Neural Network Reconstruction of the Electron Density of High Energy Density Plasmas From Under-Resolved Interferograms

P.-A. Gourdain and A. Bachmann

***Abstract*— Interferometry can accurately measure the electron density of a high energy density plasma by comparing the phase shift between a laser beam passing through the plasma and a reference beam. While the actual phase shift is continuous, the measured shift has discontinuities, since its measurement is constrained between $-\pi$ and π, an effect called "wrapping." Although many methods have been developed to recover the original, "unwrapped" phase shift, noise and under-sampling often hinder their effectiveness, requiring advanced algorithms to handle imperfect data. Analyzing an interferogram is essentially a pattern recognition task, where radial basis function neural networks (RBFNNs) excel. This work proposes a network architecture designed to unwrap the phase interferograms, even in the presence of significant aliasing and noise. Key aspects of this approach include a three-stage learning process that sequentially eliminates phase discontinuities, the ability to learn directly from the data without requiring a large training set, the ability to mask regions with missing or corrupted data trivially, and a parallel Levenberg–Marquardt algorithm (LMA) that uses local network clustering and global synchronization to accelerate computations.**

***Index Terms*— Artificial intelligence, interferometry, plasma measurements.**

I. Introduction

USUALLY, interferometry (see [1], [2], [3]) is used to measure the properties of a system (e.g., elevation and electron density) that would be impractical to measure otherwise [4]. However, the phase shift ϕ_{GT}, or "ground truth," between the two paths cannot be measured directly. Rather, only the wrapped phase shift [5] ϕ_W, bounded between $-\pi$ and π, can be computed from the interferogram. Note that the rest of this article, we will use phase instead of phase shift. As the wrapped phase ϕ_W cannot be used directly, the data need to be unwrapped, a task deceptively challenging, especially in the presence of noise. To further complicate matters, large chunks of phase data might be missing due to the presence of strong signal cutoff (often seen in magnetic resonance imaging [6]) or under-sampling, as in the dense plasma interferometry [7]. Furthermore, datasets have grown extremely large, straining serial algorithms used in phase unwrapping (e.g., functional MRI [8], interferometric synthetic aperture radar [9], shape reconstruction [10], or fringe projection profilometry [11]). Regardless of the problem, the phase unwrapping procedure must find an approximate phase ϕ, that is, such that $\phi = \phi_{GT} + o(\phi_{GT})$. The ground truth must be extracted from the intensity I given by

$$I(x, y) = A(x, y) + B(x, y)\cos^2\phi_{GT}(x, y). \tag{1}$$

Note that in the ideal case, where $A \equiv 0$ and $B \equiv 1$, we effectively measure the wrapped phase $\phi_W = W(\phi_{GT})$, where W is the wrapping operator defined as follows:

$$W(\phi) = \phi - 2k\pi \text{ with } k \in \mathbb{Z} \text{ such that } W(\phi) \in]-\pi, \pi]. \tag{2}$$

In this work, we turned the intensity given by (1) into a computed phase ϕ_W, which is bounded between $-\pi$ and π, from the filtered Fourier transform used on the interferogram [12], [13], [14]. The bounds $\pm\pi$ are defined by the Fourier transform operation, which is not capable of recovering the "unwrapped" phase.

When the phase ϕ_{GT} is well-behaved (i.e., continuous, noise free, and over-sampled) then phase unwrapping is straightforward [15]. However, when the phase is corrupted by noise or under-sampled (i.e., aliasing), unwrapping becomes difficult [16] and a variety of methods have been developed to overcome this issue. Early methods used branch-cuts [17], [18], [19], least-square algorithm [20], [21], or polynomial phase approximation [22], [23]. However, they also did not react well to noise, leading to the development of algorithms capable of handling high noise levels [24], [25], [26], [27], [28], [29] using Kalman filters [30], [31]. Machine learning algorithms were equally successful, using artificial neural networks [32], then deep learning [33], [34], [35], [36], [37] and finally convolutional networks [38], [39]. Unlike previous techniques, which tend to use the grid given by the natural data layout, machine learning is usually not relying on the physical data structure to perform the unwrapping. Compared with other phase-based measurements, high-energy density plasma interferometry [40], [41], [42] has its set of unique challenges. Typically, external noise levels are relatively low since most

Received 18 September 2023; revised 24 February 2024, 7 September 2024, and 23 November 2024; accepted 10 December 2024. Date of publication 30 December 2024; date of current version 28 January 2025. This work was supported by the NSF CAREER Award under Grant PHY-1943939. The review of this article was arranged by Senior Editor E. Schamiloglu. *(Corresponding author: P.-A. Gourdain.)*

P.-A. Gourdain is with the Department of Physics and Astronomy and the Laboratory for Laser Energetics, University of Rochester, Rochester, NY 14627 USA (e-mail: gourdain@pas.rochester.edu).

A. Bachmann is with the Department of Physics and Astronomy, University of Rochester, Rochester, NY 14627 USA, and also with the Department of Physics and Astronomy, University of California at Los Angeles, Los Angeles, CA 90024 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TPS.2024.3519032.

Digital Object Identifier 10.1109/TPS.2024.3519032

interferometers use a laser beam [43] that is extremely bright (>100 MW/cm^2). However, diffraction can generate artifacts that degrade beam quality. Furthermore, these plasmas have energy densities on the order of 1 kJ/cm^3, and the continuum light they produce as a result can cause large-scale intensity blotches embedded inside the interferogram. High energy density plasmas can also be surrounded by complex structures, which blocks part of the beam and create regions free of interference fringes. The shape of these structures is often complex [44], [45] and must be removed from the input data. Finally, with electron density gradients relatively large, the interferometry data are often under-sampled, creating zones where the interference pattern is not directly usable [46].

To deal with such practical considerations, we developed a parallel neural network architecture capable of unwrapping phase data following staged supervised learning. Parallelization is obtained by clustering neurons across contiguous regions, where overlapping neurons, called *ghost* neurons [47], are used to synchronize different networks. Unlike previous methods using Levenberg–Marquardt algorithms (LMAs) [48] or spatial derivatives [49], the proposed radial basis function neural network (RBFNN) architecture uses the well-documented neural networks [50]. However, the optimization procedure has been modified to unwrap the phase obtained from an imperfect interferogram, where noise and under-sampling are caused by inadequate digitization or data compression. Since radial basis functions are good interpolants for smooth functions [51], they guarantee that the RBFNN can interpolate [52] a phase that is smooth enough using one hidden layer. While we have not investigated multilayer perceptrons [53], [54], they are known to be excellent approximators [55] and are not that different in practice [56], [57]. The main reason we chose radial basis functions in this work hinges on keeping the Jacobian matrix, used in gradient-based training, as sparse as possible. While multilayer perceptrons use sigmoid-like functions, which are not compact, we can use compact radial basis functions [58], leading to RBFNN with sparse Jacobian matrices.

While most methods expect the phase data to be noisy, dealing with corrupted or missing phase data is not a trivial task. Removing the corrupted phase from the learning process is especially challenging when the region to exclude is geometrically complex. Furthermore, the wrapped phase data can have different sizes. Therefore, the method proposed here does not use a training dataset containing wrapped and unwrapped data as it is the case in machine learning (see [37]). Rather, we train the RBFNN only on the wrapped phase data that the RBFNN is trying to unwrap. Therefore, every new unwrapping is also a new training operation. However, based on the consideration just listed above, this approach gives the most flexibility when successive acquisitions yield phase data that is fundamentally different from the previous ones both in size and shape, as it is often the case in plasma electron density measurements.

II. Training of the Staged Neural Network Architecture

A. Preliminary Remarks

1) Condition to Detect Aliasing: Super-resolution imaging uses numerical or physical techniques that allow to effectively increase the resolution of an image [59], [60]. This technique is required when the phase was wrapped more than once across two pixels in interferometry data. This phenomenon is known as aliasing [16], [61]. It can present itself as a series of swift, consecutive jumps, a case relatively easy to detect when noise levels are low. It can also be completely inconspicuous. For instance, a phase which varies as $\phi_{\mathrm{GT}}(n) = \xi - \pi$ and $\phi_{\mathrm{GT}}(n+1) = \xi + \pi$, where $0 < \xi \ll 1$, would yield a seemingly constant wrapped phase $\phi_W(n) = \xi - \pi$ and $\phi_W(n+1) = \xi - \pi$. As a lower bound, we can see that aliasing is present when $|\partial\phi_{\mathrm{GT}}| > 2\pi$, then $W(\partial\phi_{\mathrm{GT}})$ has jumps inside $]-\pi, \pi]$. While this is only a necessary condition, it becomes sufficient when ϕ_{GT} is continuous, as we will see later, allowing us to detect the presence of aliasing inside the data.

2) Key Property of Periodic Functions: When any phase Φ is discretized, we can define its left derivative as $\partial_L\Phi(n) = \Phi(n) - \Phi(n-1)$. Using (2), we get $\mathcal{F}(\partial_L\phi_{\mathrm{GT}}(n)) = \mathcal{F}(\phi_W(n) - \phi_W(n-1) + 2(k_n - k_{n-1})\pi)$ for any 2π-periodic function $\mathcal{F}$. As a result, $\mathcal{F}(\partial_L\phi_{\mathrm{GT}}(n)) = \mathcal{F}(\partial_L\phi_W(n))$. This is true for the whole domain if we use the linear extrapolation to define the left derivative at the left boundary as $\partial_L\Phi(1) = -\Phi(1) + 2\Phi(2) - \Phi(3)$. For the right derivative, defined as $\partial_R\Phi(n) = \Phi(n+1) - \Phi(n)$, we use the same reasoning to get $\mathcal{F}(\partial_R\phi_{\mathrm{GT}}(n)) = \mathcal{F}(\partial_R\phi_W(n))$. This is valid across the whole domain if we now use the linear extrapolation of the right derivative at the right boundary as $\partial_R\Phi(N) = -\Phi(N-2) + 2\Phi(N-1) - \Phi(N)$. This property does not extend to the central derivative $\partial_C\Phi(n) = (1/2)[\Phi(n+1) - \Phi(n-1)]$ since $\mathcal{F}(\partial_C\phi_{\mathrm{GT}}(n)) = \mathcal{F}(\partial_C\phi_W(n) + (k_{n+1} - k_{n-1})\pi)$ is equal to $\mathcal{F}(\partial_C\phi_W(n))$ only when $k_n - k_{n-1}$ is even, but not when it is odd. In the end, we find

$$\mathcal{F}(\partial_{L,R}\phi_W) = \mathcal{F}(\partial_{L,R}\phi_{\mathrm{GT}}). \tag{3}$$

It happens that the second derivative $\partial^2\Phi = \Phi(n+1) - 2\Phi(n) + \Phi(n-1)$ is also invariant since $\mathcal{F}(\partial^2\phi_{\mathrm{GT}}(n)) = \mathcal{F}(\phi_W(n+1) - 2\phi_W(n) + \phi_W(n-1) + 2(k_{n+1} - 2k_n + k_{n-1})\pi) = \mathcal{F}(\partial^2\phi_W(n))$. If we linearly extrapolate the second derivatives to the domain boundaries as $\partial^2\phi(1) = \phi(1) - 3\phi(2) + 3\phi(3) - \phi(4)$ and $\partial^2\phi(N) = \phi(N-3) - 3\phi(N-2) + 3\phi(N-1) - \phi(N)$, then this result is still valid for the whole domain and we get

$$\mathcal{F}(\partial^2\phi_W) = \mathcal{F}(\partial^2\phi_{\mathrm{GT}}). \tag{4}$$

Applying the same reasoning using (3) and (4), we can easily show that

$$W(\partial_{L,R}\phi_W) = W(\partial_{L,R}\phi_{\mathrm{GT}}) \tag{5}$$

also known as the Itoh condition [15], and

$$W(\partial^2\phi_W) = W(\partial^2\phi_{\mathrm{GT}}). \tag{6}$$

3) Restriction on the Ground Truth: An important restriction arises when unwrapping the phase using RBFNNs. Since the output layer is a sum of radial basis functions, which are smooth, the output is also smooth. Therefore, the RBFNN can only unwrap a phase in which ground truth is smooth. The continuity criterion is defined by the interferometer resolution here. If the ground truth, while continuous, varies too quickly for the interferometer to measure the change, there will be a discontinuity in the signal, and the ground truth will "appear" discontinuous. However, when few discontinuities are present,

they can be hidden relatively easily from the RBFNN using a mask. This condition is usually not restrictive for interferograms generated by high energy density plasmas. If we work with a phase ϕ_{GT} that is twice-continuous and not aliased, i.e., $\partial_{L,R}\phi_{\mathrm{GT}} \in]-\pi, \pi]$, (5) gives

$$W(\partial_{L,R}\phi_W) = \partial_{L,R}\phi_{\mathrm{GT}}.$$

As a result, $W(\partial_{L,R}\phi_W)$ is continuous since $\partial_{L,R}\phi_{\mathrm{GT}}$ is continuous, regardless of how many phase jumps are present in $\partial_{L,R}\phi_W$ [15].

Since our goal is to deal with aliased phase, we can use the much less restrictive assumption $\partial^2\phi_{\mathrm{GT}} \in]-\pi, \pi]$, and (6) gives

$$W(\partial^2\phi_W) = \partial^2\phi_{\mathrm{GT}}. \tag{7}$$

Furthermore, if $\partial^2\phi_{\mathrm{GT}}$ is continuous across the whole domain then $W(\partial^2\phi_W)$ is also continuous everywhere. We will look at both assumptions in the rest of this article.

