POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Evolutional Deep Neural Networks for Partial Differential Equations

Laurea Magistrale in Mathematical Engineering - Ingegneria Matematica

Author: Luca Paparo

Advisor: Prof. Andrea Manzoni, Prof. Taraneh Sayadi

Co-advisor: Prof. Vincent Le Chenadec, Prof. Peter Schmid

Academic year: 2022-2023

1. Introduction

Partial Differential Equations (PDEs), represent a fundamental aspect of physical modeling. Many problems, ranging from the diffusion of the heat in a room, to the variation in prices in the stock market can be modeled as PDE problems. The downside of those types of problems, however, is that they become quite complex really fast, thus we can rarely expect a solution in closed form to exist. A big discussion on how to reliably, accurately and efficiently obtain a solution for those problems has been, in the last decades (and still is) under the spotlight: many different methods (such as the Finite Element Method (FEM), Spectral Methods, Finite Volumes Method and many others) have been proposed, developed and applied. While those "classical" methods have shown to be reliable and fast in a wide range of problems, they still suffer from several limitations. Because of the blooming of scientific machine learning in the last decade, many neural network based methods have been proposed for the numerical approximation of PDEs; we will refer to them as "Machine Learning Methods". Some of those methods have demonstrated to outperform classical methods in terms of efficiency and accuracy for some problems. For example, Physics Informed Neural Networks (PINNs) outperform classical methods in some situations, where some noisy solution data is available, or in condition of uncertainty regarding equation's parameters (see [6]). Other methods, such as Fourier Neural Operators [5], can accurately solve PDE problems for whole sets of initial conditions in simple geometries, and if a significant amount of information regarding the exact solution is available. In this thesis, we will discuss the Evolutional Deep Neural Network (EDNN) method [4], a machine learning based method for solving PDEs. We will delve into its advantages and limitations with respect to both classic and machine learning methods. Moreover, we will address some of the flaws of the method, proposing possible solutions.

2. The EDNN Method

Let us consider a non-linear partial differential equation in the form:

$$\frac{\partial \boldsymbol{u}}{\partial t} - \mathcal{N}_x(\boldsymbol{u}) = 0, \quad \boldsymbol{x} \in \Omega \subset \mathbb{R}^d, \tag{1}$$

where $d \in \mathbb{N} \setminus \{0\}$ and $\mathcal{N}_x(\cdot)$ is a non-linear operator, while $\boldsymbol{u}(\boldsymbol{x}) \in \mathbb{R}^o$, $o \in \mathbb{N} \setminus \{0\}$. Let us consider the following mapping carried out by a

neural network:

$$\boldsymbol{u}_h : \quad \Omega \times W \to \mathbb{R}^o,$$

where W is the set of admissible parameters of the network. Suppose that the parameters of the network depend directly on time, then, $\forall t > 0$:

$$\boldsymbol{u}_h = \boldsymbol{u}_h(\boldsymbol{x}, \mathcal{W}(t)), \quad \boldsymbol{x} \in \Omega, \ \mathcal{W}(t) \in W.$$

The initial set of parameters have to approximate as well as possible the initial condition $\boldsymbol{u}_0 : \Omega \to \mathbb{R}^o$. This is easily achievable through commonly used optimizers such as stochastic gradient descent, or Adam. To advance in time the solution, it is necessary to discretize the time interval $[0, T]$, where we want to integrate the PDE, into a finite series of $N + 1$ times:

$$0 = t_0 < t_1 < \cdots < t_N = T.$$

Advancing the solution from time t_n to time t_{n+1} means finding a suitable set of parameters $\mathcal{W}(t_{n+1})$ such that $\boldsymbol{u}_h(\cdot, \mathcal{W}(t_{n+1})) \approx \boldsymbol{u}(\cdot, t_{n+1})$. To do that, we define and look for a minimizer of the following functional:

$$\mathcal{J}(\gamma) = \frac{1}{2} \int_\Omega \left\| \frac{\partial \boldsymbol{u}_h}{\partial \mathcal{W}} \gamma - \mathcal{N}_x(\boldsymbol{u}_h) \right\|_2^2 d\boldsymbol{x}, \tag{2}$$

