Journal of Computational Physics 470 (2022) 111592

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A physically-informed deep-learning model using time-reversal for locating a source from sparse and highly noisy sensors data

Adar Kahana [a,*], Eli Turkel [a], Shai Dekel [a], Dan Givoli [b]

[a] Department of Applied Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel
[b] Department of Aerospace Engineering, Technion, Hairs 32000, Israel

ARTICLE INFO

Article history:
Received 6 November 2021
Received in revised form 7 August 2022
Accepted 29 August 2022
Available online 5 September 2022

Keywords:
Time-reversal
Inverse problems
Sensors
Learning
Physically-informed

ABSTRACT

We approximate the underwater acoustic wave problem for locating sources in that medium. We create a time dependent synthetic data-set of sensor recorded pressures, based on a small set of sensors placed in the domain, and perturb this data with high random multiplicative noise. We show that reference time-reversal based method struggles with high noise, and a naive deep-learning method also fails. We propose a method, based on physically-informed neural-networks and time-reversal, for approximating the source location even in the presence of high sensors noise.

1. Introduction

In this work we present a construction of a deep-learning (DL) based solution for the inverse problems of source refocusing, which performs well even with high levels of sensor measurement noise. We simulate acoustic waves propagating in an underwater medium and record the data, over time, at a small number of sensors placed in the domain. Given these measurements, the properties of the medium and the location of the sensors, we aim to find the area of the original source that generated these measurements.

Since we only have information at a very few locations, this is a highly ill-posed inverse problem. Inverse problems [1,6,11,14,24,25], are ill-posed and therefore the desired solution does not necessarily exist and if so, may not be unique. Furthermore, small perturbations in the data may lead to large changes in the solution. Hence, the presence of measurement noise in the sensors makes reconstructing the initial data very challenging. The application of inverse problems includes a variety of fields such as acoustics, geophysics, material testing (including non-destructive testing) and more [1–3,16,23].

In a physical experiment, an initial pulse creates a wave that propagates throughout the medium from time 0 to time T (also called the forward process). The wave pressure is then recorded in a small set of sensors placed in the interior of the domain (in section 4.3 we discuss a popular case where sensors are placed on the boundary of the domain). We can use only the recorded pressures for analysis. An example of synthetic sensor recordings is shown in Fig. 1. In this example we see of a series of synthetically generated recordings at a single sensor. The upper figure shows recorded data of an incident (free space) signal. The middle shows slightly perturbed recorded data of the same incident signal and the lower shows

* Corresponding author.
E-mail address: adarkahana@gmail.com (A. Kahana).

https://doi.org/10.1016/j.jcp.2022.111592
0021-9991/

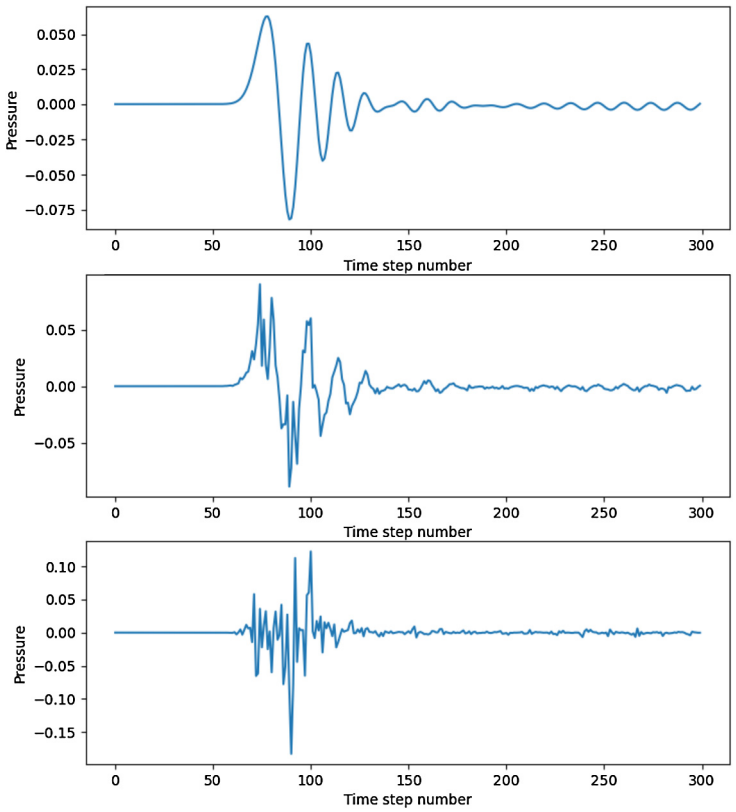


Fig. 1. Example of a series of synthetically generated recordings in a single sensor. The upper figure shows noise-less data of the incident (free space) signals while the middle and lower ones show slightly and highly perturbed signals (respectively).

a highly perturbed signal (the perturbations in this case are normally distributed multiplicative noise, further discussed in section 4.1).

It is often sufficient to find the location of the source, rather than the full initial pattern. Many studies have tackled this problem. One approach includes using the Time-Reversal (TR) [7,9,12,13,17,22] method to back propagate the waves through time until the initial source is found. One drawback is that the backward process has to be computed which is a costly procedure. Another drawback is that when using a very small number of sensors the method struggles (in real applications the number of sensors may be limited by budgets and or physical constraints). Another approach is approximating the source location from the arrival time of the incident wave [4,5,18,21,26]. In addition, one can use the adjoint method for this inverse problem [31]. These methods usually require more a-priori knowledge about the problem.

The use of machine-learning (ML) for solving PDE related problems is accelerating rapidly [19,20,27]. ML algorithms aim to fit input data to corresponding outputs. Learning methods have many strengths such as the ability to fit a non-linear problem, robustness to the problem definition, and rapid convergence (compared to optimization problems). A major drawback of learning based algorithms is that, while performing well on the specific task they are designed for, they often struggle when injected with variant data. For example, a classifier built to locate cars in an image, may fail to locate trucks even though the vehicles are very similar. Using ML for PDE related applications is a challenge since the input data often has measurement noise, which makes it difficult for the ML algorithm to fit the problem.

In this work we use a numerical scheme to synthetically generate data of forward acoustic wave propagation. We store data only in a small set of sensors placed around the domain. This mimics a physical experiment. Then, we use this data to approximate the location of the support of the source using TR and ML. We propose a deep-learning based method that performs well even with high measurement noise (noisy data). We create a method that utilizes both deep-learning and time-reversal to achieve high accuracy, and noise robustness. We formulate the problem as a data-driven problem with time dependent data at a small set of sensors. The first part of the method is a neural-network that estimates the noise level of the sensors data. We then train a neural-network for identifying the source. In this part we incorporate the TR method, known for being robust to noise [7], within the neural-network. For each sample, we infer the noise level using the noise level inference neural-network and based on that we use a TR-based neural-network trained to infer the source location for measurements with similar noise level. We discuss the inverse problem mentioned above and show the performance of the method, being able to locate sources in the presence of high measurement noise. The computations are done in a two spatial dimensions setup, but this approach can also be implemented for three dimensions with an increase in the

computational effort. In addition, we add a physically-informed loss term that incorporates the wave problem into the deep-learning algorithm and yields better results.

We point out that there are naive approaches for source refocusing using deep-learning. We initially tried a simple approach where we train a network with sensor data as inputs and the source images as outputs. For the noise-less case we achieved an excellent reconstruction. However, already at 10% of measurement noise (the noise will be defined later in this paper) the network fails even when training the network using noisy measurements. This setup is described in section 4.2. In addition, as shown in the results section (section 4), using TR alone (in the presence of high noise) does not produce results that allow one to interpret and extract the sources locations. The failure of the naive approach and the standalone TR method [13] motivates the construction of an improved methodology.