B. Construction of the Input Layer

While ϕ_W has jumps, we have shown that $W(\partial_{L,R}\phi_W)$ and $W(\partial^2\phi_W)$ are continuous if ϕ_{GT} is twice continuous. Yet, we cannot match the RBFNN output ϕ–ϕ_{GT} using gradient-based optimization since the wrapping operator W, which turns ϕ_{GT} into ϕ_W, is not differentiable. We will use here a gradient-based methods even if gradient-free methods [62], [63], [64], [65], [66], [67] have been used successfully in machine learning. Equations (3) and (4) show that we can use the differentiable sine and cosine functions instead of W, where differentiability is required. As long as ϕ_{GT} is twice continuous, these functions remove the spurious discontinuities otherwise present in $\partial_{L,R}\phi_W$ and $\partial^2\phi_W$ at every phase jump of ϕ_W.

1) Input Layer to Achieve Super-Resolution: We can now construct an input layer $i_{1,\ldots,12}$, where all the data are continuous. At every location inside the interferogram, we can get

$$\left.\begin{aligned} i_1 &= \cos(\phi_W)\\ i_2 &= \sin(\phi_W)\\ i_3 &= \cos(\partial_{xL}\phi_W)\\ i_4 &= \sin(\partial_{xL}\phi_W)\\ i_5 &= \cos(\partial_{yL}\phi_W)\\ i_6 &= \sin(\partial_{yL}\phi_W)\\ i_7 &= \cos(\partial_{xR}\phi_W)\\ i_8 &= \sin(\partial_{xR}\phi_W)\\ i_9 &= \cos(\partial_{yR}\phi_W)\\ i_{10} &= \sin(\partial_{yR}\phi_W)\\ i_{11} &= W(\partial_{xx}\phi_W)\\ i_{12} &= W(\partial_{yy}\phi_W) \end{aligned}\right\}. \tag{8}$$

We can now compare the input layer with the RBFNN output ϕ using the following set of equations:

$$\left.\begin{aligned} o_1 &= \cos(\phi)\\ o_2 &= \sin(\phi)\\ o_3 &= \cos(\partial_x\phi)\\ o_4 &= \sin(\partial_x\phi)\\ o_5 &= \cos(\partial_y\phi)\\ o_6 &= \sin(\partial_y\phi)\\ o_7 &= \cos(\partial_x\phi)\\ o_8 &= \sin(\partial_x\phi)\\ o_9 &= \cos(\partial_y\phi)\\ o_{10} &= \sin(\partial_y\phi)\\ o_{11} &= \partial_{xx}\phi\\ o_{12} &= \partial_{yy}\phi \end{aligned}\right\}. \tag{9}$$

Note that, while i_{11} and i_{12} are still using the wrapping operator, this operator is not present in the equations used to compare the input layer and the RBFNN output because we restricted the second derivative to be between $-\pi$ and π. As the wrapping W has no effect on the RBFNN output, it has completely disappeared from o_{11} and o_{12}, and we can now take their derivatives. However, this operator is still required on the left-hand side of i_{11} and i_{12} to remove the phase jumps in $\partial^2\phi_W$. Also note that we have dropped the subscripts L and R for the first derivatives of the output of the RBFNN, since it is a sum of analytical functions, which derivatives can be computed exactly.

2) Input Layer When Super-Resolution Is Not Required: The training can be substantially simplified when super-resolution is not required, i.e., $\partial\phi_{\mathrm{GT}} \in]-\pi, \pi]$. In this case, we can replace (8) with

$$\left.\begin{aligned} \underline{i}_1 &= \cos(\phi_W), \quad \underline{i}_3 = W(\partial_{xL}\phi_W), \quad \underline{i}_5 = W(\partial_{xR}\phi_W)\\ \underline{i}_2 &= \sin(\phi_W), \quad \underline{i}_4 = W(\partial_{yL}\phi_W), \quad \underline{i}_6 = W(\partial_{yR}\phi_W) \end{aligned}\right\} \tag{10}$$

and (9) with

$$\left.\begin{aligned} \underline{o}_1 &= \cos(\phi), \quad \underline{o}_3 = \partial_x\phi, \quad \underline{o}_5 = \partial_x\phi\\ \underline{o}_2 &= \sin(\phi), \quad \underline{o}_4 = \partial_y\phi, \quad \underline{o}_6 = \partial_y\phi \end{aligned}\right\}. \tag{11}$$

C. Activation Function

Throughout this article, the RBFNN will use a compact Wendland function [58] as the activation function. Such functions can be constructed easily starting from

$$\psi_{p,0}(r) = (1-r)_+^p = \begin{cases} (1-r)^p, & \text{for } 0 \le r \le 1\\ 0, & \text{for } r > 1 \end{cases}$$

and using

$$\psi_{p,q}(r) = \mathfrak{I}^q\psi_{p,0}, \qquad \text{for } 0 \le r \le 1$$

to increase the function smoothness. Here, $p,\ q \in \mathbb{N}$. The operator $\mathfrak{I}$ above is defined as $\mathfrak{I}f(r) = \int_r^\infty f(t)t\,dt$ for $0 \le r$. Wendland functions are C^k and can be computed analytically. They yield a strictly positive definite matrix in $\mathbb{R}^d$, where

$d < p$ and $k = 2q$. The subscripts of $\psi_{p,q}$ will be dropped in the rest of this article. Each neuron (x_n, y_n) in our 2-D dataset is activated using such radial basis functions [50], [68]. In this article, we use exclusively the Wendland function ψ given by

$$\psi(r) = \begin{cases} \frac{1}{3}(1-r)^6[(35r+18)r+3], & \text{for } 0 \le r \le 1 \\ 0, & \text{for } r > 1 \end{cases} \tag{12}$$

obtained for $p = 3$ and $q = 2$.

D. Output Layer

The output layer ϕ is expressed as the sum of radial basis functions $\psi_n(x, y) = \psi(r_n)$ centered on each neuron n located at (x_n, y_n) and scaled by the weight w_n. The output layer of an RBFNN with N neurons is continuous and defined as follows:

$$\phi(x, y) = \sum_{n=1}^{N} w_n(x, y)\psi(r_n(x, y)) \tag{13}$$

with

$$r_n(x, y) = \sqrt{(x - x_n)^2\rho_{x_n}^2 + (y - y_n)^2\rho_{y_n}^2}$$

where ρ_{x_n} and ρ_{y_n} are the activation distance inverses for the nth neuron along the x- and y-directions, respectively. The analytical expression of the Jacobian matrix is greatly simplified when using the inverse of the activation distance. As discussed later in this article, we need to match five constraints to give the neural network architecture super-resolution, i.e., ϕ_{GT}, $\partial_x\phi_{\text{GT}}$, $\partial_y\phi_{\text{GT}}$, $\partial_{xx}\phi_{\text{GT}}$, and $\partial_{yy}\phi_{\text{GT}}$. To match five constraints, we need to inject three degrees of freedom inside the weights as follows:

$$w_n(x, y) = a_n + b_n(x - x_n) + c_n(y - y_n). \tag{14}$$

Together with ρ_{x_n} and ρ_{y_n}, we now have five degrees of freedom per neuron. Note that the weights w_n are now local linear [69], [70] in x and y.

E. Definition of the Objective Function

1) Objective Function With Super-Resolution: We can now define the objective function $F(\mathbf{e})$, used by the training process to minimize the error vector $\mathbf{e} = [e_1, \ldots, e_N]^T$ for all neurons $n \in \{1, \ldots, N\}$

$$F(\mathbf{e}) = \mathbf{e}^{\mathrm{T}}\mathbf{e} = \sum_{n=1}^{N}\sum_{j=1}^{12} e_{jn}^2, \quad \text{with } e_{jn} = o_{jn} - i_{jn}. \tag{15}$$

The error e_{jn} is the difference between the jth input layer value computed at the location (x_n, y_n), i.e., i_{jn}, and the jth output layer value computed at the same location, i.e., o_{jn}. We expect the total error to remain high, even after full convergence, since the training will never match the left and right derivatives simultaneously. Therefore, we need to define another error $\hat{\mathbf{e}}$ to estimate when our network is fully trained

$$\hat{\mathbf{e}}^{\mathrm{T}}\hat{\mathbf{e}} = \sum_{n=1}^{N}\sum_{j=1}^{8} \hat{e}_{jn}^2 \tag{16}$$

where $\hat{e}_{jn} = \hat{o}_{jn} - \hat{i}_{jn}$, and, as shown in the equation at the bottom of the page.

2) Objective Function Without Super-Resolution: The objective function for interferograms where super-resolution is not needed is defined as $\underline{F}(\underline{\mathbf{e}})$ and should be used to minimize the error vector $\underline{\mathbf{e}} = [\underline{e}_1, \ldots, \underline{e}_N]^T$ for all neurons $n \in \{1, \ldots, N\}$

$$\underline{F}(\underline{\mathbf{e}}) = \underline{\mathbf{e}}^{\mathrm{T}}\underline{\mathbf{e}} = \sum_{n=1}^{N}\sum_{j=1}^{6} \underline{e}_{jn}^2, \quad \text{with } \underline{e}_{jn} = \underline{o}_{jn} - \underline{i}_{jn}. \tag{17}$$

As we did earlier, we can define the error $\hat{\underline{\mathbf{e}}}$ to better assess the actual convergence error

$$\hat{\underline{\mathbf{e}}}^{\mathrm{T}}\hat{\underline{\mathbf{e}}} = \sum_{n=1}^{N}\sum_{j=1}^{8} \hat{\underline{e}}_{jn}^2 \tag{18}$$

where $\hat{\underline{e}}_{jn} = \hat{\underline{o}}_{jn} - \hat{\underline{i}}_{jn}$, and

$$\left.\begin{array}{llll} \hat{\underline{i}}_1 = \underline{i}_1, & \hat{\underline{i}}_2 = \underline{i}_2, & \hat{\underline{i}}_3 = \frac{1}{2}(\underline{i}_3 + \underline{i}_5), & \hat{\underline{i}}_4 = \frac{1}{2}(\underline{i}_4 + \underline{i}_6) \\ \hat{\underline{o}}_1 = \underline{o}_1, & \hat{\underline{o}}_2 = \underline{o}_2, & \hat{\underline{o}}_3 = \underline{o}_3, & \hat{\underline{o}}_4 = \underline{o}_4 \end{array}\right\}.$$

3) Regularization: Simple Bayesian regularization [71], or more complex variants such as using Markov chain Monte Carlo [72], have been proposed to avoid over-fitting noisy data, and it is necessary in the presence of noise

$$F_R(\mathbf{e}) = \sum_{n=1}^{N}\sum_{j=1}^{12} e_{jn}^2 + \Omega\sum_{n=1}^{N}\left(a_n^2 + b_n^2 + c_n^2 + \rho_{x_n}^2 + \rho_{y_n}^2\right). \tag{19}$$

We found that Ω should be 1 during the first and second training stages since the noise has the largest impact on the second derivative of the wrapped phase. Regularization is typically not necessary during the last stage of the training and we can use $\Omega = 0$. When super-resolution is not needed, we use

$$\underline{F}_R(\underline{\mathbf{e}}) = \sum_{n=1}^{N}\sum_{j=1}^{6} \underline{e}_{jn}^2 + \Omega\sum_{n=1}^{N}\left(a_n^2 + b_n^2 + c_n^2 + \rho_{x_n}^2 + \rho_{y_n}^2\right). \tag{20}$$

$$\left.\begin{array}{llll} \hat{i}_1 = i_1, & \hat{i}_2 = i_2, & \hat{i}_3 = \frac{1}{2}(i_3 + i_7), & \hat{i}_4 = \frac{1}{2}(i_4 + i_8) \\ \hat{i}_5 = \frac{1}{2}(i_5 + i_9), & \hat{i}_6 = \frac{1}{2}(i_6 + i_{10}), & \hat{i}_7 = i_{11}, & \hat{i}_8 = i_{12} \\ \hat{o}_1 = o_1, & \hat{o}_2 = o_2, & \hat{o}_3 = o_3, & \hat{o}_4 = o_4 \\ \hat{o}_5 = o_5, & \hat{o}_6 = o_6, & \hat{o}_7 = o_{11}, & \hat{o}_8 = o_{12} \end{array}\right\}$$

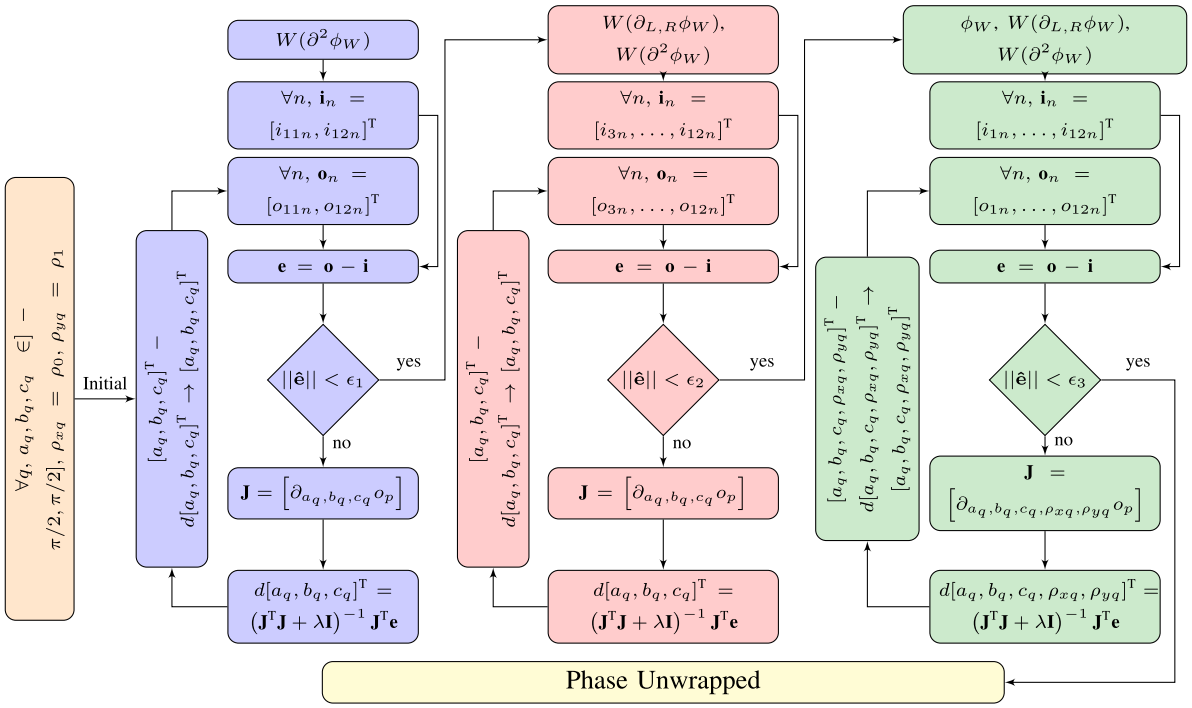

Fig. 1. Staged training of the super-resolution RBFNN with the first stage in blue, the second stage in red, and the last stage in green.