where the first term in the norm comes from the chain rule $\partial \boldsymbol{u}_h / \partial t = (\partial \boldsymbol{u}_h / \partial \mathcal{W}) * (\partial \mathcal{W} / \partial t)$. Thus, the minimizer of the functional $\mathcal{J}$ is the $\partial \mathcal{W} / \partial t$ that minimizes the L^2 error of the equation. By selecting a set of M sample points $\{\boldsymbol{x}_k\}_{k=1}^M \subset \Omega$, the optimization problem can be discretized to:

$$\begin{aligned} \boldsymbol{J}^T \boldsymbol{J} \hat{\gamma}_{opt} &= \boldsymbol{J}^T \boldsymbol{N} \\ [\boldsymbol{J}]_{ij} = \frac{\partial \boldsymbol{u}_h^i}{\partial \mathcal{W}_j}, &\qquad [\boldsymbol{N}]_i = \mathcal{N}_x(\boldsymbol{u}_h^i), \end{aligned} \tag{3}$$

where $\boldsymbol{u}_h^i := \boldsymbol{u}_h(\boldsymbol{x}^i)$ and $\hat{\gamma}_{opt} \approx \gamma_{opt}$, the minimizer of Eq. (2). For further details, we refer to [4]. Once $\dot{\mathcal{W}} \approx \hat{\gamma}_{opt}$ is computed, it is possible to march from the current time t_n to t_{n+1} using a generic explicit method. This is the main principle of the EDNN method.

2.1. Embedded Boundary Conditions

In the EDNN framework, boundary conditions are embedded in the framework, thus, showing machine precision at the boundaries. On the other hand, only periodic and Dirichlet boundary conditions are treatable in this way.
The periodic boundary conditions are implemented by transforming the initial coordinates via a periodic function $\boldsymbol{f}$ in $\bar{\Omega}$, bijective in Ω. Thus, the output of the network $\boldsymbol{u}_h(\boldsymbol{f}(\boldsymbol{x}))$ is periodic by construction.
The Dirichlet boundary conditions are enforced by applying a function $\mathcal{G}$ to the network's output $\boldsymbol{v}$, so that $\boldsymbol{u}_h(\boldsymbol{x}) := \mathcal{G}(\boldsymbol{v}(\boldsymbol{x}), \boldsymbol{x})$ respects the Dirichlet conditions. In a compact interval $I \subset \mathbb{R}$, for homogeneous Dirichlet conditions, a possible choice of $\mathcal{G}$ would be:

$$\mathcal{G}(\boldsymbol{v}(\boldsymbol{x}), \boldsymbol{x}) = \boldsymbol{v}(\boldsymbol{x}) + c_e(\boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x}_e) + c_w(\boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x}_w),$$

where $\boldsymbol{x}_e$ and $\boldsymbol{x}_w$ correspond to the boundary points and $c_e(\cdot)$ and $c_w(\cdot)$ are respectively -1 and 0 when computed in $\boldsymbol{x}_e$ and 0 and -1 when computed in $\boldsymbol{x}_w$. This can be easily generalized in two or more dimensions.

2.2. Advantages and Limitations

The EDNN method allows to accurately compute the solution in a broad range of test cases and problems. The method shows clear advantages when compared to some of the most diffused machine learning methods for PDEs. By construction, opposite to PINNs, the method can solve seeminglessly for longer periods of time. Fourier Neural Operators, on the other hand, require a large amount of well structured data from exact solutions, something not required by the EDNN method, which only requires the initial and boundary values of the problem. Finally, the method manages to solve efficiently high dimensional problems [3], unreachable by classical methods due to the curse of dimensionality.
On the other hand, EDNNs still present big limitations, that can be grouped in two categories:

- efficiency problems: in lower dimension, the integration of equations requires minutes, rather than seconds as classical method;
- boundary problems: it is possible to enforce only Dirichlet and periodic boundary conditions and, even in those two cases, they are applicable only in trivial geometries.

2.3. Our Contributions

This thesis presents two possible approaches to tackle the limitations the method has. First

of all, a way to implement a broader range of boundary conditions is addressed. This will be done by weakly enforcing the invariance of the restriction of the function at the border. In this way, with small implementation differences, Neumann, Robin and many other boundary conditions can be, with almost no effort, introduced in the framework. Moreover, no particular constraint on the geometry of the problem is necessary.
Furthermore, we propose an approach to decrease the computational time of the time integration. We will show that this can be done by minimizing the equation error (see Eq. (2)) by transforming first the represented solution in a spectral space, and then by computing this error using the transformed equation. By choosing an orthogonal basis in the spectral space, we can parallelize the evolution considering more than one network, speeding up the advancement of the solution.
In Section 3 our new approach to boundary conditions is presented and discussed. In Section 4 we will discuss the EDNN method with PseudoSpectral marching.

