Parallel Solution of Nonlinear Projection Equations in a Multi-Task Learning Framework

Dawen Wu, Abdel Lisser

Parallel Solution of Nonlinear Projection Equations in a Multi-Task Learning Framework

Dawen Wu and Abdel Lisser

***Abstract*—Nonlinear projection equations (NPEs) provide a unified framework for addressing various constrained nonlinear optimization and engineering problems. However, when it comes to solving multiple NPEs, traditional numerical integration methods are not efficient enough. This is because traditional methods solve each NPE iteratively and independently. In this paper, we propose a novel approach based on Multi-Task Learning (MTL) for solving multiple NPEs. The solution procedure is outlined as follows. First, we model each NPE as a system of ordinary differential equations (ODEs) using neurodynamic optimization. Second, for each ODE system, we use a Physics-Informed Neural Network (PINN) as the solution. Third, we use a multi-branch MTL framework, where each branch corresponds to a PINN model. This allows us to solve multiple NPEs in parallel by training a single neural network model. Experimental results show that our approach has superior computational performance, especially when the number of NPEs to be solved is large.**

***Index Terms*—Nonlinear projection equations, Multi-task learning, Ordinary differential equations, Physics-informed neural networks, Error analysis**

I. Introduction

A nonlinear projection equation (NPE) consists of a nonlinear mapping function and a feasible set, and the objective is to find the fixed point of the NPE problem. As reported in the literature [1], [2], NPEs provide a unified framework for modeling a variety of nonlinear optimization problems, including complementarity problems, variational inequalities, and equilibrium point problems. Such problems have numerous real-world applications in fields such as economics, engineering, and computer science.

Typically, NPEs are addressed through neurodynamic optimization, which models the problem as a system of ordinary differential equations (ODEs) [3]–[5]. The constructed ODE system must exhibit the global convergence property, ensuring that the state solution of the system converges to the NPE solution as the time variable approaches infinity, irrespective of the initial point. Consequently, the NPE problem is transformed into solving the state solution of the ODE system. However, this ODE system is often highly nonlinear and lacks analytical solutions. As a result, numerical integration methods such as Runge-Kutta (RK) methods or backward differential formulation are frequently employed to solve for the state solution [6].

Motivation. Traditional methods for solving NPEs typically rely on neurodynamic approaches using numerical integration. While feasible, these methods are computationally inefficient for handling large or multiple instances of NPE problems. This inefficiency is particularly pronounced in applications such as energy markets and traffic management, where it is often necessary to solve multiple nonlinear optimization problems. It is worth noting that many of these nonlinear optimization problems can be reformulated as NPE problems [1]. For example, pricing and demand management in electricity markets can be formulated as many NPE problems, and solving them iteratively is time-consuming [7]. To overcome these challenges, we propose a Multi-Task Learning (MTL) framework. This framework solves these interrelated NPE problems in parallel, significantly reducing computational cost.

The authors are with Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des Signaux et Systèmes, 3 rue Joliot Curie, 91190 Gif-sur-Yvette, France (e-mail: dawen.wu@centralesupelec.fr; abdel.lisser@l2s.centralesupelec.fr).

A. Related works

Neurodynamic optimization. Over the past few decades, a variety of neurodynamic models have been developed to address diverse constrained optimization problems, encompassing linear and quadratic programming [8], [9], general convex programming [10]–[12], biconvex optimization [13], pseudo-convex optimization problems [14], time-varying optimization problems [15]–[17]. Specialized applications have also been explored, such as supervised feature selection through fractional programming [18], task assignment in multivehicle systems [19], and mitigated TDOA localization [20]. In particular, a projection neurodynamic model for solving NPEs was introduced and demonstrated global convergence to the exact solution under mild conditions [3]. This model also exhibited both asymptotic and exponential stability without requiring a smooth nonlinear mapping. To enhance performance, a bi-projection neurodynamic model was devised to efficiently solve quadratic optimization problems [4]. Moreover, a collaborative approach combining the projection neurodynamic model with particle swarm optimization was introduced for global optimization problems [5].

Physics-informed neural networks. Another research direction explored in this paper concerns the application of deep learning for solving differential equations. This concept was initially introduced in the 1990s, where neural networks were trained to minimize a loss function that incorporated both boundary conditions and differential equations [21]. Subsequent research demonstrated that network architecture could be specifically designed to satisfy boundary conditions [22], [23]. With the rise of deep learning, this approach has regained interest for solving high-dimensional nonlinear partial differential equations [24], [25]. A significant contribution to this field is the development of physics-informed neural networks (PINNs) [26], which integrate physical laws and data errors into the loss function. The flexible network structure and

efficient training algorithms of PINNs have led to numerous successes across a wide range of computational problems in physics and engineering [27]. This expanding body of work is propelled by a collocation strategy that adapts PINNs to exploit the structural properties of the target problem. Consequently, several variations of PINNs have been developed to address distinct problem scenarios [28], [29] and enhance computational performance [30]–[33]. To facilitate the use of deep learning for solving differential equations, several packages have been developed [34], [35].

Multi-task learning. Multi-task learning (MTL) is a machine learning approach that simultaneously optimizes multiple related tasks, improving generalization and performance by sharing information and leveraging commonalities among tasks [36]. MTL algorithms primarily focus on training a single model capable of solving various tasks, and this approach has been applied to diverse fields such as reinforcement learning, computer vision, natural language processing, and robotics [37]–[43]. Addressing the challenging learning problems in MTL involves various strategies. Architectural solutions include employing multiple modules or paths, attention-based architectures, or decomposing the problem into local problems corresponding to individual tasks, which are easier to learn [44]–[47]. These local models are then integrated into a single multi-task policy using diverse distillation techniques [48].

B. Contributions

The main contributions of this paper are as follows:

- We present an MTL framework designed to solve multiple NPE problems simultaneously, in contrast to traditional numerical methods that treat each NPE problem individually. The proposed MTL framework consists of two parts, the shared layers and the task-specific layers. The shared layers are responsible for processing the common features across multiple NPE problems, while the task-specific layers are designed to handle each individual NPE.
- We conduct an error analysis for the proposed neural network approach. Our study shows that the total error in our method consists of three components: Neurodynamic Error, Generalization Error, and Optimization Error. We investigate the factors that influence these errors, including the time range, the network architecture, and the model training. Finally, we show that even a single-layer neural network is capable of converging to the optimal solution of the NPE.
- We apply the proposed MTL framework to a variety of NPE scenarios. Experimental results show that our MTL framework exhibits significant computational performance benefits, especially when a large number of instances need to be solved or when the required solution accuracy is relatively lax. In particular, when solving for 100 NPE instances with a target error reduction down to 1, our MTL framework is 3 times faster than traditional numerical methods. Moreover, we observe that the computational time for our MTL framework grows more slowly than that of traditional numerical methods as the number of instances to solve increases. These experimental results validate the efficiency of the proposed MTL framework.

C. Outline

The remainder of this paper is organized as follows. Section II provides the necessary background, including an introduction to the NPE problem and how a neurodynamic optimization approach models it. Section III reformulates an NPE problem into a neural network training problem using PINNs, accompanied by an error analysis. Section IV introduces the proposed MTL framework and demonstrates its ability to solve multiple NPE problems in parallel. Section V presents experimental results and compares them with various numerical solvers. Finally, Section VI summarizes the key findings of this paper and outlines directions for future research.

II. Neurodynamic Approach for Modelling NPE

A. NPE

Definition 1 (Nonlinear projection equation)**.** *Consider a nonlinear mapping $G : \mathbb{R}^d \to \mathbb{R}^d$ and a feasible set $\Omega \subset \mathbb{R}^d$. The projection function $P_\Omega : \mathbb{R}^d \to \Omega$ maps a vector $z \in \mathbb{R}^d$ onto Ω, such that:*

$$P_\Omega(z) = \arg\min_{x\in\Omega} \|z - x\|, \tag{1}$$

where $\|\cdot\|$ denotes the Euclidean norm.

The NPE problem, denoted by $NPE(\Omega, G)$, is to find a vector $x^* \in \Omega$ satisfying:

$$P_\Omega(x^* - G(x^*)) = x^*. \tag{2}$$

Definition 2 (Nonlinear complementarity problem)**.** *Consider a nonlinear mapping $G : \mathbb{R}^d \to \mathbb{R}^d$. The nonlinear complementarity problem, denoted by $NCP(G)$, is to find a vector $x^* \in \mathbb{R}^d$ satisfying:*

$$G(x^*) \geq 0, \quad x^* \geq 0, \quad G(x^*)^T x^* = 0. \tag{3}$$

Definition 3 (Variational inequality)**.** *Consider a nonlinear mapping $G : \mathbb{R}^d \to \mathbb{R}^d$ and a feasible set $\Omega \subset \mathbb{R}^d$. The variational inequality problem, denoted by $VI(\Omega, G)$, is to find a vector $x^* \in \Omega$ satisfying:*

$$(x - x^*)^T G(x^*) \geq 0, \quad x \in \Omega. \tag{4}$$

Proposition 1 ([1])**.** *Let $\Omega \subset \mathbb{R}^d$ be a nonempty closed convex set. Then x^* solves the problem $NCP(G)$ if and only if x^* solves $NPE(\mathbb{R}^n_+, G)$, where $\mathbb{R}^d_+ = \{x \in \mathbb{R}^d | x \geq \mathbf{0}\}$ represents the set of non-negative real vectors.*

Proposition 2 ([1])**.** *Let $\Omega \subset \mathbb{R}^d$ be a nonempty closed convex set. Then x^* solves the problem $VI(\Omega, G)$ if and only if x^* solves $NPE(\Omega, G)$.*

According to the literature [1], NPEs can be viewed as a unified framework for many nonlinear optimization problems. For example, the Karush-Kuhn-Tucker (KKT) conditions for

linear and quadratic programming problems can be represented as linear complementarity problems, and the KKT conditions for convex constraint nonlinear optimization problems can be transformed into nonlinear complementarity problems [49]. Both are recast as NPEs according to Proposition 1. Many Nash equilibria in game theory can be represented by variational inequalities [50], [51], which are then reformulated as an NPE via Proposition 2.

B. Neurodynamic Approach

Assumption 1.

