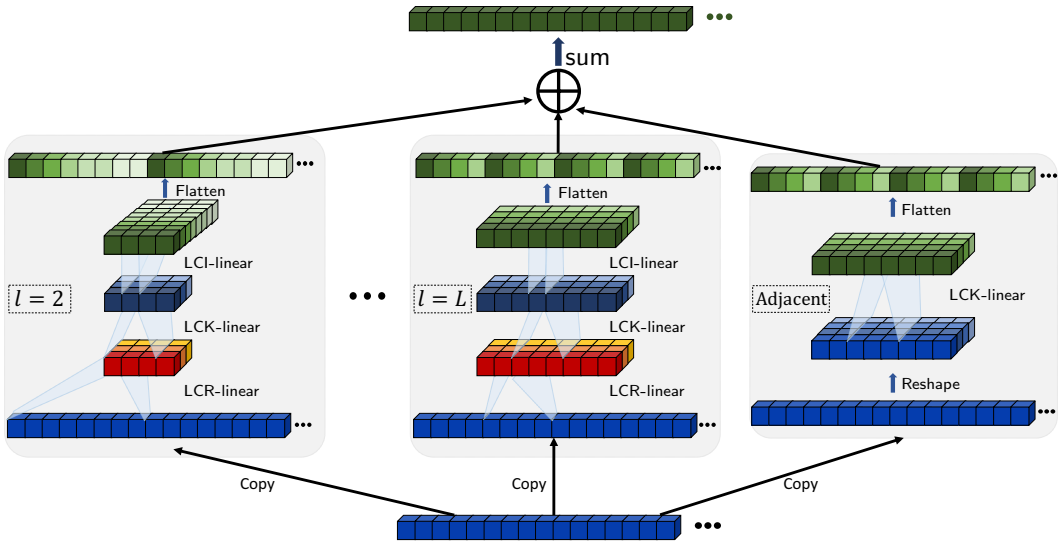


Figure 5: Neural network architecture for $\mathcal{H}$-matrices.

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

Moreover, the memory footprints of the neural network architecture and $\mathcal{H}$-matrices are asymptotically the same with respect 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:

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

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

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

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

2.3. Multi-dimensional case. Following the previous section, the extension of Algorithm 1 to the d-dimensional case can be easily deduced using the tensor product of one-dimensional cases. Consider $d = 2$ below for instance, and the generalization to higher dimensional case will be straight-forward. Suppose that we have an IE in 2D 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 in 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 3 boxes for 1D case, 3^2 boxes for 2D case. Similarly, the decomposition (2.4) on the matrix A can easily to extended for this case. Following the structure of $\mathcal{H}$-matrices, the off-diagonal blocks of $A^{(\ell)}$ can be approximated as

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

As mentioned before, we can describe the network using tensor 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 2-dimensional index $i = (i_1, i_2)$. Using the tensor notations, $U^{(\ell)}$, $V^{(\ell)}$ in (2.7), can be treated as 3-tensors of dimension $N \times N \times r$. We generalize the notion of band matrix to *band tensors*. A band tensor T satisfies that

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

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

Property 2.
1. *The 3-tensor $U^{(\ell)}$ and $V^{(\ell)}$, $\ell = 2, \cdots, L$ are block diagonal tensors with block size $N/2^\ell \times N/2^\ell \times r$;*
2. *the 4-tensor $A^{(\mathrm{ad})}$ is a block band cyclic tensor with block size $m \times m \times m \times m$ and band size $n_b^{(\mathrm{ad})} = (1,1)$;*
3. *the 4-tensors $M^{(\ell)}$, $\ell = 2, \cdots, L$ are block band cyclic tensor with block size $r \times r$ 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 2D can be represented by a 3-tensor of 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 a locally connected layer ζ with size $\alpha' \times N'_{x,1} \times N'_{x,2}$, then

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

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 four operations in (2.11) which constitute the building blocks of the neural network. The extension is trivial, the parameters N_x, s and w in the one-dimensional LC networks are replaced by their 2-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.

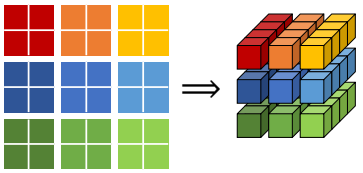


Figure 6: Diagram of $\mathsf{Reshape}[2^2, 3, 3]$ in Algorithm 2.