F. Multistage Training

1) Staged LMA: The minimization procedure needs to find the values of the basis function weights $u_n = (a_n, b_n, c_n, \rho_{x_n}, \rho_{y_n})$ for all neurons $n \in \{1, \ldots, N\}$ using a gradient-based algorithm. We used here a standard LMA [73], [74], which minimizes the error $\mathbf{e}$ (not $\hat{\mathbf{e}}$) in the sense of the least square using the Jacobian matrix $\mathbf{J}$. The solution is found by successive iterations, advancing the vector $\mathbf{u} = [u_1, \ldots, u_N]^{\mathrm{T}}$ such that $\mathbf{u}^{\mathrm{new}} = \mathbf{u} - \mathrm{d}\mathbf{u}$ with

$$\mathrm{d}\mathbf{u} = \left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I}\right)^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{e}.$$

The procedure to find λ follows a standard Levenberg–Marquardt (LM) optimization method specific to RBFNN training [75], [76], [77], [78]. This procedure can be altered, avoiding the storage of complete Jacobian matrices [79]. Regardless of the method used, the first two stages can be seen as an initialization of the weights of the RBFNN in the presence of wrapped input data. The last stage corresponds to the true optimization procedure. Since different stages are following a conventional LM optimization using the appropriate inputs and Jacobian matrices, we only summarize here the goals of the three training stages.

1) *Matching the Second Derivatives of ϕ_W:* The error vector is defined as $\mathbf{e}_n = [e_{11n}, e_{12n}]^{\mathrm{T}} = [o_{11n} - i_{11n}, o_{12n} - i_{12n}]^{\mathrm{T}}$. We only train the neural network architecture to optimize the radial basis function weights w_n at this stage using $\mathbf{u}_n = [u_{1n}, u_{2n}, u_{3n}]^{\mathrm{T}} = [a_n, b_n, c_n]^{\mathrm{T}}$. We have found that optimizing the activation distance early on does not really improve the quality of the output at this stage. The quality of convergence at this stage is crucial to super-resolution. This stage is shown in blue in Fig. 1.
2) *Matching the First and Second Derivatives of ϕ_W:* The error vector is now redefined as $\mathbf{e}_n = [e_{3n}, \ldots, e_{12n}]^{\mathrm{T}} = [o_{3n} - i_{3n}, \ldots, o_{12n} - i_{12n}]^{\mathrm{T}}$. Here again, we train the neural network architecture to optimize the radial basis function weights w_n using $\mathbf{u}_n = [u_{1n}, u_{2n}, u_{n_3}]^{\mathrm{T}} = [a_n, b_n, c_n]^{\mathrm{T}}$. This stage propagates the super-resolution information to the first derivatives of the phase. This stage is shown in red in Fig. 1.
3) *Matching ϕ_W as well as the First and Second Derivatives of ϕ_W:* The error vector is defined as $\mathbf{e}_n = [e_{1n}, \ldots, e_{12n}]^{\mathrm{T}} = [o_{1n} - i_{1n}, \ldots, o_{12n} - i_{12n}]^{\mathrm{T}}$. We now optimize the neutral network to find the basis function weight w_n and the inverse activation distances at this stage so $\mathbf{u}_n = [u_{1n}, \ldots, u_{5n}]^{\mathrm{T}} = [a_n, b_n, c_n, \rho_{x_n}, \rho_{y_n}]^{\mathrm{T}}$. This stage unwraps the phase globally, in one single sweep. This stage is shown in green in Fig. 1.

When super-resolution is not needed, the training will only try to match the first left and right derivatives, together with the wrapped phase using only two training stages.

1) *Matching the First Derivatives of ϕ_W:* The error vector is first defined as $\underline{\mathbf{e}}_n = [\underline{e}_{3n}, \ldots, \underline{e}_{6n}]^{\mathrm{T}} = [\underline{o}_{3n} - \underline{i}_{3n}, \ldots, \underline{o}_{6n} - \underline{i}_{6n}]^{\mathrm{T}}$. Here again, we train the neural network architecture to optimize the radial basis function weights w_n using $\mathbf{u}_n = [u_{1n}, u_{2n}, u_{n_3}]^{\mathrm{T}} = [a_n, b_n, c_n]^{\mathrm{T}}$. This stage propagates the super-resolution information to the first derivatives of the phase. This stage is shown in blue in Fig. 2.
2) *Matching ϕ_W as well as the First Derivatives of ϕ_W:* The error vector is defined as $\underline{\mathbf{e}}_n = [\underline{e}_{1n}, \ldots, \underline{e}_{6n}]^{\mathrm{T}} = [\underline{o}_{1n} - \underline{i}_{1n}, \ldots, \underline{o}_{6n} - \underline{i}_{6n}]^{\mathrm{T}}$. We now optimize the neutral network to find the actual basis function weight w_n and activation distances at this stage so $\mathbf{u}_n = [u_{1n}, \ldots, u_{5n}]^{\mathrm{T}} = [a_n, b_n, c_n, \rho_{x_n}, \rho_{y_n}]^{\mathrm{T}}$. This stage

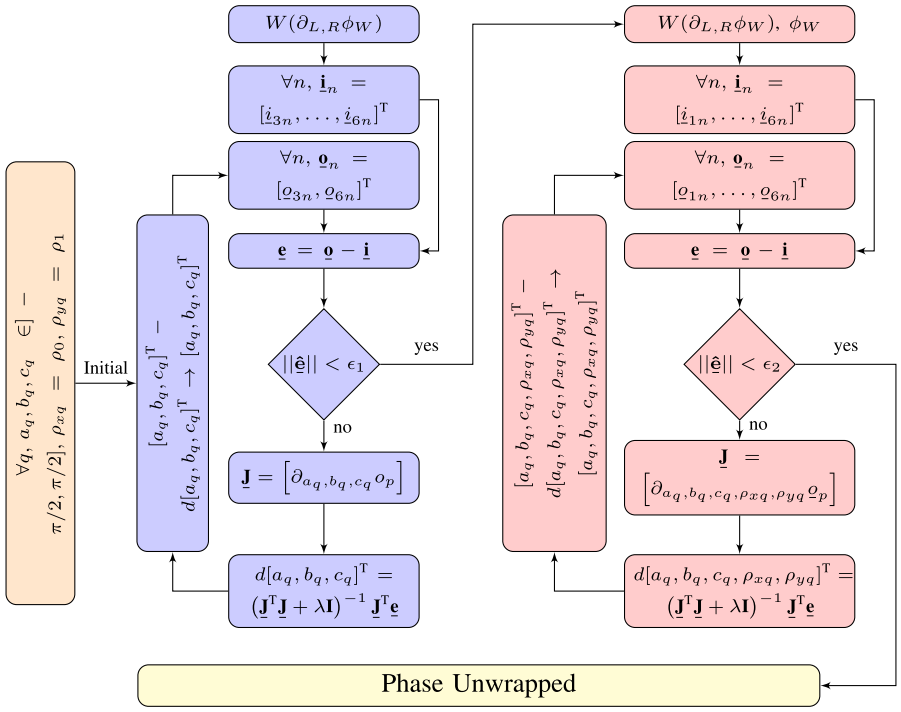

Fig. 2. Staged training of the RBFNN without super-resolution. The first stage is in blue, and the second stage is in red.

unwraps the phase globally, in one single sweep. This stage is shown in red in Fig. 2.

2) Computation of the Jacobian Matrix With Super-Resolution: The Jacobian matrix used in the last stage of the training is given by

$$\mathbf{J} = \begin{bmatrix} \partial_{u_1}\mathbf{e}_1 & \cdots & \partial_{u_N}\mathbf{e}_1 \\ \vdots & \ddots & \vdots \\ \partial_{u_1}\mathbf{e}_N & \cdots & \partial_{u_N}\mathbf{e}_N \end{bmatrix} \tag{21}$$

where

$$\partial_{u_q}\mathbf{e}_p = \begin{bmatrix} \partial_{a_q}o_{1p} & \partial_{b_q}o_{1p} & \partial_{c_q}o_{1p} & \partial_{\rho_{x_q}}o_{1p} & \partial_{\rho_{y_q}}o_{1p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \partial_{a_q}o_{12p} & \partial_{b_q}o_{12p} & \partial_{c_q}o_{12p} & \partial_{\rho_{x_q}}o_{12p} & \partial_{\rho_{y_q}}o_{12p} \end{bmatrix}.$$

Here, the matrix $\partial_{u_q}e_p$ corresponds to the partial derivative of the error e_p between the output layer and the input layer computed at the pth neuron with respect to the weights u_q of the qth neuron. Since the input layer does not depend on any neuron weights, the input values i_{1p}–i_{12p} have been dropped inside the partial derivatives, and only the output values o_{1p}–o_{12p} were retained. To form the smaller Jacobian matrix matrices necessary for the first two stages, we just need to drop the corresponding terms in the full matrix $\partial_{u_q}e_p$ for the first stage

$$\partial_{u_q}\mathbf{e}_p = \begin{bmatrix} \partial_{a_q}o_{11p} & \partial_{b_q}o_{11p} & \partial_{c_q}o_{11p} \\ \partial_{a_q}o_{12p} & \partial_{b_q}o_{12p} & \partial_{c_q}o_{12p} \end{bmatrix}$$

and the second stage

$$\partial_{u_q}\mathbf{e}_p = \begin{bmatrix} \partial_{a_q}o_{3p} & \partial_{b_q}o_{3p} & \partial_{c_q}o_{3p} \\ \vdots & \vdots & \vdots \\ \partial_{a_q}o_{12p} & \partial_{b_q}o_{12p} & \partial_{c_q}o_{12p} \end{bmatrix}.$$

All the functions used in o_{1_n}–o_{12_n} are analytical and can be differentiated, since the wrapping operator W was dropped from o_{11_n} and o_{12_n} using the condition $\partial^2\phi \in]-\pi,\pi]$. We can now compute the Jacobian matrix elements taking the partial derivative on every term in (9) with respect to $\omega \in \{a_q, b_q, c_q, \rho_{x_q}, \rho_{y_q}\}$

$$\left.\begin{array}{c} \partial_\omega o_1 = -\partial_\omega\phi\sin(\phi) \\ \partial_\omega o_2 = \partial_\omega\phi\cos(\phi) \\ \partial_\omega o_3 = -\partial_{x\omega}\phi\sin(\partial_x\phi) \\ \partial_\omega o_4 = \partial_{x\omega}\phi\cos(\partial_x\phi) \\ \partial_\omega o_5 = -\partial_{y\omega}\phi\sin(\partial_y\phi) \\ \partial_\omega o_6 = \partial_{y\omega}\phi\cos(\partial_y\phi) \\ \partial_\omega o_7 = -\partial_{x\omega}\phi\sin(\partial_x\phi) \\ \partial_\omega o_8 = \partial_{x\omega}\phi\cos(\partial_x\phi) \\ \partial_\omega o_9 = -\partial_{y\omega}\phi\sin(\partial_y\phi) \\ \partial_\omega o_{10} = \partial_{y\omega}\phi\cos(\partial_y\phi) \\ \partial_\omega o_{11} = \partial_{xx\omega}\phi \\ \partial_\omega o_{12} = \partial_{yy\omega}\phi \end{array}\right\}. \tag{22}$$

The values of the partial derivatives used in (22) are listed in the Appendix.

3) Computation of the Jacobian Matrix Without Super-Resolution: When dropping super-resolution, the Jacobian matrix $\underline{\mathbf{J}}$ of Fig. 2

$$\underline{\mathbf{J}} = \begin{bmatrix} \partial_{u_1}\underline{\mathbf{e}}_1 & \cdots & \partial_{u_N}\underline{\mathbf{e}}_1 \\ \vdots & \ddots & \vdots \\ \partial_{u_1}\underline{\mathbf{e}}_N & \cdots & \partial_{u_N}\underline{\mathbf{e}}_N \end{bmatrix} \tag{23}$$

can be computed in a similar manner. Here, the error $\partial_{u_q}\underline{\mathbf{e}}_p$ is given by

$$\partial_{u_q}\underline{\mathbf{e}}_p = \begin{bmatrix} \partial_{a_q}\underline{o}_{3p} & \partial_{b_q}\underline{o}_{3p} & \partial_{c_q}\underline{o}_{3p} \\ \vdots & \vdots & \vdots \\ \partial_{a_q}\underline{o}_{6p} & \partial_{b_q}\underline{o}_{6p} & \partial_{c_q}\underline{o}_{6p} \end{bmatrix}$$

for the first stage, and

$$\partial_{u_q}\underline{\mathbf{e}}_p = \begin{bmatrix} \partial_{a_q}\underline{o}_{1p} & \partial_{b_q}\underline{o}_{1p} & \partial_{c_q}\underline{o}_{1p} & \partial_{\rho_{x_q}}\underline{o}_{1p} & \partial_{\rho_{y_q}}\underline{o}_{1p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \partial_{a_q}\underline{o}_{6p} & \partial_{b_q}\underline{o}_{6p} & \partial_{c_q}\underline{o}_{6p} & \partial_{\rho_{x_q}}\underline{o}_{6p} & \partial_{\rho_{y_q}}\underline{o}_{6p} \end{bmatrix}$$

for the second stage.