3. Weak Imposition of Boundary Conditions

As previously pointed out, the EDNN framework has more than one major limitation regarding the boundary conditions. Besides the scarse choice of currently available boundary conditions embeddable in the network, it is only possible to solve for simple geometries. In this Section, starting from the idea of invariances (proposed in [1]), we will discuss a way to impose generic boundary conditions in generic domains. We will display two test cases:

- an advection-reaction-diffusion equation with Dirichlet boundary conditions. By solving them in an annulus, we will demonstrate the ability of the approach to well behave also in non trivial geometries;
- the homogeneous heat equation solved in 4, 5 and 7 dimensions with Neumann boundary conditions, showing that the method is able to solve for non-Dirichlet and non-periodic boundary conditions in a multidimensional setting.

3.1. Advection Reaction Diffusion in an Annulus

3.1.1 Problem and Reference Solution

let us consider the following problem:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \mu \Delta u + \boldsymbol{\beta} \cdot \nabla u + \sigma u, & \boldsymbol{x} \in \Omega,\ t > 0, \\ u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}), & \boldsymbol{x} \in \bar{\Omega}, \\ u(\boldsymbol{x}) = u_D(\boldsymbol{x}), & \boldsymbol{x} \in \partial\Omega,\ t > 0, \end{cases}$$

where $\mu = 0.05$, $\sigma = 1$ and $\beta = [-0.5, 0]$. The domain Ω is the circular ring defined as:

$$\Omega = \{(x, y) \in \mathbb{R}^2 :\ x^2 + y^2 < a \text{ et } x^2 + y^2 > b\},$$

where the external and internal radii, are $a = 1$ and $b = 0.3$. Initial and boundary functions are respectively:

$$u_0(\boldsymbol{x}) = 4(r-1)(0.3-r)), \quad r = ||\boldsymbol{x}||,\ \forall \boldsymbol{x} \in \bar{\Omega};$$
$$u_D(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega.$$

We solve the problem until the final time $T = 2$ and compare our results with the FEM solution obtained on a fine mesh counting more than 14000 nodes, selecting a time step of $\Delta t = 0.001$.

3.1.2 The EDNN Approach

Since the geometry is non-simply-connected, it is impossible to implement the boundary conditions as described in Section 2. Instead, it is possible to define the optimization problem in the following way, to include invariant quantities [1]:

$$\mathcal{W}(t) = \arg\min_{\gamma \in \Theta} \mathcal{J}(\gamma)$$
$$s.t. \quad I_k(\mathcal{W}(t)) = I_{k,0}, \qquad k = 1, 2, \dots, \quad t \geq 0,$$

where $\mathcal{J}$ is the functional defined in Eq. (2), Θ is the space of admissible parameters. Finally, I_k and $I_{k,0}$, for $k = 1, 2, \dots$, are respectively a function of the parameters and its initial value. Since any function of the solution (and its derivatives) at each point is itself a function of the parameters, we can choose the solution computed in one boundary point as an invariant. Thus, by picking a set of representative points on the boundaries and imposing there the invariance condition, it is possible to keep the solution constant at the borders. In this case, we select

equally spaced points on the internal and external boundaries, $\{x_{int,i}\}_{i=1}^{N_{int}}$ and $\{x_{ext,i}\}_{i=1}^{N_{ext}}$, where $N_{int}, N_{ext} \in \mathbb{N}$ are proportional to the internal and external radius, respectively. Thus, we impose the following invariants:

$$I_k(\mathcal{W}(t)) = u_h(\mathcal{W}(t), x_{int,k})$$
$$I_j(\mathcal{W}(t)) = u_h(\mathcal{W}(t), x_{ext,k}),$$