2. Numerical development

2.1. Mathematical formulation of the physical problem

The wave problem is given by

$$\begin{cases} \ddot{u}(\vec{x},t) = \nabla\cdot(c^2(\vec{x})\nabla u(\vec{x},t)) & \vec{x}\in\Omega, \quad t\in(0,T] \\ u(\vec{x},0) = u_0(\vec{x}) & \vec{x}\in\Omega \\ \dot{u}(\vec{x},0) = v_0(\vec{x}) & \vec{x}\in\Omega \\ u(\vec{x},t) = f(\vec{x},t) & \vec{x}\in\partial\Omega_1, \quad t\in[0,T] \\ \frac{\partial u}{\partial \mathbf{n}}(\vec{x},t) = g(\vec{x},t) & \vec{x}\in\partial\Omega_2, \quad t\in[0,T] \qquad \partial\Omega_1\cup\partial\Omega_2 = \partial\Omega \end{cases} \tag{1}$$

where $u(\vec{x},t)$ is the wave amplitude or acoustic wave pressure, and $c(\vec{x})$ is the wave propagation speed (assumed constant throughout this paper). The problem initial conditions are set by u_0, v_0 for the initial acoustic wave pressure and velocity, respectively. The boundary conditions are set by f, g for the Dirichlet and Neumann boundary condition types, respectively. Throughout this paper, Ω is considered a two dimensional rectangular domain, and the initial conditions u_0, v_0 are compactly supported in Ω. In addition, throughout the paper we discuss only Dirichlet boundary condition, so $\partial\Omega = \partial\Omega_1$. The specific problems (choice of initial and boundary conditions, domain, parameters, etc.) are defined in section 4.

For given initial conditions (u_0, v_0) and boundary conditions (f, g), there exists a unique solution. In addition, since the wave problem (1) is well-posed, small changes in the initial conditions result in small changes in the solution. Solving the problem through time (finding $u(\vec{x},t)$ that satisfies the wave equation given initial and boundary conditions) is often referred to as solving the forward problem. The source refocusing problem, which is the focus of this work, relates to solving the inverse wave propagation problem. In this case, we consider measurements of the solution $u(\vec{x},t)$ at a very small set of spatial locations and at equally spaced time steps, and use them to reconstruct the support of the initial source. In this work we only use the pressures (u), while the velocities (v), that may be nonzero, are ignored (not stored nor updated at any point, including the source velocities v_0). This inverse problem is ill-posed. In the following subsections we discuss how to approximate the solution of the forward problem, and create synthetic data for solving the inverse problem. In a real-physical experiment, the data is gathered from measurement devices.

2.2. Numerical scheme for the forward problem

We discuss a method to approximate the solution of the forward problem to create synthetically generated data. To approximate the acoustic pressures in the domain we use a finite difference scheme. We discuss the problem in two spatial dimensions, but the method can be implemented for three spatial dimensions as well. We use a rectangular grid of size $N_x \times N_y$, with equally spaced spatial nodes with lengths Δx and Δy in each axis. We choose an initial condition, specify a Dirichlet boundary condition on the entire external boundary, and compute the solution inside the domain. The method to choose an initial condition is explained in detail in section 2.3. The sensors that "record" the solution are grid points (nodes) $\{(x_k, y_k)\}_{k=1}^{N_{sensors}}$, and the number of sensors $N_{sensors}$ is much smaller than the number of nodes on the grid (typically 5-10 sensors in physical experiments).

We approximate the solution of the wave problem in space and time using the explicit compact second order accurate finite difference central difference (FDCD) scheme in both space and time. The time step is limited by the Courant-Friedrichs-Lewy condition:

$$\Delta t \le \frac{1}{c_{max}\sqrt{2\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right)}}, \tag{2}$$

where Δx and Δy are the distances between two adjacent grid points in the directions x and y respectively.

We begin with the given initial condition and use the numerical scheme to propagate the waves for N_{steps} time steps (forward process) and record the data in the sensors. After simulating the forward process we get a matrix of size $N_{steps} \times$

$N_{sensors}$, containing the synthetically generated acoustic pressure measurements received in all the sensors in each time step. The sensor data is then perturbed by a normally distributed multiplicative noise (detailed formulation given in section 4.1). We synthetically perturb the sensors data with multiplicative noise because the amount of noise scales with the acoustic pressure, and with a large amount of noise the original signal is not recoverable. It is well known that signal processing under multiplicative noise is more difficult than under additive noise, due to the nonlinear nature of multiplicative noise. Throughout the paper we discuss the multiplicative noise and in section 4.3 we explore additive noise as well. We also experiment with changing the noise distribution, using either normally or uniformly distributed noise.

2.3. Sources dataset

We formulate the source refocusing problem as a data-driven problem. We choose a random node on the grid which serves as the location of the source that starts the emission of the acoustic waves. Each random choice will be called a sample, and the total number of samples is $N_{samples}$. The initial condition is a compactly supported pulse, often implemented as a point source or a thin Gaussian (we also experiment with a thick Gaussian). We experiment with different sources and elaborate on how we choose them in section 4. For each sample we simulate the forward process and record the pressure in the sensors. Each sample has unique sensor recordings - different source locations produce different pressures in the sensors over time. As a result we get sensor recordings $\{u_m(t_n, x_k, y_k)\}_{n=1,k=1}^{N_{steps},N_{sensors}}$ for each sample $m = 1, ..., N_{samples}$. After simulating the forward process for all the samples, we get a 3-dimensional matrix of size $N_{samples} \times N_{steps} \times N_{sensors}$. This is the input to the learning algorithm. The specific choices of parameters are given in section 4. Note, that we do not store nor use the velocities $\{v_m(t_n, x_k, y_k)\}_{n=1,k=1}^{N_{steps},N_{sensors}}$. As additional inputs to the machine learning process, the velocities add another complication (more data to handle requires a deeper and wider network, resulting in more computational resources required). We expect a trade-off between higher accuracy and less efficiency when using the velocities.

The goal of the DL model is to output $\tilde{u}_0$ from which one can extract the location of the source given by the initial condition u_0. We are only interested in finding the location of the source. For training a DL algorithm we need to define each sample as input and output. The DL algorithm is trained to, given these inputs, infer the outputs. We assume the output, also referred to as label, is a compactly supported square blob inside the domain, where inside the blob each node has value 1 and outside it, the value 0. The formulation is given by:

$$u_0^m(x, y) = \begin{cases} 1, & x_m - S_{support} \le x \le x_m + S_{support} \quad \text{and} \quad y_m - S_{support} \le y \le y_m + S_{support} \\ 0, & \text{otherwise} \end{cases}, \tag{3}$$

where (x_m, y_m) are the coordinates picked for the source of the m^{th} sample, $S_{support}$ is the chosen size for the support and $u_0^m(x, y, 0)$ is the initial condition of the m^{th} sample. In section 4 we define the $S_{support}$ used in the numerical experiments. The source can be placed anywhere in the domain including on top of a sensor. In some applications, the device used to emit the wave can also record pressure - in other words, the source is also a sensor. In this work we have a single source and a small set of sensors and the location of the source is completely arbitrary. We mark the labels set as $\{\mathcal{O}_q^{source}\}_{q=1}^{N_{samples}}$. We get a 3-dimensional matrix of size $N_{samples} \times N_x \times N_y$. For each sample, the binary image of the source is the label we want to predict in order to find the location of the source. These binary images are different from the actual source pattern (shown in Fig. 8 in the leftmost and rightmost images of the noise-less case). However, since we are interested in the location of the source, we can calculate the center of mass of the binary images to get the location, and compare it to the known source location. This is also how we evaluate the performance of the method (as explained in section 4.4). The method of using binary labels was chosen so we can use the physically-informed term (introduced in section 3.4) for this model.