4) Masking and Clustering Strategies: We now focus on the initialization of our network, looking at masking, neuron clustering, and receptor connections. The mask should be chosen before the training starts and should remain the same throughout the training. Most interferometry data carries noise, discontinuities, and regions that should be dropped from the interferogram. The mask should keep only the data that can be unwrapped with minimal error propagation. The mask over discarded data should slightly overlap with useful data. This strategy allows to compute properly phase derivatives at the mask boundary rather than using extrapolations. Furthermore, the mask should neither split the data into separate regions nor have constricted regions.

The optimal number of receptors is integrated into the optimization procedure and does not have to be computed beforehand. Since we are using compact radial basis functions, any input p such that $r_{pq} = [(x_p - x_q)^2\rho_{x_q}^2 + (y_p - y_q)^2\rho_{y_q}^2]^{1/2} > 1$ will not be connected to the neuron q. The training process is initialized by choosing arbitrary values for ρ_{x_q} and ρ_{y_q}, and these values should be chosen carefully. In regions with rapid phase changes the activation distance inverses should be large.

Some neural networks can use a clustering method, such as k-means [80], [81], to improve the quality and speed of the training. However, in our case, the data pattern is rather inextricable a priori without super-resolution, which is only gained a posteriori. As a result, the shape of the mask and the distance between neurons, rather than the data inside the input layer, truly shape neuron clustering in this work. This greatly simplifies the clustering procedure, which now boils down to a straightforward graph partitioning [82] based on nearest-neighbor connections.

G. Parallel Training

Parallelization becomes necessary for moderately large datasets [83], [84], as the size of the Jacobian matrix $\mathbf{J}$, even sparse, could be difficult to handle on today's supercomputers. This is especially true for high-resolution 2-D interferograms obtained when measuring the electron density of high energy density plasmas. The basic clustering strategy described above can be used to split the main network into K nonoverlapping networks. As it is often the case with parallel codes, we introduce *ghost* neurons [47], which are duplicated neurons shared by exactly two networks. Since the training of each network is now done independently, a synchronization step is required to make sure that all output layers match seamlessly.

We used a single-nearest-neighbor search to define a single-layer of ghost neurons at the boundary between each cluster, allowing for some overlap between networks so that output layers can match seamlessly after synchronization. However, the synchronization procedure needs to keep very few of these ghost neurons to "stitch" the domains together.

1) Output Layer: The synchronization uses a constant phase Φ_k, which is added to the output layer of the network $k \in \{1, \ldots, K\}$ as follows:

$$\phi_k(x, y) = \Phi_k + \sum_{q=M_k}^{N_k} w_q \psi(r_q) + \sum_{q=M_{gk}}^{N_{gk}} w_q \psi(r_q) \tag{24}$$

where the neurons $\{M_k, \ldots, N_k\}$ are the neurons only owned by the kth network, while the neurons $\{M_{gk}, \ldots, N_{gk}\}$ are the ghost neurons of the kth network, owned by the neighbors of the kth network. The last two terms of (24) are the nonsynchronized outputs of the RBFNNs obtained using (13), and their weights are computed using the Levenberg–Marquardt procedure highlighted above. However, under such conditions, the Jacobian matrix is a block matrix given by

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 & & \\ & \ddots & \\ & & \mathbf{J}_K \end{bmatrix}.$$

All the missing elements are 0. Here, $\mathbf{J}_k$ is the Jacobian matrix of (21), but restricted to the k^{th} network, where the error $\mathbf{e}_k$ is defined by (15), also restricted to the kth network. As a result, $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ is the block diagonal, and the LMA can be solved in parallel using

$$\mathrm{d}\mathbf{u}_k = \left(\mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k + \lambda\mathbf{I}_k\right)^{-1}\mathbf{J}^{\mathrm{T}}{}_k\mathbf{e}_k, \quad \text{for } k \in \{1, \ldots, K\}.$$

If super-resolution is not needed, we will use the Jacobian matrix of (23) instead of (21). Once the training is over, we need to synchronize the output layers across all networks. As the synchronization focuses solely on Φ_k, the network parameters a_k, b_k, c_k, ρ_{x_k}, and ρ_{y_k} are constant, so the last two terms of (24) are now two constants and do not need to be computed again throughout the synchronization procedure.

2) Input Layer: For any ghost neuron q shared with the network k but owned by the network labeled l_{qk}, the value $\phi_k(x_q, y_q)$ might be initially different from the value $\phi_{l_{qk}}(x_q, y_q)$ when the staged training is over. Yet, we can synchronize the output layers across different networks by simply defining the synchronization input layer of the kth network as follows:

$$I_{qk} = \phi_{l_{qk}}(x_q, y_q) \tag{25}$$

with the corresponding output value

$$O_{qk} = \phi_k(x_q, y_q). \tag{26}$$

There is no need to use wrapping functions like sine or cosine here. We are dealing with a phase that has been unwrapped successfully for each separate network at this point. However, it remains out of synchronization across the domain. Now, the error to minimize is given by

$$\mathbf{E}^{\mathrm{T}}\mathbf{E} = \sum_{k=1}^{K}\sum_{q=M_{gk}}^{N_{gk}} E_{qk}^2, \quad \text{with } E_{qk} = O_{qk} - I_{qk}. \tag{27}$$

The error E_{qk} is the squared difference of the input layer value from (25) computed at the location (x_q, y_q) inside the kth cluster, i.e., I_{qk}, and the output layer value computed at the same location, i.e., O_{qk}.

3) Synchronization LMA: We use again the LMA to minimize the error **E** in the sense of the least square using the synchronization Jacobian matrix

$$\mathbf{J_s} = \begin{bmatrix} \partial_{\Phi_1} E_{M_{g1}} & \cdots & \partial_{\Phi_K} E_{M_{g1}} \\ \vdots & \ddots & \vdots \\ \partial_{\Phi_1} E_{N_{gK}} & \cdots & \partial_{\Phi_K} E_{N_{gK}} \end{bmatrix}. \tag{28}$$

Here, we cannot drop the input values I_{qk} from the Jacobian matrix since the input layer for the kth network may depend on a phase bias $\Phi_{k'}$ when ghost neurons in the network k are owned by the network k'. We are now using a standard LMA to solve this problem. One final parallel three-stage training can be done after the synchronization procedure to eliminate any residual errors, while keeping all Φ_k constant.

III. Accuracy of the Staged Neural Network Using Synthetic Phase

This section presents the performance of the neural network architecture for different types of synthetic phase variation with strong local aliasing. The first test looks at a smoothly varying phase. Then, we focus on phase that varies randomly. The nonmonotonic nature of the phase variation creates a new set of challenges on the top of phase aliasing, especially in the presence of a fragmented mask and high noise levels. We looked at the accuracy of the neural network architecture by computing the error between the ground-truth phase and the output layer, $\epsilon(x, y) = |\phi_{\mathrm{GT}}(x, y) - \phi(x, y)|/(2\pi)$, which is given in units of 2π rather than radians and represents the normalized error with respect to the wrapped phase ϕ_W, which spans an interval of 2π. Each network is trained until the maximum error goes below 10^{-3} or when the overall error cannot be improved.

A. Quasi-Monotonic Phase

The quasi-monotonic phase is given by

$$\phi_{\mathrm{GT}}(x, y) = \alpha(x + y) + \beta \exp\left(-\frac{x^2 + y^2}{\sigma^2}\right). \tag{29}$$

Fig. 3(a) and (b) shows the initial ground-truth phase and the digitized wrapped phase with strong aliasing, respectively, all in radians. The neural network output layer is virtually identical to ϕ_{GT}. However, the very high accuracy is obtained only after removing a constant bias that exists between the two phases. This bias is not an error. Rather it comes from a lack of absolute reference because we are measuring a phase shift and not an absolute phase. Since this bias cannot be determined from the wrapped phase shown in Fig. 3(b), we computed this bias to make the average of network output equal to the average of the ground truth, and the recovered phase is shown in Fig. 3(c). In reality, we would not have access to this information when measuring the phase. However, this limitation is physical rather than imposed by the method presented here. For $\beta < 40$, the RBFNN recovers ϕ_{GT} from the digitized phase with an error well below 10^{-3}. The error becomes quickly worse with larger values of β. After this correction, Fig. 3(d) shows that the maximum error between the RBFNN and the phase from (29) is less than 0.1%.

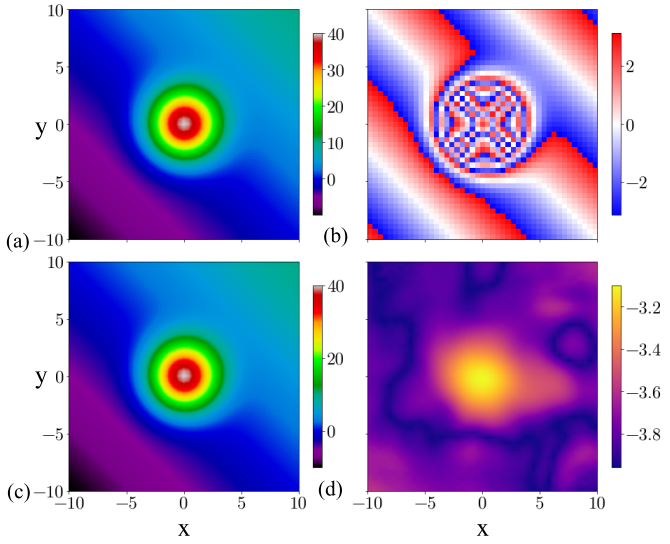

Fig. 3. (a) Ground-truth phase, (b) digitized wrapped phase, (c) RBFNN output, all in radians, and (d) output phase error on the $\log_{10}$ scale.

B. Random Phase With Masked Data

When the phase varies randomly across the domain, the neural network architecture cannot exploit any trend to recover the ground truth ϕ_{GT}. If aliasing is introduced, then it becomes very difficult to even attempt the task manually. While Fig. 4(a) shows that ϕ_{GT} does not vary wildly, the digitized, wrapped phase in Fig. 4(b) shows that a randomly varying phase is in fact relatively difficult to unwrap. Yet the output of the neural network shown in Fig. 4(c) matches well ϕ_{GT}, with an error below 0.1% shown in Fig. 4(d). The large masked regions in Fig. 4 could represent regions with low contrast or local loss of signal. The error is more homogeneously distributed compared with the quasi-monotonic phase presented in the previous section, mostly caused by global (rather than local) aliasing. There is very little change in the overall error compared with the unmasked case (not shown). Fig. 5(a) shows that aliasing is large enough to cause several sections of the wrapped phase to increase smoothly, while the ground-truth phase actually decreases. This happens in regions where the first derivative of ϕ_{GT}, as shown in Fig. 5(b), is smaller than $-\pi$, causing $W(\partial\phi_W)$ to wrap around. Note that this wrapping is not problematic since we are using the sine and cosine functions when training our neural network on first derivatives, which continuously vary throughout phase jumps.

Since the neural network architecture is trained on a dataset that contains the first and second derivatives of the phase, we can take the derivatives of the neural network architecture output to estimate the derivatives of the phase. Fig. 5(b) shows an excellent agreement with the ground-truth phase derivative. We clearly see here that the RBFNN cannot match the left and right first derivatives simultaneously, since they have different values. Rather the RBFNN matches the average, which is the central first derivative. As shown in (8), the neural network architecture uses the left and right derivatives of ϕ_W to compute the weights used in the output layer. Therefore, the derivative of ϕ, which also matches the derivative of ϕ_{GT}, is located in between the left and right derivatives of ϕ_W,

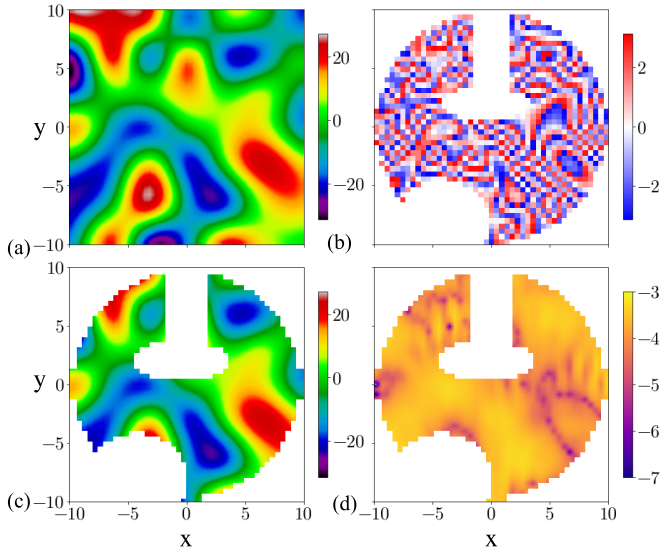

Fig. 4. (a) Ground truth of a randomly generated phase. (b) Wrapped phase with asked regions. The mask is just used to demonstrate the RBFNN capabilities rather than hiding poorly resolved regions here. (c) Phase obtained by the RBFNN. (d) Error on the $\log_{10}$ scale.

as expected [see Fig. 5(b)]. As a result, using the error $\hat{\mathbf{e}}$ given in (18) makes more sense than using $\mathbf{e}$. Based on the assumptions that $\phi_{\mathrm{GT}} \in]-\pi, \pi]$ and ϕ_{GT} is continuous, we see that $W(\partial_x^2 \phi_W)$ has no jump since $W(\partial_x^2 \phi_W) = \partial_x^2 \phi_{\mathrm{GT}}$.

