



HAL
open science

Enhancing Neurodynamic Approach with Physics-Informed Neural Networks for Solving Non-Smooth Convex Optimization Problems

Dawen Wu, Abdel Lisser

► **To cite this version:**

Dawen Wu, Abdel Lisser. Enhancing Neurodynamic Approach with Physics-Informed Neural Networks for Solving Non-Smooth Convex Optimization Problems. *Neural Networks*, 2023, 168, pp.419-430. 10.1016/j.neunet.2023.08.014 . hal-04370995

HAL Id: hal-04370995

<https://hal.science/hal-04370995v1>

Submitted on 4 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Enhancing Neurodynamic Approach with Physics-Informed Neural Networks for Solving Non-Smooth Convex Optimization Problems

Dawen Wu^a (dawen.wu@centralesupelec.fr), Abdel Lisser^a (abdel.lisser@l2s.centralesupelec.fr)

^a Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France

Corresponding Author:

Dawen Wu

Address: Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France

Tel: (+33) 750798387

Email: dawen.wu@centralesupelec.fr

Enhancing Neurodynamic Approach with Physics-Informed Neural Networks for Solving Non-Smooth Convex Optimization Problems

Dawen Wu^{a,*}, Abdel Lisser^a

^a*Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France*

Abstract

This paper proposes a deep learning approach for solving non-smooth convex optimization problems (NCOPs), which have broad applications in computer science, engineering, and physics. Our approach combines neurodynamic optimization with physics-informed neural networks (PINNs) to provide an efficient and accurate solution. We first use neurodynamic optimization to formulate an initial value problem (IVP) that involves a system of ordinary differential equations for the NCOP. We then introduce a modified PINN as an approximate state solution to the IVP. Finally, we develop a dedicated algorithm to train the model to solve the IVP and minimize the NCOP objective simultaneously. Unlike existing numerical integration methods, a key advantage of our approach is that it does not require the computation of a series of intermediate states to produce a prediction of the NCOP. Our experimental results show that this computational feature results in fewer iterations being required to produce more accurate prediction solutions. Furthermore, our approach is effective in finding feasible solutions that satisfy the NCOP constraint.

Keywords: Non-smooth convex optimization problem, Neurodynamic optimization, Physics-informed neural network, Numerical integration method, Ordinary differential equation

1. Introduction

Non-smooth convex optimization problems (NCOPs) are an important class of optimization problems that arise in various real-world applications. While traditional optimization methods have been successful in solving many smooth convex optimization problems, they are not well-suited to handle the nonsmoothness of the objective functions in NCOPs. This has led to the development of alternative approaches, such as neurodynamic optimization.

Neurodynamic optimization is a promising approach for solving NCOPs (Qin & Xue, 2014). It involves the implementation of a circuit-based neurodynamic model to solve the optimization problem in real time. This approach has many potential applications, such as in resource allocation (Jia et al., 2021), feature selection (Wang et al., 2023), and coordination of multi-manipulator systems (Hou et al., 2010).

The use of neurodynamic optimization to solve NCOPs typically involves three steps. First, a first-order

*Corresponding author

Email address: dawen.wu@centralesupelec.fr, abdel.lisser@12s.centralesupelec.fr (Abdel Lisser)

12 ordinary differential equation (ODE) system is used to model the NCOP. The ODE system must be shown
13 to have a globally convergent property, meaning that the state solution of the system converges to the
14 optimal solution of the NCOP as time approaches infinity, regardless of the starting point. Second, classical
15 numerical integration methods such as Runge-Kutta methods or backward differentiation formulas are used
16 to numerically obtain the state solution of the ODE system (Burden et al., 2015). Third, the state at the end
17 of a given time range, called the end state, is taken as to be the predicted solution of the NCOP. However,
18 this approach has a drawback. That is, the numerical integration method has to compute all the intermediate
19 states, from the initial state to the end state, to solve the optimization problem, which makes it inefficient.

20 To overcome this limitation, we propose a novel approach that combines neurodynamic optimization
21 with physics-informed neural networks to provide an efficient and accurate solution. Unlike the traditional
22 numerical integration method, our improved approach can directly predict the entire state solution, including
23 the end state. This approach allows for efficient and accurate optimization of NCOPs without requiring the
24 computation of all intermediate states.