- *The function $G(\cdot)$ is locally Lipschitz continuous.*
- *The feasible set Ω is a box-constrained set, defined as $\Omega = \{x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \mid l_i \leq x_i \leq h_i\}$, where l_i and h_i denote the lower and upper bounds of x_i respectively. In this case, the projection function in Eq. (1) is reduced to $P_\Omega(x) = \left(P_\Omega^1(x_1), P_\Omega^2(x_2), \ldots, P_\Omega^d(x_d)\right)^T$, where $P_\Omega^i(x_i)$, $i \in \{1, 2, \ldots, d\}$ is defined as:*

$$P_\Omega^i(x_i) = \begin{cases} l_i & \text{if } x_i < l_i, \\ x_i & \text{if } l_i \leq x_i \leq h_i, \\ h_i & \text{if } x_i > h_i. \end{cases} \tag{5}$$

Consider a time-dependent function $y : \mathbb{R} \to \mathbb{R}^n$, where $y(t)$ denotes the state at time t. The objective of neurodynamic optimization is to design a first-order ODE system to govern $y(\cdot)$. In this paper, we utilize the projection neurodynamic model proposed by [3] to model the NPE, wherein the ODE system is defined as follows:

$$\frac{dy}{dt} = -G\left(P_\Omega(y)\right) + P_\Omega(y) - y. \tag{6}$$

To simplify the discussion, we define:

$$\Phi(y) = -G\left(P_\Omega(y)\right) + P_\Omega(y) - y. \tag{7}$$

Hence, the ODE system (6) can be expressed as $\frac{dy}{dt} = \Phi(y)$.

Definition 4 (State solution)**.** *Given an ODE system $\frac{dy}{dt} = \Phi(y)$, where $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ and an initial condition $(t_0 \in \mathbb{R}, y_0 \in \mathbb{R}^d)$, a vector-valued function $y : \mathbb{R} \to \mathbb{R}^d$ is deemed the state solution if it satisfies the ODE system $\frac{dy}{dt} = \Phi(y)$ and the initial condition $y(t_0) = y_0$. Given a time range $[t_0, T]$, $y(T)$ is denoted as the end state of the interval.*

Theorem 1 ([3])**.** *Consider an NPE problem, $NPE(\Omega, G)$, and let Assumption 1 hold. For any initial condition, (t_0, y_0), the state solution of the ODE system (6) converges to the optimal solution of $NPE(\Omega, G)$ as time t approaches infinity, i.e,*

$$\lim_{t \to \infty} y(t) = x^*, \tag{8}$$

where x^* is an optimal solution of $NPE(\Omega, G)$.

In particular, if $NPE(\Omega, G)$ contains only one optimal solution x^*, then the ODE system is globally asymptotically stable at x^*.

Initial value problem (IVP) construction. In practice, to solve the ODE system of Eq. (6) requires the construction of an IVP, which consists of the following three:

- The ODE system, which is given in Eq. (6).
- An initial condition (t_0, y_0), which is user-specified and represents the starting point of the state solution.
- A time range $[t_0, T]$, which is user-specified and represents the domain of the state solution to be solved.

After constructing the IVP, $y(t)$, $t \in [t_0, T]$ represents the state solution over the time range of $[t_0, T]$. According to Theorem 1, the end state, $y(T)$, serves as a predicted solution to the NPE, i.e., $y(T) \approx x^*$. The state solution is typically determined by a numerical method, such as the Runge-Kutta method. In the next section, we show how to use the neural network to solve the IVP, thus avoiding the use of numerical methods.

III. Reformulation of an NPE as a Neural Network Learning Problem

In this section, we illustrate the process of reformulating an NPE problem as a neural network training problem. Section III-A presents a modified PINN and shows how it solves for both the ODE system and the NPE. Section III-B presents the loss and objective functions used to train the neural network. Section III-C examines the error composition of the proposed method and discusses its convergence properties.

A. PINN Model

Model description. We use the following PINN model to solve the NPE problem,

$$\hat{y}(t; w) = y_0 + \left(1 - e^{-(t - t_0)}\right) N(t; w), \quad t \in [t_0, T], \tag{9}$$

where $N(t; w)$ denotes a fully connected neural network with trainable parameters w. The given time range is $[t_0, T]$. The auxiliary function $\left(1 - e^{-(t-t_0)}\right)$ ensures that the neural network always satisfies the initial condition $\hat{y}(t = t_0; w) = y_0$, irrespective of the model parameters w. Fig. 1(A) describes the PINN model.

Construction technique. The designed PINN model, as described by Eq. (9), employs the construction technique proposed by Lagaris et al. [22]. This technique aims to modify the output of the neural network $N(t; w)$ so that it inherently satisfies initial or boundary conditions, independent of the values of the trainable parameters w. This construction technique has also been used in other PINN studies [28], [52], [53]. In our work, we adapt this technique to the specific requirements of our problem setting, which considers only the initial conditions (t_0, y_0).

Approximate state solution to the ODE system. As shown in Fig. 1 (B), the proposed model itself approximates the state solution to the ODE system (6) over the time range $[t_0, T]$, that is,

$$\hat{y}(t; w) \approx y(t), \quad t \in [t_0, T], \tag{10}$$

where $y(\cdot)$ represents the true state solution of the ODE system. Although the input time t of the model $\hat{y}(t; w)$ can be any real number, we only regard $\hat{y}(t; w)$ as the solution of the ODE within the time range $[t_0, T]$. Thus, we constrain the input to $t \in [t_0, T]$.

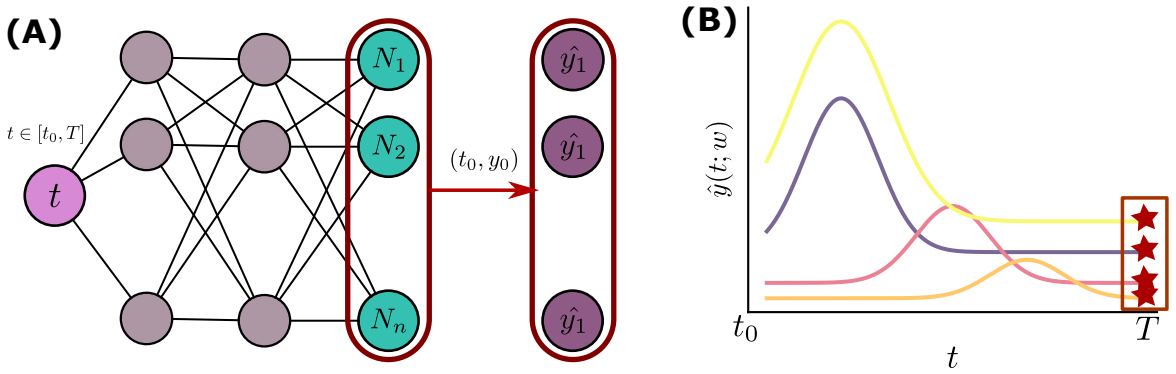

Fig. 1: **(A):** The PINN model, where $[N_1, N_2, \ldots, N_n]$ is the output of the neural network, and $[\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n]$ is the output of the model. **(B):** The PINN model as an approximate state solution to the ODE system, where the end state marked with red stars is the prediction to the NPE.

Predicted solution to the NPE. The end state of the proposed model, denoted by $\hat{y}(t = T; w)$, serves as the predicted solution to the NPE (2), as illustrated in Fig. 1 (B). The following equation demonstrates how $\hat{y}(t = T; w)$ approximates the optimal solution x^*:

$$\hat{y}(t = T; w) \approx y(T) \approx x^*, \tag{11}$$

where $\hat{y}(t = T; w) \approx y(T)$ indicates that the end state of our model approximates the true end state, and $y(T) \approx y^*$ is derived from Theorem 1, indicating that $y(T)$ resolves the NPE.

B. Training Objective

Loss function. The loss function for the proposed neural network is defined as:

$$\mathcal{L}(t, w) = \left\| \frac{\partial \hat{y}(t; w)}{\partial t} - \Phi(\hat{y}(t; w)) \right\|, \tag{12}$$

where $\Phi(\cdot)$ refers to the ODE system (6), which corresponds to the NPE to be solved. $\Phi(\hat{y}(t; w))$ represents the expected derivative according to the ODE system. $\frac{\partial \hat{y}(t;w)}{\partial t}$ denotes the actual derivative of the model, which can be computed using automatic differentiation tools such as PyTorch or JAX [54], [55]. $\mathcal{L}(t, w)$ expresses the difference between the two at time t and with network parameters w.

Incorporating the NPE into the loss function. First, the NPE is reformulated as an ODE system using neurodynamic optimization. This ODE system is then integrated into the loss computation process. Initially, a neural network is a versatile framework without a specific aim to solve a particular NPE. By incorporating the reformulated NPE as an ODE system into the loss function, the neural network is trained to solve both the ODE system and the NPE.

Objective function. Assuming a fixed neural network architecture, the objective function is defined as:

$$J(w) = \int_{t_0}^{T} \left\| \frac{\partial \hat{y}(t; w)}{\partial t} - \Phi(\hat{y}(t; w)) \right\| dt, \tag{13}$$

which is the integral of the loss function over the time range $[t_0, T]$. The loss value $\mathcal{L}(t, w)$ represents the error of the model at time t, while the objective function $J(w)$ represents the total error of the model over the time range $[t_0, T]$.

Batch loss. However, the objective function $J(w)$ is computationally intractable due to its integral component. Hence, in practice, the model is trained by minimizing the following batch loss:

$$\mathcal{L}(\mathbb{T}, w) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \mathcal{L}(t, w), \tag{14}$$

where $\mathbb{T}$ is a set of randomly sampled times from the interval $[t_0, T]$, and $|\mathbb{T}|$ denotes the size of the set. In this manner, the integral in the objective function $J(w)$ can be approximated by a sum of loss values over the set of sampled times. By minimizing the batch loss, the model can be effectively trained to solve the NPE.

C. Error Analysis

Introduction. In this subsection, we analyze the prediction error of the PINN solution, denoted by $\hat{y}(T; w)$, as given in Eq. (11). We discuss the various components that contribute to this error. We then examine how the choice of network architecture and time range $[t_0, T]$, two important hyperparameters, affect the error composition. Finally, we show that there exists a neural network that can accurately solve the target NPE problem, supported by universal approximation theorems of neural networks and the global convergence theorem of neurodynamic optimization.