$\forall k \in \{1, \dots, N_{int}\}$, $\forall j \in \{N_{int}+1, \dots, N_{int}+N_{ext}\}$.
it is possible to incorporate the invariance condition in the stepping of the solution by, first of all, considering the problem in Eq. (3) in "least squares form", i.e. as

$$\hat{\gamma}_{opt} := \arg\min_{\gamma \in \Theta} ||\boldsymbol{J}\gamma - \boldsymbol{N}||_2^2,$$

so as the solution in a least squares sense of the problem $\boldsymbol{J}\hat{\gamma}_{opt} = \boldsymbol{N}$. The invariance constraint can be weakly enforced, by advancing the solution through the following least squares problem [2] (for $t > 0$):

$$\begin{bmatrix} \boldsymbol{J}(\mathcal{W}(t)) \\ \nabla \boldsymbol{I}^T(\mathcal{W}(t)) \end{bmatrix} \hat{\gamma}_{opt} = \begin{bmatrix} \boldsymbol{N}(\mathcal{W}(t)) \\ \boldsymbol{0} \end{bmatrix},$$

where $\boldsymbol{I} := [I_1, \dots, I_{N_{ext}+N_{int}}]$.

3.1.3 Numerical Results

For this test, we consider a feedforward neural network, with three hidden layers of 25 neurons each. The activation function is the hyperbolic tangent. We consider a time step $\Delta t = 0.01$, with a 4^{th} order Runge-Kutta explicit marching scheme. In Figure 1 we showcase the comparison between EDNN and reference solution. The EDNN method with the new implementation of the boundary conditions permits to accurately integrate the solution. Throughout the simulation, the maximum relative error stays below 2.39%, compared to the FEM solution.

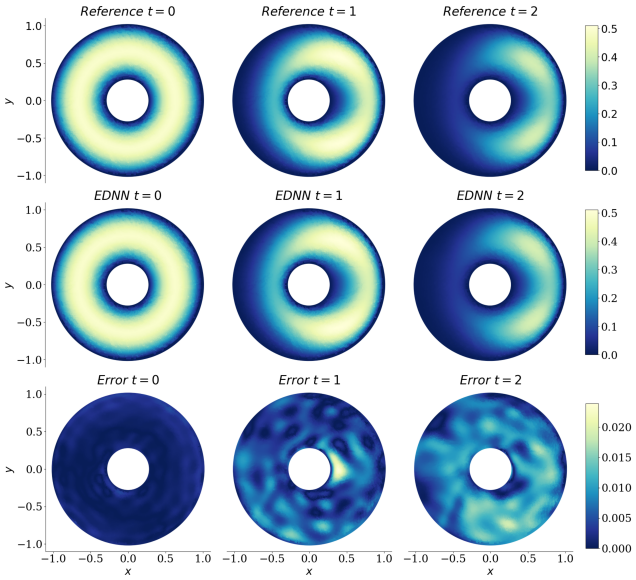

Figure 1: On each row (from top to bottom) reference solution, EDNN solution and relative error between the two at (from left to right) $t = 0$, $t = 1$ and $t = 2$.

3.2. Multidimensional Heat Equation

The more general selection of possible domains is not the only advantage this approach provides. Since it is possible to impose the invariance constraint to every function of the parameter, we can easily generalize this approach to the most common boundary conditions. We take as an example the homogeneous Neumann boundary conditions. Although constraint of this kind are necessary in many cases, no way to impose them has been yet proposed. In the following, we will show that our approach is viable to impose those conditions also in multidimensional settings.
let us consider the following homogeneous Heat Equation in $N \in \mathbb{N} \setminus \{0\}$ dimensions:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \Delta u, & \boldsymbol{x} \in \Omega_N := \left(-\dfrac{1}{2}, \dfrac{1}{2}\right)^N, t > 0 \\ u(\boldsymbol{x}, 0) = \dfrac{1}{N}\displaystyle\sum_{i=0}^{N}(4x_i^3 - 3x_i + 1), & \boldsymbol{x} \in \Omega_N \\ \dfrac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x}, t) = 0, & \forall \boldsymbol{x} \in \partial\Omega_N,\ t > 0, \end{cases}$$

with $N = 4, 5, 7$. The same approach taken in Section 3.1 would fail in this case for two reasons:

- computing the parameter's derivative as $\boldsymbol{J}\gamma = \boldsymbol{N}$ would require too many discretization points;
- it is infeasible to take a representative set

of points at the boundary and imposing the invariance condition for each of them.