Using the notations above we extend Algorithm 1 to the 2D case in Algorithm 2. We crucially remark that the $\mathsf{Reshape}[r^2, n_1, n_2]$ function in Algorithm 2 is not the usual major column based reshaping. It reshapes a 2-tensor T with size $rn_1 \times rn_2$ to 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 $\mathsf{Reshape}[2^2, 3, 3]$. 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 neural network. In this section, we extend the aforementioned NN architecture to represent a general nonlinear mapping 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 one-dimensional case below for simplicity, and the

generalization to higher dimensions follows directly as in subsection 2.3.

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

1: u $= 0$;
2: **for** $\ell = 2$ to L **do**
3: $\quad \xi_0 = \mathsf{LCR}[\mathsf{linear}; N, 2^\ell, r](v)$;
4: $\quad$ **for** $k = 1$ to K **do**
5: $\quad\quad \xi_k = \mathsf{LCK}[\phi; 2^\ell, r, r, 2n_b^{(\ell)} + 1](\xi_{k-1})$;
6: $\quad$ **end for**
7: $\quad u = u + \mathsf{LCI}[\mathsf{linear}; 2^\ell, r, \frac{N}{2^\ell}](\xi_K)$;
8: **end for**
9: $\xi_0 = \mathsf{Reshape}[m, 2^L](v)$;
10: **for** $k = 1$ to K **do**
11: $\quad \xi_k = \mathsf{LCK}[\phi; 2^L, m, m, 2n_b^{(\mathrm{ad})} + 1](\xi_{k-1})$;
12: **end for**
13: $u^{(\mathrm{ad})} = \mathsf{Flatten}(\xi_K)$;
14: $u = u + u^{(\mathrm{ad})}$;

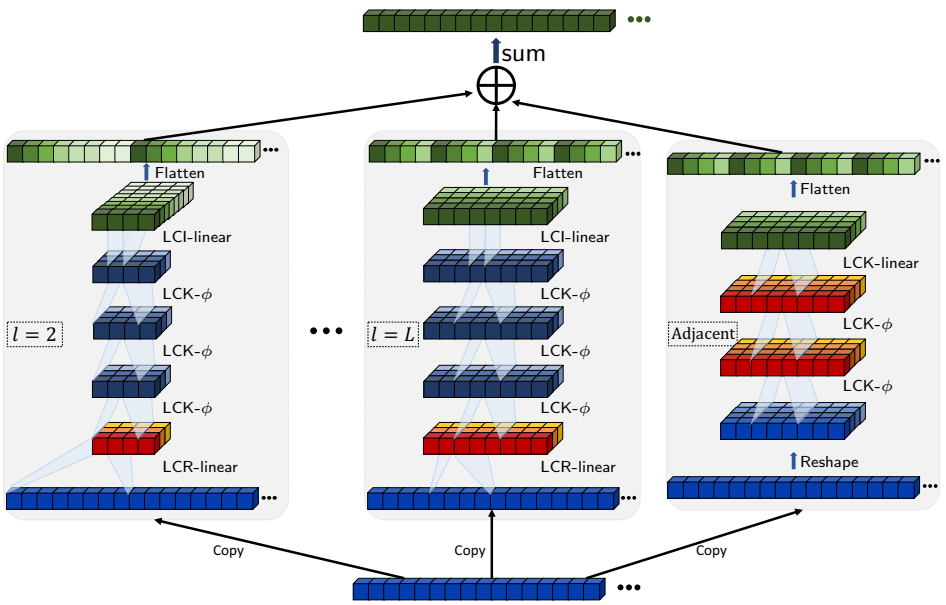


Figure 7: Multiscale neural network architecture for nonlinear mappings, which is an extension of the neural network architecture for $\mathcal{H}$-matrices Figure 5. ϕ is an activation function.

3.1. Algorithm and architecture. NN can represent nonlinearity 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 linear activation functions in these layers. Particularly, we also use linear activation function for the last layer of the adjacent part, *i.e.* the ϕ in line 11 in Algorithm 3 is linear when $k = K$.

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))

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

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

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

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 neural network 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 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.

