SIAM J. SCI. COMPUT.
Vol. 41, No. 5, pp. A3182–A3201

SWITCHNET: A NEURAL NETWORK MODEL FOR FORWARD AND INVERSE SCATTERING PROBLEMS*

YUEHAW KHOO† AND LEXING YING‡

Abstract. We propose a novel neural network architecture, SwitchNet, for solving wave equation based inverse scattering problems via providing maps between the scatterers and the scattered field (and vice versa). The main difficulty of using a neural network for this problem is that a scatterer has a global impact on the scattered wave field, rendering a typical convolutional neural network with local connections inapplicable. While it is possible to deal with such a problem using a fully connected network, the number of parameters grows quadratically with the size of the input and output data. By leveraging the inherent low-rank structure of the scattering problems and introducing a novel switching layer with sparse connections, the SwitchNet architecture uses far fewer parameters and facilitates the training process. Numerical experiments show promising accuracy in learning the forward and inverse maps between the scatterers and the scattered wave field.

Key words. Helmholtz equation, inverse problem, artificial neural network, operator compression

AMS subject classifications. 65R10, 65T50

DOI. 10.1137/18M1222399

1. Introduction.

In this paper, we study the forward and inverse scattering problems via the use of artificial neural networks (NNs). In order to simplify the discussion, we focus on the time-harmonic acoustic scattering in two-dimensional space. The inhomogeneous media scattering problem with a fixed frequency ω is modeled by the Helmholtz operator

$$Lu := \left(-\Delta - \frac{\omega^2}{c^2(x)} \right) u, \tag{1}$$

where $c(x)$ is the velocity field. In many settings, there exists a known background velocity field $c_0(x)$ such that $c(x)$ is identical to $c_0(x)$ except in a compact domain Ω. By introducing the *scatterer* $\eta(x)$ compactly supported in Ω,

$$\frac{\omega^2}{c(x)^2} = \frac{\omega^2}{c_0(x)^2} + \eta(x), \tag{2}$$

we can equivalently work with $\eta(x)$ instead of $c(x)$. Note that in this definition $\eta(x)$ scales quadratically with the frequency ω. However, as ω is assumed to be fixed throughout this paper, this scaling does not affect any discussion below.

In many real-world applications, $\eta(\cdot)$ is unknown. The task of the inverse problem is to recover $\eta(\cdot)$ based on some observation data $d(\cdot)$. The observation data $d(\cdot)$ is

*Submitted to the journal's Methods and Algorithms for Scientific Computing section October 23, 2018; accepted for publication (in revised form) July 18, 2019; published electronically October 15, 2019.

https://doi.org/10.1137/18M1222399

Funding: This work 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, the National Science Foundation under award DMS-1818449, and the GCP Research Credits Program from Google. This work was done when the first author held a post-doctoral scholar position at Stanford University.

†Department of Statistics, The University of Chicago, Chicago, IL 60637 (ykhoo@uchicago.edu).

‡Department of Mathematics and ICME, Stanford University, Stanford, CA 94305 (lexing@stanford.edu), and Facebook AI Research, Menlo Park, CA 94025.

often a quantity derived from the Green's function $G = L^{-1}$ of the Helmholtz operator L and, therefore, it depends closely on $\eta(\cdot)$. This paper is an exploratory attempt to construct efficient approximations to the forward map $\eta \to d$ and the inverse map $d \to \eta$ using the modern tools from machine learning and artificial intelligence. Such approximations are highly useful for the numerical solutions of the scattering problems: an efficient map $\eta \to d$ provides an alternative to expensive partial differential equation (PDE) solvers for the Helmholtz equation; an efficient map $d \to \eta$ is more valuable as it allows us to solve the inverse problem of determining the scatterers from the scattering field, without going through the usual iterative process (readers interested in the inversion problem are referred to [1, 4, 2] and the references therein).

In the last several years, deep NN has become the go-to method in computer vision, image processing, speech recognition, and many other machine learning applications [21, 32, 15, 12]. More recently, methods based on NN have also been applied to solving PDEs. Based on the way that the NN is used, these methods for solving the PDE can be roughly separated into two different categories. For the methods in the first category [19, 31, 6, 13, 17, 9], instead of specifying the solution space via the choice of basis (as in finite element method or Fourier spectral method), NN is used for representing the solution. Then an optimization problem, for example, a variational formulation, is solved in order to obtain the parameters of the NN and hence the solution to the PDE. Similar to the use of an NN for regression and classification purposes, the methods in the second category such as [26, 14, 16, 11] use an NN to learn a map that goes from the coefficients in the PDE to the solution of the PDE. As in machine learning, the architecture design of an NN for solving PDE usually requires the incorporation of the knowledge from the PDE domain such that the NN architecture is able to capture the behavior of the solution process.

This paper takes a deep learning approach to learn both the forward and inverse maps. For the Helmholtz operator (1), we propose an NN architecture for determining the forward and inverse maps between the scatterer $\eta(\cdot)$ and the observation data $d(\cdot)$ generated from the scatterer. Although this task looks similar to computer vision problems such as image segmentation, denoising, and super-resolution where the map between the two images has to be determined, the nature of the map in our problem is much more complicated. In many image processing tasks, the value of a pixel at the output generally only depends on a neighborhood of that pixel at the input layer. However, for the scattering problems, the input and output are often defined on different domains and, due to wave propagation, each location of the scatterer can influence every point of the scattered field. Therefore, the connectivity in the NN has to be wired in a nonlocal fashion, rendering a typical NN with local connectivity insufficient. This leads to the development of the proposed *SwitchNet*. The key idea is the inclusion of a novel low-complexity *switch layer* that sends information between all pairs of sites effectively, following the ideas from butterfly factorizations [24]. The same factorization was used earlier in the architecture proposed [22], but the network weights there are hard-coded and not trainable. We note that recently, U-net [30], a convolutional NN, was been applied to solve the full waveform inversion problem [33]. However, in [33], U-net is used to provide a map between a low-frequency input field to a smooth output (by a suitable preprocessing on the high-frequency full waveform). Our approach directly provides mappings between the high-frequency waveform and the scatterers. Besides the inverse mapping, we are also able to obtain the forward mapping.

The paper is organized as follows. In section 2, we discuss about some preliminary results concerning the Helmholtz equation. In section 3, we study the so-called far field

pattern of the scattering problem, where the sources and receivers can be regarded as placed at infinity. We propose SwitchNet to determine the maps between the far field scattering pattern and the scatterer. In section 4, we turn to the setting of a seismic imaging problem. In this problem, the sources and receivers are at a finite distance but yet well separated from the scatterer.

2. Preliminary. The discussion of this paper shall focus on the two-dimensional case. Here, we summarize the mathematical tools and notations used in this paper. As mentioned above, the scatterer $\eta(x)$ is compactly supported in a domain Ω, whose diameter is of $O(1)$. For example, one can think of Ω as the unit square centered at the origin. In (1), the Helmholtz operator is defined on the whole space $\mathbb{R}^2$ with the radiative (Sommerfeld) boundary condition [8] specified at infinity. Since the scatterer field $\eta(x)$ is localized in Ω, it is convenient to truncate the computation domain to Ω by imposing the *perfectly matched layer* [3] that approximates the radiative boundary condition.

In a typical numerical solution of the Helmholtz operator, Ω is discretized by a Cartesian grid $X \subset \Omega$ at the rate of a few points per wavelength. As a result, the number of grid points N per dimension is proportional to the frequency ω. We simply use $\{x\}_{x\in X}$ to denote the discretization points of this $N \times N$ grid X. The Laplacian operator $-\Delta$ in the Helmholtz operator is typically discretized with local numerical schemes, such as the finite difference method [20]. Via this discretization, we can consider the scatterer field η, discretized at the points in X, as a vector in $\mathbb{R}^{N^2}$ and the Helmholtz operator L as a matrix in $\mathbb{C}^{N^2\times N^2}$.