Notations Setup. Fig. 2-(A) illustrates the error decomposition for the PINN solution to an NPE problem. The mathematical notations used in the figure are explained below:

- $\mathcal{H}$ denotes a network architecture, represented by a set of neural networks with the same architecture. For example, for an architecture with a single hidden layer consisting of 100 neurons, $\mathcal{H}$ contains all the neural networks under that particular architecture.
- x^* is the optimal solution for the NPE being solved.
- $y(T)$ represents the end state of the true state solution over the time range $[t_0, T]$.
- $\hat{y}(T; w)$ is the practical PINN solution obtained, for example, after 1000 training iterations.

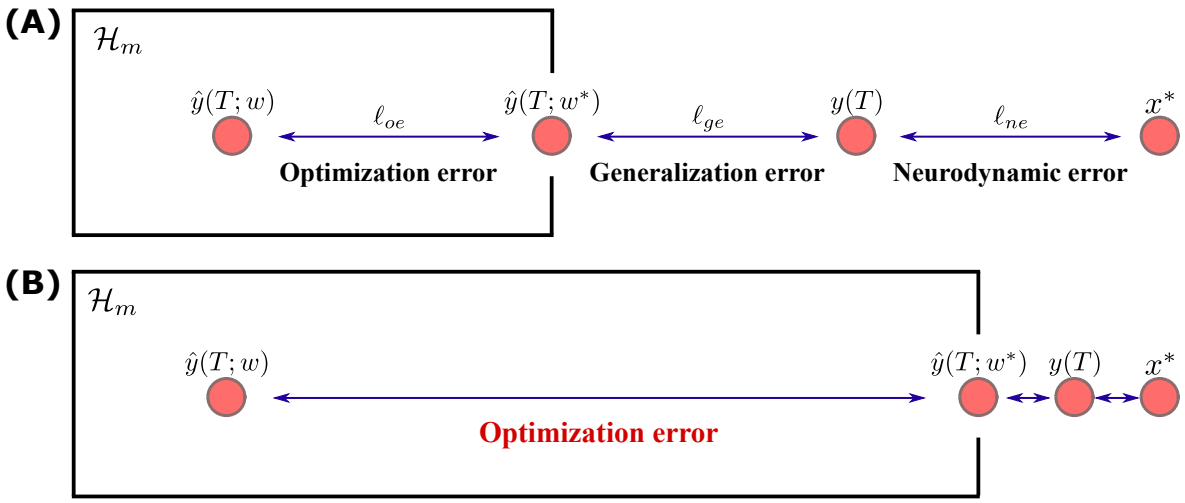

Fig. 2: **(A)** Error decomposition showcasing optimization, generalization, and neurodynamic errors. **(B)** The extreme case of infinitely large network architecture and time range.

- $\hat{y}(T;w^*)$ is the best achievable solution within the architecture $\mathcal{H}$, where w^* is the minimizer of the objective function (13).

Error Decomposition. We define the total error between the predicted and the optimal solution as

$$\ell_{\text{total}} = |\hat{y}(T;w) - x^*|. \tag{15}$$

As shown in Fig. 2-(A), the total error ℓ_{total} consists of the following three components:

$$\ell_{\text{total}} = \ell_{\text{oe}} + \ell_{\text{ge}} + \ell_{\text{ne}}. \tag{16}$$

Below, we explain each of the error components.

- *Optimization Error* is defined as $\ell_{\text{oe}} = |\hat{y}(T;w) - \hat{y}(T;w^*)|$. The optimization error captures the discrepancy between the actual PINN solution and the optimal solution achievable under the same neural network architecture $\mathcal{H}$. This error is due to the inherent limitations of the training process in reaching a global optimum, often settling for a local one instead. This type of error is ubiquitous in various deep learning tasks and has been the subject of extensive research [56], [57].
- *Generalization Error* is given by $\ell_{\text{ge}} = \|\hat{y}(T;w^*) - y(T)\|$. The generalization error is due to the representational capacity of the chosen network architecture $\mathcal{H}$. For example, consider a single-layer network with 100 neurons. Even if the global optimum w^* is reached, there remains an unavoidable gap to $y(T)$.
- *Neurodynamic Error* is denoted by $\ell_{\text{ne}} = \|y(T) - x^*\|$. This error is independent of the neural network and is due solely to the limited size of the chosen time range $[t_0, T]$. According to Theorem 1, as T approaches infinity, $y(T)$ converges to the optimal solution, i.e., $\lim_{T\to\infty} y(T) = x^*$. In practice, however, one can only choose a finitely large $[t_0, T]$, which leads to this error.

Network Architecture and Time Range. As described above, the two hyperparameters, the network architecture $\mathcal{H}$ and the time range $[t_0, T]$, play a critical role in the composition of the error. The implications of the choice of these hyperparameters are discussed below:

- The choice of the network architecture $\mathcal{H}$ affects the generalization error. As the complexity of the network architecture increases, characterized by an increase in the number of hidden layers and neurons, the upper bound on the representational capability of the network increases, thereby reducing the generalization error. However, a more complex architecture poses training challenges and increases the difficulty of minimizing the optimization error.
- The choice of the time range $[t_0, T]$ affects the neurodynamic error. For a fixed network architecture $\mathcal{H}$, a larger time range tends to reduce the neurodynamic error, while potentially increasing the generalization error. The reduction of the neurodynamic error is due to the convergence properties outlined in Theorem 1. Conversely, the increased generalization error arises because a more complex network is required to accurately represent the true state solution $y(t)$ for $t \in [t_0, T]$.

Towards Zero ℓ_{ge} and ℓ_{ne} As shown in Fig. 2-(B), in an extreme scenario, where the network architecture tends towards infinite complexity and the time range $[t_0, T]$ becomes infinitely large, both the generalization error ℓ_{ge} and the neurodynamic error ℓ_{ne} converge to zero. This focuses the objective solely on minimizing the optimization error, i.e., $\ell_{\text{total}} = \ell_{\text{oe}}$. In this context, the NPE problem transforms into a conventional neural network training task, similar to various deep learning challenges. Advanced research in neural network training can thus be leveraged to better address the NPE problem. Despite the theoretical appeal, training a neural network under these conditions introduces inherent challenges. In particular, although ℓ_{ge} and ℓ_{ne} tend to zero, ℓ_{oe} will inevitably increase, complicating the optimization task.

Next, we demonstrate that within a single-layer fully connected network architecture, there exists a neural network that makes $\hat{y}(T;w)$ the optimal solution to the NPE problem. First, we give a formal definition of the network architecture

under consideration, followed by a proposition establishing its existence, and finally, we conclude with some remarks.

Definition 5. *Consider a neural network that takes inputs over the interval $[t_0, T]$ and contains a single hidden layer with k hidden units, d output units. We denote such a network architecture as $\mathfrak{C}^{(k)}(\psi)$, where ψ specifies the activation function of the hidden units. For a network with an arbitrary number of hidden units, the architecture is defined as:*

$$\mathfrak{C}(\psi) = \bigcup_{k=1}^{\infty} \mathfrak{C}^{(k)}(\psi). \tag{17}$$

Assumption 2. *The activation function of (17) is non-constant, bounded, and continuous.*

Proposition 3. *Consider an NPE problem denoted as $NPE(\Omega, G)$, and x^* is the optimal solution to this problem. Let Assumptions 1 and 2 hold. Consider any initial condition (t_0, y_0) for the ODE system (6). Then, there exists a neural network $N \in \mathfrak{C}(\psi)$ such that $\hat{y}(T; W)$ is arbitrarily close to x^* as T tends to infinity. Formally,*

$$\lim_{T \to \infty} \hat{y}(T; w) = x^*. \tag{18}$$

Remark 1. *The choice of the activation function significantly impacts the neural network's ability to globally approximate the optimal solution. As stated in Lemma 2, the activation function must be non-constant, bounded, and continuous. Consequently, the widely used rectified linear unit (ReLU) activation function is unsuitable due to its unbounded nature. Instead, the hyperbolic tangent (tanh) activation function is a more appropriate choice, as it fulfills all the three requirements.*

Remark 2. *Proposition 3 establishes the existence of a single-layer neural network that allows the PINN model (9) to provide an optimal solution to the target NPE problem. This existence result is jointly derived from the global convergence theorem in neurodynamic optimization and the universal approximation theorem for neural networks. Specifically, the global convergence theorem ensures that the state solution $y(t)$ converges to the optimal solution as time t tends to infinity. Subsequently, the universal approximation theorem guarantees that there exists a neural network capable of approximating the corresponding $y(t)$.*

IV. Multi-Task Learning for Multiple NPEs

In this section, we introduce the multi-task learning (MTL) framework employed in our study, which allows us to address multiple NPE instances using a single neural network, thereby eliminating the need for repeated individual solutions. In Section IV-A, we present the MTL framework, the loss function, and the training objective. In Section IV-B, we present the training procedure of the MTL model.

A. Multi-Task Learning Framework

MTL framework description. We employ an MTL framework based on the model in (9) that facilitates the simultaneous training of a single model for solving multiple NPEs. As shown in Fig. 3, the model comprises two parts: shared layers and task-specific layers. The shared layers learn common features among NPEs, promoting knowledge transfer across different NPEs and improving generalization capabilities. The task-specific layers generate task-specific predictions for each NPE.

Model input and output. The MTL model can be expressed by the following equation:

$$MTL(t; w) = \left(\hat{y}^{(1)}(t; w), \hat{y}^{(2)}(t; w), \ldots, \hat{y}^{(n)}(t; w)\right). \tag{19}$$

The model takes an input of $t \in [t_0, T]$. It is essential to note that the MTL model requires the same time range $[t_0, T]$ for all NPEs, as they share the input. The model has n outputs, each of which corresponds to an NPE to be solved. The MTL solutions for the multiple NPEs can be directly generalized from Section III. Specifically, each output, $\hat{y}^{(i)}$ for $t \in [t_0, T]$, is the approximated state solution for the i-th ODE system, which in turn is derived from the i-th NPE. When choosing the time to $t = T$, the output $\hat{y}^{(i)}(t = T; w)$ is the prediction for the i-th NPE. For each NPE, we can provide a distinct initial condition $(t_0^{(i)}, y_0^{(i)})$.