4.1. Implementation. Our implementation of MNN uses Keras [10], a high-level application programming interface (API) running, in this case, on top of TensorFlow [1] (a library of toolboxes for training neural networks). 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 [12]. 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 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 the test are run on GPU with data type `float32`. All 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) are problem dependent.

4.2. NLSE with inhomogeneous background potential. The nonlinear Schrödinger equation (NLSE) is widely used in quantum physics to describe the single particle properties of the Bose-Einstein condensation phenomenon [40, 2]. Here we study the NLSE with inhomogeneous background potential $V(x)$

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

with period boundary condition, to find its ground state $u_G(x)$. 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 [3] 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. In the following, we study MNN on 1D and 2D cases, respectively.

The potential V is chosen as

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

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 $T \sim \mathcal{U}(2,4) \times 10^{-3}$.

368 **4.2.1. One-dimensional case.** For the one-dimensional case, the number of discretization
369 points is $N = 320$, and we set $L = 6$ and $m = \frac{N}{2^L} = 5$. In all the tests, the number of test
370 samples is the same as that the number of train samples if not properly specified. We perform
371 numerical experiments to study the behavior of MNN for different number of channels r,
372 different number of CK layers K, different number of Gaussians n_g and different number of
373 training samples $N_{\text{samples}}^{\text{train}}$.

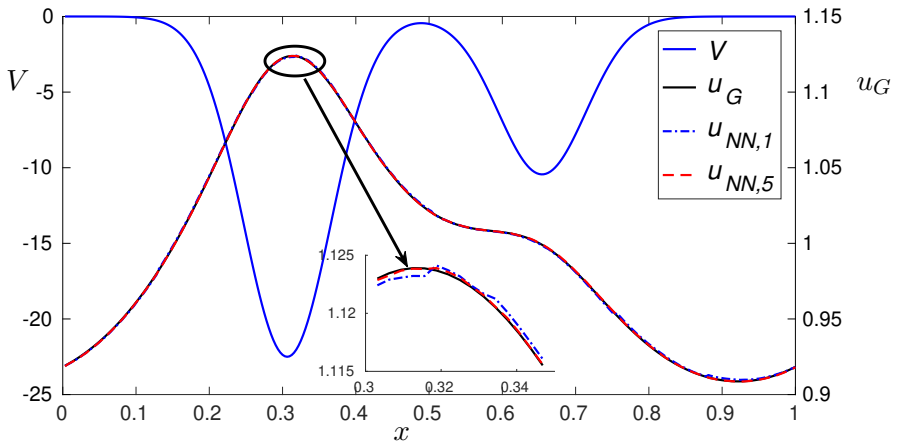


Figure 8: An example of the potential V and its corresponding solution u_G and predicted solution $u_{NN,K}$ by MNN with $r = 6$, $n_g = 2$.

$N_{\text{samples}}^{\text{train}}$	$N_{\text{samples}}^{\text{test}}$	Training error	Validation error
500	5000	2.1e-4	2.4e-4
1000	5000	1.8e-4	2.0e-4
5000	5000	1.4e-4	1.5e-4
20000	20000	1.5e-4	1.5e-4

Table 1: 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 $n_g = 2$.

r	N_{params}	Training error	Validation error
2	1895	4.0e-4	4.0e-4
4	4555	2.0e-4	2.0e-4
6	8535	1.4e-4	1.5e-4

Table 2: Relative error in approximating the ground state of NLSE for different number of channels r for 1D case with $K = 5$, $n_g = 2$ and $N_{\text{samples}}^{\text{train}}$= 5000.

K	N_{params}	Training error	Validation error
1	3343	7.5e-4	7.5e-4
3	5939	2.1e-4	2.1e-4
5	8535	1.4e-4	1.5e-4

Table 3: Relative error in approximating the ground state of NLSE for different number of CK layers K for 1D case with $r = 6$, $n_g = 2$ and $N_{\text{samples}}^{\text{train}} = 5000$.

n_g	Training error	Validation error
2	1.4e-4	1.5e-4
4	2.6e-4	2.7e-4
6	2.6e-4	2.6e-4
8	2.8e-4	2.9e-4

Table 4: Relative error in approximating the ground state of NLSE for different number of Gaussians n_g for 1D case with $K = 5$, $r = 6$ and $N_{\text{samples}}^{\text{train}} = 5000$.