Fig. 5 clearly shows how the neural network architecture can recover the ground truth ϕ_{GT}, without explicitly unwrapping it. The output of the neural network and its derivatives are continuous by construction, since they are the sum of continuous radial basis functions. At the end of the first stage of the training, the second derivative of the neural network matches directly the second derivative of the wrapped phase, which is continuous since $W(\partial_x^2 \phi_W) = \partial_x^2 \phi_{\mathrm{GT}}$ and $\partial_x^2 \phi_{\mathrm{GT}}$ is continuous. At the end of the first stage, the output of the neural network is continuous since the output is continuous by construction. At the end of the second stage, the network output matches the first derivatives of the wrapped phase via the sine and cosine functions. This approach hides the phase jumps created by the wrapping operator W when aliasing exists. Again, at the end of this stage, the output of the neural network architecture is also continuous since it is the sum of continuous functions. During the third stage, where the network is trained to match the wrapped phase values via the sine and cosine functions, its output again remains continuous. Therefore, the training process forced the output of the neural network architecture to match the sine and cosine of the wrapped phase, and the radial basis functions used to build this network forced the output to be continuous, allowing to remove the jumps of the wrapped phase.

C. Random Phase With Noise

When there is no aliasing, the noise can be removed from the wrapped phase using standard filtering techniques specifically developed for interferograms, such as the fringe smoothing approach [85], local fringe frequency estimation [86], windowed Fourier filtering [87], [88], or Gabor filter local frequency [89]. Any of these techniques can be applied

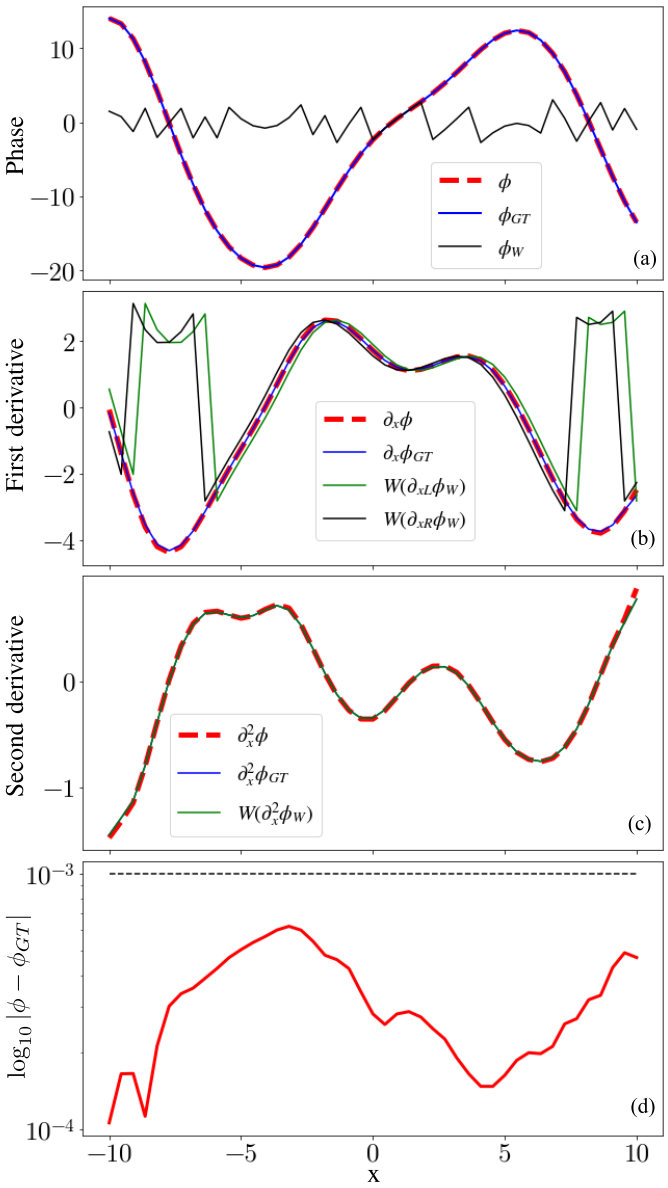

Fig. 5. (a) Ground-truth phase, neural network output phase, and wrapped phase of Fig. 4 along the x-direction, together with their (b) first and (c) second derivatives along the x-direction. (d) Output phase error on the $\log_{10}$ scale.

to the wrapped phase before feeding it to the neural network architecture. When filtering the wrapped phase, we can detect locations with noisy data by computing the phase residues and mask out locations where the residues lead to a nonconservative result [90], providing that the ground-truth phase is conservative (e.g., interferogram of topographic data). Filtering can also be done during the unwrapping procedure [91], [92], [93], [94], [95] but cannot be applied here as the filtering procedure is deeply dependent on the unwrapping method. However, when aliasing is present, direct filtering becomes more problematic. For one, the method of residue cannot be used reliably. Furthermore, aliasing can act like noise and it becomes difficult to differentiate between good data that was wrapped multiple times and noisy data.

To look at the impact of noise on the neural network performance for the strongly aliased phase, we added noise

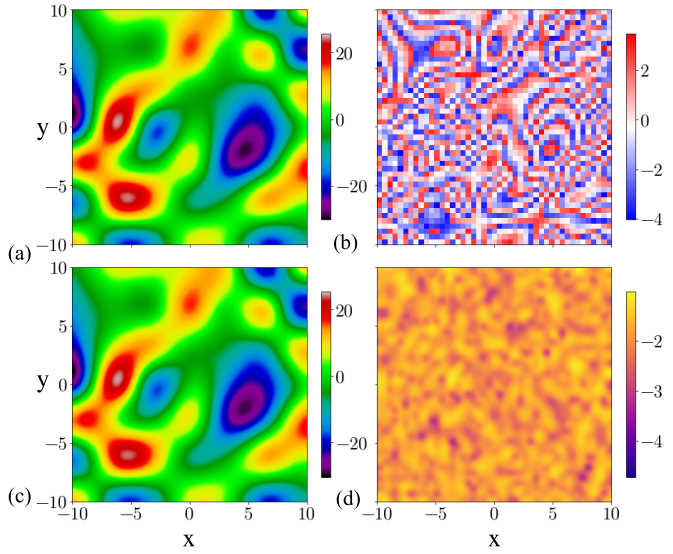

Fig. 6. (a) Ground-truth phase, (b) digitized wrapped phase, (c) RBFNN output, all in radians, and (d) phase error on the $\log_{10}$ scale with a noise level $\gamma = 0.1\%$ or 10% of the maximum value of the wrapped phase.

$\mathcal{N}(x, y) \in [-1, 1]$ to the wrapped phase as follows:

$$\phi_W(x, y) = W(\phi_{\mathrm{GT}}(x, y)) + \gamma\pi\mathcal{N}(x, y) \tag{30}$$

where γ is a constant controlling the maximum noise level. Fig. 6 shows that the neural network architecture recovers the ground truth ϕ_{GT} with an error that is on the order of the noise level added to the wrapped phase. The neural network architecture tends to perform well for $\gamma < 0.1$, but tends to develop O(1) error when $\gamma > 0.1$. Similar results are found with masked data. Therefore, without a specific noise filtering strategy working on the aliased wrapped phase, we find that the neural network remains reliable for noise levels below 10% of the wrapped phase. Larger noise levels will require some filtering beforehand. Fig. 7 shows how the regularization avoids over-fitting of the network output, limiting the impact of the noise on unwrapped phase.

D. Comparison With Other Methods

Since the proposed RBFNN gives excellent results compared with cases where the phase is quasi-monotonic or random, with or without noise, it is worth comparing it to standard methods used in the literature. The comparison done here is clearly not exhaustive. Furthermore, we are not trying to show that the RBFNN is superior to other methods. Each method is usually designed with a particular application in mind and a method working well in a given set of conditions can have subpar results when faced with another. Overall, our method requires large Jacobian matrices and should be used when the phase has been heavily digitized or when regions have missing or corrupted phase.

Fig. 8 shows that the RBFNN performs as well as the discrete cosine transform (DCT) of [16] just before super-resolution is needed, at which points the DCT method fails. Note that the precision of the RBFNN can be improved up to an error of 10^{-4} by reducing the limit of the convergence error, while there is no mechanism to decrease the error of the

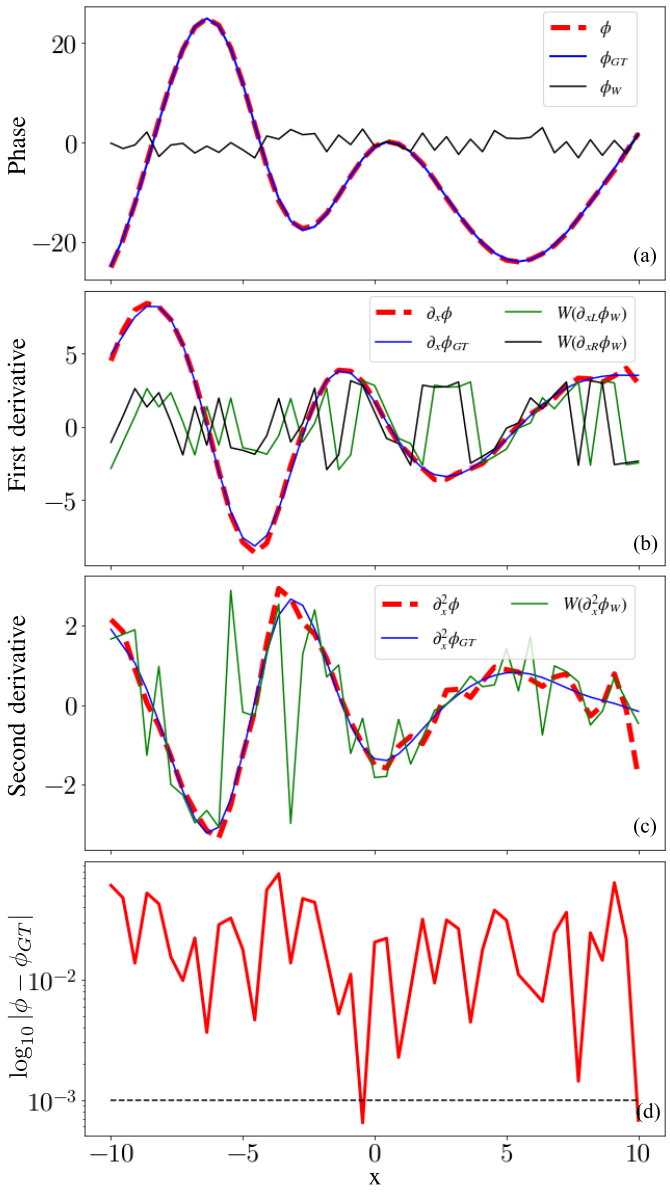

Fig. 7. (a) Ground-truth phase, neural network output phase, and wrapped phase of Fig. 6 along the x-direction with a noise level $\gamma = 0.1\%$ or 10% of the maximum value of the wrapped phase. Their (b) first and (c) second derivatives along the x-direction. (d) Output phase error on the $\log_{10}$ scale.

DCT method, except by increasing the resolution. However, the DCT is extremely fast and returned a solution in 38 ms. The RBFNN required 48 s on hardware with 48 cores at 2 GHz and for a 50×50 grid to arrive at an error on the order of 10^{-3}. The RBFNN was written in python which can substantially slow down computations. However, all the functions were accelerated using Numba, and the LMA was done in parallel. Further acceleration could be gained using GPUs.

We also compared our results against a convolutional long short-term memory (LSTM) network described in [39]. This deep-learning method is one of the best performers with phase unwrapping algorithms according to the authors. The LSTM took 229 s to create the learning set, 1894 s to train on this set, and 1 s to compute the solution on a 256×256 grid. Note

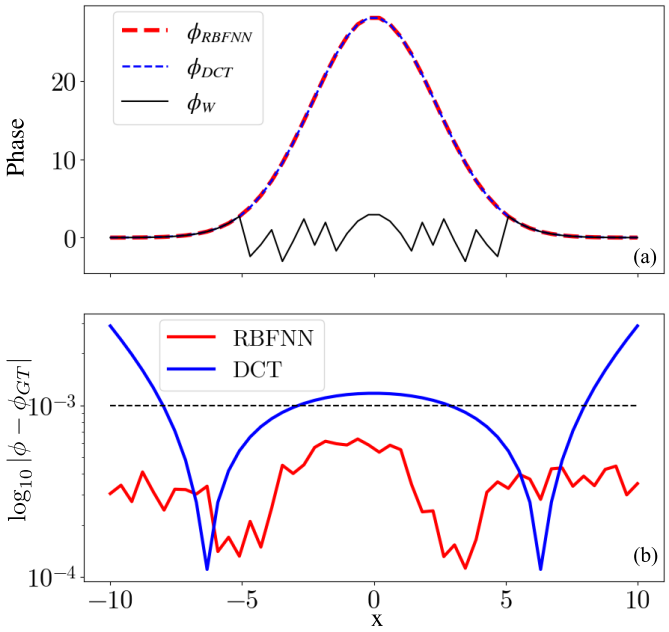

Fig. 8. (a) Phase unwrapped in radians using the DCT of [16] and the proposed RBFNN, together with the wrapped phase. (b) Output phase error on the $\log_{10}$ scale.