Using the background velocity field $c_0(x)$, we first introduce the background Helmholtz operator $L_0 = -\Delta - \omega^2/c_0^2$. With the help of L_0, one can write L in a perturbative way as

$$L = L_0 - E, \quad E = \operatorname{diag}(\eta), \tag{3}$$

where E is viewed as a perturbation. By introducing the background Green's function

$$G_0 := L_0^{-1}, \tag{4}$$

one can write down a formal expansion for the Green's function $G = L^{-1}$ of the η-dependent Helmholtz operator L:

$$\begin{aligned} G &= (L_0(I - G_0E))^{-1} \\ &\sim (I + G_0E + G_0EG_0E + \cdots)G_0 \\ &\sim G_0 + G_0EG_0 + G_0EG_0EG_0 + \cdots \\ &:= G_0 + G_1 + G_2 + \cdots, \end{aligned} \tag{5}$$

which is valid when the scatterer field $\eta(x)$ is sufficiently small. The last line of the above equation serves as the definition of the successive terms of the expansion (G_1, G_2, and so on). As G_0 can be computed from the knowledge of the background velocity field $c_0(x)$, most data gathering processes (with appropriate postprocessing) focus on the difference $G - G_0 = G_1 + G_2 + \cdots$ instead of G itself.

A usual experimental setup consists of a set of *sources* S and a set of *receivers* R:

$$S = \{s\}_{s\in S}, \quad R = \{r\}_{r\in R}.$$

The data gathering process usually involves three steps: (1) impose an external force or incoming wave field via some sources, (2) solve for the scattering field either computationally or physically, (3) gather the data with receivers at specific locations or

directions. The second step is modeled by the difference of the Green's function $G - G_0$, as we mentioned above. As for the other steps, it is convenient at this point to model the first step with a source-dependent operator Π_S and the third one with a receiver-dependent operator Π_R. We shall see later how these operators are defined in more concrete settings. By putting these components together, one can set the observation data d abstractly as

$$\begin{aligned} d &= \Pi_R(G - G_0)\Pi_S = \Pi_R(G_0EG_0 + G_0EG_0EG_0 + \cdots)\Pi_S \\ &= (\Pi_R G_0)(E + EG_0E + \cdots)(G_0\Pi_S). \end{aligned} \tag{6}$$

In this paper, we focus on two scenarios: far field pattern and seismic imaging. We start with far field pattern first to motivate and introduce SwitchNet. We then move on to the seismic case by focusing on the main differences.

3. SwitchNet for far field pattern.

3.1. Problem setup. In this section, we consider the problem of determining the map from the scatterer to the far field scattering pattern, along with its inverse map. Without loss of generality, we assume that the diameter of the domain Ω is of $O(1)$ after appropriate rescaling. The background velocity $c_0(x)$ is assumed to be 1 since the far field pattern experiments are mostly performed in free space.

In this problem, both the sources and the receivers are indexed by a set of unit directions in $\mathbb{S}^1$. The source associated with a unit direction $s \in S \subset \mathbb{S}^1$ is an incoming plane wave $e^{i\omega s\cdot x}$ pointing at direction s. It is well known that the scattered wave field, denoted by $u_s(x)$, at a large distance takes the following form [8]:

$$u_s(x) = \frac{e^{i\omega|x|}}{\sqrt{|x|}}\left(u_s^\infty\left(\frac{x}{|x|}\right) + o(1)\right),$$

where the function $u_s^\infty(\cdot)$ is defined on the unit circle $\mathbb{S}^1$. The receiver at direction $r \in R \subset \mathbb{S}^1$ simply records the quantity $u_s^\infty(r)$ for each s. The set of observation data d is then defined to be

$$d(rs) = u_s^\infty(r).$$

Figure 1 provides an illustration of this experimental setup. Henceforth, we assume that both R and S are chosen to be a set of uniformly distributed directions on $\mathbb{S}^1$. Their size, denoted by M, typically scales linearly with frequency ω.

This data gathering process can be put into the framework of (6). First, one can think of the source prescription as a limiting process that produces in the limit

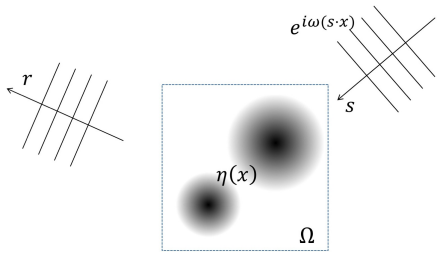


FIG. 1. *Illustration of the incoming and outgoing waves for a far field pattern problem. The scatterer $\eta(x)$ is compactly supported in the domain Ω. The incoming plane wave points at direction s. The far field pattern is sampled at each receiver direction r.*

the incoming wave $e^{i\omega s\cdot x}$. The source can be considered to be located at the point $-s\rho$ for the direction $s \in \mathbb{S}^1$ with the distance $\rho \in \mathbb{R}^+$ going to infinity. In order to compensate for the geometric spreading of the wave field and also the phase shift, the source magnitude is assumed to scale like $\sqrt{\rho}e^{-i\omega\rho}$ as ρ goes to infinity. Under this setup, we have

$$\begin{aligned} &\lim_{\rho\to\infty}(G_0\Pi_S)(x,s) \\ &= \lim_{\rho\to\infty}(1/\sqrt{\rho})e^{i\omega|x-(-s\rho)|}\sqrt{\rho}e^{-i\omega\rho} \\ &= \lim_{\rho\to\infty}(1/\sqrt{\rho})e^{i\omega(\rho+s\cdot x)}\sqrt{\rho}e^{-i\omega\rho} \\ &= e^{i\omega s\cdot x}. \end{aligned} \tag{7}$$

Similarly, one can also regard the receiver prescription as a limiting process. The receiver is located at point $r\rho'$ for a fixed unit direction $r \in \mathbb{S}^1$ with $\rho' \in \mathbb{R}^+$ going to infinity. Again in order to to compensate for the geometric spreading and the phase shift, one scales the received signal with $\sqrt{\rho'}e^{-i\omega\rho'}$. As a result, we have

$$\begin{aligned} &\lim_{\rho'\to\infty}(\Pi_R G_0)(r,x) \\ &= \lim_{\rho'\to\infty}(1/\sqrt{\rho'})e^{i\omega|r\rho'-x|}\sqrt{\rho'}e^{-i\omega\rho'} \\ &= \lim_{\rho'\to\infty}(1/\sqrt{\rho'})e^{i\omega(\rho'-r\cdot x)}\sqrt{\rho'}e^{-i\omega\rho'} \\ &= e^{-i\omega r\cdot x}. \end{aligned} \tag{8}$$

In this limiting setting, one redefines the observation data as

$$d = \lim_{\rho,\rho'\to\infty}(\Pi_R G_0)(E + EG_0E + \cdots)(G_0\Pi_S). \tag{9}$$

Now taking the two limits (7) and (8) under consideration, one arrives at the following representation of the observation data $d(r,s)$ for $r \in R$ and $s \in S$:

$$d(r,s) = \sum_{x\in X}\sum_{y\in X} e^{-i\omega r\cdot x}(E + EG_0E + \cdots)(x,y)e^{i\omega s\cdot y}. \tag{10}$$

3.2. Low-rank property. The intuition behind the proposed NN architecture comes from examining (10) when E (or η) is small. In such a situation, we simply retain the term that is linear in E. Using the fact that $E = \mathrm{diag}(\eta)$, (10) becomes

$$d(r,s) \approx \sum_{x\in X} e^{i\omega(s-r)\cdot x}\eta(x)$$

for $r \in R \subset \mathbb{S}^1$ and $s \in S \subset \mathbb{S}^1$. This linear map takes $\eta(x)$ defined on a Cartesian grid $X \subset \Omega$ to $d(r,s)$ defined on yet another Cartesian grid $R\times S \subset \mathbb{S}^1\times\mathbb{S}^1$. Recalling that both R and S are of size M and working with a vectorized $d \in \mathbb{C}^{M^2}$, we can write the above equation compactly as

$$d \approx A\eta, \tag{11}$$

where the element of the matrix $A \in \mathbb{C}^{M^2\times N^2}$ at $(r,s) \in R\times S$ and $x \in X$ is given by

$$A(rs,x) = \exp(i\omega(s-r)\cdot x). \tag{12}$$

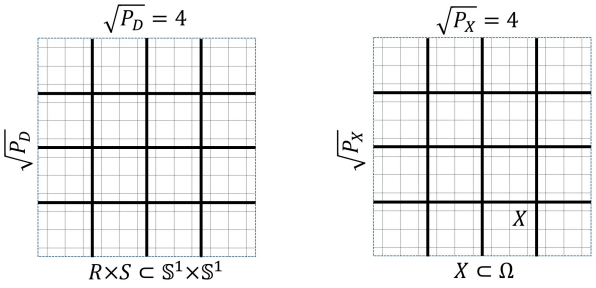


FIG. 2. *Illustration of the partitions used in Theorem* 1. *The fine grids stand for the Cartesian grids X and $R \times S$. The bold lines are the boundary of the squares of the partitions.*