Usually, the number of samples should be greater than that of parameters to avoid overfitting. But in neural network, it has been consistently found that the number of samples can be less than that of parameters [48, 49] We present the numerical results for different $N_{\text{samples}}^{\text{train}}$with $K = 5$, $r = 6$ and $n_g = 2$ in Table 1. In this case, the number of parameters is $N_{\text{params}} = 8535$. There is no overfitting even $N_{\text{samples}}^{\text{train}} = 500$, and the error is only slightly larger than that when $N_{\text{samples}}^{\text{train}} = 20000$. For the case $N_{\text{samples}}^{\text{train}} = 5000$, the error is close to that for $N_{\text{samples}}^{\text{train}} = 20000$. This allows us to train MNN with $N_{\text{samples}}^{\text{train}} < N_{\text{params}}$. This feature is quite useful for high-dimensional case, because for high-dimensional case, $N_{\text{params}}$is usually very large and generating samples is expensive.

Table 2 presents the numerical results for different number of channels (*i.e.* the rank of the $\mathcal{H}$-matrix) r with $K = 5$, $n_g = 2$ and $N_{\text{samples}}^{\text{train}} = 5000$. As r increases, we find that the error first consistently decreases and then stagnates. We use $r = 6$ for the 1D NLSE below to balance between efficiency and accuracy.

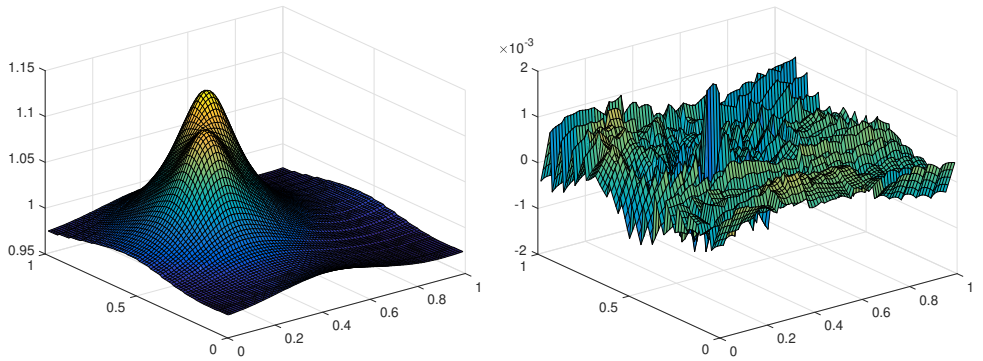
Similarly, Table 3 presents the numerical results for different number of CK layers K with $r = 6$, $n_g = 2$ and $N_{\text{samples}}^{\text{train}} = 5000$. The error consistently decreases with respect to the increase of K, as NN can represent increasingly more complex functions with respect to the depth of the network. However, after a certain threshold, adding more layers provides very marginal gains in accuracy. In practice, $K = 5$ is a good choice for the NLSE for 1D case.

Table 4 presents the numerical results for different number of Gaussians in the potential V, for fixed $K = 5$, $r = 6$ and $N_{\text{samples}}^{\text{train}} = 5000$. Table 4 shows that within the class of input functions considered here, MNN is not sensitive to the complexity of the input. In particular, it shows that for the cases considered, increasing the number of wells only marginally increases the error.

Across the results in Tables 2 to 4, the validation errors are very close to the corresponding

training errors, and no overfitting is observed. Figure 8 presents a sample for the potential V and its corresponding solution and prediction solution by MNN. The prediction solution agrees with the target solution very well.

4.2.2. Two-dimensional case. For two-dimensional case, the number of discretization points is $n = 80 \times 80$, and we set $L = 4$ and $m = 5$. In all the tests, the number of test data is same as that of the train data. We perform simulation to study the behavior of MNN for different number of channels r, different number of CK layers K and different number of Gaussians n_g. As is discussed in 1D case, MNN allows $N_{\text{samples}}^{\text{train}} < N_{\text{params}}$. In all the tests, the number of samples is always 20000 and numerical test shows no overfitting for all the test.