25 Here we want to clarify the difference between our work and Neural ODEs. (Chen et al., 2018). Neural
26 ODEs are a class of neural networks that interpret transformations within the model as a continuous process
27 and use ODEs to characterize this process. Training such a model involves solving an ODE using numerical
28 integration methods, and recent research bypasses the use of numerical integration methods to improve
29 computational efficiency (Biloš et al., 2021). However, it is important to emphasize that the primary goal of
30 a neural ODE is not to solve a particular ODE per se, but rather to address traditional supervised learning
31 challenges, such as time series modeling. In our case, the task at hand is to solve an ODE system derived
32 from an NCOP. In this context, the neural network acts as a surrogate model that is specifically trained to
33 approximate the solution of the ODE system.

34 *1.1. Related Works*

35 **Neurodynamic Optimization.** Neurodynamic optimization is a class of methods that model con-
36 strained optimization problems using ODE systems. This approach was first introduced by Hopfield & Tank
37 (1985) to solve the traveling salesman problem. Since then, neurodynamic optimization has been applied to
38 a wide range of optimization problems, including linear and quadratic programming problems (Xia & Wang,
39 2000), general convex programming problems (Kennedy & Chua, 1988; Xia et al., 2008), biconvex optimiza-
40 tion problems (Che & Wang, 2018), global optimization problems (Che & Wang, 2019), and pseudoconvex
41 problems (Guo et al., 2011; Bian et al., 2018; Liu et al., 2022). These methods typically use the Lyapunov
42 stability theorem to prove that the constructed ODE system has a global convergence property, which means
43 that any state solution of the ODE system converges to an optimal solution of the target problem.

44 **Physics-Informed Neural Networks (PINNs).** Another line of research included in our work is the
45 use of deep learning to solve differential equations. As a representative approach, PINNs (Raissi et al., 2019)
46 are a class of neural networks that integrate physical laws described by PDEs and boundary conditions into
47 the training losses. Thus, the neural network is trained to become the solution of the PDE. PINNs have a

48 wide range of successful applications, such as identifying material properties (Shukla et al., 2021a), optimizing
49 sensor locations (Jagtap et al., 2022b), and investigating complex fluid dynamics, including supersonic and
50 high-speed flows (Jagtap et al., 2022a; Mao et al., 2020). To facilitate the use and application of PINNs,
51 several software packages have been developed and are available to the scientific community (Lu et al., 2021;
52 Chen et al., 2020).

53 Numerous methods and theoretical explorations have been introduced to improve the computational
54 performance of PINNs. From the theoretical side, research has been conducted to understand the error
55 estimates associated with PINNs. Mishra & Molinaro (2022) investigates the generalization error of PINNs
56 in approximating a class of inverse problems (Mishra & Molinaro, 2022), and De Ryck et al. (2022) provides
57 error estimates for the PINN approximation of the Navier-Stokes equations. On the methodological side, a
58 prominent strategy is the development of domain decomposition methods, which divide the problem domain
59 into several subdomains, each of which is addressed by a separate PINN. This approach creates interconnected
60 PINN solutions that allow independent treatment of each subdomain while preserving the interconnections
61 across the partitioned domains (Jagtap et al., 2020c; Shukla et al., 2021b; Hu et al., 2022a; Penwarden et al.,
62 2023; Hu et al., 2022b). To further improve the performance of PINNs, adaptive activation functions have
63 been investigated, which add learnable parameters to the activation function (Jagtap et al., 2020b,a, 2022b;
64 Jagtap & Karniadakis, 2023). These innovations in PINNs are reshaping the landscape of solution methods
65 for solving complex scientific and engineering problems.

66 *1.2. Contributions*

67 Our paper presents several key contributions:

- 68 • We propose a novel approach for solving NCOPs that enhances neurodynamic optimization with PINNs,
69 avoiding the use of numerical integration methods.
- 70 • We improve the PINN to adapt to the NCOP by incorporating the initial condition into the neural
71 network and using only one neural network to predict all system states.
- 72 • We design a specialized training algorithm that leverages the structure of NCOPs to optimize the
73 performance of our proposed approach. The neural network is trained to simultaneously satisfy the
74 ODE system and minimize the NCOP objective function.
- 75 • Our experimental results demonstrate several advantages of our approach over classical numerical inte-
76 gration methods and vanilla PINNs. Our approach requires fewer iterations to produce more accurate
77 predicted solutions, and it finds a feasible solution to the problem more quickly.