2.4. Time-reversal

The idea behind the TR procedure is to solve the wave equation "backwards in time" and examine events that occurred in the past, exploiting the reversibility property of the acoustic wave equation in a non-dissipative medium. The TR procedure is known to be robust to measurement noise [7,9,12,15]. We are interested in utilizing the TR process to compensate for the sensitivity to measurement noise, often experienced when using deep-learning algorithms.

For the same time interval $[0, T]$, used in the forward process, we substitute $\tau = T - t$ and solve the same wave equation (1) from $\tau = 0$ to $\tau = T$. This is called the backward process. After the backward process is finished we find the pressure at the initial time, $u_0(\vec{x})$. This method does not work for dissipative problems, such as heat dissipation. It relies on the time reversibility property of the wave problem. When substituting τ, the second derivative in time is multiplied by a minus twice, so the coefficient of the time derivative stays the same. For equations with an odd time derivative, the term changes sign, and the problem changes.

If we had access to the pressure in the domain at every node on the grid, at times T and $T - \Delta t$, we could reconstruct $u_0(\vec{x})$ perfectly. This is based on the time reversibility property of the wave equation. However, in our study, we have access only to the sensor recorded data. Since the number of sensors is much smaller than the number of nodes on the

grid, perfect reconstruction of the initial condition is not possible. Fewer sensors imply a worse reconstruction of the initial condition. Using the TR process we still manage to get good, yet partial reconstruction of the initial condition.

The partial reconstruction of the initial condition still holds important insights. First, if one is interested in the source location (coordinates only), the reconstruction often shows a distinct peak in that location. This can be leveraged to find the source. The information from each of these images can help identify the source. In addition, the TR method has been empirically proven robust to noise, an attribute which we want to leverage as well.

3. Deep-learning framework

We introduce a new design composed of four blocks:

1. A pre-processing neural-network that estimates the noise level of the sensors recordings.
2. TR based backward process.
3. Convolutional layers that sharpen the TR results.
4. Physically-informed layers, generating a loss term used only during training.

The input to the first block are the sensors recordings. Based on the noise level, the sensor recordings are sent to a refocusing network that has been trained on measurements with a similar noise level. The TR process recreates the source image from the noisy measurements and the convolutional layers sharpen this source image. Finally, we discuss the physically-informed loss term that utilizes the wave equation and helps the convolutional part of the network train better. We now discuss how each block functions and the contribution to the overall method.

3.1. Noise inference network

Neural networks are trained to receive specific input data, and infer corresponding output data. When there is large variation between the inputs, the networks tend to struggle with producing the outputs. In this case, the variance comes from the measurement noise. When training the source inference network (described in section 3.3) using a training set composed of representative samples with different noise distributions, the network failed to converge. The proposed solution is to train several source refocusing networks, so that each network is trained with input and output data created using the same noise distribution. An illustration of the inference process of multiple trained source refocusing networks is given in Fig. 2c.

We focus on normally distributed multiplicative noise $\mathcal{N}(0,\sigma^2)$. We calculate different noise levels for different values of σ. We create l sets of samples such that all the measurements in a set are perturbed with normally distributed multiplicative noise with σ_l. In this case all the samples have similar noise level and a source inference network can be trained for this set of samples (as shown in section 4).

To execute this, we first create a noise inference network. The purpose of this network is to infer the noise level of the recorded pressures of each sample and act as a gate, deciding where to send each sample. This network receives noisy sensors recordings and infers the noise variance σ (single scalar output). After inferring σ we find the closest σ_l and the sample is sent to a source inference network trained on a set of σ_l noise level samples. This network has five convolutional layers (with 32 one-dimensional filters of length 7), followed by four fully-connected layers of sizes 1000, 500, 100 and 1 respectively. The architecture was designed by trial and error until high accuracy in inferring the noise level was achieved. This idea can be extended to support other noise distributions as well. For example, in section 4.3 we test the method with uniformly distributed additive noise.

3.2. TR backward process

The TR block is composed of a set of FDCD layers (described below). Each FDCD layer computes a single time-step. By cascading multiple FDCD layers, the computation of the backward process is done inside the network. This brings many possibilities for adding weights to the network. For example, adding weights after each step to train the network to refine the analytic backward process with respect to the output.

FDCD layer

The goal is to create a custom layer in Keras [10] where the input is the solution on the domain at two consecutive time-steps and the output is the image at the next time step. The input of the backward step is the sensor recordings. Therefore, we first create an initialization layer that receives the sensors recordings as input and builds two images of the domain, one for time step T and one for time step $T-1$, initialized with zeros. On the sensors nodes, we replace the zeros with the recorded senor data at these two time steps. Then, we use FDCD to produce an image of the domain at time step $T-2$, and replace the computed values at the sensor nodes with the recorded ones (ignoring the computed values). We do this iteratively until reaching time step 0.

In more details, we approximate the solution of (1) using finite-differences. Using a simplified notation:

$$u^n_{i,j} \approx u(x_{min} + i\Delta x, y_{min} + j\Delta y, n\Delta t), \quad i = 0, ..., N_x, \quad j = 0, ..., N_y, \quad n = 0, ..., N_t,$$

where x_{min} and y_{min} are the lower bounds of the x, y axes, respectively. Assuming a constant velocity on the entire grid, we want to approximate $u_{tt}=c^2(u_{xx}+u_{yy})$. The central-difference approximation is given by:

$$\frac{u_{i,j}^{n+1}-2u_{i,j}^{n}+u_{i,j}^{n-1}}{\Delta t^2}=c^2\left(\frac{u_{i+1,j}^{n}-2u_{i,j}^{n}+u_{i-1,j}^{n}}{\Delta x^2}+\frac{u_{i,j+1}^{n}-2u_{i,j}^{n}+u_{i,j-1}^{n}}{\Delta y^2}\right). \tag{4}$$

For the forward process we solve for u^{n+1} while for the backward process we solve for u^{n-1}.

For the backward process, the FDCD layer receives as input the images of the next u^{n+1} and current u^n time steps and all the sensor recordings. The layer computes u^{n-1} time step using (4). We substitute the approximate values only at the sensors nodes with the recorded values available to us at the corresponding time step (another common method to assimilate the recorded values is to average them with the approximate values). The output of this layer is the current u^n and the u^{n-1} time steps. We cascade n FDCD layers so that the output of the last one is u_0.

This implementation uses Keras and is part of the network itself. This gives the flexibility to later on add trainable weights to the FDCD layer and create a trainable backward process. In addition, a non-linear activation can be attached between the time steps, resulting in a non-linear backward process computation. These are useful in the presence of measurement noise where the problem is highly challenging. We experimented with applying these but decided to introduce them in a future publication.

3.3. Convolutional sharpening

Observing the output of the last FDCD layer (intermediate hidden layer) we see a distinct reconstruction of the initial condition, in the case without measurement noise. It is present in the middle image of a) and b) in Fig. 8. For high measurement noise the TR method struggles as presented in images (c-f) in Fig. 8. Therefore, we need to enhance the image to do the refocusing.

After the TR backward layers we apply convolution layers to extract a clear image of the support of the source from the backward output. The output of the backward step of the TR process is an approximation of the initial condition u^0, which is an image of size $N_x \times N_y$, and it is used as the input to the convolutional sharpening block. We apply five convolution layers with a varying number of filters to this reconstructed initial condition. A schematic of the network architecture (with both the TR and convolutional sharpening blocks) is given in Fig. 2a.