MTL loss. We now define the loss function for the MTL model consisting of n NPEs. The MTL loss function is given by:

$$\mathcal{L}_{\text{MTL}}(t, w) = \frac{1}{n} \sum_{i=1}^{n} \left\| \frac{\partial \hat{y}^{(i)}(t; w)}{\partial t} - \Phi^{(i)}(\hat{y}^{(i)}(t; w)) \right\|, \tag{20}$$

where $\Phi^{(i)}(\cdot)$ represents the ODE system derived from the i-th NPE. The MTL loss in (20) essentially extends the loss function (12) to accommodate multiple NPEs. Similarly, we formulate the objective function and the batch loss for the MTL model as follows:

$$J_{\text{MTL}}(w) = \frac{1}{n} \sum_{i=1}^{n} \int_{t_0}^{T} \left\| \frac{\partial \hat{y}^{(i)}(t; w)}{\partial t} - \Phi^{(i)}(\hat{y}^{(i)}(t; w)) \right\| dt, \tag{21}$$

$$\mathcal{L}_{\text{MTL}}(\mathbb{T}, w) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \mathcal{L}_{\text{MTL}}(t, w). \tag{22}$$

The MTL loss function allows the model to learn from multiple NPEs simultaneously, thereby enhancing its overall performance and generalization capabilities. It is important to note that in the loss functions (20), (21), and (22), we assume that the weights assigned to each NPE are equal. In practice, one has the flexibility to assign weights as needed.

Shared information between tasks. The shared layers of the MTL framework output a hidden feature vector in their final layer. This hidden feature vector serves as a representation of the shared information between multiple NPE tasks. However, due to the black-box nature of neural networks, it is challenging to interpret or explain the specific meaning encapsulated by this hidden feature vector. This is analogous to the difficulty of interpreting a single feature vector or feature

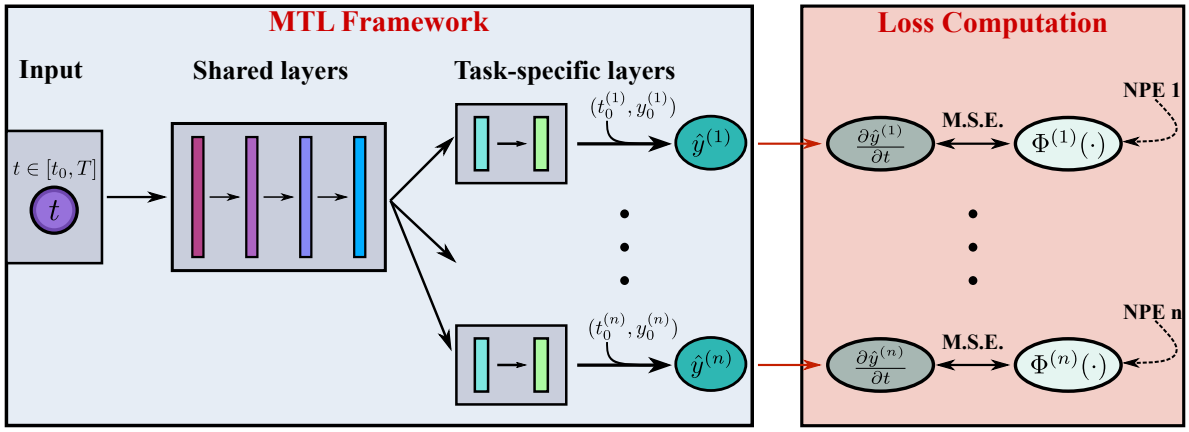

Fig. 3: **MTL model for solving multiple NPEs.** This figure illustrates the process of solving n NPEs using the MTL framework. For each NPE indexed by $i = 1, 2, \ldots, n$, $(t_0^{(i)}, y_0^{(i)})$, $\hat{y}^{(i)}$, and $\Phi(\cdot)^{(i)}$ represent the initial condition, the approximate state solution, and the corresponding ODE system, respectively. M.S.E. stands for mean square error.

map in any neural network, as its semantics are generally not readily understandable.

B. Training Procedure

NPE error. To assess the effectiveness of a prediction x_{pred} in solving the NPE (2), we introduce a performance metric called the NPE error. This metric is defined as follows:

$$\text{NE}(x_{\text{pred}}) = \|P_\Omega(x_{\text{pred}} - G(x_{\text{pred}})) - x_{\text{pred}}\|_\infty . \tag{23}$$

The NPE error quantifies the deviation of the prediction x_{pred} from the true solution of the NPE, providing an accurate assessment of how well the model has learned to solve the given problem. By minimizing the NPE error during training, we ensure that the model's predictions closely adhere to the properties of the NPE and improve its overall performance.

Pipeline. Algorithm 1 outlines the process for utilizing the proposed MTL framework to tackle multiple NPEs concurrently. Let $I = \{1, 2, \ldots, n\}$ denote the index set for the NPEs. First, a time range $[t_0, T]$ and a set of initial conditions $\{(t_0^{(i)}, y_0^{(i)})\}_{i \in I}$ must be given to formulate the IVPs. Then, the MTL model is initialized with n task-specific heads, where each head $\hat{y}^{(i)}(t; w)$ corresponds to the initial condition $(t_0^{(i)}, y_0^{(i)})$. The model is fine-tuned by applying gradient descent to the loss function (22) to improve the approximation. Notably, our solver relies exclusively on deep learning infrastructure, eliminating the need for traditional optimization or numerical integration solvers.

Best Prediction w.r.t NPE error. An important feature of Algorithm 1 is that it uses the NPE error (23) as a metric to evaluate how well the prediction solves the NPE problem. At each iteration, the algorithm calculates the NPE error for the current iteration, denoted as $\text{NE}_{\text{curr}}^{(i)}$, and compares it to the smallest NPE error encountered so far, denoted as $\text{NE}_{\text{best}}^{(i)}$. The terms $x_{\text{curr}}^{(i)}$ and $x_{\text{best}}^{(i)}$ represent the prediction at the current iteration and the best prediction encountered so far, respectively. If $\text{NE}_{\text{curr}}^{(i)}$ is less than $\text{NE}_{\text{best}}^{(i)}$, it indicates that the model has identified a better prediction during this iteration. As a result, the algorithm updates $\text{NE}_{\text{best}}^{(i)}$ to equal $\text{NE}_{\text{curr}}^{(i)}$ and retains the best prediction as $x_{\text{best}}^{(i)} = x_{\text{curr}}^{(i)}$. Such a mechanism ensures that the optimal prediction obtained by the model is maintained throughout the training process, ultimately improving the overall performance of the algorithm. Algorithm 1 presents the for-loop iteration over the index set I, which in practice can easily be replaced by a parallel approach using Numpy to enhance computational efficiency.

V. Experiments

Experimental Setup. We use PyTorch 1.12.1 [54] to implement the proposed MTL framework and JAX 0.4.1 [55] to implement the ODE system. Regarding the network architecture for the MTL framework, the shared layers consist of three fully connected layers of 50, 100, and 100 neurons, respectively. The task-specific layers contain two fully connected layers with 100 and 50 neurons, respectively. The activation function used is tanh. The optimizer used for training is Adam [58], with a learning rate of 0.001 and a batch size of 128. All three subsections use $t_0 = 0$ and $y_0 = \mathbf{0}$ as the initial condition. Sections V-A and V-C choose $[0, 10]$ as the time range, while Section V-B considers different choices of time ranges. The numerical solvers used for comparison are Runge-Kutta 45 (RK45), Runge-Kutta 23 (RK23), Dormand-Prince 853 (DOP853), Backward Differentiation Formula (BDF), Radau, and LSODA. All of these methods are available in the Scipy library [59].

Evaluation Metrics. We evaluate the performance of the MTL framework using the following three metrics.

- NPE Error: As defined in Eq. (23), this metric measures how well a prediction solves the NPE problem.
- Mean Square Error (MSE) Loss: This metric measures how well a predicted state solution solves the ODE system.
- Computation Time: This metric measures the computational efficiency.

Algorithm 1 Training of the MTL model to solve multiple NPEs

Input: A time range: $[t_0, T]$;
n number of NPEs: $\{NPE(\Omega^{(1)}, G^{(1)}), NPE(\Omega^{(2)}, G^{(2)}), \ldots, NPE(\Omega^{(n)}, G^{(n)})\}$;
n number of initial conditions: $\{(t_0^{(1)}, y_0^{(1)}), (t_0^{(2)}, y_0^{(2)}), \ldots, (t_0^{(n)}, y_0^{(n)})\}$.
Output: Predictions for the NPEs, denoted by $\{x_{\text{best}}^{(1)}, x_{\text{best}}^{(2)}, \ldots, x_{\text{best}}^{(n)}\}$.

function NPEs SOLVER
  Derive ODE systems, $\{\Phi^{(1)}(\cdot), \Phi^{(2)}(\cdot), \ldots, \Phi^{(n)}(\cdot)\}$, corresponding to the given NPEs using (6).
  Instantiate the MTL model based on the given initial conditions.
  Initialize $\{\text{NE}_{\text{best}}^{(1)}, \text{NE}_{\text{best}}^{(2)}, \ldots, \text{NE}_{\text{best}}^{(n)}\}$, where $\text{NE}_{\text{best}}^{(i)} = \text{NE}(\hat{y}^{(i)}(t = T; w))$.
  while iteration $\leq$ maximum iteration **do**
    Sample $\mathbb{T} \sim U(t_0, T)$ ▷ $U(t_0, T)$ is the uniform distribution with interval $[t_0, T]$
    Compute the MTL batch loss: $\mathcal{L}_{\text{MTL}}(\mathbb{T}, w)$ ▷ Forward propagation
    Update weights: $w \leftarrow \nabla_w \mathcal{L}_{\text{MTL}}(\mathbb{T}, w)$ ▷ Backward propagation
    for $i = 1, 2, \ldots, n$ **do**
      Obtain the prediction from the i-th head of the MTL model: $x_{\text{curr}}^{(i)} = \hat{y}^{(i)}(t = T; w)$
      Project $x_{\text{curr}}^{(i)}$ onto the feasible set Ω using (1): $x_{\text{curr}}^{(i)} = P_\Omega(x_{\text{curr}}^{(i)})$
      $\text{NE}_{\text{curr}}^{(i)} = \text{NE}(x_{\text{curr}}^{(i)})$ ▷ Calculate the NPE error of $x_{\text{curr}}^{(i)}$ by (23).
      if $\text{NE}_{\text{curr}}^{(i)} < \text{NE}_{\text{best}}^{(i)}$ **then**
        $\text{NE}_{\text{best}}^{(i)} = \text{NE}_{\text{curr}}^{(i)}$ ▷ Update $\text{NE}_{\text{best}}^{(i)}$.
        $x_{\text{best}}^{(i)} = x_{\text{curr}}^{(i)}$ ▷ Update the best prediction.
      end if
    end for
  end while
  return $\{x_{\text{best}}^{(1)}, x_{\text{best}}^{(2)}, \ldots, x_{\text{best}}^{(n)}\}$
end function