78 *1.3. Outline*

79 The remaining sections are organized as follows: Section 2 provides the necessary background for un-
80 derstanding the paper, including an introduction to the NCOP and the neurodynamic optimization used to

81 model it. In Section 3, we present our proposed neural network for solving the NCOP. Section 4 details the
 82 design of the loss function and the training algorithm for the neural network. Section 5 presents the results
 83 of our experiments on solving the NCOP using the proposed approach, and compares our approach with the
 84 existing methods. Finally, Section 6 summarizes the main results of the paper and outlines possible directions
 85 for future research.

86 2. Neurodynamic Approach for Modelling NCOP

87 Section 2.1 introduces the NCOP, which is the goal of this paper. In Section 2.2, we show how to model
 88 the NCOP using a neurodynamic approach. In Section 2.3, we introduce the PINN method for solving PDEs.

89 2.1. NCOP

We consider the following optimization problem:

$$\left\{ \begin{array}{l} \min_x f(x) \\ \text{s.t.} \\ g(x) \leq \mathbf{0} \\ Ax = b, \end{array} \right. \quad (1)$$

90 where $x = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ is the decision variables, and $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function.
 91 $g(x) = (g_1(x), g_2(x), \dots, g_m(x))^T : \mathbb{R}^n \rightarrow \mathbb{R}^m$ represents the inequality constraints, and $u = (u_1, u_2, \dots, u_m)$
 92 represents the dual variables of the inequality constraints. $Ax = b$ represents the equality constraints with
 93 $A \in \mathbb{R}^{k \times n}$ and $b \in \mathbb{R}^k$. n , m , and k denote the number of decision variables, inequality constraints, and
 94 equality constraints, respectively.

95 In this paper, we consider the case where $f(x)$ and $g(x)$ are convex but not necessarily smooth, and A is
 96 of full row rank. We denote x^* and u^* as the optimal primal and dual solutions, respectively.

Definition 1 (Subgradient and subdifferential). *A vector $l \in \mathbb{R}^n$ is a subgradient of $f : \mathbb{R}^n \rightarrow \mathbb{R}$ at $x \in \text{dom } f$ if the following holds*

$$f(z) \geq f(x) + l^T(z - x), \quad \forall z \in \text{dom } f. \quad (2)$$

97 *The set of all subgradients of f at x is called the subdifferential of f at x and is denoted by $\partial f(x)$.*

98 2.2. Neurodynamic Approach

Now, let $x : \mathbb{R} \rightarrow \mathbb{R}^n$ and $u : \mathbb{R} \rightarrow \mathbb{R}^m$ be some time dependent functions. The aim of a neurodynamic approach is to construct a first-order ODE system to govern $x(t)$ and $u(t)$, such that they will settle down to the optimal primal and dual solutions of the NCOP (1). In this paper, the two-layer neurodynamic approach

in Qin & Xue (2014) is adopted, where the ODE system is described as follows:

$$\begin{aligned}\frac{dx}{dt} &\in -(I - U) [\partial f(x) + \partial g(x)^T (U + g(x))^+] - A^T \rho(Ax - b), \\ \frac{du}{dt} &= \frac{1}{2} (-U + (U + g(x))^+),\end{aligned}\tag{3}$$

where $U = A^T (AA^T)^{-1} A$, $I \in \mathbb{R}^{n \times n}$ is the identity matrix, $\rho(s) = (\tilde{\rho}(s_1), \tilde{\rho}(s_2), \dots, \tilde{\rho}(s_k))^T$, and for $i = 1, 2, \dots, k$,

$$\tilde{\rho}(s_i) = \begin{cases} 1 & \text{if } s_i > 0, \\ [-1, 1] & \text{if } s_i = 0, \\ -1 & \text{if } s_i < 0. \end{cases}\tag{4}$$