Since the output of the TR block is an image of size $N_x \times N_y$ we chose convolution layers. Convolutions are suitable for images because they are able to capture local connections in pixels close to each other. They are commonly used for imaging applications that involve DL. The convolution kernels we use are of size 5×5. The intuition for this kernel size is the desired compactly supported binary sources mentioned in section 2.3. The 5×5 kernels capture the edges of the compactly supported binary sources. The number of channels (number of convolution filters) chosen are $16, 32, 32$ and 16, and they were chosen by trial and error.

The convolution layers, in this case, are used to sharpen the initial condition image reconstruction, and extract the binary support of the source. After the network is trained and the weights of the convolution layers are fitted for the problem, the output of the network is a clear image of the same size, showing the location of the source. The convolution layers are followed by the Rectified Linear Unit (ReLU) which is a non-linear activation function ($x \to max(0, x)$). We exploit a strength of the neural network - the ability to fit non-linear problems. We tested the method also using adaptive activation functions [29,30], which have been useful for PDE related DL models. In this case they were not effective. A plausible explanation is that the proposed method approximates the medium properties (binary images) and not the solution of the PDE itself.

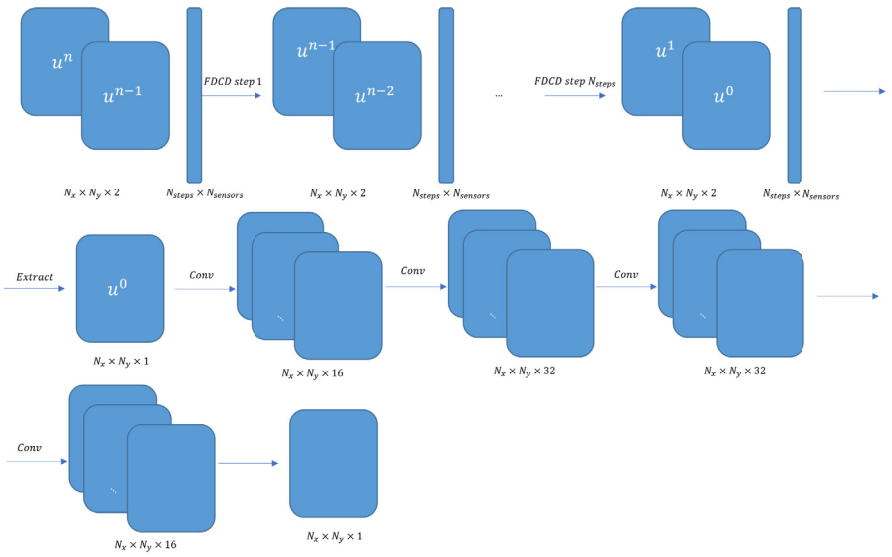
In the presence of high measurement noise, the outputs of the TR process become harder to interpret. Solving source refocusing problems, traditional methods commonly define a score function for the source compact region and locate the source based on higher scores [13,15,17]. These may fail because with high measurement noise the behavior of the refocusing changes and a unified scoring function that supports all cases may not exist. However, using the convolutional sharpening block, the source inference network does learn these extreme cases as presented in section 4. Moreover, with varying levels of noise the task becomes even more challenging. When training multiple source inference networks to infer sources from measurements that have a similar noise level, the convolution layers are trained uniquely for measurements with that noise level, resulting in a more accurate network per noise level.

The last layer of the network is followed by a sigmoid activation ($x \to \frac{1}{1+e^{-x}}$). The sigmoid function serves as a good differentiable approximation to the Heaviside step function. Each pixel in the output of the network is a probability so after using the sigmoid activation these probabilities are centered around 0 and 1, suitable to the binary output we want.

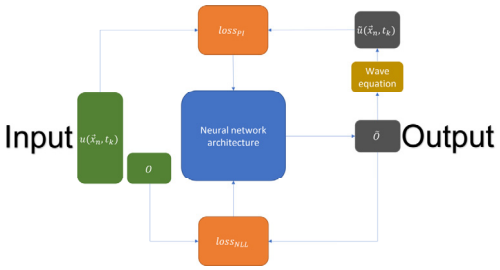
The loss function we used to train the network is the Negative Log-Likelihood (NLL) defined by:

$$NLL=-\frac{1}{N_{samples}\cdot N_x \cdot N_y}\sum_{q=1}^{N_{samples}}\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}(\mathcal{O}_q)_{i,j}log((\tilde{\mathcal{O}}_q)_{i,j}), \tag{5}$$

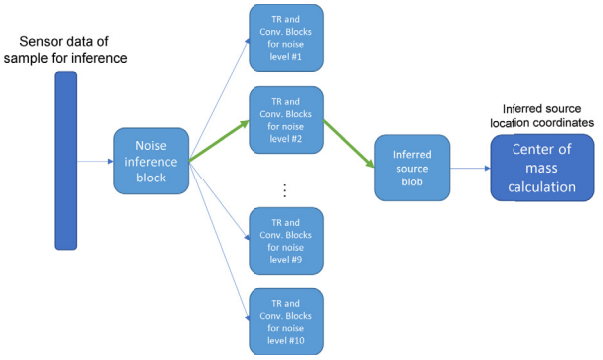
for the set of true labels $\{\mathcal{O}_q\}_{q=1}^{N_{samples}}$ and predicted labels $\{\tilde{\mathcal{O}}_q\}_{q=1}^{N_{samples}}$.



(a) Schematic of the proposed TR and convolutional sharpening blocks network architecture



(b) Illustration of the usage of both NLL loss and PI loss during the training phase



(c) Illustrating of inference using the trained models

Fig. 2. Overall illustration of the proposed method. Expanding the neural network block of b) is a), and each TR and Conv. Block in c) is b) trained for a different noise level.

3.4. Physically-informed loss term

We use the physical problem to create a unique loss term based on it, in addition to the loss term defined in (5). This makes the network "aware" of the physical problem and utilizes this knowledge to converge faster and produce more accurate results. We propose a new approach for including the numerical scheme used to approximate the forward solution of (1) over time, inside the network and during the training phase.

To create the physically-informed (PI) loss term, the sharpened estimation of the initial condition is used to forward propagate an approximate solution. Recall that the input to the source inference network is the data in the sensors and the labels. During training the predicted labels are compared to the true labels using the NLL loss as explained in section 3.3. We propose to use the predicted labels, in addition to the NLL loss computation, as inputs to the forward wave propagation process. We use FDCD directly with the estimated source image (the predicted reconstruction of the initial condition) to forward propagate the waves for N_{steps} and compute the sensors recordings as explained in section 2.2. We have the true sensor recordings $\{u_q(t_n, x_k, y_x)\}_{n=1,k=1}^{N_{steps},N_{sensors}}$ for each sample $q = 1, ..., N_{samples}$ and the ones calculated from the forward process with the reconstructed image as $\{\tilde{u}_q(t_n, x_k, y_x)\}_{n=1,k=1}^{N_{samples},N_{sensors}}$. We compare these values via the Mean Squared Error (MSE) given by

$$PI = \frac{1}{N_{samples} \cdot N_{steps} \cdot N_{sensors}} \sqrt{\sum_{q=1}^{N_{samples}} \sum_{n=1}^{N_{steps}} \sum_{k=1}^{N_{sensors}} |u_q(t_n, x_k, y_x) - \tilde{u}_q(t_n, x_k, y_x)|^2}. \tag{6}$$

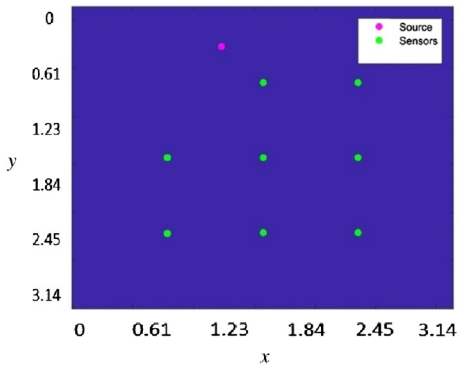


Fig. 3. Sensors placement used in the experiments. In this example we have a random source at $x = 1.14$, $y = 0.42$.