(a) u_{NN} (b) $u_G - u_{NN}$

Figure 9: Prediction solution of MNN for $K = 5$, $r = 6$ and $n_g = 2$ and its error with respect to the reference solution.

r	N_{params}	Training error	Validation error
2	33371	4.9e-4	4.9e-4
6	57323	1.5e-4	1.5e-4
10	100955	1.4e-4	1.4e-4

Table 5: Relative error in approximating the ground state of NLSE for different number of channels r for 2D case with $K = 5$, $n_g = 2$ and $N_{\text{samples}}^{\text{train}} = 20000$.

Tables 5 and 6 present the numerical results for different number of channels r and different number of CK layers K, respectively. We find that similar to the 1D case, the choice of parameters $r = 6$ and $K = 5$ also yield accurate results in the 2D case. Table 7 presents the numerical results for different number of Gaussians in the potential V for $K = 5$, $r = 6$ and $N_{\text{samples}}^{\text{train}} = 20000$. We can find that MNN is also not sensitive to the complexity of the number of Gaussians. Figure 9 presents the prediction of a sample and its corresponding error with respect to the reference.

K	N_{params}	Training error	Validation error
1	16939	6.9e-4	7.0e-4
3	37131	2.1e-4	2.1e-4
5	57323	1.5e-4	1.5e-4
7	77515	1.4e-4	1.4e-4

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

n_g	Training error	Validation error
2	1.5e-4	1.5e-4
4	1.5e-4	1.5e-4
6	2.4e-4	2.4e-4
8	2.2e-4	2.2e-4

Table 7: Relative error in approximating the ground state of NLSE for different number of Gaussians n_g for the 2D case with $r = 6$, $K = 5$ and $N_{\text{samples}}^{\text{train}} = 20000$.

4.3. Kohn-Sham map. Kohn-Sham density functional theory [26, 30] is the most widely used electronic structure theory. It requires the solution of the following set of nonlinear eigenvalue equations (real arithmetic assumed for all quantities):

$$\begin{aligned} &\left(-\frac{1}{2}\Delta + V[\rho](x)\right)\psi_i(x) = \varepsilon_i\psi_i(x), \ \ x \in \Omega = [-1,1)^d \\ &\int_\Omega \psi_i(x)\psi_j(x)\,\mathrm{d}x = \delta_{ij}, \quad \rho(x) = \sum_{i=1}^{n_e} |\psi_i(x)|^2. \end{aligned} \tag{4.5}$$

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 non-decreasingly, and $\rho(x)$ is the electron density, which satisfies the constraint

$$\rho(x) \geq 0, \quad \int_\Omega \rho(x)\,\mathrm{d}x = n_e. \tag{4.6}$$

The Kohn-Sham equations (4.5) need to be solved self-consistently, which can also viewed as solving the following fixed point map

$$\rho = \mathcal{F}_{\text{KS}}[V[\rho]]. \tag{4.7}$$

Here the mapping $\mathcal{F}_{\text{KS}}[\cdot]$ from V to ρ is called the Kohn-Sham map, which for a fixed potential is reduced to a linear eigenvalue problem, and it constitues the most computationally intensive step for solving (4.5). We seek to approximate the Kohn-Sham map using a multiscale neural network, whose output was regularized so it satisfies (4.6).

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

Table 8: Relative error on the approximation of the Kohn-Sham map for different r, with $K = 6$, $N^{\text{train}}_{\text{samples}}$=16000, and $N^{\text{test}}_{\text{samples}}$=4000.

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

$$V(x) = -\sum_{i=1}^{n_g} \sum_{j\in\mathbb{Z}^d} c_i \exp\left(-\frac{(x - r_i - 2j)^2}{2\sigma^2}\right), \qquad x \in [-1, 1)^d, \tag{4.8}$$

where d is the dimension and $r_i \in [-1, 1)^d$. We set $\sigma = 0.05$ for 1D and $\sigma = 0.2$ for 2D case. The coefficients c_i are randomly chosen following the uniform distribution $\mathcal{U}([0.8, 1.2])$, and the centers of the Gaussian wells r_i, are chosen randomly under the constraint that $|r_i - r_{i'}| > 2\sigma$. The Kohn-Sham map is discretized using a pseudo-spectral method [46], and solved by a standard eigensolver.

4.3.1. One-dimensional case. We generated 7 data sets using 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 8 shows that there is no overfitting, even at this level of accuracy and number of parameters. This behavior is found in all the numerical examples, thus we only report the test error in what follows.