To simplify the discussion, we denote $y(t) = (x(t)^T, u(t)^T)^T$ and define:

$$\Phi(y) = \begin{bmatrix} -(I - U) [\partial f(x) + \partial g(x)^T (U + g(x))^+] - A^T \rho(Ax - b) \\ \frac{1}{2} (-U + (U + g(x))^+) \end{bmatrix}.\tag{5}$$

99 Thus, the ODE system (3) can be written as $\frac{dy}{dt} \in \Phi(y)$.

100 **Definition 2** (State solution). Consider an ODE system $\frac{dy}{dt} \in \Phi(y)$, where $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Given $(t_0 \in$
101 $\mathbb{R}, y_0 \in \mathbb{R}^n)$, a vector value function $y : \mathbb{R} \rightarrow \mathbb{R}^n$ is called a state solution, if it satisfies the ODE system
102 $\frac{dy}{dt} \in \Phi(y)$ and the initial condition $y(t_0) = y_0$.

103 In particular, we call $y(t)$ the state at time t . Given a time interval $[t_0, T]$, we call $y(T)$ the end state on
104 that time interval.

Theorem 1 (Qin & Xue (2014)). Consider a NCOP (1) and its derived ODE system (3). Given any initial
condition $y(t_0) = y_0$, the state solution $y(t)$ of the ODE system converges to an optimal solution y^* as time
 t approaches infinity, i.e.

$$\lim_{t \rightarrow \infty} y(t) = y^*,\tag{6}$$

105 where $y^* = (x^{*T}, u^{*T})^T$, x^* and u^* are the optimal primal and dual solutions to the NCOP.

106 In particular, if the NCOP contains only one optimal solution x^* , then the ODE system is called globally
107 asymptotically stable at y^* .

108 **Initial Value Problem (IVP) Construction.** In practice, in order to use the neurodynamic approach
109 to solve the NCOP, we need to construct an IVP consisting of three components: 1) the ODE system (3), 2)
110 an initial condition $y(t_0) = y_0$, and 3) a time range $t \in [t_0, T]$. $y(t)$ for $t \in [t_0, T]$ represents the state solution
111 of this IVP problem over the time range $[t_0, T]$, where the end state, $y(T)$, is considered to be the predicted
112 solution to the NCOP. According to Theorem 1, the larger the time range $[t_0, T]$, the closer $y(T)$ is to the
113 optimal solution y^* of the NCOP.

114 *2.3. Physics-Informed Neural Networks (PINNs)*

115 In this subsection, we introduce the powerful framework of PINNs for solving PDEs. For clarity and to
 116 make this subsection self-contained, please note that the mathematical notations used in this subsection are
 117 independent of those used in other sections. The mathematical notations should be interpreted only in the
 118 context of this subsection.

A typical PDE problem can be expressed in the following general form:

$$\begin{aligned} D_x(u; \lambda) &= f(x), & x \in \Omega \subset \mathbb{R}^d \\ \mathcal{B}_k(u) &= g_k(x), & x \in \partial\Omega \subset \mathbb{R}^d, \quad \text{for } k = 1, 2, \dots, n_b, \end{aligned} \quad (7)$$

119 where $D_x(\cdot)$ is the differential operator, and $u : \Omega \cup \partial\Omega \rightarrow \mathbb{R}$ is the solution to be found. λ denotes the PDE
 120 parameters. $\mathcal{B}_k(\cdot)$ denotes to the boundary conditions, which can be of the Dirichlet, Neumann, or mixed
 121 type. For problems involving temporal dynamics, time t is considered as part of x , and the initial conditions
 122 can be treated as a unique type of boundary condition.

123 Let $\{x_b^{(i)}, u_b^{(i)}\}_{i=1}^{N_b}$ and $\{x_d^{(i)}\}_{i=1}^{N_d}$ be the sets of randomly selected training points and residual points,
 124 respectively. These points are usually drawn from an unknown distribution. Let u_Θ be a surrogate function
 125 based on a neural network with model parameters Θ . The goal of the PINN approach is to learn a surrogate
 126 function u_Θ to approximate the solution u for a given PDE.