Since the forward process is a differentiable procedure, the automatic differentiation mechanism in Keras [10,8] recognizes it and the loss can be used in the training process. The PI loss acts as a penalty term when training the network. Therefore, it does not replace the NLL loss. The loss we use in training is a combination of these two losses, given by $loss = \alpha NLL + (1-\alpha)PI$. To choose α we run the training process for a few steps and print the values of NLL and PI. In this case, the losses were of the same order of magnitude and thus we choose $\alpha = 0.5$. We illustrate the usage of the PI loss during the training step in Fig. 2b.

We emphasize that the data used for the forward propagation inside the PI loss computation is the binary supports (ground truth labels, see equation (3)) and probability images inferred by the convolutional sharpening block (predictions). They are different from the actual initial condition used to create the sensors data (ground truth inputs), which is the Gaussian sources. The convolutional sharpening block predicts the binary supports (blobs) from the TR output image. Therefore, applying the FDCD to the predicted image and comparing to the network inputs is an incorrect comparison, since the former uses a blob initial condition, and the latter a Gaussian. The correct comparison is done by applying FDCD once on the prediction (predicted blob), and once on the true label (true blob). It makes sense to compare them since the blob acts as a source, able to generate propagating waves. In addition, if the network is able to make accurate predictions, the probability images should be close to the ground truth images, and the sensor recordings in both cases are comparable. In this case, we have a two dimensional step function and thus a non-smooth initial condition. When creating the sources in such a way, we expect numerical artifacts to appear when using the explicit finite-differences scheme to simulate the forward process. However, these artifacts can be neglected since the learning algorithm finds the patterns in the data regardless of these artifacts, as shown in section 4. In addition, we compute the forward process for both the true binary source and predicted binary source as desired.

4. Numerical tests and results

4.1. Setup

We created 5,000 samples as described in section 2. The grid size we used is 128×128. The size of the medium is $[0, \pi] \times [0, \pi]$. We used 8 fixed sensors as shown in Fig. 3. The generated source locations are completely arbitrary. Identical samples (choosing the same source location more than once) were generated and included in the data-set. We emphasize that the small number of samples was chosen deliberately based on the limited available computation resources. In addition, in the noise-less case the number of nodes on the grid is $128^2 = 16,384$, so using only 5,000 samples does not cover the number of possible samples. With noise, each sample is perturbed randomly so we have an infinite number of possible samples and we use only 5,000. For these settings, the total storage size of the dataset is 1.5 GB.

The parameters we used are $c = 1484 \frac{m}{s}$ (average acoustic wave propagation speed in the Mediterranean sea), $\Delta x = \Delta y = \frac{\pi}{127} \approx 0.025$ and $\Delta t = \frac{1}{c\sqrt{2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)}} \approx 8.33 \cdot 10^{-6}$. We chose the number of time steps as 500 such that, for these parameters, the wave-front impacts all the sensors (for every source location). A large number of time steps causes higher memory consumption and slower training, and 500 is a good balance between enough recorded data in the sensors for the model to train on and a small data-set consuming less computational resources.

The initial condition used for the forward wave propagation when creating the data is a small compactly supported Gaussian eruption. The boundary condition we use is a homogeneous Dirichlet condition on both the vertical and horizontal boundaries, mimicking an acoustic wave propagation physical experiment conducted inside a pool. Thus, reflections from the boundaries return to the sensors and more data about the wave propagation is recorded, compared to using absorbing boundaries (mimicking an open boundary, waves propagate in an ocean for example) where there are no reflections. We experimented with the absorbing boundary conditions as well and the results of that specific experiment are given later in this section.

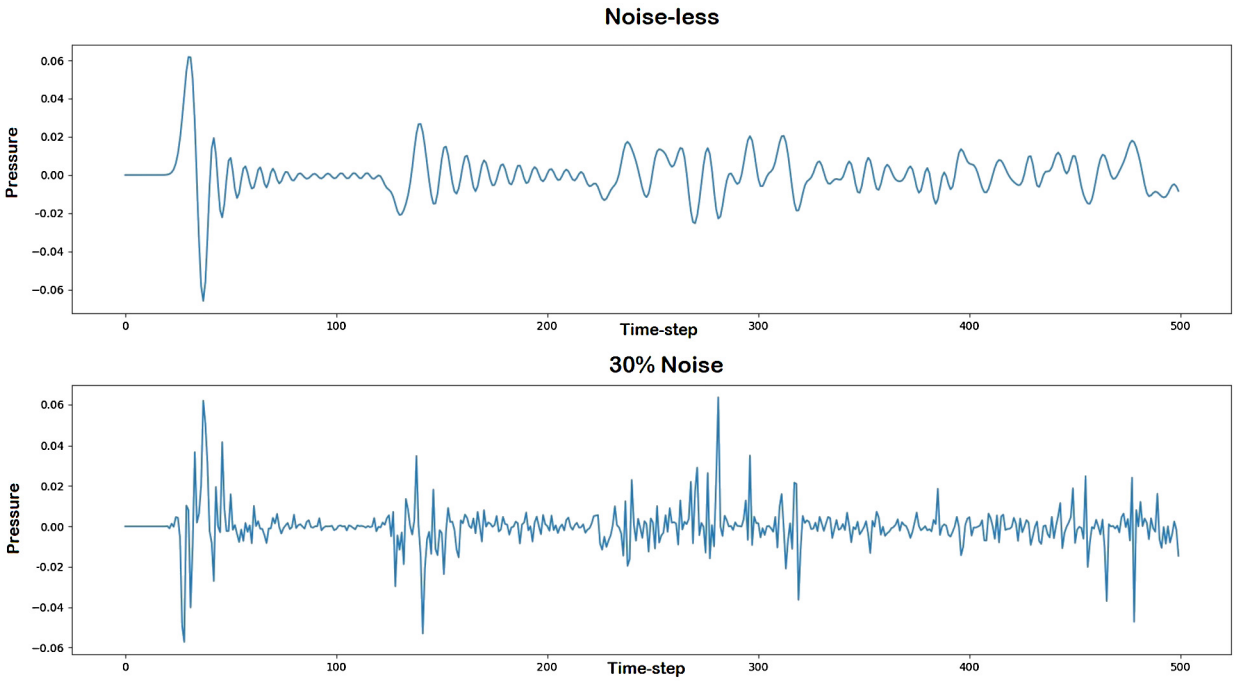


Fig. 4. Comparison between a noise-less sensor recording and the same sensor recording with $\sigma = 0.3$ noise applied to it in the same scenario.

We add synthetic noise to the sensors data using normally distributed multiplicative noise. Each recording is multiplied as follows: $\{u(t_n, x_k, y_k) \cdot \mathcal{N}(0, \sigma^2)\}_{n=1,k=1}^{N_{steps}, N_{sensors}}$. The parameter σ is the variance of the noise. In our experiments we used the sequence $\sigma = 0, 0.1, 0.2, ..., 1$, so we have 11 different noise level distributions. Fig. 4 presents the impact of 30% noise on a sensor recording. We observe from this figure that already at 30% noise the clean signal is barely noticeable. We train 11 networks such that each network trains on a specific noise strength up to 100% noise.