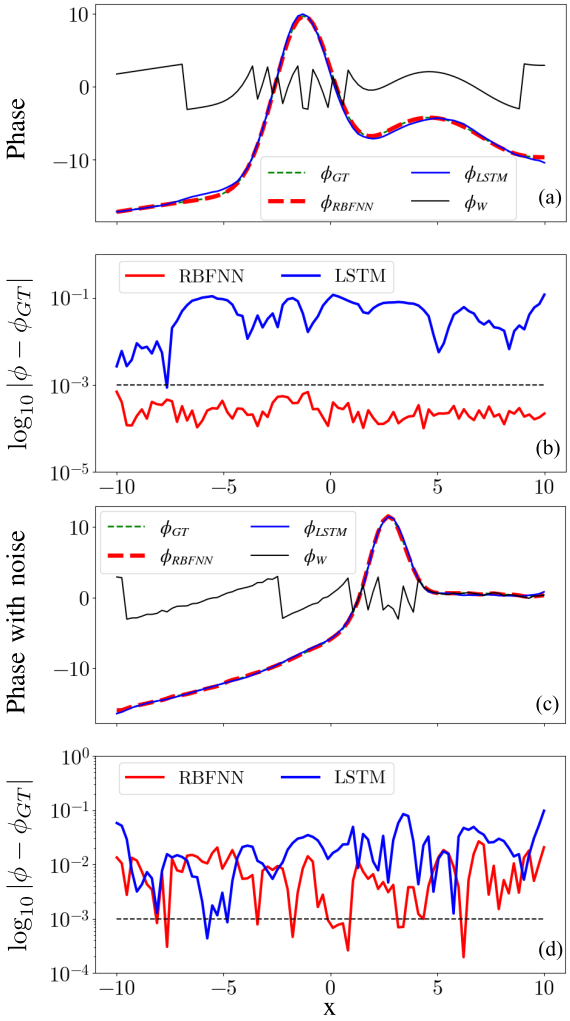

Fig. 9. (a) Phase unwrapped in radians using the LSTM network of [39] and the proposed RBFNN, together with the wrapped phase. (b) Output phase error on the $\log_{10}$ scale. (c) Another phase unwrapping when noise is present. (d) Corresponding error.

that the network can actually be trained an order of magnitude faster by using GPUs, but we used CPUs here to keep the same hardware in both tests. Fig. 9(a) shows the initial wrapped phase, part of the test deck of the LSTM network python notebook, together with the unwrapped phase done by both methods. Fig. 9(b) shows that the LSTM network gave a much higher error for the standard parameters coming with the code. We believe that the higher error does not come from an inherent issue of the LSTM network but from the fact than the training set included a noisy phase. The proposed RBFNN took 220 s for a grid that was 64 × 64 using the hardware described above.

Once we move to noisy data, the two methods follow ϕ_{GT} in a similar fashion, as shown in Fig. 9(c), the error is noise limited [see Fig. 9(d)]. It is important to note that the LSTM algorithm requires that the size of the grid of the phase to be unwrapped matches the size of the training dataset phase. Also, there is no provision to allow for masking regions where the phase is not usable. As a result, if each new phase unwrapping requires a different mask, the LSTM network has to go through a new learning step.

IV. Accuracy of the Staged Neural Network on Experimental Interferograms

We now use the staged training on real interferograms generated by the interaction of a green laser beam with a high energy density plasma [97]. The phase shift corresponds to the line-average electron density [98] of the plasma. The plasmas were generated by using a multipin radial foil configuration [96] connected to the electrodes of a pulsed-power driver [99]. In this case, we do not know the ground truth. Therefore, we assessed the quality of the unwrapping procedure by looking at the difference between the measured wrapped phase and the neural network output layer. The final error, $\epsilon(x, y) = |W(\phi_W(x, y)) - W(\phi(x, y)|)/(2\pi)$, is given in units of 2π. It is the normalized error with respect to the wrapped phase ϕ_W, which spans an interval of 2π.

The interferogram is presented in Fig. 10(a). The measurement is based on shearing [7] rather than Mach–Zehnder interferometry. The former uses a single reference path which is insensitive to mechanical vibrations, greatly affecting the fringe pattern. It is possible to use a reference phase, without plasma, and subtract it from the measurement done when a plasma is present. The difference in phase is proportional to the line-average electron density. Starting with the region of interest shown in Fig. 10(a), the Fourier transform gives a spectrum that is symmetric with respect to the origin since the phase data are real-valued. We use a single square filter to isolate the dominant modes, but excluding the origin, where the dc component is located. The inverse Fourier transform is now complex valued since the filter broke the symmetry with respect to the origin. The phase of each complex values corresponds to the wrapped phase measured by the interferogram [12], [13], [14] and shown in Fig. 10(b).

The data are then down-sampled by a factor of 6 × 6 to compress the interferogram [as shown in Fig. 10(c)].

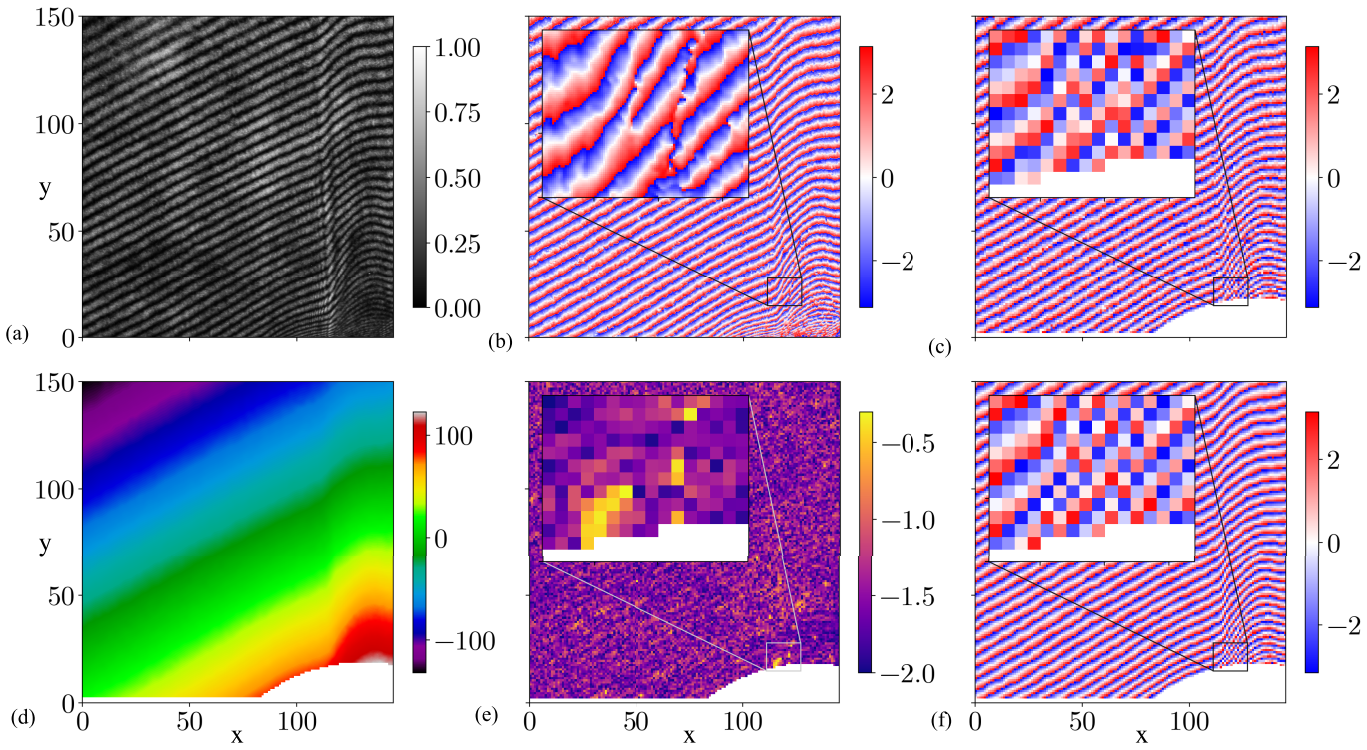

Fig. 10. (a) Normalized interferogram of the left side of a hollow plasma jet. The jet is symmetric with respect to the right axis. (b) Wrapped phase after applying Fourier filtering to keep the dominant modes. (c) Digitized down-sampled phase with the mask to hide the regions where phase data should not be used. (d) Actual output ϕ of the RBFNN. (e) $\log_{10}$ error between the wrapped phase ϕ_W and $W(\phi)$. (f) Wrapped RBFNN output is given for reference. All phase data are in radians. The zoomed panels highlight the region with the strongest aliasing. Note that the radius at the base of the jet is 1 mm (see [96], for more information on the jet geometry).

Using this data, we get a total of 150×140 neurons in the output layer. The input layer has a total of $12 \times 150 \times 140$ inputs when super-resolution is needed (8) and $6 \times 150 \times 140$ inputs where it is not (10). The only sample in the training set is the data itself, as given in Fig. 10(c). While the compression is not necessary to demonstrate the efficacy of the RBFNN, this compression created a region with strong aliasing [the zoomed portion of Fig. 10(c)]. In reality, a strongly under-resolved interferogram may have a fringe count so high that some regions will have very low contrast. The Fourier filtering used to turn the experimental data, which form follows (1), into phase data cannot handle this high-density low-contrast situation. Note that, if the second derivative of the phase falls outside of $]-\pi, \pi]$, the RBFNN will also fail regardless of the quality of the filtering. The mask drops the data where fringes could not be resolved clearly, some edge data corrupted by the Fourier filtering (see Fig. 10(b) near the x-axis), and under-resolved fringes, partly caused by the Fourier filtering. We hid the data using a disk-shaped mask. We then trained the neural network architecture with super-resolution. The output of the network is presented in Fig. 10(d). The bump in electron density caused by the plasma jet appears clearly in the figure. The error between the measured wrapped phase of Fig. 10(c) and the wrapped value of the output of the RBFNN of Fig. 10(d) is shown in (e). This error is on the order of 10%, leading to an average error that is comparable to the noise recorded by the interferometer and clearly visible in the insert of Fig. 10(b). We see two types of error larger than 10% in this figure. The error that is randomly distributed throughout is caused by a local phase jump when the noise of the wrapped phase is close to $-\pi$ or π. This noise is not present in the output of the network due to regularization. The second type of error is closer to a true error, as the RBFNN has some difficulty to unwrap the phase accurately [region shown in the zoomed insert of Fig. 10(e)]. This error is coming from the 6×6 compression ratio, which has aliased the phase slightly beyond the capabilities of the RBFNN. However, this error is reduced down to noise error if we use a 5×5 compression ratio. It is important to note that this error did not propagate to the neighboring neurons. If we consider the low, average error level of Fig. 10(e) and the smoothness of the output, the RBFNN unwrapped the phase successfully. The wrapped output is shown in Fig. 10(f) and can be compared with the measured phase in Fig. 10(c). Without super-resolution, the RBFNN was not able to unwrap the phase.

Since the shearing interferometer is mechanically stable, we can measure accurately the density of the jet by subtracting the background phase from Fig. 10(d). Following the exact same procedure, we can process the same region of the interferogram without any plasma. In this case, the fringe pattern is relatively periodic, as shown in Fig. 11(a). We get the wrapped background phase using the same Fourier filter as the one used for the interferogram with plasma. Since the pattern of the interferogram is clearly resolved, we trained the RBFNN without super-resolution, leading to the output presented in Fig. 11(b). It is interesting to note that the error between the wrapped output layer and the data, as shown

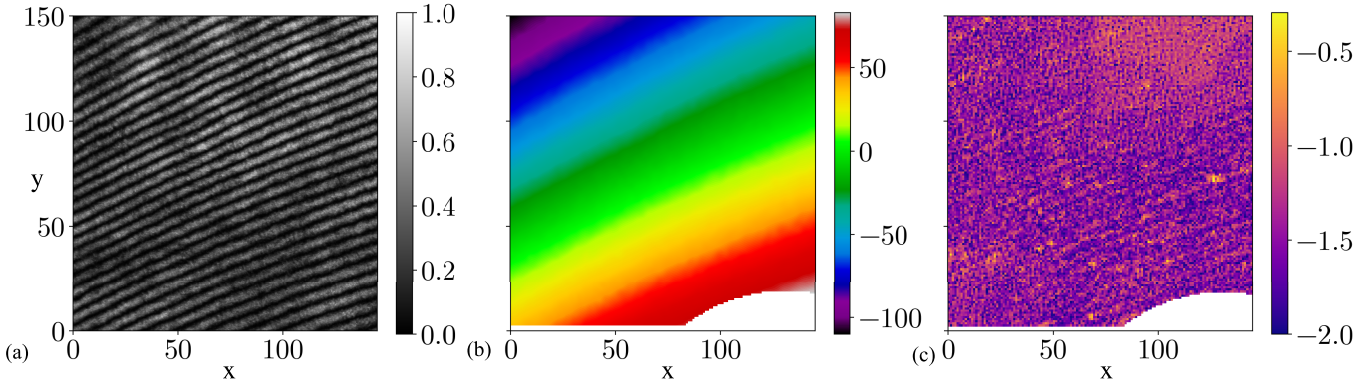

Fig. 11. (a) Normalized interferogram with no plasma present, (b) RBFNN output in radians ϕ, and (c) $\log_{10}$ error between the wrapped phase ϕ_W and $W(\phi)$.

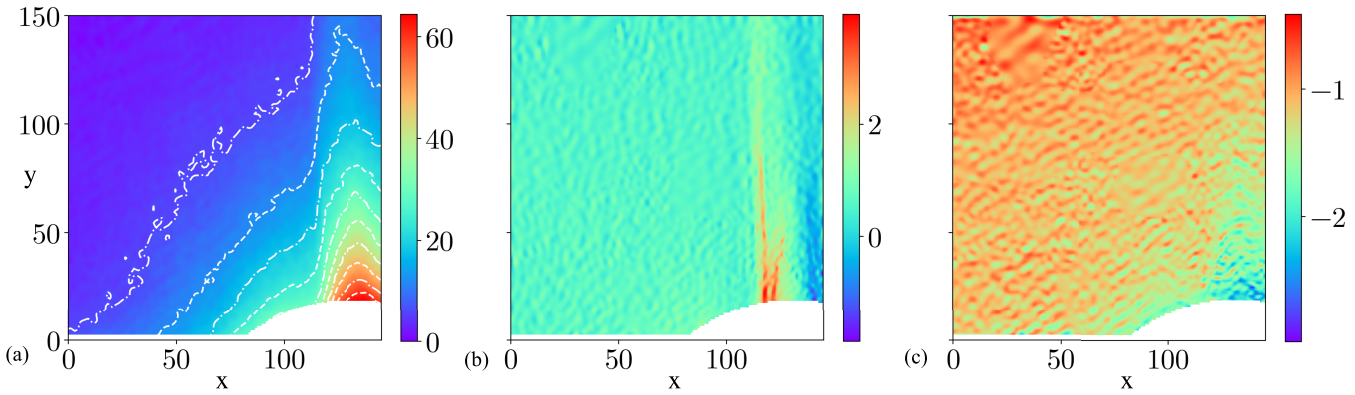

Fig. 12. (a) Line average phase shift and its derivatives along (b) horizontal and (c) vertical directions for a hollow plasma jet. The line average density is directly proportional to the line average phase shift. The axes units are arbitrary.

in Fig. 11(c), is similar to the error when the plasma is present. This indicates that the error is mostly caused by noise. Once the background phase is removed from the phase with plasma, we get the line average density of the jet presented in Fig. 12(a). While noise is present in the density measurement, its source has been filtered by our earlier Fourier transform. We believe that the density fluctuations seen in the RBFNN output derivatives shown in Fig. 12(b) and (c) do not carry any physical information of the density itself. As a result, an Abel inversion technique that is robust to significant noise levels (see [100]) should be used to compute the volume electron density. We can note the difference in smoothness between domains due to the optimization of the activation distance during the last stage.