The following theorem concerning the matrix A plays a key role in the design of our NN. Let us first partition Ω uniformly into $\sqrt{P_X} \times \sqrt{P_X}$ Cartesian squares of side-length equal to $1/\sqrt{\omega}$. Here we assume that $\sqrt{P_X}$ is an integer. Note that, since the diameter of Ω is of $O(1)$, $\sqrt{P_X} \approx \sqrt{\omega}$. This naturally partitions the set of grid points X into P_X subgroups depending on which square each point belongs to. We shall denote these subgroups by $X_0, \ldots, X_{P_X-1}$. Similarly, we also partition $\mathbb{S}^1 \times \mathbb{S}^1$ uniformly (in the angular parameterization) into $\sqrt{P_D} \times \sqrt{P_D}$ squares $D_0, \ldots, D_{P_D-1}$ of side-length equal to $1/\sqrt{\omega}$. $\sqrt{P_D}$ is also assumed to be an integer, and obviously $\sqrt{P_D} \approx \sqrt{\omega}$. This further partitions the set $R \times S$ into P_D subgroups depending on which square they belong to. We shall denote these subgroups by $D_0, \ldots, D_{P_D-1}$. Figure 2 illustrates the partition for $\sqrt{P_X} = \sqrt{P_D} = 4$.

CLAIM 1. *For any D_i and X_j, the submatrix*

$$A_{ij} := [A(rs, x)]_{(r,s)\in D_i, x\in X_j} \tag{13}$$

is numerically low-rank.

Proof. The proof of this theorem follows the same line of argument in [5, 34, 23], and below we outline the key idea. Denote the center of D_i by (r_i, s_i) and the center of X_j by x_j. For each $(r, s) \in D_i$ and $x \in X_j$, we write

$$\begin{aligned}\exp(i\omega(s-r)\cdot x) = \exp(i\omega((s-r)-(s_i-r_i))\cdot(x-x_j))\cdot \\ \exp(i\omega(s_i-r_i)\cdot x)\cdot\exp(i\omega(s-r)\cdot x_j)\cdot\exp(-i\omega(s_i-r_i)\cdot x_j).\end{aligned} \tag{14}$$

Note that for fixed D_i and X_j each of the last three terms is either a constant or depends only on x or (r, s). As a result, $\exp(i\omega(s-r)\cdot x)$ is numerically low-rank if and only if the first term $\exp(i\omega((s-r)-(s_i-r_i))\cdot(x-x_j))$ is so. Such a low-rank property can be derived from the conditions concerning the side-lengths of D_i and X_j. More precisely, since (r, s) resides in D_i with center (r_i, s_i), then

$$|(s-r)-(s_i-r_i)| \le \frac{1}{\sqrt{\omega}}. \tag{15}$$

Similarly, as x resides in X_j with center x_j, then

$$|x-x_j| \le \frac{1}{\sqrt{\omega}}. \tag{16}$$

Multiplying these two estimates results in the estimate

$$\omega|((s-r)-(s_i-r_i))\cdot(x-x_j))| \le 1 \tag{17}$$

for the phase of $\exp(i\omega((s-r)-(s_i-r_i))\cdot(x-x_j))$. Therefore,

$$\exp(i\omega((s-r)-(s_i-r_i))\cdot(x-x_j)) \tag{18}$$

for $(r,s)\in D_i$ or $x\in X_j$ is nonoscillatory and hence can be approximated effectively by applying, for example, Chebyshev interpolation in both the (r,s) and x variables. Since the degree of the Chebyshev polynomials only increases polylogarithmically with respect to the desired accuracy, $\exp(i\omega((s-r)-(s_i-r_i))\cdot(x-x_j))$ is numerically low-rank by construction. This proves that the submatrix A_{ij} defined in (13) is also numerically low-rank. ∎

3.3. Matrix factorization. In this subsection, we show that Theorem 1 guarantees a *low-complexity* factorization of the matrix A. Let the row and column indices of $A\in\mathbb{C}^{M^2\times N^2}$ be partitioned into index sets $\{D_i\}_{i=0}^{P_D-1}$ and $\{X_j\}_{j=0}^{P_X-1}$, respectively, as in Theorem 1. To simplify the presentation, we assume $P_X=P_D=P$, $|X_0|=\cdots|X_{P-1}|=N^2/P$, and $|D_0|=\cdots=|D_{P-1}|=M^2/P$.

Since the submatrix

$$A_{ij}:=[A(rs,x)]_{rs\in D_i,x\in X_j}$$

is numerically low-rank, assume that

$$A_{ij}\approx U_{ij}V_{ij}^*, \tag{19}$$

where $U_{ij}\in\mathbb{C}^{M^2/P\times t}$ and $V_{ij}\in\mathbb{C}^{N^2/P\times t}$. Here t can be taken to be the maximum of the numerical ranks of all submatrices A_{ij}. Theorem 1 implies that t is a small constant.

By applying (19) to each block A_{ij}, A can be approximated by

$$\begin{bmatrix} U_{00}V_{00}^* & U_{01}V_{01}^* & \cdots & U_{0(P-1)}V_{0(P-1)}^* \\ U_{10}V_{10}^* & U_{11}V_{11}^* & \cdots & U_{1(P-1)}V_{1(P-1)}^* \\ \vdots & & \ddots & \vdots \\ U_{(P-1)0}V_{(P-1)0}^* & U_{(P-1)1}V_{(P-1)1}^* & \cdots & U_{(P-1)(P-1)}V_{(P-1)(P-1)}^* \end{bmatrix}. \tag{20}$$

The next step is to write (20) into a factorized form. First, introduce U_i and V_j

$$U_i=\begin{bmatrix}U_{i0},U_{i1},\ldots,U_{i(P-1)}\end{bmatrix}\in\mathbb{C}^{M^2/P\times tP},\quad V_j=\begin{bmatrix}V_{0j},V_{1j},\ldots,V_{(P-1)j}\end{bmatrix}\in\mathbb{C}^{N^2/P\times tP}, \tag{21}$$

and define in addition

$$U=\begin{bmatrix}U_0 & & & \\ & U_1 & & \\ & & \ddots & \\ & & & U_{P-1}\end{bmatrix}\in\mathbb{C}^{M^2\times P^2t},\quad V^*=\begin{bmatrix}V_0{}^* & & & \\ & V_1{}^* & & \\ & & \ddots & \\ & & & V_{P-1}{}^*\end{bmatrix}\in\mathbb{C}^{P^2t\times N^2}. \tag{22}$$

In addition, introduce

$$\Sigma=\begin{bmatrix} \Sigma_{00} & \Sigma_{01} & \cdots & \Sigma_{0(P-1)} \\ \Sigma_{10} & \Sigma_{11} & \cdots & \Sigma_{1(P-1)} \\ \vdots & & \ddots & \vdots \\ \Sigma_{(P-1)0} & \Sigma_{(P-1)1} & \cdots & \Sigma_{(P-1)(P-1)} \end{bmatrix}\in\mathbb{C}^{P^2t\times P^2t}, \tag{23}$$

where the submatrix $\Sigma_{ij} \in \mathbb{C}^{Pt\times Pt}$ itself is a $P \times P$ block matrix with blocks of size $t\times t$. Σ_{ij} is defined to be zero everywhere except at the (j,i)th $t\times t$ block, which hosts a $t \times t$ identity matrix. In order to help understand the NN architecture discussed in the sections below, it is imperative to understand the meaning of Σ. Let us assume for simplicity that $t = 1$. Then for an arbitrary vector $z \in \mathbb{C}^{P^2}$, Σz essentially performs a "switch" that shuffles z as follows:

$$(\Sigma z)(jP + i) = z(iP + j), \quad i, j = 0, \ldots, P - 1. \tag{24}$$

In other words, the "switch" Σz amounts to reshaping z into a square matrix and transposing the matrix, followed by a vectorization of that matrix.