Instead, we solve the least squares problem

$$\begin{bmatrix} \boldsymbol{J}^T\boldsymbol{J}(\mathcal{W}(t)) \\ \nabla \boldsymbol{I}^T(\mathcal{W}(t)) \end{bmatrix} \hat{\gamma}_{opt} = \begin{bmatrix} \boldsymbol{J}^T\boldsymbol{N}(\mathcal{W}(t)) \\ \mathbf{0} \end{bmatrix}.$$

In this scenario, the solution's quality, apart from boundary conditions, hinges on the precise approximation of two integrals, as detailed in [3, 4]. The integrals approximation is done using the basic Monte Carlo method, resilient to increased dimensions. Furthermore, we choose the L^2 norm of the normal derivatives at the boundaries as a singular invariant.

For a comparative analysis, we juxtapose the EDNN solutions against approximations of the exact solutions. This comparison involves accounting for $N_{m,4} = 24^4$ and $N_{m,5} = 16^5$ modes in the heat equation's solution for 4 and 5 spatial dimensions, respectively. These are integrated using 35^4 and 16^5 Gauss-Legendre points for each dimension. When extending to 7 spatial dimensions, approximating the exact solution becomes virtually impossible. Nonetheless, it can be easily shown that the integral of the solution remains constant, while the solution itself converges to a constant value $u_{\infty,7} \equiv I/|\Omega_7|$. Here, I represents the integral of the solution over the domain, and $|\Omega_7| = 1$ denotes the measure of the domain.

3.3. Numerical Results

In our experiments, we employed a consistent network configuration across all three cases. Specifically, we utilized a feedforward network comprising three hidden layers, each containing 25 neurons. The chosen activation function was the hyperbolic tangent. For temporal discretization, we set the time step $\Delta t = 0.01$ and adopted an explicit Runge-Kutta method of 4^{th} order for time stepping. The L^2 error of the invariant at the boundary was computed using Gauss-Legendre integration. The integration employed 15^3, 10^4, and 5^6 points for the 3, 4, and 6-dimensional cases, respectively.

Figure 2 illustrates (projecting on the first two dimensions) the results at time $T = 1$ for the 4D and 5D scenarios. In these dimensions, the final pointwise relative errors were approximately 5.68% and 8.25%, respectively. This approach to handle boundary conditions demonstrates the capability to accurately compute the solution of the problem, even in higher-dimensional spaces. Furthermore, the significance of imposing boundary conditions is highlighted. In Figure 3, we present a comparison of the mean absolute error at the boundary under two different scenarios. The constrained version of our model successfully represents the solution at the boundary with good fidelity throughout its evolution.

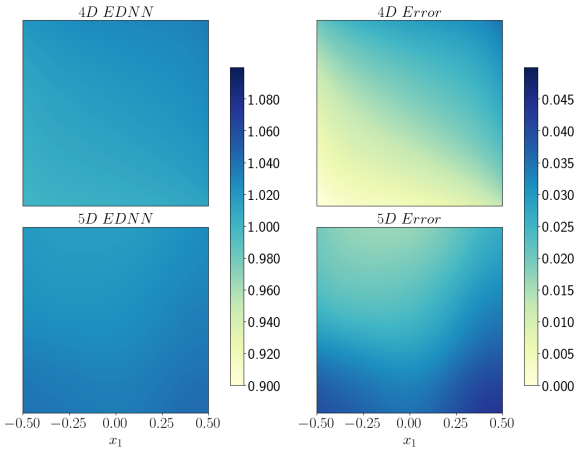

Figure 2: EDNN solution and error at $T = 1$ in 4D and 5D.

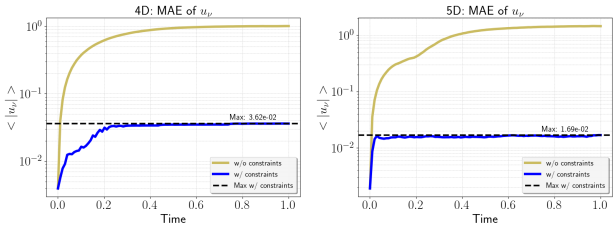

Figure 3: Mean absolute error at the boundaries with and without constraints in 4D and 5D.

The final section of our analysis presents the mean and variance of the solution in a seven-dimensional spatial domain. Notably, the mean is equivalent to the integral, given that the measure of the domain is 1. Additionally, we compare these findings with those from the non-constrained case. Although a direct comparison of the solution to a reference standard is unfeasible, precluding definitive arguments regarding its pointwise accuracy, the results depicted in Figure 4 indicate that the weakly constrained solution effectively preserves the integral and successfully attains the correct steady state.