V. Conclusion

The proposed RBFNN incorporates the functions necessary to deal with aliased interferograms by combining: 1) scattered neuron placement, allowing to discard relatively easily corrupted data while keeping data carrying high fidelity information; 2) the use of a mask to hide external geometries, which are often present in phase measurement; and 3) a regularization scheme which can filter noise very effectively. The RBFNN can unwrapped the phase extracted from an interferogram by comparing measurements with the output of the RBFNN through sine and cosine functions. These functions hide the existence of any discontinuity in the wrapped phase from the training set. Taking into account that the RBFNN output is continuous by construction, the neural network architecture yields a phase that is fully unwrapped once the error between the input and output layers has been minimized. As the network is trained to match the first and second derivatives of the phase, high-fidelity gradients can be computed directly from the RBFNN output since the impact of noise was limited by regularization. The network structure allows a clustering strategy where parallelization is easy to implement. It transforms a dense matrix into a block diagonal matrix, speeding up the training substantially.

This work did not attempt to do any filtering in the pre-learning stage, except from a Fourier filter, which was mostly used to get the complex amplitude field, allowing to compute the wrapped phase readily. However, filtering techniques can be used in conjunction with the proposed algorithm. While regularization does filter data by limiting over-fitting, it should not be considered a very effective filter. First, the regularization parameter is global. Second, the regularization is static, and there is no mechanism in place in our training that can optimize it.

While the RBFNN presented here requires more memory and computational power than a more basic phase unwrapping algorithm (see [20]), errors are relatively easy to detect and remain local, as shown in Fig. 10(e). Combined with the proposed parallelization strategy, the unwrapping time can be reduced substantially. While the training procedures with and without super-resolution are clearly separated in this work, it is

possible to use phase derivative averages to find regions where super-resolution is required (i.e., $|W(\phi_w)| > \pi$) and regions where it is not (i.e., $|W(\phi_w)| < \pi$). The Jacobian matrix can be adapted locally to each method seamlessly. However, this criterion is not absolute and super-resolution should be used as much as possible. While we have not discussed local interferogram defects, such as branch cuts or line splitting, they can be masked and easily removed from the original dataset, allowing the neural network to perform the phase unwrapping without much difficulty.

APPENDIX

This section lists the analytic functions necessary to compute the Jacobian matrices used in this article. The output layer is $\phi(x,y) = \sum_{q=1}^{N} w_q \psi(r_q)$ with $r_q = ((x-x_q)^2\rho_{x_q}^2 + (y-y_q)^2\rho_{y_q}^2 + \epsilon)^{1/2}$. Furthermore, $\psi' = (d/dr)\psi$ and $\psi'' = (d^2/dr^2)\psi$. ϵ is used in computations to avoid a possible division by 0, which only happens numerically. The problematic terms w_q/r_q found below are multiplied by ψ' and $\lim_{r_q\to 0}\psi' \propto r_q$ for radial basis functions.

A. Partial Derivatives With Respect to the RBFNN Parameters

$$\partial_{a_q}\phi = \psi(r_q)$$

$$\partial_{b_q}\phi = (x-x_q)\psi(r_q)$$

$$\partial_{c_q}\phi = (y-y_q)\psi(r_q)$$

$$\partial_{\rho_{x_q}}\phi = \frac{w_q(x-x_q)^2\rho_{x_q}}{r_q}\psi'(r_q)$$

$$\partial_{\rho_{y_q}}\phi = \frac{w_q(y-y_q)^2\rho_{y_q}}{r_q}\psi'(r_q).$$

B. Partial Derivatives Along x

$$\partial_x\phi = \sum_{q=1}^{N} b_q\psi(r_q) + \frac{w_q(x-x_q)\rho_{x_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{xx}\phi = \sum_{q=1}^{N}\left[\frac{(w_q + b_q(x-x_q))\rho_{x_q}^2}{r_q} - \frac{w_q(x-x_q)^2\rho_{x_q}^4}{r_q^3}\right]\psi'(r_q) + \frac{w_q(x-x_q)^2\rho_{x_q}^4}{r_q^2}\psi''(r_q)$$

$$\partial_{xa_q}\phi = \frac{(x-x_q)\rho_{x_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{xxa_q}\phi = \left[\frac{\rho_{x_q}^2}{r_q} - \frac{(x-x_q)^2\rho_{x_q}^4}{r_q^3}\right]\psi'(r_q) + \frac{(x-x_q)^2\rho_{x_q}^4}{r_q^2}\psi''(r_q)$$

$$\partial_{xb_q}\phi = \psi(r_q) + \frac{(x-x_q)^2\rho_{x_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{xxb_q}\phi = \left[\frac{3(x-x_q)\rho_{x_q}^2}{r_q} - \frac{(x-x_q)^3\rho_{x_q}^4}{r_q^3}\right]\psi'(r_q) + \frac{(x-x_q)^3\rho_{x_q}^4}{r_q^2}\psi''(r_q)$$

$$\partial_{xc_q}\phi = \frac{(x-x_q)(y-y_q)\rho_{x_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{xxc_q}\phi = \left[\frac{(y-y_q)\rho_{x_q}^2}{r_q} - \frac{(x-x_q)^2(y-y_q)\rho_{x_q}^4}{r_q^3}\right]\psi'(r_q) + \frac{(x-x_q)^2(y-y_q)\rho_{x_q}^4}{r_q^2}\psi''(r_q).$$

C. Partial Derivatives Along y

$$\partial_y\phi = \sum_{q=1}^{N} c_q\psi(r_q) + \frac{w_q(y-y_q)\rho_{y_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{yy}\phi = \sum_{q=1}^{N}\left[\frac{(w_q + c_q(y-y_q))\rho_{y_q}^2}{r_q} - \frac{w_q(y-y_q)^2\rho_{y_q}^4}{r_q^3}\right] \times \psi'(r_q) + \frac{w_q(y-y_q)^2\rho_{y_q}^4}{r_q^2}\psi''(r_q)$$

$$\partial_{ya_q}\phi = \frac{(y-y_q)\rho_{y_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{yya_q}\phi = \left[\frac{\rho_{y_q}^2}{r_q} - \frac{(y-y_q)^2\rho_{y_q}^4}{r_q^3}\right]\psi'(r_q) + \frac{(y-y_q)^2\rho_{y_q}^4}{r_q^2}\psi''(r_q)$$

$$\partial_{yb_q}\phi = \frac{(x-x_q)(y-y_q)\rho_{y_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{yyb_q}\phi = \left[\frac{(x-x_q)\rho_{y_q}^2}{r_q} - \frac{(x-x_q)(y-y_q)^2\rho_{y_q}^4}{r_q^3}\right]\psi'(r_q) + \frac{(x-x_q)(y-y_q)^2\rho_{y_q}^4}{r_q^2}\psi''(r_q)$$

$$\partial_{yc_q}\phi = \psi(r_q) + \frac{(y-y_q)^2\rho_{y_q}^2}{r_q}\psi'(r_q)$$

$$\partial_{yyc_q}\phi = \left[\frac{3(y-y_q)\rho_{y_q}^2}{r_q} - \frac{(y-y_q)^3\rho_{y_q}^4}{r_q^3}\right]\psi'(r_q) + \frac{(y-y_q)^3\rho_{y_q}^4}{r_q^2}\psi''(r_q).$$

REFERENCES

[1] F. Jahoda and G. Sawyer, "Optical refractivity of plasmas," *Methods Exp. Phys.*, vol. 9, pp. 1–48, Jan. 1971.

[2] C. M. Vest, *Holographic Interferometry*. Hoboken, NJ, USA: Wiley, 1979.

[3] P. Hariharan, *Basics of Interferometry*. Amsterdam, The Netherlands: Elsevier, 2010.

[4] A. V. Oppenheim and J. S. Lim, "The importance of phase in signals," *Proc. IEEE*, vol. 69, no. 5, pp. 529–541, May 1981.

[5] J. Blackledge, *Quantitative Coherent Imaging*. London, U.K.: Academic Press, 1989.

[6] S. Witoszynskyj, A. Rauscher, J. R. Reichenbach, and M. Barth, "Phase unwrapping of MR images using ΦUN—A fast and robust region growing algorithm," *Med. Image Anal.*, vol. 13, no. 2, pp. 257–268, Apr. 2009.

[7] G. S. Sarkisov, "Shearing interferometer with air wedge for electron plasma diagnostics in a dense plasma," *Instrum. Exp. Techn.*, vol. 39, no. 5, pp. 110–114, 1996.

[8] O. W. Stanley, A. B. Kuurstra, L. M. Klassen, R. S. Menon, and J. S. Gati, "Effects of phase regression on high-resolution functional MRI of the primary visual cortex," *NeuroImage*, vol. 227, Feb. 2021, Art. no. 117631.

[9] H. Yu, Y. Lan, Z. Yuan, J. Xu, and H. Lee, "Phase unwrapping in InSAR: A review," *IEEE Geosci. Remote Sens. Mag.*, vol. 7, no. 1, pp. 40–58, Mar. 2019.

[10] S. Zhang, X. Li, and S.-T. Yau, "Multilevel quality-guided phase unwrapping algorithm for real-time three-dimensional shape reconstruction," *Appl. Opt.*, vol. 46, no. 1, pp. 50–57, Jan. 2007.

[11] S. S. Gorthi and P. Rastogi, "Fringe projection techniques: Whither we are?" *Opt. Lasers Eng.*, vol. 48, no. 2, pp. 133–140, Feb. 2010.

[12] M. Takeda, H. Ina, and S. Kobayashi, "Fourier-transform method of fringe-pattern analysis for computer-based topography and interferometry," *J. Opt. Soc. Amer.*, vol. 72, no. 1, pp. 156–160, 1982.

[13] W. W. Macy, "Two-dimensional fringe-pattern analysis," *Appl. Opt.*, vol. 22, no. 23, p. 3898, Dec. 1983.

[14] C. Roddier and F. Roddier, "Interferogram analysis using Fourier transform techniques," *Appl. Opt.*, vol. 26, no. 9, p. 1668, May 1987.

[15] K. Itoh, "Analysis of the phase unwrapping algorithm," *Appl. Opt.*, vol. 21, no. 14, p. 2470, 1982.

[16] D. C. Ghiglia and M. D. Pritt, *Two-dimensional Phase Unwrapping: Theory, Algorithms, and Software*, 1st ed., Hoboken, NJ, USA: Wiley, 1998.

[17] R. Cusack, J. M. Huntley, and H. T. Goldrein, "Improved noise-immune phase-unwrapping algorithm," *Appl. Opt.*, vol. 34, no. 5, p. 781, 1995.

[18] R. M. Goldstein and C. L. Werner, "Radar interferogram filtering for geophysical applications," *Geophys. Res. Lett.*, vol. 25, no. 21, pp. 4035–4038, Nov. 1998.

[19] D. Zheng and F. Da, "A novel algorithm for branch cut phase unwrapping," *Opt. Lasers Eng.*, vol. 49, no. 5, pp. 609–617, May 2011.

[20] D. C. Ghiglia and L. A. Romero, "Robust two-dimensional weighted and unweighted phase unwrapping that uses fast transforms and iterative methods," *J. Opt. Soc. Amer. A, Opt. Image Sci.*, vol. 11, no. 1, pp. 107–117, Jan. 1994.

[21] X. Wang, S. Fang, and X. Zhu, "Weighted least-squares phase unwrapping algorithm based on a non-interfering image of an object," *Appl. Opt.*, vol. 56, no. 15, p. 4543, 2017.

[22] V. Katkovnik, J. Astola, and K. Egiazarian, "Phase local approximation *(PhaseLa)* technique for phase unwrap from noisy data," *IEEE Trans. Image Process.*, vol. 17, no. 6, pp. 833–846, Jun. 2008.

[23] S. S. Gorthi and P. Rastogi, "Piecewise polynomial phase approximation approach for the analysis of reconstructed interference fields in digital holographic interferometry," *J. Opt. A, Pure Appl. Opt.*, vol. 11, no. 6, Jun. 2009, Art. no. 065405.

[24] M. Servin, J. L. Marroquin, D. Malacara, and F. J. Cuevas, "Phase unwrapping with a regularized phase-tracking system," *Appl. Opt.*, vol. 37, no. 10, p. 1917, 1998.

[25] O. Loffeld, H. Nies, S. Knedlik, and W. Yu, "Phase unwrapping for SAR interferometry—A data fusion approach by Kalman filtering," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 1, pp. 47–58, 2007.

[26] X. Xie and Y. Lee, "Enhanced phase unwrapping algorithm based on unscented Kalman filter, enhanced phase gradient estimator, and path-following strategy," *Appl. Opt.*, vol. 53, no. 18, pp. 4049–4060, 2014.

[27] X. M. Xie and Q. N. Zeng, "Efficient and robust phase unwrapping algorithm based on unscented Kalman filter, the strategy of quantizing paths-guided map, and pixel classification strategy," *Appl. Opt.*, vol. 54, no. 31, p. 9294, 2015.