We split the generated data into three groups: training set, validation set and testing set. The 5,000 samples in the training data are split so that 4,500 (90%) are the training set and the remaining 500 (10%) are the validation set, and we create additional 1,000 samples to compose the testing set. The training set is used to train the network (both input and output are injected during training). The validation set is used to evaluate the performance during training. While training, after each epoch we calculate the loss using the validation set, in addition to the loss computed using the training set that is used for learning the weights of the network. By examining the validation loss we conclude if the network is converging (train and validation losses are decreasing), saturating (validation loss stopped decreasing) or if the network is over-fitting the training data (increasing validation loss and decreasing training loss). We save the network weights after the network converges and before it starts saturating or over-fitting. The testing set is used to evaluate the network performance on a data-set it has not seen during training and ensure that the network is able to generalize (infer the output for samples not included in the training set).

We used 400 epochs and a batch size of 64 for training. We trained the networks on a Tesla T4 GPU for 3 days (about 8 hours per model of the 11 mentioned models). After the model finished training, inference takes less than a second per sample (applicable for real-time purposes).

We first present the contribution of the PI loss to the training process. In Fig. 5 we show a comparison between the computed NLL losses over the validation set during training, once with applying the PI loss term and once without it. As a regulative term, we observe that the PI loss improves the convergence rate. The loss comparison presented here is for a noisy data training. Without noise, the validation loss drops from 10^{-2} to 10^{-6}. Fig. 8 presents visual aspects of the method. We show two different examples of source locations. For each example, we show the cases of 0%, 50% and 100% measurement noise. The leftmost images show the true image of the source, the middle ones show the output of the TR block and the rightmost show the output of the network. We observe that with high noise the TR images do not show a distinct image of the source. We are interested only in the location of the source, so we compute the center of mass of the rightmost images as explained in section 4.4.

4.2. Naive network

We first test a naive approach for the same problem. The naive approach involves neither the TR process nor a physically informed loss term. We did use the noise level inference and train 11 naive networks (one per noise level). The architecture used for the naive network includes:

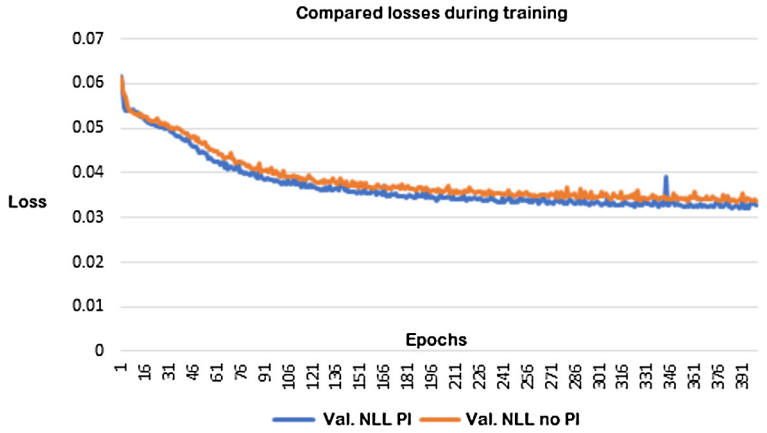


Fig. 5. Comparison between NLL loss of validation set computed during training per epoch, with and without PI loss term.

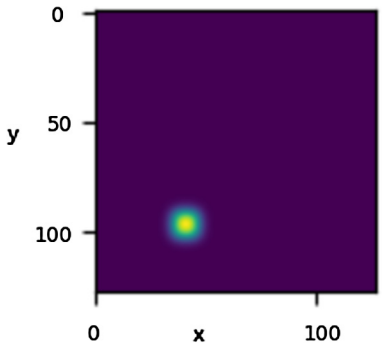


Fig. 6. Example of a thick Gaussian initial condition.

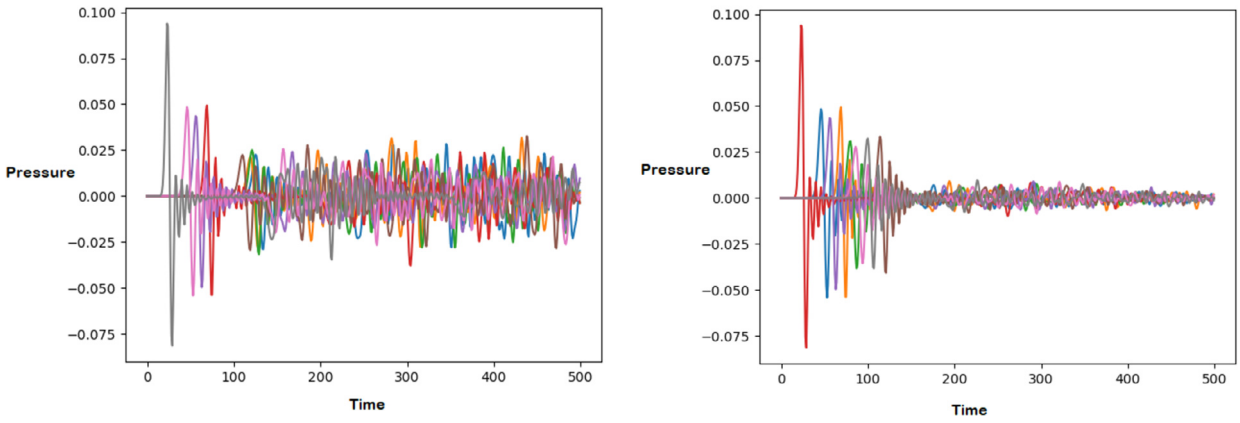
1. Column stacking the sensors data into a $N_{steps} \cdot N_{sensors} \times 1$.
2. Dense layer of size 4096×4096. The output of this layer is of size 4096×1.
3. Reshape into $64 \times 64 \times 1$.
4. Convolution layer with 4 filters of size 8×8. The output of this layer is of size $64 \times 64 \times 4$.
5. Reshape into 128×128.

We use this architecture for all 11 networks. This architecture gives an excellent source refocusing for the noise-less case but fails when adding noise. Results are given in Table 1. From 10% and upwards the naive network gave the same result as the 50% and 100% in the table. With noise, the reconstruction of the source using the naive approach was equivalent to a random guess.

4.3. Test scenarios

In addition to the test setup mentioned above we conducted tests for different scenarios. In each scenario we create a suitable data-set and train new networks according to that data. Due to the large complexity of the physically informed loss function, the scenarios were tested without it. We expect even better results for these scenarios when using the physically informed loss. The conducted tests are as follows:

- **Noise distribution:** We tested the method with noise distributions other than the normally distributed Gaussian noise to show the method is applicable in different scenarios. First, we use uniformly distributed noise: $\mathcal{U}(0, \sigma)$ where $\sigma = 0, 0.1, ..., 1$. Then, we use additive noise instead of multiplicative, by $\{u(t_n, x_k, y_k) \cdot (1 + \mathcal{U}(0, \sigma^2))\}_{n=1,k=1}^{N_{steps}, N_{sensors}}$. We present the results for the three interesting cases: a) multiplicative normally distributed noise, b) additive normally distributed noise, and c) additive uniformly distributed noise. As expected, the results with additive noise are much better than with multiplicative noise. The results are summarized in Table 1.
- **Initial condition:** We tested the method with a different initial condition. Now, we used thick Gaussians that are much different from the thin ones (examples are shown in the leftmost images of Fig. 8). An example of such thick Gaussian is shown in Fig. 6. The results in this case are similar to the previous results.