A. *Solving Multiple NPEs with the Proposed MTL Framework*

Problem Definition. Consider the following NPE problem,

$$P_\Omega(x^* - G^r(x^*)) = x^*, \tag{24}$$

where $r = [a, b, c, d] \in \mathbb{R}^4$,

$$G^r(x) = \begin{bmatrix} 3x_1 - \frac{a}{x_1+1} + 5x_2 - 13 \\ 1.2x_1 + bx_2 \\ cx_3 + 8x_4 \\ 1x_3 + 2x_4 - \frac{4}{x_4+2} - d \end{bmatrix},$$
$$\Omega = \{x \in \mathbb{R}^4 \mid 1 \leq x_1 \leq 100, -3 \leq x_2 \leq 100, -10 \leq x_3 \leq 100, 1 \leq x_4 \leq 100\}, \tag{25}$$

and $P_\Omega(\cdot)$ is defined in (5). r is the problem data, and selecting distinct values of r yields different NPE instances.

Problem Set Construction. We create a problem set by uniformly sampling r from the interval $[1, 10]^4$. The set of the multiple NPEs is denoted as $\{NPE(G^{r_1}, \Omega), NPE(G^{r_2}, \Omega), \ldots, NPE(G^{r_n}, \Omega)\}$, where n is the number of instances. These NPE instances share the same feasible region Ω and a majority of the nonlinear function $G^r(\cdot)$, distinguished only by the problem parameter r. Traditional numerical methods solve this set of instances individually, treating each instance as an independent problem. In contrast, our proposed MTL framework solves all these NPE instances simultaneously in a parallel manner.

Training of the MTL Framework. Fig. 4 shows the training of different numbers of NPE instances using our proposed MTL framework. Specifically, Fig. 4(A) displays the loss values, indicating the solution accuracy to the ODE systems, while Fig. 4(B) shows the NPE error, indicating the solution accuracy to the NPE instances. With only 1000 iterations, all MSE losses converge from about 150 to less than 1, and all NPE errors converge from about 13 to less than 1. Notably, the significant reduction in MSE loss occurs primarily between the 100th and 1000th iteration, while the reduction in NPE error occurs between the 10th and 100th iteration. The asynchronous decrease of these two metrics suggests that the NPE error can be reduced even if the MSE loss is not reduced or even increased. These results benefit from the use of the evlautation metric (23) in Algorithm 1.

Performance of the MTL Framework. Fig. 5 demonstrates the computational efficiency of the MTL framework compared to traditional numerical solvers. Here, the number of NPE instances is fixed at 100. Experimental results show that our MTL framework outperforms the numerical solvers across various target accuracies. In particular, when the required target accuracy is relaxed, such as requiring an NPE error of 1, the computational efficiency advantage of the MTL framework is significant, at least 3 times faster than RK45 (the best-performing numerical solver in this case). When the required target accuracy is stringent, such as requiring an NPE error of 0.05, the computational efficiency of the MTL framework still leads over various numerical solvers.

B. *Application to Hopfield Network*

The goal of this subsection is to apply the proposed MTL framework for solving equilibrium points in a Hopfield network. This subsection also serves as a hyperparameter study to

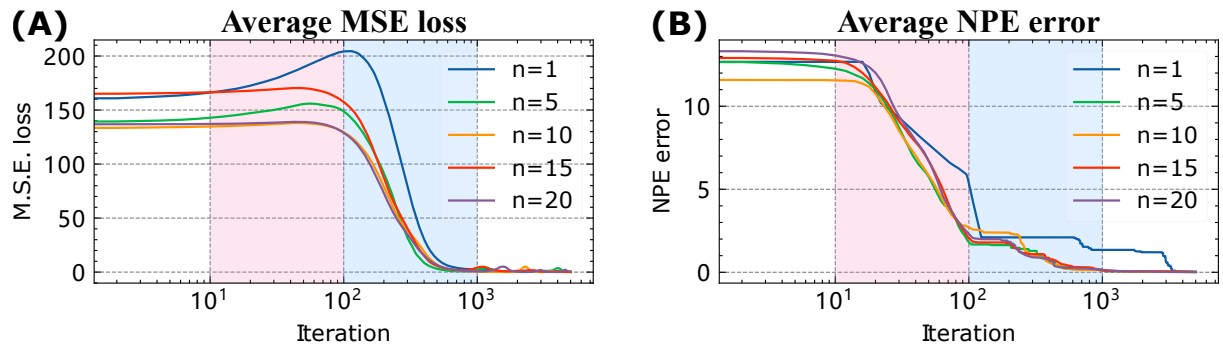

Fig. 4: Training of the MTL framework. n denotes the number of NPE instances to be solved. **(A)** Average MSE loss versus training iteration. **(B)** Average NPE error versus training iteration.

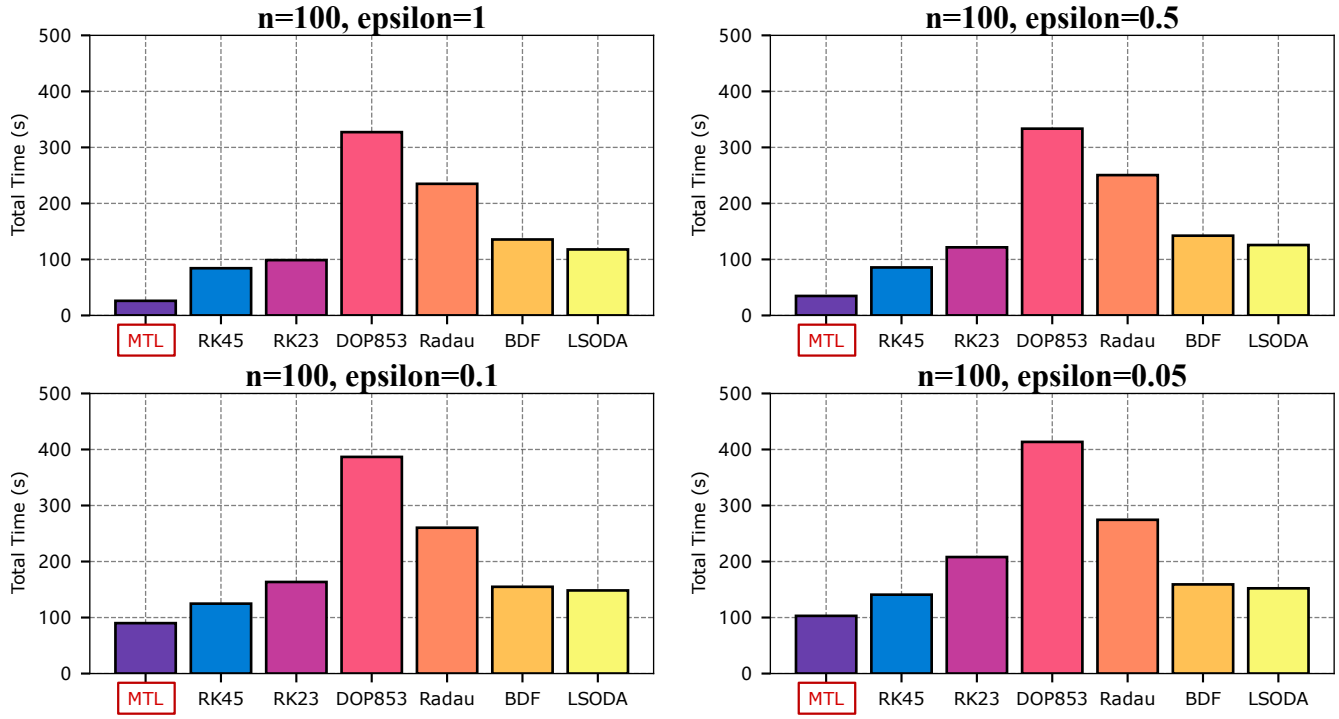

Fig. 5: Comparison of computation time between the MTL framework and numerical solvers. Computation time is measured in seconds. Each subplot shows the total times required to achieve a specific accuracy for 100 NPE instances using a particular solver.

investigate the performance of the framework under different time ranges.

Problem Definition. Consider the following Hopfield network:

$$\frac{dx}{dt} = -x + TP_{\Omega}(x) + q, \tag{26}$$

where $T \in \mathbb{R}^{10\times 10}$ is defined as

$$T = \begin{bmatrix} 3 & 3 & \dots & 3 \\ 3 & 3 & \dots & 3 \\ \vdots & \vdots & \ddots & \vdots \\ 3 & 3 & \dots & 3 \\ -3 & -3 & \dots & -3 \end{bmatrix}. \tag{27}$$

The feasible set Ω is defined as

$$\Omega = \{x \in \mathbb{R}^{10} \mid -1 \leq x \leq 1\}. \tag{28}$$

Our goal is to find x^* such that $\frac{dx}{dt} = 0$. The equilibrium point computation of the Hopfield network has wide applications in various types of combination optimization problems [60]–[62].

Problem Set Construction. We construct a set of instances by selecting several different $q \in \mathbb{R}^{10}$ in Eq. (26). Specifically, we uniformly sample 50 different q from the interval $[-1, 1]^{10}$ to create a problem set. Next, we use the proposed MTL framework to solve these 50 instances in parallel.