With the above definitions for U, V, and Σ, the approximation in (20) can be written compactly as

$$A \approx U\Sigma V^*. \tag{25}$$

Notice that although A has $M^2 \times N^2$ entries, using the factorization (25), A can be stored using $tP(M^2 + P + N^2)$ entries. In this paper, $P \approx \max(M, N)$ and M and N are typically on the same order. Therefore, instead of $O(N^4)$, one only needs $O(N^3)$ entries to parameterize the map A approximately using (25). Such a factorization is also used in [24] for the compression of Fourier integral operators.

We would like to comment on another property that may lead to further reduction in the parameters used for approximating A. Let us focus on any two submatrices A_{ij} and A_{ik} of A. For two regions X_j and X_k, where the center of X_j and X_k are x_j and x_k, respectively, $X_k = X_j + (x_k - x_j)$. Let $(r, s) \in D_i$. For $x \in X_j$ and $x' = x + (x_k - x_j) \in X_k$, we have

$$\begin{aligned} \exp(i\omega(s - r) \cdot x) &= g_1(r, s)h((r, s), x), \\ \exp(i\omega(s - r) \cdot x') &= g_2(r, s)h((r, s), x), \end{aligned} \tag{26}$$

where

$$\begin{aligned} g_1(r, s) = \exp(i\omega(s - r) \cdot x_j), \quad g_2(r, s) = \exp(i\omega(s - r) \cdot x_k), \\ h((r, s), x) = \exp(i\omega(s - r) \cdot (x - x_j)). \end{aligned} \tag{27}$$

Therefore, the low-rank factorizations of A_{ij} and A_{ik} are solely determined by the factorization of $h(rs, x)$. This implies that it is possible to construct low-rank factorizations for A_{ij} and A_{ik},

$$A_{ij} \approx U_{ij}V_{ij}^*, \quad A_{ik} \approx U_{ik}V_{ik}^* \tag{28}$$

such that $V_{ij}^* = V_{ik}^*$. Since this is true for all possible j, k, one can pick low-rank factorizations so that $V_0 = V_1 = \cdots = V_{P-1}$.

As a final remark in this section, this low-complexity factorization (25) for A can be easily converted to one for A^* since

$$A^* \approx V\Sigma^*U^*, \tag{29}$$

where U, Σ, V are provided in (21), (22), and (23).

3.4. Neural networks. Based on the low-rank property of A in section 3.2 and its low-complexity factorization in section 3.3, we propose new NN architectures for representing the inverse map $d \to \eta$ and the forward map $\eta \to d$.

3.4.1. NN for the inverse map $d \to \eta$. As pointed out earlier, $d \approx A\eta$ when η is sufficiently small. The usual filtered back-projection algorithm [28] solves the inverse problem $d \to \eta$ via

$$\eta \approx (A^*A + \epsilon I)^{-1} A^* d, \tag{30}$$

where ϵ is the regularization parameter. In the far field pattern problem, $(A^*A+\epsilon I)^{-1}$ can be understood as a deconvolution operator. To see this, a direct calculation reveals that

$$(A^*A)(x,y) = \sum_{rs \in R \times S} e^{i\omega(s-r)\cdot y} e^{-i\omega(s-r)x} = \sum_{rs \in R \times S} e^{-i\omega(s-r)(x-y)} \tag{31}$$

for $x, y \in X$. (31) shows that A^*A is a translation-invariant convolution operator. Therefore, the operator $(A^*A+\epsilon I)^{-1}$, as a regularized inverse of A^*A, simply performs a deconvolution. In summary, the above discussion shows that in order to obtain η from the scattering pattern d in the regime of small η, one simply needs to apply sequentially to d

- the operator A^*,
- a translation-invariant filter that performs the deconvolution $(A^*A + \epsilon I)^{-1}$.

Although these two steps might be sufficient when η is small, a nonlinear solution is needed when η is not so. For this purpose, we propose a nonlinear NN *SwitchNet* for the inverse map. There are two key ingredients in the design of SwitchNet.

- The first key step is the inclusion of a `Switch` layer that sends local information globally, as depicted in Figure 4. The structure of the `Switch` layer is designed to mimic the matrix-vector multiplication of the operator $A^* \approx V\Sigma^*U^*$ in (20). However, unlike the fixed coefficients in (20), as an NN layer, the `Switch` layer allows for tunable coefficients and learns the right values for the coefficients from the training data. This gives the architecture a great deal of flexibility.
- The second key step is to replace the linear deconvolution in the back-projection algorithm with a few convolution (`Conv`) layers with nonlinearities. This enriches the architecture with nonlinear capabilities when approximating the nonlinear inverse map. We use multiple layers since this enables us to aggregate information from a larger window, in case the kernel $(A^*A + \epsilon I)^{-1}$ is rather global.

Algorithm 1. SwitchNet for the inverse map $d \to \eta$ of far field pattern.

Require: $t, P_D, P_X, N, w, \alpha, L, d \in \mathbb{C}^{M \times M}$
Ensure: $\eta \in \mathbb{C}^{N \times N}$

$d_1 \leftarrow \texttt{Vect}[P_D](d)$
$d_2 \leftarrow \texttt{Switch}[t, P_D, P_X, N^2](d_1)$
$e_0 \leftarrow \texttt{Square}[P_X](d_2)$
for ℓ from 0 to $L-1$ **do**
    $\tilde{e}_{\ell+1} \leftarrow \texttt{Conv}[w, \alpha](e_\ell)$
    $e_{\ell+1} \leftarrow \texttt{ReLU}(\tilde{e}_{\ell+1})$
end for
$\eta \leftarrow \texttt{Conv}[w, 1](e_L)$
return η

The pseudocode for SwitchNet is summarized in Algorithm 1. The input d is a $\mathbb{C}^{M\times M}$ matrix, while the output η is a $\mathbb{C}^{N\times N}$ matrix. The first three steps of Algorithm 1 mimic the application of the operator $A^* \approx V\Sigma^* U^*$. The `Switch` layer does most of the work, while the `Vect` and `Square` layers are simply operations that reshape the input and output data to the correct matrix form at the beginning and the end of the `Switch` layer. In particular, `Vect` groups the entries of the two-dimensional field d according to squares defined by the partition $D_0, \ldots, D_{P_D-1}$, and `Square` does the opposite. The remaining lines of Algorithm 1 simply apply the `Conv` layers with window size w and channel number α.

These basic building blocks of SwitchNet are detailed in the following subsection. We also take the opportunity to include the details of the pointwise multiplication `PM` layer that will be used in later on.

3.4.2. Layers for SwitchNet. In this section we provide the details for the layers that are used in SwitchNet. Henceforth, we assume that the entries of a tensor are enumerated in the Python convention, i.e., going through the dimensions from the last one to the first. One operation that will be used often is a *reshape*, in which a tensor is changed to a different shape with the same number of entries and with the enumeration order of the entries kept unchanged.

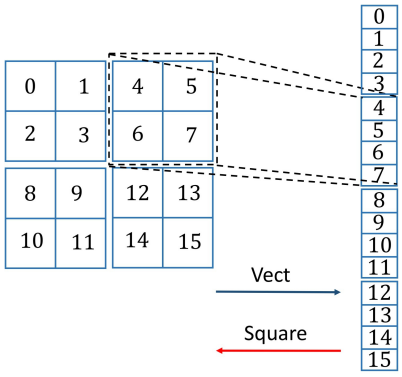


FIG. 3. *An illustration of the* `Vect` *and* `Square` *layers. Detailed descriptions of the layers are provided in section* 3.4.2. *For the purpose of illustration we let* $P = 4$. *The* `Vect` *layer vectorizes a* 4×4 *matrix on the left-hand side, according to the partitioning by* 2×2 *blocks, to give the size* 16 *vector on the right-hand side. The* `Square` *layer is simply the adjoint map of the* `Vect` *layer.*

Vectorize layer (Figure 3). $z_{\mathrm{O}} = \mathtt{Vect}[P](z_{\mathrm{I}})$ with input $z_{\mathrm{I}} \in \mathbb{C}^{n\times n}$. Henceforth we assume that $\sqrt{P}$ is an integer and $\sqrt{P}$ divides n. This operation partitions z_{I} into $\sqrt{P} \times \sqrt{P}$ square sub-blocks of equal size. Then each sub-block is vectorized, and the vectorized sub-blocks are stacked together as a vector in $\mathbb{C}^{n^2}$. Intuitively, these operations cluster the nearby entries in a sub-block together. The details of the `Vect` layer are given in the following:

- Reshape the z_{I} to a $\sqrt{P} \times \frac{n}{\sqrt{P}} \times \sqrt{P} \times \frac{n}{\sqrt{P}}$ tensor.
- Swap the second and the third dimensions to get a $\sqrt{P} \times \sqrt{P} \times \frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$ tensor.
- Reshape the result to an n^2 vector and set it to z_{O}.