From Table 9 we can observe that as we increase r the error decreases sharply. Figure 10 depict 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 properly 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 9 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 10. We can observe that the error decreases sharply as the depth of the network increases, and then stagnates as K becomes large.

4.3.2. Two-dimensional case. The discretization is the standard extension to 2D using tensor products, using a 64×64 grid. In this case we only used $n_g = 2$ and we followed the same number of training and test samples as that in the 1D case. We fixed $K = 6$, $L = 4$,

$n_g \backslash r$	2	4	6	8	10
2	6.7e-4	3.3e-4	2.8e-4	3.3e-4	1.8e-4
3	7.3e-4	3.6e-4	3.0e-4	2.0e-4	2.8e-4
4	8.5e-4	4.1e-4	2.9e-4	3.8e-4	2.4e-4
5	9.0e-4	5.9e-4	4.0e-4	3.6e-4	3.6e-4
6	6.3e-4	4.8e-4	3.8e-4	4.0e-4	4.2e-4
7	8.6e-4	5.5e-4	3.9e-4	3.7e-4	3.5e-4
8	1.2e-3	5.1e-4	3.7e-4	4.5e-4	3.7e-4

Table 9: Relative test error on the approximation of the Kohn-Sham map for different ranks r, with fixed $K = 6$ and $N_{\text{samples}}^{\text{train}}$=16000.

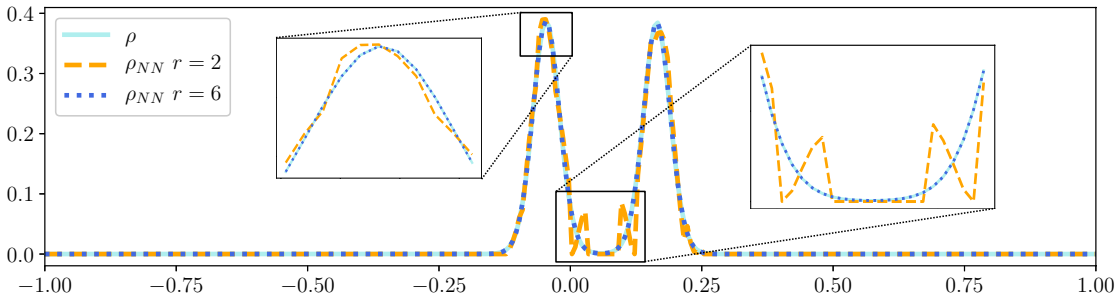


Figure 10: Estimation using two different multiscale networks with $r = 2$, and $r = 6$; with $K = 6$, and $L = 5$ fixed.

and we trained the network for different number of channels, r. The results are displayed in Table 11, 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 Kohn-Sham map as shown in Figure 11. Figure 11a shows the output of neural network for a test sample and Figure 11b shows the approximation error with respect to the reference.

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 neural network, which mainly consists of three sub-networks: restriction network, kernel network, and interpolation network. The three sub-networks 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 neural networks (CNN) when the mapping is equivariant to translation. Using the parameterized nonlinear Schrödinger

$n_g \backslash K$	2	4	6	8	10
2	1.4e-3	3.1e-4	2.8e-4	3.5e-4	2.3e-4
3	2.0e-3	5.4e-4	3.0e-4	5.6e-4	5.3e-4
4	1.9e-3	5.8e-4	2.8e-4	6.0e-4	7.1e-4
5	1.8e-3	7.2e-4	4.0e-4	8.0e-4	7.4e-4
6	2.1e-3	7.3e-4	3.8e-4	6.7e-4	6.7e-4
7	2.2e-3	7.9e-4	3.8e-4	7.4e-4	5.8e-4
8	2.0e-3	8.8e-4	3.7e-4	6.7e-4	6.8e-4