Training of the MTL Framework. Fig. 6 shows the training of the MTL framework to solve these 50 instances, with the time range set to $[0, 10]$. Unlike Fig. 4, which shows average results, Fig. 6 explicitly shows the results for each instance. We observe that the initial NPE errors vary between instances, ranging from 2 to 18. At the 600th training iteration, the NPE errors for all instances fall below 0.3. At the 1500th

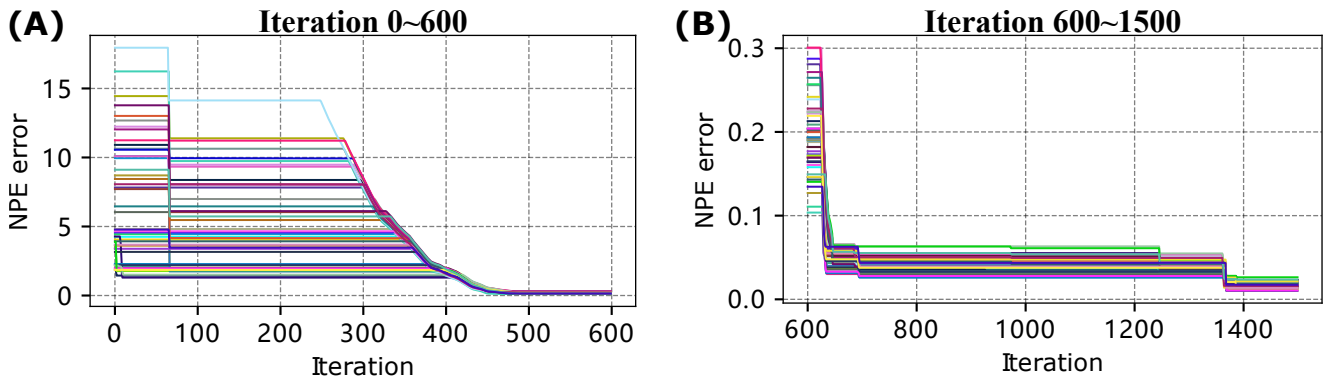

Fig. 6: NPE error versus training iteration for the 50 instances of Eq. (26). **(A)**: the NPE errors from iteration 0 to 600. **(B)**: the NPE errors from iteration 600 to 1500.

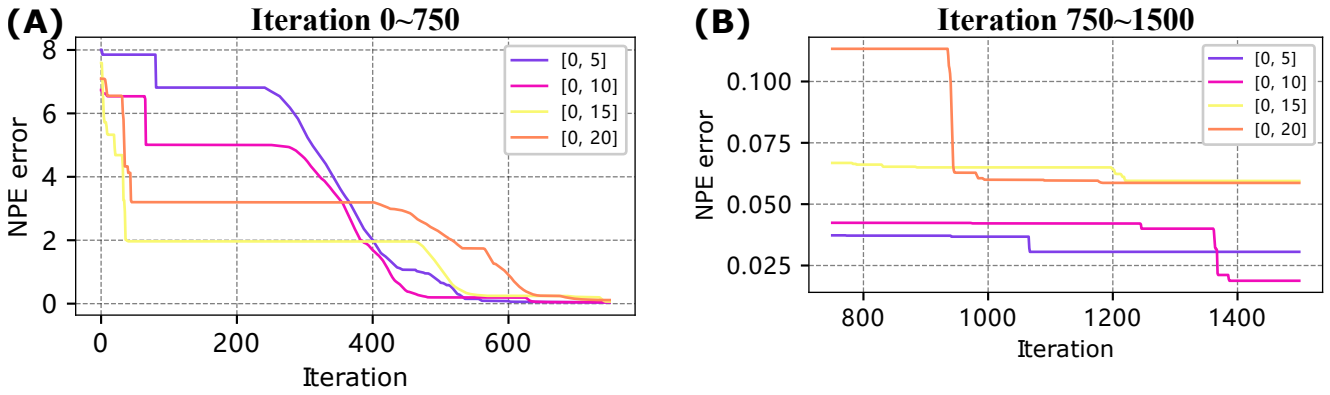

Fig. 7: Comparison of average NPE errors over different time ranges. **(A)**: the average NPE errors from iteration 0 to 750. **(B)**: the average NPE errors from iteration 750 to 1500.

training iteration, the NPE errors are further reduced to below 0.05. These experimental results demonstrate the effectiveness of our MTL framework in solving the equilibrium points for these 50 instances.

Selection of Time Range. As shown in Fig. 7, the choice of the time range significantly influences the convergence speed of the MTL framework. There is no a universally optimal choice for the time range. Instead, it should be determined based on the training resources allocated. For example, in Fig. 7, the time range $[0, 5]$ gives the best results for 750 iterations, whereas $[0, 10]$ gives the best results for 1500 iterations. Here are some guidelines on how to choose an appropriate time range based on the number of training iterations:

- If computational resources are limited and a quick result is desired, choose a smaller time range, such as $[0, 5]$.
- If computational resources are sufficient, a larger time range should be chosen, as this will result in reduced neurodynamic errors, consistent with the discussion in Section III-C.
- In most scenarios, the choice of time range should be customized to align with the available computational resources. For example, we find that the time range $[0, 10]$ works best for 1500 training iterations.

C. Application to Variational Inequality

The goal of this subsection is to apply the proposed MTL framework for solving variational inequalities. This subsection also compares the proposed MTL framework with various iterative methods.

Problem Definition. Consider the following variational inequality:

$$(x - x^*)^T G^r(x^*) \geq 0, \quad x \in \Omega. \tag{29}$$

$G^r(x)$ is parameterized by $r = [a, b, c, d] \in \mathbb{R}^4$, and

$$G^r(x) = \begin{pmatrix} 2x_1 e^{x_1^2+(x_2-1)^2} + x_1 - x_2 - x_3 + a \\ 2(x_2-1)e^{x_1^2+(x_2-1)^2} - x_1 + 2x_2 + 2x_3 + b \\ \frac{-1}{\sqrt{x_1+x_2+3*x_3+x_4}} + x_1 + x_2 + x_3 + x_4 + c \\ x_3 + 2x_4 - \frac{4}{x_4+2} + d \end{pmatrix}. \tag{30}$$

Ω is defined as

$$\Omega = \{x \in \mathbb{R}^4 \mid x \geq 0\}. \tag{31}$$

By Proposition 2, Eq. (29) is reformulated as an NPE problem.

Problem Set Construction. We construct a problem set by choosing different vectors r in Eq. (29), and each r corresponds to a particular instance. Specifically, we sample r with a uniform distribution over the interval $[-10, 10]^4$. In the following, we apply our proposed MTL framework to solve the problem set with different numbers of instances.

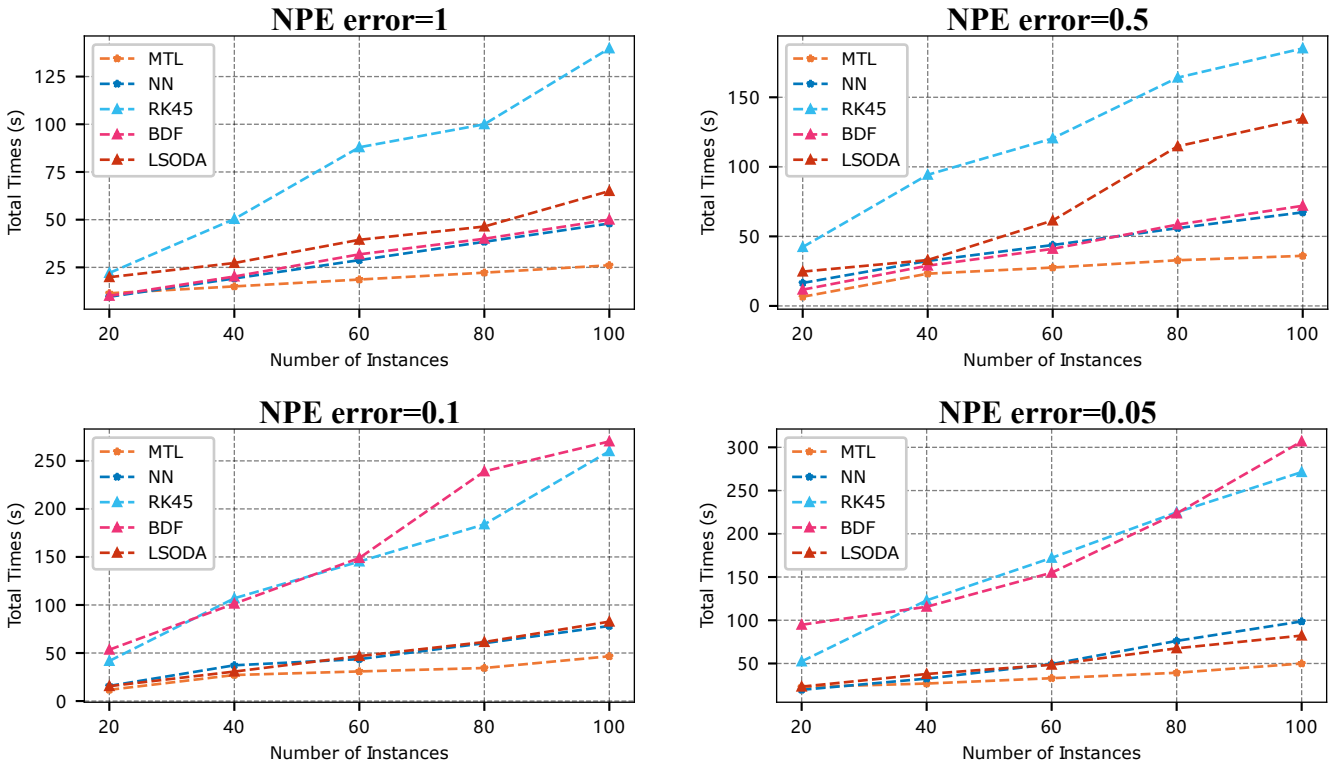

Fig. 8: Efficiency comparison of various solvers over different NPE error thresholds. The y-axis indicates the total computation time in seconds, while the x-axis indicates the number of instances.