Square layer (Figure 3). $z_{\mathrm{O}} = \mathtt{Square}[P](z_{\mathrm{I}})$ with input $z_{\mathrm{I}} \in \mathbb{C}^{n^2}$, where $\sqrt{P}$ is an integer. The output is $z_{\mathrm{O}} \in \mathbb{R}^{n\times n}$. Essentially, as the adjoint operator of the `Vect` layer, this layer fills up each square sub-block of the matrix z_{O} with a segment of entries in z_{I}. The details are given as follows:

- Reshape the z_{I} to a $\sqrt{P} \times \sqrt{P} \times \frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$ tensor.
- Swap the second and the third dimensions to get a $\sqrt{P} \times \frac{n}{\sqrt{P}} \times \sqrt{P} \times \frac{n}{\sqrt{P}}$ tensor.
- Reshape the result to an $n \times n$ matrix and set it to z_{O}.

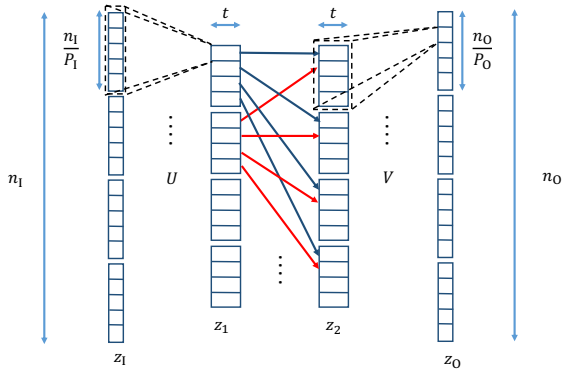


FIG. 4. *An illustration of the* `Switch` *layer; the detailed description of it is provided in section* 3.4.2. *For the purpose of illustration we let* $n_{\mathrm{I}} = n_{\mathrm{O}} = 20, P_{\mathrm{I}} = P_{\mathrm{O}} = 4$.

Switch layer (Figure 4). $z_{\mathrm{O}} = \texttt{Switch}[t, P_{\mathrm{I}}, P_{\mathrm{O}}, n_{\mathrm{O}}](z_{\mathrm{I}})$ with input $z_{\mathrm{I}} \in \mathbb{C}^{n_{\mathrm{I}}}$. It is assumed that n_{I} and n_{O} are integer multiples of P_{I} and P_{O}, respectively. This layer consists the following steps.

- Apply U^T to z_{I}:

$$z_1 = U^T z_{\mathrm{I}} \in \mathbb{C}^{P_{\mathrm{O}} P_{\mathrm{I}} t},$$

$$U^T = \begin{bmatrix} U_0^T & & \\ & \ddots & \\ & & U_{P_{\mathrm{I}}-1}^T \end{bmatrix}, \quad U_0^T, \ldots, U_{P_{\mathrm{I}}-1}^T \in \mathbb{C}^{tP_{\mathrm{O}} \times \frac{n_{\mathrm{I}}}{P_{\mathrm{I}}}}.$$

- Reshape z_1 to be a $\mathbb{C}^{P_{\mathrm{O}} \times P_{\mathrm{I}} \times t}$ tensor. Here we follow the Python convention of going through the dimensions from the last one to the first one. Then a permutation is applied to swap the first two dimensions to obtain a tensor of size $\mathbb{C}^{P_{\mathrm{O}} \times P_{\mathrm{I}} \times t}$. Finally, the result is reshaped to a vector $z_2 \in \mathbb{C}^{P_{\mathrm{I}} P_{\mathrm{O}} t}$, again going through the dimensions from the last to the first.
- Apply V to z_2:

$$z_{\mathrm{O}} = V z_2 \in \mathbb{R}^{n_{\mathrm{O}}},$$

$$V = \begin{bmatrix} V_0 & & \\ & \ddots & \\ & & V_{P_{\mathrm{O}}-1} \end{bmatrix}, \quad V_0, \ldots, V_{P_{\mathrm{O}}-1} \in \mathbb{C}^{\frac{n_{\mathrm{O}}}{P_{\mathrm{O}}} \times tP_{\mathrm{I}}}.$$

Here the nonzero entries of U, V are the trainable parameters. The `Switch` layer is illustrated in Figure 4.

Convolution layer. $z_{\mathrm{O}} = \texttt{Conv}[w, c_{\mathrm{O}}](z_{\mathrm{I}})$ with input $z_{\mathrm{I}} = \mathbb{C}^{n \times n \times c_{\mathrm{I}}}$. Here $c_{\mathrm{I}}, c_{\mathrm{O}}$ denote the input and output channel numbers and w denotes the window size. In this paper we only use the convolution layer with stride 1 and with zero padding:

$$z_{\mathrm{O}}(k_1,k_2,k_3) = \sum_{l_1=\max(0,k_1-\frac{w-1}{2})}^{\min(n-1,k_1+\frac{w-1}{2})} \sum_{l_2=\max(0,k_2-\frac{w-1}{2})}^{\min(n-1,k_2+\frac{w-1}{2})} \sum_{l_3=0}^{c_{\mathrm{I}}-1} W\left(l_1-k_1+\frac{w-1}{2}, l_2-k_2+\frac{w-1}{2}, l_3, k_3\right) z_{\mathrm{I}}(l_1,l_2,l_3) + b(k_3) \tag{32}$$

with $k_1,k_2 = 0,\dots,n-1,\ k_3 = 0,\dots,c_{\mathrm{O}}-1$. Here w is assumed to be odd in the presentation. Both $W \in \mathbb{C}^{w\times w\times c_{\mathrm{I}}\times c_{\mathrm{O}}}$ and $b \in \mathbb{C}^{c_{\mathrm{O}}}$ are trainable parameters.

ReLU layer. $z_{\mathrm{O}} = \texttt{ReLU}(z_{\mathrm{I}})$ with input $z_{\mathrm{I}} = \mathbb{C}^{n\times n\times c}$ where $\mathrm{ReLU}(z_{\mathrm{I}}) = \max(0, z_{\mathrm{I}})$ pointwise.

Pointwise multiplication layer. $z_{\mathrm{O}} = \texttt{PM}(z_{\mathrm{I}})$ with input $z_{\mathrm{I}} \in \mathbb{C}^{n\times n\times c_{\mathrm{I}}}$. It is defined as

$$z_{\mathrm{O}}(k_1,k_2) = W(k_1,k_2) z_{\mathrm{I}}(k_1,k_2) + b(k_1,k_2), \tag{33}$$

$k_1,k_2 = 0,\dots,n-1$. Both $W \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^{n\times n}$ are trainable parameters.

We remark that, among these layers, the `Switch` layer has the most parameters. If the input and output to the `Switch` layer both have size $n \times n$, the number of parameter is $2tPn^2$, where P is the number of squares that partition the input field and t is the rank of the low-rank approximation.

3.4.3. NN for the forward map $\eta \to d$. We move on to discuss the parameterization of the forward map $\eta \to d$. The proposal is based on the simple observation that *the inverse of the inverse map is the forward map.*

More precisely, we simply reverse the architecture of the inverse map proposed in Algorithm 1. This results in an NN presented Algorithm 2. The basic architecture of this NN involves applying a few layers of `Conv` first, followed by a `Switch` layer that mimics $A \approx U\Sigma V^*$.

Algorithm 2. SwitchNet for the forward map $\eta \to d$ of far field pattern.