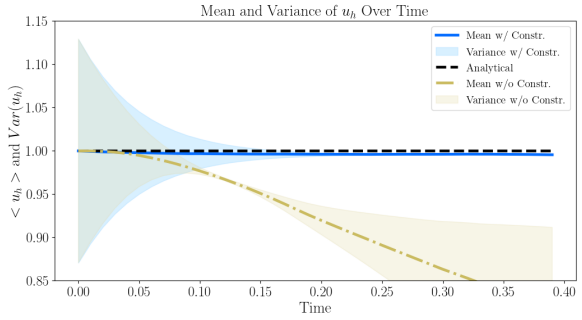

Figure 4: Mean and variance of the 7D solution with and without constraints. The black dashed line represents the exact value of the mean.

Our discussion has highlighted a new approach of addressing boundary conditions, enabling the resolution of higher-dimensional problems with non-Dirichlet and non-periodic boundaries. Traditional methods and popular machine learning techniques like PINNs and FNOs struggle with these problems due to the exponential rise in computational demands. However, the EDNN method makes solving these complex, higher-dimensional issues feasible. To illustrate this, Table 1 presents the average processing times and their standard deviations, aggregated over 10 separate runs. The simulations are run on an Intel Core i5-11300H processor running at 4.4GHz and using 8Gb of RAM.

	4D	**5D**	**7D**
Mean (s)	299.7	437.3	874.1
Standard Deviation (s)	1.418	3.274	5.156

Table 1: Mean computational time and standard deviation over 10 runs.

Furthermore, it is important to note that in all three scenarios, the solution is derived using between 1451 to 1526 parameters. This demonstrates the feasibility of computing accurate solutions with a relatively low number of degrees of freedom.

4. PseudoSpectral-EDNN

In this section, we address a critical limitation of the EDNN method: its high computational cost in low-dimensional settings. To mitigate this issue, we propose an approach that involves transforming the equations into spectral space, significantly reducing computation time. Notably, when a pseudospectral method is applicable, it offers substantial improvements in computational efficiency while maintaining the meshless structure of the method. We will further explore how this technique integrates with EDNN, and discuss additional non-trivial benefits that emerge from this integration.

4.1. The Methodology

Let us consider a transform $\mathcal{T}$ that maps a function space X into a finite function space V, $dim(V) = N \in \mathbb{N} \setminus \{0\}$. Moreover, let us suppose that $\mathcal{B} = \{v_k\}_{k=1}^N$, $v_k \in V$ forms a basis of V. Then, $\forall v \in V$, $\exists!\ \hat{\boldsymbol{\alpha}} = [\hat{\alpha}_1, \dots, \hat{\alpha}_N] \in \mathbb{R}^N$ such that:

$$v = \sum_{k=1}^{N} \hat{\alpha}_k v_k.$$

The choice of X depends on the activation functions of the network and its structure. The choice of V depends on the spectral method we select and on X. Then, calling $u_h(\cdot, \mathcal{W}(t))$ the output of the network:

$$\hat{\boldsymbol{\alpha}}(\mathcal{W}(t)) \leftrightarrow v(\mathcal{W}(t)) = \mathcal{T}(u_h(\mathcal{W}(t))).$$

Considering that $\mathbb{R}^N$ is isomorphic to V, we will use the abuse of notation $\mathcal{T}(u_h) = \hat{\boldsymbol{\alpha}}$ for simplicity. Then, the time advancement is regulated by the tranformed equation:

$$\frac{\partial \hat{\boldsymbol{\alpha}}}{\partial t} = \mathcal{T}(\mathcal{N}(u_h)),$$

which is a system of ODEs. At each time step, the functional presented in Eq. (2) is rewritten as:

$$\hat{\mathcal{J}}(\gamma, \mathcal{W}(t)) = \frac{1}{2} \left|\left| \frac{d\hat{\boldsymbol{\alpha}}(\mathcal{W}(t))}{d\mathcal{W}} \gamma - \mathcal{T}(\mathcal{N}(u_h)) \right|\right|^2.$$

Subsequently, this leads to a system of equations akin to Eq. (3). The efficiency in computational cost is enhanced due to two key factors:

1. The implementation of efficient spectral space transformations, such as the Fast Fourier Transform or the Fast Chebyshev Transform, circumvents the need for computationally intensive automatic differentiation to compute the right hand side of the equation at the collocation points.
2. The use of an orthogonal basis for $\mathcal{B}$ allows for the (almost) independent advancement of each α_k, enabling parallel processing and further improving efficiency.