Description of Fig. 8. Fig. 8 shows the total computation time required by different solvers to solve a varying number of instances and at different target accuracies. In the figure,

- 'MTL' denotes the MTL framework proposed in this paper.
- 'NN' refers to solving multiple NPE problems iteratively and individually using the PINN model (9), where one PINN model corresponds to one instance.
- 'RK45' and 'BDF' are explicit and implicit numerical solvers, respectively.
- 'LSODA' is a numerical solver that flexibly switches between the above two solvers.

Note that only the proposed MTL framework is a parallel solution algorithm, the other four methods are iterative.

Comparative Efficiency Analysis of Solvers. Based on the analysis of our experimental results, we draw the following conclusions:

- For all NPE error thresholds, the computational time of our proposed MTL framework increases as the number of instances increases. However, its growth rate is significantly slower than that of other solvers, indicating that MTL is more efficient when dealing with large numbers of instances.
- While the NN method performs comparably to the MTL framework with fewer instances, its computation time increases more rapidly as the number of instances increases, suggesting that it is less adept at handling large instance sets than MTL.
- Conventional numerical solvers, RK45, BDF, and LSODA, show relatively consistent performance over different NPE error thresholds. However, their computation times are higher than those of the MTL and NN solvers. In particular, LSODA performs relatively better than the other two numerical methods for this problem.
- As the NPE error threshold decreases, i.e. as the solution accuracy requirement becomes more stringent, the computation time increases for all solvers. This indicates that higher accuracy requirements place greater demands on computational resources. In this scenario, our MTL solver continues to maintain a leading position.

VI. Conclusion

In this paper, we have proposed a novel methodology for solving NPEs. The proposed approach employs three important tools: (1) neurodynamic optimization for modeling NPEs as ODE systems, (2) PINNs for solving the ODE systems, and (3) MTL framework for integrating multiple PINNs. In addition, we have conducted a comprehensive error analysis on the proposed method. Experimental results show that our approach is efficient and accurate, outperforming traditional numerical integration methods. The proposed method holds great promise for efficiently addressing various constrained nonlinear optimization problems.

Below we suggest some future directions for our work:

- Investigate advanced network architectures or training methods from the field of PINNs to further improve the performance of the proposed approach.
- Investigate advanced MTL frameworks to improve or understand the shared knowledge across multiple NPEs.

- Adapt the proposed approach to solve other types of nonlinear optimization problems by collaborating with various neurodynamic techniques.

A. Appendix

In this Appendix, we provide a proof for Proposition 3. The structure of the appendix is organized as follows: A.1 and A.2 contain two lemmas concerning the universal approximation theorem for neural networks. Subsequently, in A.3, we employ these two lemmas to furnish the proof for Proposition 3.

The following metric is used to measure the closeness between two functions.

Definition 6. *$f : [t_0, T] \to \mathbb{R}^d$ and $g : [t_0, T] \to \mathbb{R}^d$ are two functions. The closeness between the two functions is defined as follows*

$$\rho(f, g) = \sup_{t\in[t_0,T]} \|f(t) - g(t)\|_\infty , \tag{32}$$

where $\|\cdot\|_\infty$ is the infinity norm.

A.1 Lemma 1 and its proof

Lemma 1. *Let $y : [t_0, T] \to \mathbb{R}^d$ be a continuous function, and choose a non-constant, bounded, and continuous activation function $\psi(\cdot)$. Then, for any $\epsilon > 0$, there exists a neural network $N \in \mathfrak{C}(\psi)$ such that*

$$\rho(y, N) < \epsilon. \tag{33}$$

Proof. Let $C([t_0, T], \mathbb{R}^n)$ be the space of all continuous functions from the interval $[t_0, T]$ to $\mathbb{R}^n$, equipped with the metric ρ defined in (32).

Define the closure of $\mathfrak{C}(\psi)$, denoted as $\hat{\mathfrak{C}}$, as the set of all functions that can be uniformly approximated by functions from $\mathfrak{C}(\psi)$ with arbitrary precision. In other words, if $h \in \hat{\mathfrak{C}}$, then for any $\epsilon > 0$, there exists a neural network $N \in \mathfrak{C}(\psi)$ such that $\rho(h, N) < \epsilon$.

Since $\psi(\cdot)$ is non-constant, bounded, and continuous, $\hat{\mathfrak{C}}$ contains non-constant functions and is closed under addition and scalar multiplication. Furthermore, $\hat{\mathfrak{C}}$ separates points. By the Stone-Weierstrass theorem, $\hat{\mathfrak{C}}$ is dense in $C([t_0, T], \mathbb{R}^n)$.

Now, fix $y \in C([t_0, T], \mathbb{R}^n)$ and $\epsilon > 0$. Since $\hat{\mathfrak{C}}$ is dense in $C([t_0, T], \mathbb{R}^n)$, there exists a function $h \in \hat{\mathfrak{C}}$ such that $\rho(h, y) < \epsilon/2$.

Since $h \in \hat{\mathfrak{C}}$, there exists a neural network $N \in \mathfrak{C}(\psi)$ that approximates h within an arbitrary degree of accuracy. Choose a neural network N such that $\rho(h, N) < \epsilon/2$. Therefore, we have: $\rho(N, y) \le \rho(y, h) + \rho(h, N) < \epsilon/2 + \epsilon/2 = \epsilon$

This completes the proof, showing that a feedforward neural network with one hidden layer and a finite number of neurons can approximate the continuous function y with an error less than ϵ under the closeness metric $\rho(\cdot, \cdot)$. □

A.2 Lemma 2 and its proof

Lemma 2. *Let $y : [t_0, T] \to \mathbb{R}^d$ be a continuous function and choose a non-constant, bounded, and continuous activation function $\psi(\cdot)$. Then, for any $\epsilon > 0$, there exists a neural network $N \in \mathfrak{C}(\psi)$ such that*

$$\rho(y, \hat{y}) < \epsilon, \tag{34}$$

where $\hat{y} : [t_0, T] \to \mathbb{R}^n$ is the PINN model in Eq. (9).

Proof. First, we construct a function $z(t) = y(t) - y_0$ to be approximated.

By Lemma 1, there exists a neural network $N' \in \mathfrak{C}(\psi)$ that approximates z with an arbitrary degree of accuracy. That is, for any $\epsilon > 0$, there exists $N' \in \mathfrak{C}(\psi)$ such that

$$\rho(z, N') < \frac{\epsilon}{2}. \tag{35}$$

Now, let's define a new function $N(t; w) = \frac{N'(t;w)}{\left(1-e^{-(t-t_0)}\right)}$. Note that since N' is a neural network with activation function ψ, it follows that N is also a neural network with activation function ψ, and thus $N \in \mathfrak{C}(\psi)$.

We construct the neural network-based function $\hat{y}(t; w) = y_0 + \left(1 - e^{-(t-t_0)}\right) N(t; w)$. Now we need to show that $\rho(y, \hat{y}) < \epsilon$.

We have:

$$\begin{aligned} \rho(y, \hat{y}) &= \rho(y, y_0 + \left(1 - e^{-(t-t_0)}\right) N) \\ &= \rho(y_0 + z, y_0 + N') \\ &= \rho(z, N') \\ &< \frac{\epsilon}{2} < \epsilon. \end{aligned} \tag{36}$$

This completes the proof, showing that there exists a PINN model $\hat{y}(t; w)$ which approximates the continuous function y with arbitrarily small ϵ under the closeness metric $\rho(\cdot, \cdot)$. □

A.3 Proof of Proposition 3

Proposition 3. *Consider an NPE problem denoted as $NPE(\Omega, G)$, and x^* is the optimal solution to this problem. Let Assumptions 1 and 2 hold. Consider any initial condition (t_0, y_0) for the ODE system (6). Then, there exists a neural network $N \in \mathfrak{C}(\psi)$ such that $\hat{y}(T; W)$ is arbitrarily close to x^* as T tends to infinity. Formally,*

$$\lim_{T\to\infty} \hat{y}(T; w) = x^*. \tag{37}$$

Proof. Consider the ODE system (6) that models $NPE(\Omega, G)$, and let the state solution of the ODE system be $y : [t_0, +\infty) \to \mathbb{R}^d$. By Theorem 1, we have

$$\lim_{t\to\infty} y(t) = x^*, \tag{38}$$

where x^* is an optimal solution of $NPE(\Omega, G)$.

Let $\epsilon > 0$ be an arbitrary positive number representing the desired accuracy. Consider an increasingly large sequence of time ranges $\{[t_0, T_n]\}$ with $T_n \to +\infty$ as $n \to \infty$.

For each interval $[t_0, T_n]$, apply Lemma 2 to obtain the PINN model $\hat{y}_n(t; w_n)$ that approximates $y(t)$ on the interval $[t_0, T_n]$ with an error less than ϵ, i.e.,

$$\rho(\hat{y}_n, y) < \epsilon. \tag{39}$$

By the definition of $\rho(\cdot,\cdot)$ in Eq. (32), at the end state $t = T_n$, it holds that

$$\|\hat{y}_n(t = T_n; w_n) - y(t = T_n)\| < \epsilon \tag{40}$$

As $n \to \infty$, $T_n \to +\infty$. Therefore,

$$\lim_{n\to\infty} \hat{y}_n(t = T_n; w_n) = \lim_{t\to\infty} y(t). \tag{41}$$

Combining (41) with (38), we conclude that

$$\lim_{n\to\infty} \hat{y}_n(t = T_n; w_n) = x^*. \tag{42}$$

□

REFERENCES

[1] P. T. Harker and J.-S. Pang, "Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications," *Mathematical programming*, vol. 48, no. 1-3, pp. 161–220, 1990.

[2] S. M. Robinson, "Normal maps induced by linear transformations," *Mathematics of Operations Research*, vol. 17, no. 3, pp. 691–714, 1992.

[3] Y. Xia and G. Feng, "A new neural network for solving nonlinear projection equations," *Neural Networks*, vol. 20, no. 5, pp. 577–589, 2007.

[4] Y. Xia and J. Wang, "A Bi-Projection Neural Network for Solving Constrained Quadratic Optimization Problems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 2, pp. 214–224, 2016.

[5] H. Che and J. Wang, "A collaborative neurodynamic approach to global and combinatorial optimization," *Neural Networks*, vol. 114, pp. 15–27, 2019.