Require: $t, P_D, P_X, M, w, \alpha, L, \eta \in \mathbb{R}^{N\times N}$
Ensure: $d \in \mathbb{R}^{M\times M\times 2}$

$\eta_0 \leftarrow \eta$
for ℓ from 0 to $L-1$ **do**
  $\tilde{\eta}_{\ell+1} \leftarrow \texttt{Conv}[w,\alpha](\eta_\ell)$
  $\eta_{\ell+1} \leftarrow \texttt{ReLU}(\tilde{\eta}_{\ell+1})$
end for
$d_1 \leftarrow \texttt{Conv}[w,1](\eta_L)$
$d_2 \leftarrow \texttt{Vect}[P_X](d_1)$
$d_3 \leftarrow \texttt{Switch}[t,P_X,P_D,M^2](d_2)$
$d \leftarrow \texttt{Square}[P_D](d_3)$
return d

We would also like to mention yet another possibility to parameterize the forward map $\eta \to d$ via a recurrent NN [27]. Let

$$E_{\mathrm{eff}} = E + EG_0E + EG_0EG_0E + \cdots =: E_1 + E_2 + E_3 + \cdots. \tag{34}$$

One can leverage the following recursion,

$$E_{k+1} = EG_0E_k, \quad k = 1,2,\dots K, \tag{35}$$

to approximate E_{eff} by treating each E_k as an $N^2 \times N^2$ image and using a recurrent NN. At the kth level of the recurrent NN, it takes E_k and E as inputs and outputs E_{k+1}. More specifically, in order to go from E_k to E_{k+1}, one first apply G_0 to each column of the image E_k; then each row of the image is reweighted by the diagonal matrix E. Stopping at the Kth level for a sufficiently large K, E_{eff} can be approximated by

$$E_{\text{eff}} \approx \sum_{i=1}^{K+1} E_i. \tag{36}$$

Once holding such an approximation to E_{eff}, we plug it into (10):

$$d(r,s) = \sum_{x\in X}\sum_{y\in X} e^{i\omega r\cdot x} E_{\text{eff}}(x,y) e^{-i\omega s\cdot y} = \sum_{x\in X} e^{i\omega r\cdot x}\left(\sum_{y\in X} E_{\text{eff}}(x,y) e^{-i\omega s\cdot y}\right). \tag{37}$$

This shows that the map from E_{eff} to d can be realized by applying a matrix product to E_{eff} first on the y-dimension, then on the x-dimension. If we view applying the Green's function G_0 as applying a convolution layer in an NN, the above discussion shows that the forward map can be obtained by first applying a recurrent NN followed by a convolutional NN. The main drawback of this approach is the large memory requirement (i.e., $N^2 \times N^2$) to store each individual E_k. In addition, the use of a recurrent NN may lead to difficulty in training [29] due to the issue of exploding or vanishing gradient. Moreover, since the weights for parameterizing G_0 are shared over multiple layers in the recurrent NN, one might not be able to efficiently use back-propagation, which may lead to a longer training time. These are the main reasons why we decided to adopt the approach in Algorithm 2.

3.5. Numerical results. In this section, we present numerical results of SwitchNet for far field pattern at a frequency $\omega \approx 60$. The scatterer field $\eta(x)$ supported in $\Omega = [-0.5, 0.5]^2$ is assumed to be a mixture of Gaussians:

$$\sum_{i=1}^{n_s} \beta \exp\left(-\frac{|x-c_i|^2}{2\sigma^2}\right) \tag{38}$$

where $\beta = 0.2$ and $\sigma = 0.015$. When preparing the training and testing examples, the centers $\{c_i\}_{i=1}^{n_s}$ of the Gaussians are chosen to be uniformly distributed within Ω. The number n_s of the Gaussians in the mixture is set to vary between 2 and 4. In the numerical experiments, the domain $\Omega = [-0.5, 0.5]^2$ is discretized by an 80×80 Cartesian grid X. To discretize the source and receiver directions, we set both R and S to be a set of 80 equally spaced unit directions on $\mathbb{S}^1$. Therefore, in this example, $N = M = 80$.

In Algorithm 1, the parameters are specified as $t = 3$ (rank of the low-rank approximation), $P_X = 8^2$, $P_D = 4^2$, $w = 10$ (window size of the convolution layers), $\alpha = 18$ (channel number of the convolution layers), and $L = 3$ (number of convolution layers), resulting in 3100K number of parameters. The parameters for Algorithm 2 are chosen to be $t = 4$, $P_X = 8^2$, $P_D = 4^2$, $w = 10$, $\alpha = 24$, and $L = 3$, with a total of 4200K parameters. Note that for both algorithms the number of parameters is significantly less than those of a fully connected NN, which has at least $80^4 = 40960$K parameters.

SwitchNet is trained with the ADAM optimizer [18] in Keras [7] with a step size of 0.002 and a mini-batch size of size 200. The optimization is run for 2500 epochs.

Both the training and testing data sets are obtained by numerically solving the forward scattering problem with an accurate finite difference scheme with a perfectly matched layer. In the experiment, 12.5K pairs of (η, d) are used for training, and another 12.5K pairs are reserved for testing. The errors are reported using the mean relative errors

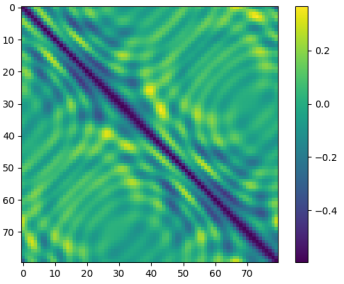
$$\frac{1}{N_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}\frac{\|d_i^{\text{NN}}-d_i\|_F}{\|d_i\|_F},\quad \frac{1}{N_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}\frac{\|\eta_i^{\text{NN}}-\eta_i\|_F}{\|\eta_i\|_F}, \tag{39}$$

where d_i^{NN} and d_i denote the predicted and ground truth scattering patterns, respectively, for the ith testing sample, and η_i^{NN} and η_i denote the predicted and ground truth scatterer field, respectively. Here $\|\cdot\|_F$ is the Frobenius norm.

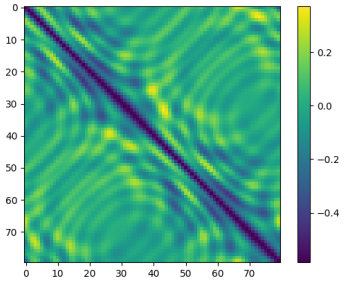
Table 1 summarizes the test errors for Gaussian mixtures with different choices of n_s. For the purpose of illustration, we show the predicted d and η by SwitchNet along with the ground truth in Figure 5 for one typical test sample.

TABLE 1
Prediction error of SwitchNet for the maps $\eta \to d$ and $d \to \eta$ for far field pattern.

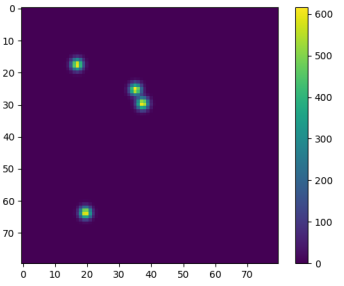
n_s	Forward map	Inverse map
2	9.4e-03	1.2e-02
3	4.0e-02	1.4e-02
4	4.8e-02	2.4e-02



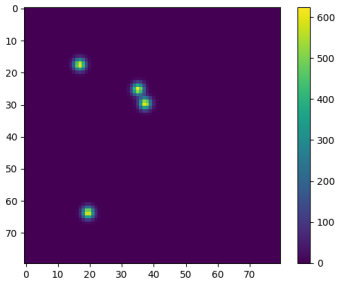
(a) Ground truth scattering pattern d.



(b) Predicted scattering pattern d^{NN}.



(c) Ground truth scatterers $\eta = \omega^2(1/c^2 - 1/c_0^2)$.



(d) Predicted scatterers η^{NN}.