(a) Sensors data without ABC

(b) Sensors data with ABC

Fig. 7. Examples of sensors recordings with and without ABC.

- **Sensors placement:** In many applications the sensors cannot be placed as in Fig. 3. For example, in underwater acoustics one often places the sensors floating on the water surface. We tested the method where all the 8 sensors are placed on the boundary. We chose to place the sensors at the top boundary, and placed them slightly below the actual boundary to avoid surface waves influencing the results.
- **Absorbing boundaries:** We tested the network with absorbing boundary conditions (ABCs) instead of reflecting conditions. Absorbing boundaries are used to mimic open boundaries (for example, a wave propagating in the ocean) while reflective boundaries mimic hard bounded physical domain (for example, a wave propagating in a pool). We implemented Mur's [28] ABC in both the forward and backward solvers. For enforcing the standard first order Mur ABC on the left boundary, we used $\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x}$ on that boundary. Every other boundary has a corresponding formula. We used the same ABC for both the forward and backward processes, changing the sign when solving backwards since we have a first derivative of t (recall substituting $\tau = T - t$ in the TR process). There are other methods to implement ABCs. We chose this implementation since it is computationally light and can be easily implemented and used inside the network (both for the FDCD layers and the PI loss). We noticed a different behavior of the recorded data in the sensors which is due to the loss of energy. An example of sensors recording with and without ABCs is given in Fig. 7. Notice that with the ABC the recorded pressures are much lower, showing effective absorption of the energy at the edges. Interestingly the results with ABCs are better than with reflecting boundaries.

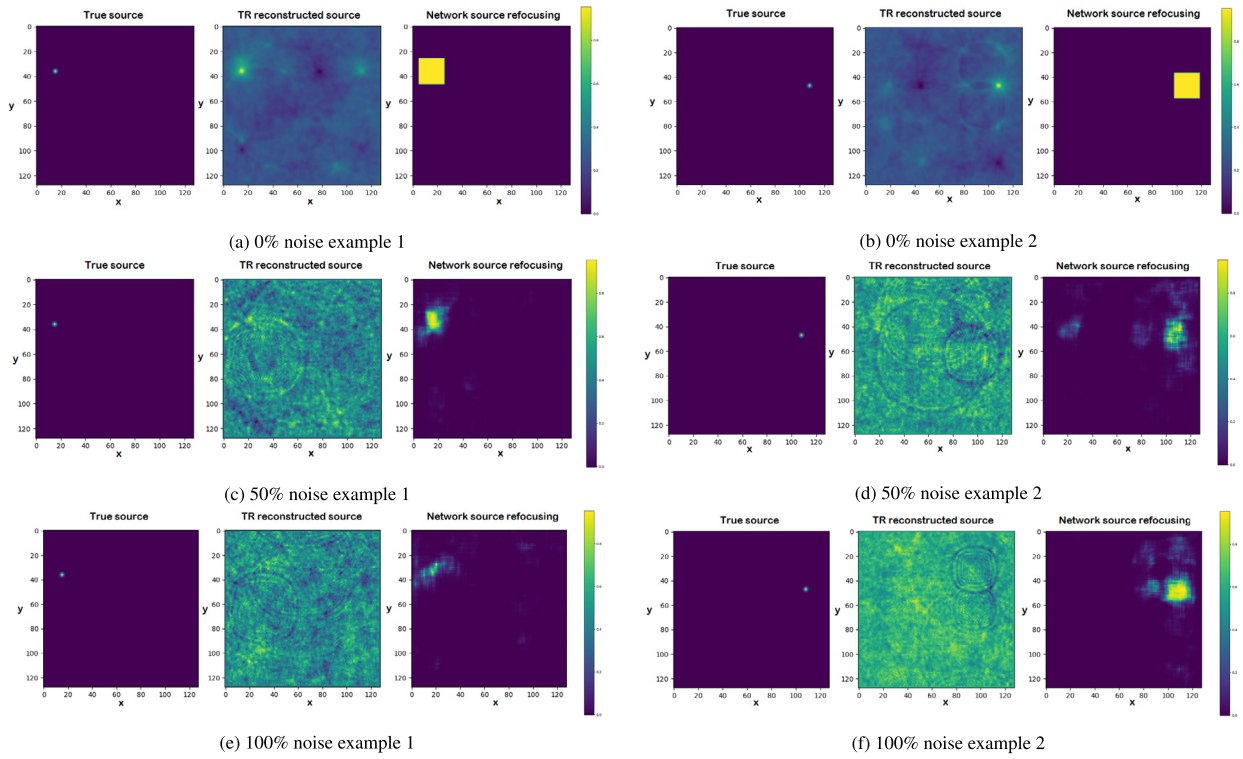
4.4. Evaluation

For evaluating the source refocusing success, we calculate the distance between the true source location and the center of mass of the predicted source. As shown in the rightmost images of (a-f) in Fig. 8, the output of the model is a probability image where the source location is the center of the segment. We calculate the total mass by $\mathcal{M}_q = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\tilde{\mathcal{O}}_q^{source}(x_i, y_j)$ per sample ($q = 1, ..., N_{samples}$). We get the coordinates of the center of mass by $\tilde{\mathcal{R}}_q = \frac{1}{\mathcal{M}_q}\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}(x_i, y_j)\tilde{\mathcal{O}}_q^{source}(x_i, y_j)$. We find the distance between the true source location $\mathcal{R}_q$ and $\tilde{\mathcal{R}}_q$ and measure the success of the method by the statistical measures (mean, median and standard deviation) of the distances calculated over the testing set. The results are given in Table 1. The physical domain chosen for this test is $[0, \pi] \times [0, \pi]$. The calculated distances are based on this domain size. We also calculate the distances in terms of pixels on the 128×128 grid as well.

We observe in Fig. 8 that the middle images (plain TR output) look arbitrary and it is difficult for a human to interpret the location of the source from them. However, the network output is sharper, showing the exact area where the source is. Computing the center of mass of the network source refocusing images we retrieve the source location accurately. Table 1 shows the statistical measures of the location results gathered from all the experiments conducted in this work.

5. Conclusion

We presented an approach for source refocusing in an underwater medium. We used a finite difference scheme for approximating the acoustic wave equation and formulated the problem of interest as a data-driven problem. We proposed a novel method to deal with high measurement noise. We created a method based on four blocks. The first is a prior network to infer the noise distribution of the sensors. Then, we incorporated the TR method into the deep-learning model. We then used convolution layers and trained several networks to retrieve the sources according to the noise distribution of the input signal. Last, we proposed a PI loss term as a penalty to the network during training. This term is based on the



(a) 0% noise example 1

(b) 0% noise example 2

(c) 50% noise example 1

(d) 50% noise example 2

(e) 100% noise example 1

(f) 100% noise example 2

Fig. 8. Visual examples of outputs of the source refocusing networks for different noise levels. Left is the true source image, middle is the output of plain TR and right is the network inference output.

Table 1
Statistical aggregation of the source refocusing results. The distances in pixels are given in brackets.