[28] Z. Cheng et al., "Practical phase unwrapping of interferometric fringes based on unscented Kalman filter technique," *Opt. Exp.*, vol. 23, no. 25, pp. 32337–32349, Feb. 2015.

[29] R. Kulkarni and P. Rastogi, "Phase unwrapping algorithm using polynomial phase approximation and linear Kalman filter," *Appl. Opt.*, vol. 57, no. 4, p. 702, 2018.

[30] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[31] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proc. IEEE*, vol. 92, no. 3, pp. 401–422, Mar. 2004.

[32] W. Schwartzkopf, T. E. Milner, J. Ghosh, B. L. Evans, and A. C. Bovik, "Two-dimensional phase unwrapping using neural networks," in *Proc. 4th IEEE Southwest Symp. Image Anal. Interpret.*, Apr. 2000, pp. 274–277.

[33] K. Wang, Y. Li, Q. Kemao, J. Di, and J. Zhao, "One-step robust deep learning phase unwrapping," *Opt. Exp.*, vol. 27, no. 10, pp. 15100–15115, 2019.

[34] W. Yin et al., "Temporal phase unwrapping using deep learning," *Sci. Rep.*, vol. 9, no. 1, p. 20175, Dec. 2019.

[35] T. Zhang et al., "Rapid and robust two-dimensional phase unwrapping via deep learning," *Opt. Exp.*, vol. 27, no. 16, pp. 23173–23185, 2019.

[36] Y. Qin, S. Wan, Y. Wan, J. Weng, W. Liu, and Q. Gong, "Direct and accurate phase unwrapping with deep neural network," *Appl. Opt.*, vol. 59, no. 24, p. 7258, Aug. 2020.

[37] K. Wang, Q. Kemao, J. Di, and J. Zhao, "Deep learning spatial phase unwrapping: A comparative review," *Adv. Photon. Nexus*, vol. 1, no. 1, Aug. 2022, Art. no. 014001.

[38] F. Yang, T.-A. Pham, N. Brandenberg, M. P. Lutolf, J. Ma, and M. Unser, "Robust phase unwrapping via deep image prior for quantitative phase imaging," *IEEE Trans. Image Process.*, vol. 30, pp. 7025–7037, 2021.

[39] M. V. Perera and A. De Silva, "A joint convolutional and spatial quad-directional LSTM network for phase unwrapping," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 4055–4059.

[40] C. W. Domier, W. A. Peebles, and N. C. Luhmann, "Millimeter-wave interferometer for measuring plasma electron density," *Rev. Scientific Instrum.*, vol. 59, no. 8, pp. 1588–1590, Aug. 1988.

[41] C. Thaury et al., "Probing electron acceleration and X-ray emission in laser-plasma accelerators," *Phys. Plasmas*, vol. 20, no. 6, Jun. 2013, Art. no. 063101.

[42] G. F. Swadling et al., "Diagnosing collisions of magnetized, high energy density plasma flows using a combination of collective Thomson scattering, Faraday rotation, and interferometry," *Rev. Scientific Instrum.*, vol. 85, no. 11, p. 11, Nov. 2014.

[43] S. V. Lebedev et al., "Laboratory astrophysics and collimated stellar outflows: The production of radiatively cooled hypersonic plasma jets," *Astrophysical J.*, vol. 564, no. 1, pp. 113–119, Jan. 2002.

[44] D. J. Ampleford et al., "Supersonic radiatively cooled rotating flows and jets in the laboratory," *Phys. Rev. Lett.*, vol. 100, no. 3, Jan. 2008, Art. no. 035001.

[45] H. R. Hasson et al., "Design of a 3-D printed experimental platform for studying the formation and magnetization of turbulent plasma jets," *IEEE Trans. Plasma Sci.*, vol. 48, no. 11, pp. 4056–4067, Nov. 2020.

[46] P.-A. Gourdain and C. E. Seyler, "Impact of the Hall effect on high-energy-density plasma jets," *Phys. Rev. Lett.*, vol. 110, no. 1, Jan. 2013, Art. no. 015002.

[47] L. N. Long and A. Gupta, "Scalable massively parallel artificial neural networks," *J. Aerosp. Comput., Inf., Commun.*, vol. 5, no. 1, pp. 3–15, Jan. 2008.

[48] R. Kulkarni and P. Rastogi, "Direct unwrapped phase estimation in phase shifting interferometry using Levenberg–Marquardt algorithm," *J. Opt.*, vol. 19, no. 1, Jan. 2017, Art. no. 015608.

[49] P. Gao, B. Yao, J. Han, L. Chen, Y. Wang, and M. Lei, "Phase and amplitude reconstruction from a single carrier-frequency interferogram without phase unwrapping," *Appl. Opt.*, vol. 47, no. 15, p. 2760, May 2008.

[50] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, no. 3, pp. 321–355, 1988.

[51] R. L. Hardy, "Multiquadric equations of topography and other irregular surfaces," *J. Geophys. Res.*, vol. 76, no. 8, pp. 1905–1915, Mar. 1971.

[52] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comput.*, vol. 3, no. 2, pp. 246–257, Jun. 1991.

[53] M.-S. Chen and M. T. Manry, "Power series analyses of back-propagation neural networks," in *Proc. Seattle Int. Joint Conf. Neural Netw. (IJCNN)*, 1991, pp. 295–300.

[54] M. T. Manry, H. Chandrasekaran, and C.-H. Hsieh, "Signal processing using the multilayer perceptron," in *Handbook of Neural Network Signal Processing*. Boca Raton, FL, USA: CRC Press, 2018, pp. 1–2.

[55] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[56] D. Casasent and E. Barnard, "Adaptive clustering neural net for piecewise nonlinear discriminant surfaces," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 1990, pp. 423–428.

[57] A. Sarajedini and R. Hecht-Nielson, "The best of both worlds: Casasent networks integrate multilayer perceptrons and radial basis functions," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 3, 1992, pp. 905–910.

[58] H. Wendland, "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree," *Adv. Comput. Math.*, vol. 4, no. 1, pp. 389–396, Dec. 1995.

[59] Z. Wang, J. Chen, and S. C. H. Hoi, "Deep learning for image super-resolution: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3365–3387, Oct. 2021.

[60] M. Bizhani, O. H. Ardakani, and E. Little, "Reconstructing high fidelity digital rock images using deep convolutional neural networks," *Sci. Rep.*, vol. 12, no. 1, pp. 1–14, Mar. 2022.

[61] R. Kulkarni and P. Rastogi, "Simultaneous unwrapping and low pass filtering of continuous phase maps based on autoregressive phase model and wrapped Kalman filtering," *Opt. Lasers Eng.*, vol. 124, Jan. 2020, Art. no. 105826.

[62] R. Datta and R. G. Regis, "A surrogate-assisted evolution strategy for constrained multi-objective optimization," *Expert Syst. Appl.*, vol. 57, pp. 270–284, Sep. 2016.

[63] J. Müller et al., "Surrogate optimization of deep neural networks for groundwater predictions," *J. Global Optim.*, vol. 81, no. 1, pp. 203–231, May 2020.

[64] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[65] L. R. Rere, M. I. Fanany, and A. M. Arymurthy, "Simulated annealing algorithm for deep learning," *Pro. Comput. Sci.*, vol. 72, pp. 137–144, Jan. 2015.

[66] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks," in *Proc. IEEE Swarm Intell. Symp. (SIS)*, Apr. 2003, pp. 110–117.

[67] M. Carvalho and T. Ludermir, "Particle swarm optimization of feed-forward neural networks with weight decay," in *Proc. 6th Int. Conf. Hybrid Intell. Syst. (HIS)*, Dec. 2006, p. 5.

[68] M. D. Buhmann, *Radial Basis Function Networks*. Boston, MA, USA: Springer, 2010, pp. 823–827.

[69] Y. Chen, B. Yang, and J. Dong, "Time-series prediction using a local linear wavelet neural network," *Neurocomputing*, vol. 69, nos. 4–6, pp. 449–465, Jan. 2006.

[70] V. Nekoukar and M. T. Hamidi Beheshti, "A local linear radial basis function neural network for financial time-series forecasting," *Int. J. Speech Technol.*, vol. 33, no. 3, pp. 352–356, Feb. 2009.

[71] D. J. MacKay, "Bayesian interpolation," *Neural Comput.*, vol. 4, no. 3, pp. 415–447, May 1992.

[72] E. Sariev and G. Germano, "Bayesian regularized artificial neural networks for the estimation of the probability of default," *Quant. Finance*, vol. 20, no. 2, pp. 311–328, Feb. 2020.

[73] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quart. J. Appl. Math.*, vol. 2, no. 2, pp. 164–168, Jul. 1944.

[74] P. E. Gill and W. Murray, "Algorithms for the solution of the non-linear least-squares problem," *SIAM J. Numer. Anal.*, vol. 15, no. 5, pp. 977–992, Oct. 1978.

[75] D. Gorinevsky, "An approach to parametric nonlinear least square optimization and application to task-level learning control," *IEEE Trans. Autom. Control*, vol. 42, no. 7, pp. 912–927, Jul. 1997.

[76] S. McLoone, M. D. Brown, G. Irwin, and A. Lightbody, "A hybrid linear/nonlinear training algorithm for feedforward neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 669–684, Jul. 1998.

[77] H. Peng, T. Ozaki, V. Haggan-Ozaki, and Y. Toyoda, "A parameter optimization method for radial basis function type models," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 432–438, Mar. 2003.

[78] T. Xie, H. Yu, J. Hewlett, P. Rozycki, and B. Wilamowski, "Fast and efficient second-order method for training radial basis function networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 4, pp. 609–619, Apr. 2012.

[79] B. M. Wilamowski and H. Yu, "Improved computation for Levenberg–Marquardt training," *IEEE Trans. Neural Netw.*, vol. 21, no. 6, pp. 930–937, Jun. 2010.

[80] E. W. Forgy, "Analysis of multivariate data: Efficiency vs interpretabilty of classifications," *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.

[81] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.

[82] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Jan. 1998.

[83] A. H. Ribeiro and L. A. Aguirre, "'Parallel training considered harmful?': Comparing series-parallel and parallel feedforward network training," *Neurocomputing*, vol. 316, pp. 222–231, Nov. 2018.

[84] S. Günther, L. Ruthotto, J. B. Schroder, E. C. Cyr, and N. R. Gauger, "Layer-parallel training of deep residual neural networks," *SIAM J. Math. Data Sci.*, vol. 2, no. 1, pp. 1–23, Jan. 2020.

[85] P. Berardino, G. Fornaro, R. Lanari, and E. Sansosti, "A new algorithm for surface deformation monitoring based on small baseline differential SAR interferograms," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 11, pp. 2375–2383, Nov. 2002.

[86] Q. Feng, H. Xu, Z. Wu, Y. You, W. Liu, and S. Ge, "Improved Goldstein interferogram filter based on local fringe frequency estimation," *Sensors*, vol. 16, no. 11, p. 1976, 2016.

[87] Q. Kemao, "Two-dimensional windowed Fourier transform for fringe pattern analysis: Principles, applications and implementations," *Opt. Lasers Eng.*, vol. 45, no. 2, pp. 304–317, Feb. 2007.

[88] Q. Kemao, W. Gao, and H. Wang, "Windowed Fourier-filtered and quality-guided phase-unwrapping algorithm," *Appl. Opt.*, vol. 47, no. 29, p. 5420, 2008.

[89] J. C. Estrada, J. L. Marroquin, and O. M. Medina, "Reconstruction of local frequencies for recovering the unwrapped phase in optical interferometry," *Sci. Rep.*, vol. 7, no. 1, pp. 1–10, Jul. 2017.

[90] R. M. Goldstein, H. A. Zebker, and C. L. Werner, "Satellite radar interferometry: Two-dimensional phase unwrapping," *Radio Sci.*, vol. 23, no. 4, pp. 713–720, Jul. 1988.

[91] T. J. Flynn, "Two-dimensional phase unwrapping with minimum weighted discontinuity," *J. Opt. Soc. America A*, vol. 14, no. 10, pp. 2692–2701, Oct. 1997.

[92] M. Costantini, "A novel phase unwrapping method based on network programming," *IEEE Trans. Geosci. Remote Sens.*, vol. 36, no. 3, pp. 813–821, May 1998.

[93] G. Fornaro, A. Pauciullo, and E. Sansosti, "Phase difference-based multichannel phase unwrapping," *IEEE Trans. Image Process.*, vol. 14, no. 7, pp. 960–972, Jul. 2005.

[94] H. Yu, Z. Li, and Z. Bao, "Residues cluster-based segmentation and outlier-detection method for large-scale phase unwrapping," *IEEE Trans. Image Process.*, vol. 20, no. 10, pp. 2865–2875, Oct. 2011.

[95] B. Tayebi, F. Sharif, and J.-H. Han, "Smart filtering of phase residues in noisy wrapped holograms," *Sci. Rep.*, vol. 10, no. 1, Oct. 2020, Art. no. 16965.

[96] P.-A. Gourdain et al., "The impact of Hall physics on magnetized high energy density plasma jets," *Phys. Plasmas*, vol. 21, no. 5, May 2014, Art. no. 056307.

[97] R. P. Drake, *Introduction To High-energy-density Physics*. Cham, Switzerland: Springer, 2018.

[98] I. H. Hutchinson, *Principles of Plasma Diagnostics*, 2nd ed., Cambridge, U.K.: Cambridge Univ. Press, 2002.

[99] J. B. Greenly, J. D. Douglas, D. A. Hammer, B. R. Kusse, S. C. Glidden, and H. D. Sanders, "A 1MA, variable risetime pulse generator for high energy density plasma research," *Rev. Scientific Instrum.*, vol. 79, no. 7, Jul. 2008, Art. no. 073501.

[100] X.-F. Li, L. Huang, and Y. Huang, "A new Abel inversion by means of the integrals of an input function with noise," *J. Phys. A, Math. Theor.*, vol. 40, no. 2, p. 347, Feb. 2006.