FIG. 5. *Results for a typical instance of the far field pattern problem with $n_s = 4$.* (a) *The ground truth scattering pattern.* (b) *The scattering pattern predicted by SwitchNet with a* 4.9e-02 *relative error.* (c) *The ground truth scatterers.* (d) *The scatterers predicted by SwitchNet with a* 2.4e-02 *relative error.*

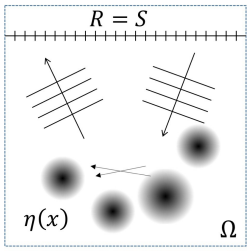


FIG. 6. *Illustration of a simple seismic imaging setting. The sources (S) and receivers (R) are located near the surface level (top) of the domain Ω. The scatterer field η(x) is assumed to be well separated from the sources and the receivers.*

4. SwitchNet for seismic imaging.

4.1. Problem setup. This section considers a two-dimensional model problem for seismic imaging. The scatterer $\eta(x)$ is again assumed to be supported in a domain Ω with an $O(1)$ diameter, after appropriate rescaling. Ω is discretized with a Cartesian grid $X = \{x\}_{x\in X}$ at the rate of at least a few points per wavelength. Compared to the source and receiver configurations in section 3.1, the experiment setup here is simpler. One can regard both $S = \{s\}_{s\in S}$ and $R = \{r\}_{r\in R}$ to be equal to a set of uniformly sampled points along a horizontal line near the top surface of the domain. The support of η is at a certain distance below the top surface so that it is well separated from the sources and the receivers (see Figure 6 for an illustration of this configuration).

The source and receiver operators in (6) take a particularly simple form. For the sources, the operator $(G_0\Pi_S)$ is simply given by sampling:

$$(G_0\Pi_S)(x, s) = G_0(x, s).$$

Similarly for the receivers, the operator $(\Pi_R^T G_0)$ is given by

$$(\Pi_R G_0)(r, y) = G_0(r, y).$$

After plugging these two formulas back into (6), one arrives at the following representation of the observation data $d(r, s)$ for $r \in R$ and $s \in S$:

$$d(r, s) = \sum_{x\in X}\sum_{y\in X} G_0(r, x)(E + EG_0E + \cdots)(x, y)G_0(y, s).$$

4.2. Low-rank property. Following the approach taken in section 3.2, we start with the linear approximation under the assumption that $\eta(x)$ is weak. Since $E = \mathrm{diag}(\eta)$, the first-order approximation is

$$d(r, s) \approx \sum_{x\in X} G_0(r, x)G_0(x, s)\eta(x). \tag{40}$$

By regarding η as a vector in $\mathbb{R}^{N^2}$ and d as a vector $\in \mathbb{C}^{M^2}$, one obtains the linear system

$$d \approx A\eta, \quad A \in \mathbb{C}^{M^2 \times N^2}, \tag{41}$$

where the element A at $(r, s) \in R \times S$ and $x \in X$ is given by

$$A(rs, x) = G_0(r, x)G_0(x, s) = G_0(r, x)G_0(s, x). \tag{42}$$

Under the assumptions that the sources S and receivers R are well separated from the support of $\eta(x)$ and that $c_0(x)$ varies smoothly, the matrix A satisfies a low-rank property similar to Theorem 1. To see this, we again partition X into Cartesian squares $X_0, \ldots, X_{P_X-1}$ of side-length equal to $1/\sqrt{\omega}$. Since $R = S$ is now the restriction of X on the surface level, this partition also induces a partitioning for $R \times S$. When $c_0(x)$ varies smoothly, it is shown (see, for example, [10]) that the restriction of the matrix $[G_0(r, x)]_{r\in R, x\in X}$ (or $[G_0(s, x)]_{s\in S, x\in X}$) to each piece of the partitioning is numerically low-rank. Since the matrix A is obtained by taking the Khatri–Rao product [25] of $[G_0(r, x)]_{r\in R, x\in X}$, $[G_0(s, x)]_{s\in S, x\in X}$, the low-rank property is preserved with the guarantee that the rank at most squares in the worst case.

By following the same argument in section 3.3, one can show that the matrix A has a low-complexity matrix factorization $A \approx U\Sigma V^*$ of exactly the same structure as (20). The corresponding factorization for A^* is $A^* \approx V\Sigma^* U^*$.

4.3. Neural networks. Based on the low-rank property in section 4.2, we propose here SwitchNet for seismic imaging.

4.3.1. NN for the inverse map $d \to \eta$. When the linear approximation is valid (i.e., (41) holds), η can be obtained from d via a filtered projection approach (called migration in the seismic community),

$$\eta \approx (A^*A + \epsilon I)^{-1} A^* d, \tag{43}$$

where ϵI is a regularizing term. Since A^* has a low-complexity factorization $A^* \approx V\Sigma^* U^*$, the application A^* to a vector can be represented by a `Switch` layer.

Concerning the $(A^*A + \epsilon I)^{-1}$ term, note that

$$(A^*A)(x, y) \approx \sum_{rs\in R\times S} \overline{G_0(x, r)G_0(x, s)}G_0(r, y)G_0(s, y), \tag{44}$$

which, unlike (31), is no longer a translation-invariant kernel as the data gathering setup is not so. For example, even when the background velocity $c_0(x) = 1$, the different terms of the Green's function $G_0(\cdot)$ in (44) scale like

$$\frac{1}{\sqrt{|x-r|}}, \quad \frac{1}{\sqrt{|x-s|}}, \quad \frac{1}{\sqrt{|y-r|}}, \quad \frac{1}{\sqrt{|x-s|}},$$

which fail to give a translation-invariant kernel of form $K(x - y)$. As a direct consequence, the operator $(A^*A + \epsilon I)^{-1}$ is not translation-invariant either.

In order to capture the loss of translation invariance, we include an extra pointwise multiplication layer `PM` (defined in section 3.4.2) when dealing with the inverse map. The pseudocode of the NN for the inverse map is given in Algorithm 3.

Algorithm 3. SwitchNet for the inverse map $d \to \eta$ of seismic imaging.

Require: $t, P_D, P_X, N, w, \alpha, L, d \in \mathbb{C}^{M\times M}$
Ensure: $\eta \in \mathbb{R}^{N\times N}$
$d_1 \leftarrow \texttt{Vect}[P_D](d)$
$d_2 \leftarrow \texttt{Switch}[t, P_D, P_X, N^2](d_1)$
$e_0 \leftarrow \texttt{Square}[P_X](d_2)$
for ℓ from 0 to $L-1$ **do**
  $\tilde{e}_{\ell+1} \leftarrow \texttt{Conv}[w, \alpha](e_\ell)$
  $e_{\ell+1} \leftarrow \texttt{ReLU}(\tilde{e}_{\ell+1})$
end for
$\eta \leftarrow \texttt{Conv}[w, 1](e_L)$
$\eta \leftarrow \texttt{PM}(\eta)$
return η

Algorithm 4. SwitchNet for the forward map $\eta \to d$ of seismic imaging.

Require: $t, P_D, P_X, M, w, \alpha, L, \eta \in \mathbb{C}^{N\times N}$
Ensure: $d \in \mathbb{C}^{M\times M}$
$\eta_0 \leftarrow \texttt{PM}(\eta)$
for ℓ from 0 to $L-1$ **do**
  $\tilde{\eta}_{\ell+1} \leftarrow \texttt{Conv}[w, \alpha](\eta_\ell)$
  $\eta_{\ell+1} \leftarrow \texttt{ReLU}(\tilde{\eta}_{\ell+1})$
end for
$d_1 \leftarrow \texttt{Conv}[w, 1](\eta_L)$
$d_2 \leftarrow \texttt{Vect}[P_X](d_1)$
$d_3 \leftarrow \texttt{Switch}[t, P_X, P_D, M^2](d_2)$.
$d \leftarrow \texttt{Square}[P_D](d_3)$
return d

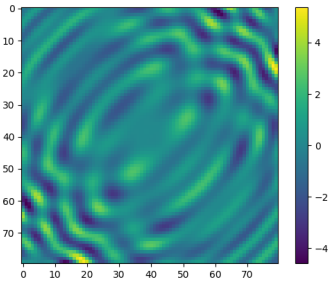
4.3.2. NN for the forward map $\eta \to d$. As in section 3.4.3, for the forward map from $\eta \to d$, we simply reverse the architecture of the NN for the inverse map in Algorithm 3. For completeness we detail its structure in Algorithm 4. The main difference between Algorithm 2 and Algorithm 4 is again the inclusion of an extra pointwise multiplication layer.