Network type	Noise level	Mean distance	Median distance	Distance standard deviation
Naive network	0%	0.1378 (5.569)	0.0247 (1.0)	0.4631 (18.72)
	50%	1.0964 (44.322)	1.1263 (45.53)	0.4016 (16.235)
	100%	1.0964 (44.322)	1.1263 (45.53)	0.4016 (16.235)
Proposed network	0%	0.0162 (0.6534)	0.0247 (1)	0.0146 (0.5917)
	50%	0.2046 (8.2722)	0.1749 (7.0711)	0.1407 (5.689)
	100%	0.2107 (8.5172)	0.1749 (7.0711)	0.1538 (6.2194)
Test: Uniformly distributed additive noise	0%	0.0085 (0.3451)	0.0 (0.0)	0.0129 (0.5224)
	50%	0.0104 (0.4199)	0.0 (0.0)	0.0135 (0.5438)
	100%	0.0178 (0.7185)	0.0247 (1.0)	0.0133 (0.5365)
Test: Uniformly distributed multiplicative noise	0%	0.0117 (0.4716)	0.0 (0.0)	0.0143 (0.5775)
	50%	0.0095 (0.3857)	0.0 (0.0)	0.0136 (0.5479)
	100%	0.0233 (0.9424)	0.0247 (1.0)	0.012 (0.4857)
Test: Thick Gaussians (initial conditions)	0%	0.0505 (2.0407)	0.0247 (1.0)	0.1358 (5.4878)
	50%	0.2587 (10.4577)	0.2281 (9.2195)	0.1639 (6.6257)
	100%	0.2402 (9.7114)	0.204 (8.2462)	0.154 (6.2252)
Test: Sensors placed on top	0%	0.0104 (0.4208)	0.0 (0.0)	0.0136 (0.5504)
	50%	0.2179 (8.8085)	0.1584 (6.4031)	0.1865 (7.5412)
	100%	0.2234 (9.0292)	0.1732 (7.0)	0.1849 (7.4734)
Test: ABC	0%	0.0147 (0.5951)	0.0247 (1.0)	0.0142 (0.5734)
	50%	0.1421 (5.7435)	0.1106 (4.4721)	0.1129 (4.5621)
	100%	0.1373 (5.5519)	0.1049 (4.2426)	0.1131 (4.5702)

acoustic wave equation. We showed that this term enhances the training process (better convergence). While training may be complex, inference is very quick and takes less than a second. We showed that using a naive DL approach does not work with high noise, and TR alone struggles with high multiplicative noise (see Fig. 8). The proposed method combining TR and deep-learning yields accurate results even when the reference methods fail. We tested the method under various conditions and the results are satisfactory, showing that the method can be used for many applications. We intend as future work to extend the method for finding the location, shape and size of unknown scatterers inside the domain. Preliminary results

show that using the method we can retrieve some information about the scatterer, mainly the location, but to get the shape and size with high noise is challenging. In addition, another interesting direction is recovering the shape of the source, and not only the location as presented in this work.

CRediT authorship contribution statement

Adar Kahana: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Eli Turkel:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Shai Dekel:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Dan Givoli:** Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] U. Albocher, A.A. Oberai, P.E. Barbone, I. Harari, Adjoint-weighted equation for inverse problems of incompressible plane-stress elasticity, Comput. Methods Appl. Mech. Eng. 198 (30–32) (2009) 2412–2420.
[2] R.V. Allen, Automatic earthquake recognition and timing from single traces, Bull. Seismol. Soc. Am. 68 (5) (1978) 1521–1532.
[3] M. Baer, U. Kradolfer, An automatic phase picker for local and teleseismic events, Bull. Seismol. Soc. Am. 77 (4) (1987) 1437–1445.
[4] E. Amitt, D. Givoli, E. Turkel, Combined arrival-time imaging and time reversal for scatterer identification, Comput. Methods Appl. Mech. Eng. 313 (2017) 279–302.
[5] F. Assous, M. Kray, F. Nataf, E. Turkel, Time reversed absorbing condition: application to shape reconstruction, Inverse Probl. 27 (6) (June 2011).
[6] P.E. Barbone, A.A. Oberai, I. Harari, Adjoint-weighted variational formulation for direct solution of inverse heat conduction problem, Inverse Probl. 23 (2007) 2325–2342.
[7] C. Bardos, M. Fink, Mathematical foundations of the time reversal mirror, Asymptot. Anal. 29 (2002) 157–182.
[8] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.
[9] P. Blomgren, G. Papanicolaou, H. Zhao, Super-resolution in time-reversal acoustics, J. Acoust. Soc. Am. 111 (2002) 230–248.
[10] F. Chollet, Keras, https://keras.io/.
[11] D. Colton, R. Kress, Inverse Acoustic and Electromagnetic Scattering Theory, 3rd edn., Applied Mathematical Sciences, vol. 93, Springer, New York, 2013, pp. xiv+405.
[12] M. Fink, F. Wu, D. Cassereau, R. Mallart, Imaging through inhomogeneous media using time reversal mirrors, Ultrason. Imag. 13 (1991) 179–199.
[13] D. Givoli, E. Turkel, Time reversal with partial information for wave refocusing and scatterer identification, Comput. Methods Appl. Mech. Eng. 213 (2012) 223–242.
[14] V. Isakov, Inverse Problems for Partial Differential Equations, 2nd edn., Applied Mathematical Sciences, vol. 127, Springer, 2006.
[15] A. Kahana, E. Turkel, D. Givoli, Convective wave equation and time reversal process for source refocusing, J. Comput. Acoust. 26 (02) (2018) 1850016.
[16] L. Kuperkoch, T. Meier, J. Lee, W. Friederich, Automated determination of P-phase arrival times at regional and local distances using higher order statistics, Geophys. J. Int. 181 (2) (2010) 1159–1170.
[17] I. Levi, E. Turkel, D. Givoli, Time reversal for elastic wave refocusing and scatterer location recovery, J. Comput. Acoust. 23 (2015) 1450013, 1–29.
[18] T. Levin, E. Turkel, D. Givoli, Obstacle identification using the TRAC algorithm, Int. J. Numer. Methods Eng. 118 (2) (2019) 61–92.
[19] H. Niu, E. Reeves, Peter Gerstoft, Source localization in an ocean waveguide using supervised machine learning, J. Acoust. Soc. Am. 142 (2017) 1176.
[20] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (686–707) (2019).
[21] C.D. Saragiotis, L.J. Hadjileontiadis, S.M. Panas, PAI-S/K: a robust automatic seismic P phase arrival identification scheme, IEEE Trans. Geosci. Remote Sens. 40 (6) (2002) 1395–1404.
[22] R. Seidl, E. Rank, Iterative time reversal based flaw identification, Comput. Math. Appl. 72 (4) (2016) 879–892.
[23] R. Sleeman, T. Van Eck, Robust automatic P-phase picking: an on-line implementation in the analysis of broadband seismogram recordings, Phys. Earth Planet. Inter. 113 (1999) 265–275.
[24] A. Tarantola, Inverse Problem Theory: Methods for Data Fitting and Model Parameter Estimation, Elsevier, Amsterdam and New York, 1987, pp. xvi+613.
[25] C.R. Vogel, Computational Methods for Inverse Problems, Frontiers in Applied Mathematics, vol. 23, SIAM, Philadelphia, PA, 2002, pp. xvi+183.
[26] J. Wang, Z. Xiao, C. Liu, D. Zhao, Z. Yao, Deep learning for picking seismic arrival times, J. Geophys. Res., Solid Earth 124 (7) (2019) 6612–6624.
[27] W. Zhu, G.C. Beroza, PhaseNet: a deep-neural-network-based seismic arrival-time picking method, Geophys. J. Int. 216 (1) (2019) 261–273, Oxford University Press.
[28] G. Mur, Absorbing boundary conditions for difference approximations to the multi-dimensional wave equation, Math. Comput. 47 (176) (1986) 437–459.
[29] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.
[30] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proc. R. Soc. A 476 (2239) (2020) 20200334.
[31] D. Givoli, A tutorial on the adjoint method for inverse problems, Comput. Methods Appl. Mech. Eng. 380 (2021) 113810.