The loss function for PINNs can be expressed as:

$$\mathcal{L}(\Theta) = W_b \text{MSE}_b \left(\Theta; \{x_b^{(i)}, u_b^{(i)}\}_{i=1}^{N_b} \right) + W_d \text{MSE}_d \left(\Theta; \{x_d^{(i)}\}_{i=1}^{N_d} \right), \quad (8)$$

where W_b and W_d represent the weights for the data and residual losses, respectively. $\text{MSE}_b \left(\Theta; \{x_b^{(i)}, u_b^{(i)}\}_{i=1}^{N_b} \right) +$
 and $\text{MSE}_d \left(\Theta; \{x_d^{(i)}\}_{i=1}^{N_d} \right)$ are given by:

$$\begin{aligned} \text{MSE}_b \left(\Theta; \{x_b^{(i)}, u_b^{(i)}\}_{i=1}^{N_b} \right) &= \frac{1}{N_b} \sum_{i=1}^{N_b} \left| u^{(i)} - u_\Theta \left(x_b^{(i)} \right) \right|^2, \\ \text{MSE}_d \left(\Theta; \{x_d^{(i)}\}_{i=1}^{N_d} \right) &= \frac{1}{N_d} \sum_{i=1}^{N_d} \left| D_x(u_\Theta(x_d^{(i)}); \lambda) - f(x_d^{(i)}) \right|^2, \end{aligned} \quad (9)$$

127 where MSE_b measures the data mismatch term, which enforces the boundary conditions as constraints.
 128 MSE_d evaluates the PDE residual at a finite set of collocation points. The neural network parameters Θ are
 129 determined by minimizing the loss function in (8).

130 A distinct advantage of PINNs is their ability to integrate both experimental and synthetic training data
 131 into the loss function. One of the central mechanisms facilitating this integration is the computation of the
 132 PDE residual using automatic differentiation (Baydin et al., 2018), which provides an efficient and accurate
 133 evaluation of the PDE operator. Thus, PINNs transform the task of solving a PDE into an optimization
 134 problem, where the global minimum of the loss function signifies the solution to the PDE. This capability

135 positions PINNs as a grid-free method, alleviating the often cumbersome process of mesh generation.

136 **Remark 1.** The ODE system (3) under consideration is a particular case of the PDE problem (7). The key
 137 differences are as follows:

- 138 • The solution of the ODE system is a function with only one input, while the PDE solution is a function
 139 with multiple inputs.
- 140 • The ODE system involves multiple differential equations, each corresponding to a state dynamic, whereas
 141 the PDE typically involves only a single equation with multiple variables.

142 Mathematically, PINNs typically aim to find a solution $u : \mathbb{R}^d \rightarrow \mathbb{R}$, where d is the number of variables.
 143 In contrast, the solution sought in the ODE system is $u : \mathbb{R} \rightarrow \mathbb{R}^n$, where n is the number of differential
 144 equations in the system.