4.4. Numerical results. In the numerical experiments, we set $\Omega = [-0.5, 0.5]^2$ and discretize it by a 64×64 Cartesian grid. As mentioned before, the sources S and the receivers R are located on a line near the top surface of Ω, similar to the setting in Figure 6. This line is discretized uniformly with $M = 80$ points. Therefore, the size of η and d are 64×64 and 80×80, respectively. We assume a Gaussian mixture model for η as in (38), where $\beta = 0.2, \sigma = 0.015$. Unlike before, the centers $\{c_i\}_{i=1}^{n_s}$ are kept away from the top surface of Ω in order to ensure that they are well separated from the sources and receivers.

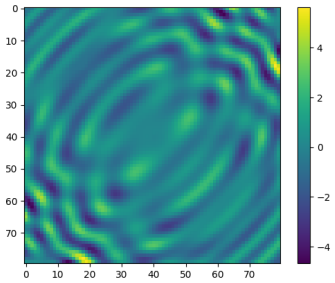
In Algorithm 3 and Algorithm 4, the parameters are set to be $t = 3$, $P_X = 8^2$, $P_D = 4^2$, $N = 64$, $M = 80$, $w = 8$, $\alpha = 18$, and $L = 3$, resulting in NNs with 2900K parameters. The procedure of training the NNs is the same as the one used in section 3.5. Table 2 presents the test errors for this model problem. The predicted and the ground truth d, η are visually compared in Figure 7 for one typical test sample.

TABLE 2
Prediction error of SwitchNet for the maps $\eta \to d$ and $d \to \eta$ for seismic imaging.

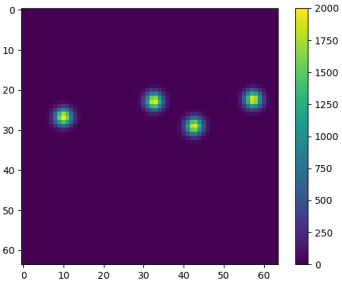
n_s	Forward map	Inverse map
2	5.6e-02	2.1e-02
3	7.7e-02	2.2e-02
4	8.0e-02	5.1e-02



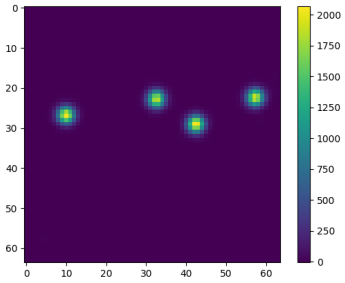
(a) Ground truth scattering pattern d.



(b) Predicted scattering pattern d^{NN}.



(c) Ground truth scatterers $\eta = \omega^2(1/c^2 - 1/c_0^2)$.



(d) Predicted scatterers η^{NN}.

FIG. 7. *Results for a typical instance of the seismic imaging setting with $n_s = 4$. (a) The ground truth scattering pattern. (b) The scattering pattern predicted by SwitchNet with a* 7.7*e*-02 *relative error. (c) The ground truth scatterers. (d) The scatterers predicted by SwitchNet with a* 6.9*e*-02 *relative error.*

5. Discussion. In this paper, we introduce an NN, SwitchNet, for approximating forward and inverse maps arising from the time-harmonic wave equation. For these maps, local information at the input has a global impact at the output; therefore, they generally require the use of a fully connected NN in order to parameterize them. Based on certain low-rank property that arises in the linearized operators, we are able to replace a fully connected NN with the sparse SwitchNet, thus reducing complexity dramatically. Furthermore, unlike convolutional NNs with local filters, the proposed SwitchNet connects the input layer with the output layer globally. This enables us to represent a highly oscillatory wave field resulting from scattering problems and to solve for the associated inverse problems.

REFERENCES

[1] T. Arens, A. Lechleiter, and D. R. Luke, *MUSIC for extended scatterers as an instance of the factorization method*, SIAM J. Appl. Math., 70 (2009), pp. 1283–1304.

[2] G. Bao, P. Li, J. Lin, and F. Triki, *Inverse scattering problems with multi-frequencies*, Inverse Problems, 31 (2015), 093001.

[3] J.-P. Berenger, *A perfectly matched layer for the absorption of electromagnetic waves*, J. Comput. Phys., 114 (1994), pp. 185–200.

[4] C. Borges, A. Gillman, and L. Greengard, *High resolution inverse scattering in two dimensions using recursive linearization*, SIAM J. Imaging Sci., 10 (2017), pp. 641–664.

[5] E. Candès, L. Demanet, and L. Ying, *A fast butterfly algorithm for the computation of Fourier integral operators*, Multiscale Model. Simul., 7 (2009), pp. 1727–1750.

[6] G. Carleo and M. Troyer, *Solving the quantum many-body problem with artificial neural networks*, Science, 355 (2017), pp. 602–606.

[7] F. Chollet, *Keras*, http://keras.io, 2017.

[8] D. Colton and R. Kress, *Inverse Acoustic and Electromagnetic Scattering Theory*, 3rd ed., Appl. Math. Sci. 93, Springer, New York, 2013, https://doi.org/10.1007/978-1-4614-4942-3.

[9] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12.

[10] B. Engquist and H. Zhao, *Approximate separability of the Green's function of the Helmholtz equation in the high frequency limit*, Comm. Pure Appl. Math., 71 (2018), pp. 2220–2274, https://doi.org/10.1002/cpa.21755.

[11] Y. Fan, L. Lin, L. Ying, and L. Zepeda-Núnez, *A Multiscale Neural Network Based on Hierarchical Matrices*, preprint, arXiv:1807.01883, 2018.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016.

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

[14] J. Han, L. Zhang, R. Car, and W. E, *Deep potential: A general representation of a many-body potential energy surface*, Commun. Comput. Phys., 23 (2018), pp. 629–639.

[15] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science, 313 (2006), pp. 504–507.

[16] Y. Khoo, J. Lu, and L. Ying, *Solving Parametric PDE Problems with Artificial Neural Networks*, preprint, arXiv:1707.03351, 2017.

[17] Y. Khoo, J. Lu, and L. Ying, *Solving for High Dimensional Committor Functions Using Artificial Neural Networks*, preprint, arXiv:1802.10275, 2018.

[18] D. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, preprint, arXiv:1412.6980, 2014.

[19] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Netw., 9 (1998), pp. 987–1000.

[20] S. Larsson and V. Thomée, *Partial Differential Equations with Numerical Methods*, Texts Appl. Math. 45, Springer-Verlag, Berlin, 2009, paperback reprint of the 2003 edition.

[21] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[22] Y. Li, X. Cheng, and J. Lu, *Butterfly-Net: Optimal Function Representation Based on Convolutional Neural Networks*, preprint, arXiv:1805.07451, 2018.

[23] Y. Li and H. Yang, *Interpolative butterfly factorization*, SIAM J. Sci. Comput., 39 (2017), pp. A503–A531.

[24] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying, *Butterfly factorization*, Multiscale Model. Simul., 13 (2015), pp. 714–732.

[25] S. Liu and G. Trenkler, *Hadamard, Khatri-Rao, Kronecker, and other matrix products*, Int. J. Inf. Syst. Sci, 4 (2008), pp. 160–177.

[26] Z. Long, Y. Lu, X. Ma, and B. Dong, *PDE-Net: Learning PDEs from Data*, preprint, arXiv:1710.09668, 2017.

[27] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, *Recurrent neural network based language model*, in Proceedings of the Eleventh Annual Conference of the International Speech Communication Association, 2010.

[28] F. Natterer, *The Mathematics of Computerized Tomography*, Classics Appl. Math. 32, SIAM, Philadelphia, 2001, https://doi.org/10.1137/1.9780898719284, reprint of the 1986 original.

[29] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks*, in Proceedings of the International Conference on Machine Learning, 2013, pp. 1310–1318.

[30] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2015, pp. 234–241.
[31] K. Rudd and S. Ferrari, *A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks*, Neurocomputing, 155 (2015), pp. 277–285.
[32] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks, 61 (2015), pp. 85–117.
[33] Z. Wei and X. Chen, *Deep-learning schemes for full-wave nonlinear inverse scattering problems*, IEEE Trans. Geosci. Remote Sensing, 57 (2018), pp. 1849–1860.
[34] L. Ying, *Sparse Fourier transform via butterfly algorithm*, SIAM J. Sci. Comput., 31 (2009), pp. 1678–1694.