[6] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical analysis*. Cengage learning, 2015.

[7] V. V. Singh and A. Lisser, "Variational inequality formulation for the games with random payoffs," *Journal of Global Optimization*, vol. 72, pp. 743–760, 2018.

[8] D. Tank and J. Hopfield, "Simple 'neural' optimization networks: An a/d converter, signal decision circuit, and a linear programming circuit," *IEEE Transactions on Circuits and Systems*, vol. 33, no. 5, pp. 533–541, 1986.

[9] Q. Liu and J. Wang, "A one-layer recurrent neural network with a discontinuous hard-limiting activation function for quadratic programming," *IEEE Transactions on Neural Networks*, vol. 19, no. 4, pp. 558–570, 2008.

[10] M. P. Kennedy and L. O. Chua, "Neural Networks for Nonlinear Programming," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 5, pp. 554–562, 1988.

[11] Y. Xia, H. Leung, and J. Wang, "A projection neural network and its application to constrained optimization problems," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 4, pp. 447–458, 2002.

[12] Z. Guo, Q. Liu, and J. Wang, "A one-layer recurrent neural network for pseudoconvex optimization subject to linear equality constraints," *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 1892–1900, 2011.

[13] H. Che and J. Wang, "A two-timescale duplex neurodynamic approach to biconvex optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 8, pp. 2503–2514, 2018.

[14] N. Liu, J. Wang, and S. Qin, "A one-layer recurrent neural network for nonsmooth pseudoconvex optimization with quasiconvex inequality and affine equality constraints," *Neural Networks*, vol. 147, pp. 1–9, 2022.

[15] Z. Zhang, T. Chen, M. Wang, and L. Zheng, "An exponential-type anti-noise varying-gain network for solving disturbed time-varying inversion systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3414–3427, 2020.

[16] Z. Zhang, Y. Lu, L. Zheng, S. Li, Z. Yu, and Y. Li, "A new varying-parameter convergent-differential neural-network for solving time-varying convex qp problem constrained by linear-equality," *IEEE Transactions on Automatic Control*, vol. 63, no. 12, pp. 4110–4125, 2018.

[17] Z. Zhang, X. Deng, and L. Zheng, "A review on varying-parameter convergence differential neural network," *Neurocomputing*, vol. 490, pp. 54–65, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222002983

[18] Y. Wang, X. Li, and J. Wang, "A neurodynamic optimization approach to supervised feature selection via fractional programming," *Neural Networks*, vol. 136, pp. 194–206, 2021.

[19] J. Wang, J. Wang, and Q.-L. Han, "Multivehicle task assignment based on collaborative neurodynamic optimization with discrete hopfield networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 12, pp. 5274–5286, 2021.

[20] W. Xiong, C. Schindelhauer, H. C. So, J. Bordoy, A. Gabbrielli, and J. Liang, "Tdoa-based localization with nlos mitigation via robust model transformation and neurodynamic optimization," *Signal Processing*, vol. 178, p. 107774, 2021.

[21] M. W. M. G. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *Communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, 1994. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303

[22] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[23] K. S. McFall and J. R. Mahan, "Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1221–1233, 2009.

[24] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[25] S. Huang, W. Feng, C. Tang, and J. Lv, "Partial Differential Equations Meet Deep Neural Networks: A Survey," *arXiv preprint arXiv:2211.05567*, 2022.

[26] 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," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118307125

[27] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (PINNs) for fluid mechanics: A review," *Acta Mechanica Sinica*, pp. 1–12, 2022.

[28] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, "Physics-Informed Neural Networks with Hard Constraints for Inverse Design," *SIAM Journal on Scientific Computing*, vol. 43, no. 6, pp. B1105–B1132, 2021. [Online]. Available: https://doi.org/10.1137/21M1397908

[29] D. Zhang, L. Guo, and G. E. Karniadakis, "Learning in Modal Space: Solving Time-Dependent Stochastic PDEs Using Physics-Informed Neural Networks," *SIAM Journal on Scientific Computing*, vol. 42, no. 2, pp. A639–A665, 2020. [Online]. Available: https://doi.org/10.1137/19M1260141

[30] Z. Fang, "A high-efficient hybrid physics-informed neural networks based on convolutional neural network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5514–5526, 2021.

[31] A. D. Jagtap and G. E. Karniadakis, "Extended Physics-informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition based Deep Learning Framework for Nonlinear Partial Differential Equations." in *AAAI Spring Symposium: MLPS*, 2021.

[32] R. Sharma and V. Shankar, "Accelerated Training of Physics Informed Neural Networks (PINNs) using Meshless Discretizations," *arXiv preprint arXiv:2205.09332*, 2022.

[33] S. Rezaei, A. Harandi, A. Moeineddin, B. X. Xu, and S. Reese, "A mixed formulation for physics-informed neural networks as a potential solver for engineering problems in heterogeneous domains: Comparison with finite element method," *Computer Methods in Applied Mechanics and Engineering*, vol. 401, p. 115616, 2022. [Online]. Available: https://doi.org/10.1016/j.cma.2022.115616

[34] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A Deep Learning Library for Solving Differential Equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021. [Online]. Available: http://dx.doi.org/10.1137/19M1274067

[35] F. Chen, D. Sondak, P. Protopapas, M. Mattheakis, S. Liu, D. Agarwal, and M. D. Giovanni, "NeuroDiffEq: A Python package for solving differential equations with neural networks," *Journal of Open Source Software*, vol. 5, no. 46, p. 1931, 2020. [Online]. Available: https://doi.org/10.21105/joss.01931

[36] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[37] B. Bakker and T. Heskes, "Task clustering and gating for bayesian multitask learning," *J. Mach. Learn. Res.*, vol. 4, no. null, p. 83–99, dec 2003. [Online]. Available: https://doi.org/10.1162/153244304322765658

[38] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[39] M. Crawshaw, "Multi-task learning with deep neural networks: A survey," *arXiv preprint arXiv:2009.09796*, 2020.

[40] S. Sodhani, A. Zhang, and J. Pineau, "Multi-task reinforcement learning with context-based representations," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9767–9779.

[41] D. Kollias, "Abaw: Valence-arousal estimation, expression recognition, action unit detection & multi-task learning challenges," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 2328–2336.

[42] Z. Zhang, W. Yu, M. Yu, Z. Guo, and M. Jiang, "A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods," *arXiv preprint arXiv:2204.03508*, 2022.

[43] M. Mayr, F. Ahmad, K. Chatzilygeroudis, L. Nardi, and V. Krueger, "Skill-based multi-objective reinforcement learning of industrial robot tasks with planning and knowledge integration," in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2022, pp. 1995–2002.

[44] C. Rosenbaum, T. Klinger, and M. Riemer, "Routing networks: Adaptive selection of non-linear functions for multi-task learning," *arXiv preprint arXiv:1711.01239*, 2017.

[45] S. Vandenhende, S. Georgoulis, B. De Brabandere, and L. Van Gool, "Branched multi-task networks: deciding what layers to share," *arXiv preprint arXiv:1904.02920*, 2019.

[46] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool, "Multi-task learning for dense prediction tasks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3614–3633, 2021.

[47] K.-K. Maninis, I. Radosavovic, and I. Kokkinos, "Attentive single-tasking of multiple tasks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1851–1860.

[48] W. M. Czarnecki, R. Pascanu, S. Osindero, S. Jayakumar, G. Swirszcz, and M. Jaderberg, "Distilling policy distillation," in *The 22nd international conference on artificial intelligence and statistics*. PMLR, 2019, pp. 1331–1340.

[49] S. C. Billups and K. G. Murty, "Complementarity problems," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1-2, pp. 303–318, 2000.

[50] F. Facchinei, A. Fischer, and V. Piccialli, "On generalized Nash games and variational inequalities," *Operations Research Letters*, vol. 35, no. 2, pp. 159–164, 2007.

[51] V. V. Singh and A. Lisser, "Variational inequality formulation for the games with random payoffs," *Journal of Global Optimization*, vol. 72, no. 4, pp. 743–760, 2018. [Online]. Available: https://doi.org/10.1007/s10898-018-0664-8

[52] G. Pang, L. Lu, and G. E. Karniadakis, "fpinns: Fractional physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2603–A2626, 2019.

[53] P. L. Lagari, L. H. Tsoukalas, S. Safarkhani, and I. E. Lagaris, "Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions," *International Journal on Artificial Intelligence Tools*, vol. 29, no. 05, p. 2050009, 2020.

[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[55] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[56] L. Chizat and F. Bach, "On the global convergence of gradient descent for over-parameterized models using optimal transport," *Advances in neural information processing systems*, vol. 31, 2018.

[57] J. F. Bonnans, K. Liu, N. Oudjane, L. Pfeiffer, and C. Wan, "Large-scale nonconvex optimization: randomization, gap estimation, and numerical resolution," *arXiv preprint arXiv:2204.02366*, 2022.

[58] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[59] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, A. Vijaykumar, A. P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. N. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. A. Nicholson, D. R. Hagen, D. V. Pasechnik, E. Olivetti, E. Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. A. Price, G.-L. Ingold, G. E. Allen, G. R. Lee, H. Audren, I. Probst, J. P. Dietrich, J. Silterra, J. T. Webber, J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. L. Schönberger, J. V. de Miranda Cardoso, J. Reimer, J. Harrington, J. L. C. Rodríguez, J. Nunez-Iglesias, J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. J. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. A. Brodtkorb, P. Lee, R. T. McGibbon, R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. J. Pingel, T. P. Robitaille, T. Spura, T. R. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. O. Halchenko, and Y. Vázquez-Baeza, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, mar 2020. [Online]. Available: http://www.nature.com/articles/s41592-019-0686-2

[60] J. J. Hopfield and D. W. Tank, ""Neural" computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[61] W. E. Lillo, D. C. Miller, S. Hui, and S. H. Zak, "Synthesis of brain-state-in-a-box (bsb) based associative memories," *IEEE transactions on Neural Networks*, vol. 5, no. 5, pp. 730–737, 1994.

[62] J. K. Paik and A. K. Katsaggelos, "Image restoration using a modified hopfield network," *IEEE Transactions on image processing*, vol. 1, no. 1, pp. 49–63, 1992.