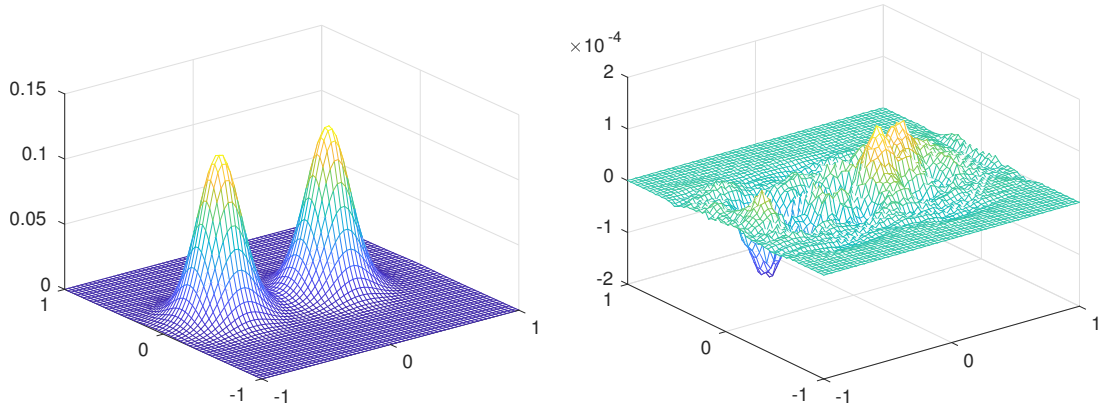
Table 10: Relative test error on the approximation of the Kohn-Sham map for different K and fixed rank $r = 6$, and $N_{\text{samples}}^{\text{train}} = 16000$.

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

Table 11: Relative errors on the approximation of the Kohn-Sham map for 2D case for different r and $K = 6$, $N_{\text{samples}}^{\text{train}}$=16000 and $N_{\text{samples}}^{\text{test}}$=4000.

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 multiscale neural network based on hierarchical matrices. The architecture can be altered depending on the target application. Some of the possible modifications and extensions are listed below. 1) In this work, the neural network 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 methods, $\mathcal{H}^2$-matrices and wavelets. 2) The input, u, and output, v, in this work are periodic. The network can be directly extended to the non-periodic 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 $A^{(\text{ad})}$ 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[24]. 6) It is known that $\mathcal{H}$-matrices can well approximate smooth kernels but become



(a) ρ_{NN} (b) $|\rho_{NN} - \rho|$

Figure 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.

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.

Acknowledgements. 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, et al., *Tensorflow: A system for large-scale machine learning.*, in OSDI, vol. 16, 2016, pp. 265–283.

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

[3] W. Bao and Q. Du, *Computing the ground state solution of Bose–Einstein condensates by a normalized gradient flow*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1674–1697.

[4] 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*, Comptes Rendus Mathematique, 339 (2004), pp. 667 – 672, https://doi.org/10.1016/j.crma.2004.08.006.

[5] J. Berg and K. Nyström, *A unified deep artificial neural network approach to partial differential equations in complex geometries*, arXiv preprint arXiv:1711.06464, (2017).

[6] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Engineering analysis with boundary elements, 27 (2003), pp. 405–422.

[7] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Mathematics of Computation, 31 (1977), pp. 333–390.

[8] S. Chan and A. H. Elsheikh, *A machine learning approach for efficient uncertainty quantification using multiscale methods*, Journal of Computational Physics, 354 (2018), pp. 493 – 511, https://doi.org/10.1016/j.jcp.2017.10.034.

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

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

[11] N. Cohen, O. Sharir, and A. Shashua, *On the expressive power of deep learning: A tensor analysis*, arXiv preprint arXiv:1603.00988, (2018).

[12] T. Dozat, *Incorporating nesterov momentum into adam*, 2015.

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

[14] Y. Efendiev, J. Galvis, G. Li, and M. Presho, *Generalized multiscale finite element methods. nonlinear elliptic equations*, Communications in Computational Physics, 15 (2014), pp. 733–755, https://doi.org/10.4208/cicp.020313.041013a.

[15] Y. Efendiev and T. Hou, *Multiscale finite element methods for porous media flows and their applications*, Applied Numerical Mathematics, 57 (2007), pp. 577 – 596, https://doi.org/10.1016/j.apnum.2006.07.009. Special Issue for the International Conference on Scientific Computing.

[16] 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, 2002, https://ub-madoc.bib.uni-mannheim.de/744/1/TR-02-008.pdf.

[17] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of computational physics, 73 (1987), pp. 325–348.