To better explain the second point, let us consider two neural networks, denoted as $NN1$ and $NN2$. These networks are associated with $\hat{\boldsymbol{\alpha}}_1$ and $\hat{\boldsymbol{\alpha}}_2$, chosen to form a partition of $\hat{\boldsymbol{\alpha}}$. Furthermore, let $\mathcal{W}$ be defined as $\mathcal{W} := [\mathcal{W}_1, \mathcal{W}_2]$, where $\mathcal{W}_1$ and $\mathcal{W}_2$ represent the parameters of the first and second network, respectively. The Jacobian matrix, represented as $[\boldsymbol{J}]_{ij} = \frac{\partial \hat{\boldsymbol{\alpha}}_i}{\partial \mathcal{W}_j}$, takes the form of a block matrix:

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{J}_2 \end{bmatrix}, \text{ where } \boldsymbol{J}_1 = \frac{\partial \hat{\boldsymbol{\alpha}}_1}{\partial \mathcal{W}_1}, \ \boldsymbol{J}_2 = \frac{\partial \hat{\boldsymbol{\alpha}}_2}{\partial \mathcal{W}_2}.$$

Consequently, the construction of the Jacobian and the resolution of the linear systems can be independently executed by the two networks, which only need to communicate for assembling the right-hand side of the transformed equation.

4.2. A Test Case

We now consider the Allen-Cahn equation, which is a well-known model in reaction-diffusion systems. The equation is defined as:

$$\begin{cases} \dfrac{\partial u}{\partial t} = 0.0001 \dfrac{\partial^2 u}{\partial x^2} - 5u^3 + 5u, & x \in (-1,1), \quad t \in [0,1] \\ u(x,0) = x^2 \cos(\pi x), & x \in [-1,1] \\ u(t,-1) = u(t,1), & t \in [0,1] \\ \dfrac{\partial u}{\partial x}(t,-1) = \dfrac{\partial u}{\partial x}(t,1). & t \in [0,1] \end{cases}$$

To analyze its evolution over time, we adopt EDNNs with the Fourier pseudospectral method. This approach transforms the problem into Fourier space, where the equation is represented as:

$$\begin{aligned} \frac{\partial u_h}{\partial t} &= 0.0001 \frac{\partial^2 u_h}{\partial x^2} - 5u_h^3 + 5u_h \\ &\Downarrow \boldsymbol{\mathcal{F}}\{\cdot\} \\ \frac{\partial \hat{\boldsymbol{\alpha}}}{\partial t} &= -0.0001 \boldsymbol{k}^2 \hat{\boldsymbol{\alpha}} - 5\boldsymbol{\mathcal{F}}\{u_h^3\} + 5\hat{\boldsymbol{\alpha}}. \end{aligned} \tag{4}$$

In this context, $\boldsymbol{\mathcal{F}}\cdot$ denotes the discrete Fourier transform operator, and $\boldsymbol{k}$ represents the array of sampled frequencies. The transformation process begins with computing the output of the network across a set of nodes, where the quantity of nodes matches the number of modes in the system. This results in a vector that encapsulates the network's output. Subsequently, to transition into the Fourier domain, the fast Fourier transform (FFT) algorithm is applied to this vector. This application of FFT efficiently converts the data into a series of coefficients, represented by the vector $\hat{\boldsymbol{\alpha}}$. A separate FFT is applied to deal with the non-linear term. Finally, the advancement is done as previously described.

4.3. Numerical Results

In the following, we will consider a feedforward neural network featuring a hyperbolic tangent activation function and three hidden layers, each comprising 20 nodes. Our focus is on comparing the performance in terms of computational time and accuracy between one, two, and four networks employing the PseudoSpectral-EDNN (PS-EDNN) method. Each network configuration tackles the equation, which involves 400 modes, in a distinct manner: a single network processes all modes, while the two-network setup divides them evenly, with each handling 200 modes. In the four-network arrangement, each network is responsible for 100 modes. We benchmark these configurations against the classical EDNN method, ensuring a fair comparison by equating the size of the linear systems solved by both the EDNN and the one-network PS-EDNN. We consider this approach fair, since the linear system's dimension often represents the main computational challenge. For consistency, all simulations are conducted with identical initial training across different network setups. This uniformity is feasible as the network structures remain constant across all the simulations, differing only in how the update law of the parameters is computed.