[18] M. A. Grepl, Y. Maday, N. C. Nguyen, and A. T. Patera, *Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations*, ESAIM: Mathematical Modelling and Numerical Analysis, 41 (2007), p. 575–605, https://doi.org/10.1051/m2an:2007031.

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

[20] W. Hackbusch, L. Grasedyck, and S. Börm, *An introduction to hierarchical matrices*, Math. Bohem., 127 (2002).

[21] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic: general complexity estimates*, Journal of Computational and Applied Mathematics, 125 (2000), pp. 479–501.

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

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

[24] 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, 2016, pp. 770–778.

[25] 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 Magazine, 29 (2012), pp. 82–97, https://doi.org/10.1109/MSP.2012.2205597.

[26] P. Hohenberg and W. Kohn, *Inhomogeneous electron gas*, Physical review, 136 (1964), p. B864.

[27] 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.

[28] Y. Khoo, J. Lu, and L. Ying, *Solving parametric PDE problems with artificial neural networks*, arXiv preprint arXiv:1707.03351, (2017).

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

[30] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Physical review, 140 (1965), p. A1133.

[31] 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, USA, 2012, Curran Associates Inc., pp. 1097–1105.

[32] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015).

[33] 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.

[34] Y. Li, X. Cheng, and J. Lu, *Butterfly-Net: Optimal function representation based on convolutional neural networks*, arXiv preprint arXiv:1805.07451, (2018).

[35] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix–vector*

multiplication, Journal of Computational Physics, 230 (2011), pp. 4071–4087.
[36] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, *Deep neural nets as a method for quantitative structure–activity relationships*, Journal of Chemical Information and Modeling, 55 (2015), pp. 263–274, https://doi.org/10.1021/ci500747n. PMID: 25635324.
[37] Y. Maday, O. Mula, and G. Turinici, *Convergence analysis of the generalized empirical interpolation method*, SIAM Journal on Numerical Analysis, 54 (2016), pp. 1713–1731, https://doi.org/10.1137/140978843.
[38] S. Mallat, *A wavelet tour of signal processing: the sparse way*, in A wavelet tour of signal processing: the sparse way, Academic press, Boston, third ed., 2008, https://doi.org/10.1016/B978-0-12-374370-1.50001-9.
[39] H. Mhaskar, Q. Liao, and T. Poggio, *Learning functions: When is deep better than shallow*, arXiv preprint arXiv:1603.00988, (2018).
[40] L. Pitaevskii, *Vortex lines in an imperfect Bose gas*, Sov. Phys. JETP, 13 (1961), pp. 451–454.
[41] M. Raissi and G. E. Karniadakis, *Hidden physics models: Machine learning of nonlinear partial differential equations*, Journal of Computational Physics, 357 (2018), pp. 125 – 141, https://doi.org/10.1016/j.jcp.2017.11.039.
[42] K. Rudd, G. D. Muro, and S. Ferrari, *A constrained backpropagation approach for the adaptive solution of partial differential equations*, IEEE Transactions on Neural Networks and Learning Systems, 25 (2014), pp. 571–584, https://doi.org/10.1109/TNNLS.2013.2277601.
[43] 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.
[44] K. Spiliopoulos and J. Sirignano, *Dgm: A deep learning algorithm for solving partial differential equations*, arXiv preprint arXiv:1708.07469, (2018).
[45] 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, Inc., 2014, pp. 3104–3112.
[46] L. Trefethen, *Spectral Methods in MATLAB*, Society for Industrial and Applied Mathematics, 2000, https://doi.org/10.1137/1.9780898719598.
[47] Y. Wang, C. W. Siu, E. T. Chung, Y. Efendiev, and M. Wang, *Deep multiscale model learning*, arXiv preprint arXiv:1806.04830, (2018).
[48] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, *Understanding deep learning requires rethinking generalization*, arXiv:1611.03530, (2016).
[49] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, *Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising*, IEEE Transactions on Image Processing, 26 (2017), pp. 3142–3155.
[50] L. Zhang, J. Han, H. Wang, R. Car, and W. E, *Deepcg: constructing coarse-grained models via deep neural networks*, arXiv preprint arXiv:1802.08549, (2018).
[51] L. Zhang, H. Wang, and W. E, *Adaptive coupling of a deep neural network potential to a classical force field*, arXiv preprint arXiv:1806.01020, (2018).