The simulations are performed using a time step of $\Delta t = 0.001$ and are continued up to a final time of $T = 4$. The marching rule applied is an explicit 4th order Runge-Kutta scheme. Figure 5 illustrates the mean and standard deviations of simulation times per iteration across 5 runs. The simulations are run on a MacBook Pro with M2 processor and 16Gb of RAM. The implementation of the PS-EDNN single network scheme enhances processing speed. This improvement primarily stems from the integration of the FFT, which simplifies the assembly of the right hand side. This method proves more efficient than the computation of derivatives at the points of computation through automatic differentiation, as required in the standard EDNN framework. In scenarios involving multiple networks, the speedup is not linear, as expected.

Nonetheless, configurations with two and four networks demonstrate substantial time savings.

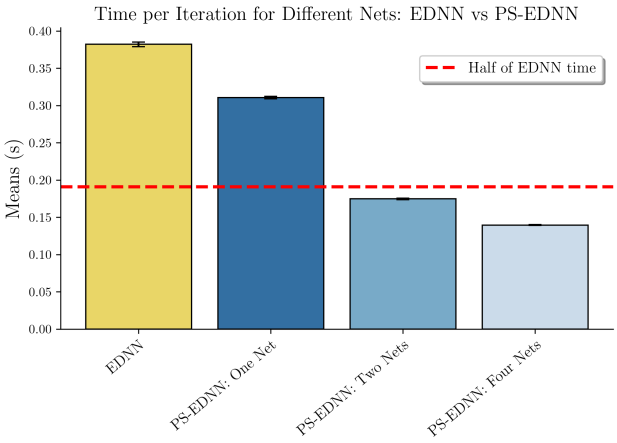

Figure 5: Computational time per iteration, computing mean and standard deviation over 5 runs.

Specifically, the two-network setup achieves a speedup of approximately 2.23× relative to the standard EDNN, and the four-network arrangement reaches about a 2.74× speedup. These enhancements are achieved without compromising the accuracy of the final results. The mean squared error for each case is consistently low, as detailed in Table 2.

	EDNN	**1-net**	**2-nets**	**4-nets**
MSE	9.25e-5	4.16e-5	1.18e-4	1.07e-4

Table 2: Mean Squared Error of the exact solution compared to the reference one at $T = 4$. From left two right: the EDNN, PS-EDNN and mulit-net PS-EDNN errors.

5. Conclusions

This report has provided an overview of the Evolutional Deep Neural Network (EDNN) method for solving PDEs. The EDNN method shows promise in accurately and efficiently solving complex, high-dimensional PDE problems that pose major challenges for traditional numerical methods. However, some key limitations were highlighted regarding efficiency in lower dimensions and inability to impose diverse boundary conditions in complex geometries. To address these issues, two main contributions were proposed and validated through numerical experiments:

1. A technique to weakly impose boundary conditions, enabling the imposition of Dirichlet, Neumann, Robin, and other boundary conditions in non-trivial domains.
2. An integration with pseudospectral methods to improve computational efficiency.

However, despite these improvements, the computational time and accuracy still do not match classical numerical methods. In conclusion, this work has summarized recent advancements that expand the applicability of EDNNs to broader classes of PDEs. Addressing limitations related to boundaries and efficiency represents an important progress toward positioning EDNN as a versatile technique for scientific computing. Additional efforts are required, particularly regarding exploring additional enhancements in accuracy and efficiency.

References

[1] W. Anderson and M. Farazmand. Evolution of nonlinear reduced-order solutions for pdes with conserved quantities. *J. Sci. Comput.*, 44(1):A176–A197, 2022.

[2] W. Anderson and M. Farazmand. Fast and scalable computation of shape-morphing nonlinear solutions with application to evolutional neural networks, 2023. arXiv:2207.13828 [math.DS].

[3] J. Bruna, B. Peherstorfer, and E. Vanden-Eijnden. Neural galerkin scheme with active learning for high-dimensional evolution equations, 2022. arXiv:2203.01360 [math.NA].

[4] Y. Du and T. Zaki. Evolutional deep neural network. *Physical Review E*, 104, 10 2021.

[5] Z. Li, N. Kovachki, K. Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. arXiv:2010.08895 [cs.LG].

[6] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comp. Phys.*, 378:686–707, 2019.