145 **3. Modified PINN Model**

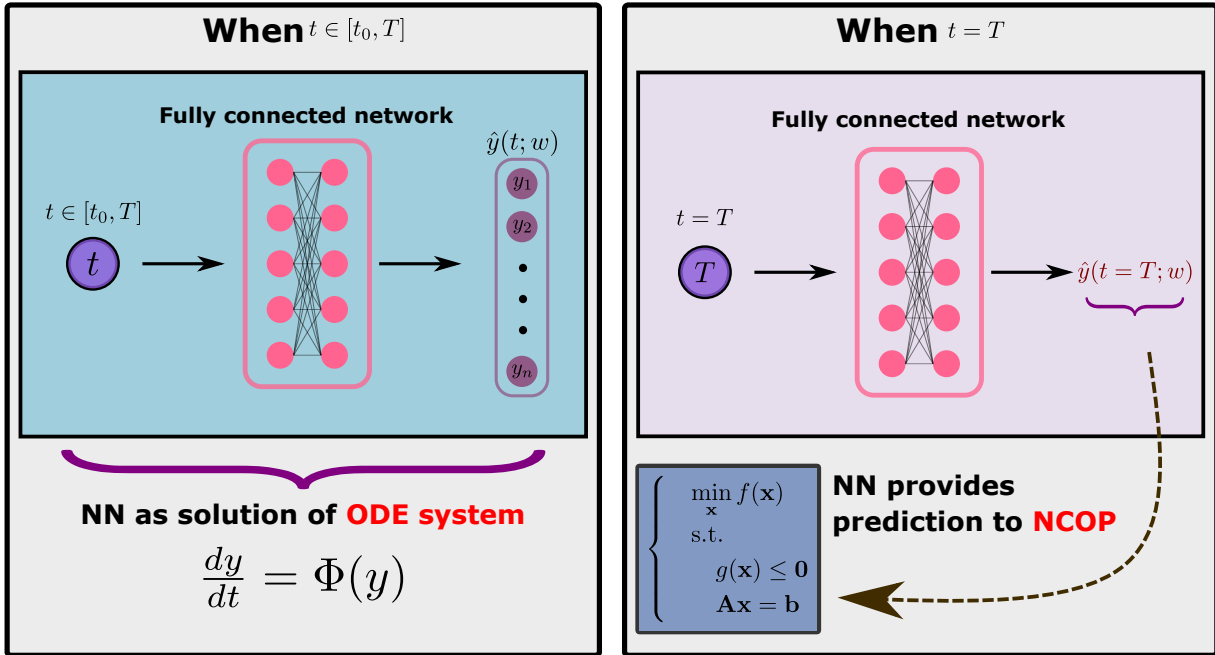


Figure 1: **Neural network solution for the ODE system and the NCOP.** **Left:** When $t \in [t_0, T]$, the model $\hat{y}(t; w)$ itself is considered to be an approximate state solution of the ODE system. **Right:** When $t = T$, the model returns the prediction, $(\hat{x}, \hat{u}) = \hat{y}(t = T; w)$, where \hat{x} and \hat{u} represent the primal and dual predictions for the NCOP, respectively.

Model Description. We propose a modified PINN model to solve the NCOP. Our model can be expressed by the following equation:

$$\hat{y}(t; w) = y_0 + \left(1 - e^{-(t-t_0)}\right) N(t; w), \quad t \in [t_0, T], \quad (10)$$

146 where $N(t; w)$ is a fully connected neural network with trainable parameters w . y_0 is a given initial point for
 147 the ODE system. $[t_0, T]$ is a given time range. The auxiliary function $(1 - e^{-(t-t_0)})$ ensures that the neural
 148 network always satisfies the initial condition $\hat{y}(t = t_0; w) = y_0$ regardless of w .

149 **Improvements to PINN.** The proposed model (10) can be seen as a modified version of the PINN
 150 to adapt to the considered problem. In particular, we make three improvements. First, we use the Lagaris
 151 method to handle the initial conditions of the ODE system (Lagaris et al., 1998). Second, we use only one
 152 neural network to handle all the differential equations of the entire ODE system, whereas, in the original
 153 method, each neural network handles only one differential equation of the system. Finally, we use the
 154 exponential form for the auxiliary function, which has been shown to improve convergence, as demonstrated
 155 in previous studies (Mattheakis et al., 2020). Our experiments confirm this improvement in convergence,
 156 which is likely due to the reduced influence of the neural network as time progresses further from the initial
 157 time.

Approximate State Solution to the ODE. As shown in Figure 1 (Left), the proposed model (10)
 itself is an approximate state solution of the ODE system (3) on the time range $[t_0, T]$, i.e.,

$$\hat{y}(t; w) \approx y(t), \quad t \in [t_0, T], \quad (11)$$

158 where $y(t)$ is the true state solution of the ODE system. While the input time t of the model $\hat{y}(t; w)$ can be
 159 take any real number, we specifically use $\hat{y}(t; w)$ as the solution of the ODE over the time range $[t_0, T]$. As
 160 a result, we restrict the input of $\hat{y}(t; w)$ to the time range $t \in [t_0, T]$.

Predicted Solution to the NCOP. The end state of the proposed model, i.e., $\hat{y}(t = T; w)$, is used
 as the predicted solution to the NCOP (1), as shown in Figure 1 (Right). The following equation shows how
 $\hat{y}(t = T; w)$ approximates the optimal solution y^* :

$$\hat{y}(t = T; w) \approx y(T) \approx y^*. \quad (12)$$

161 Here, $\hat{y}(t = T; w) \approx y(T)$ indicates that the end state of our model approximates the true end state, and
 162 $y(T) \approx y^*$ comes from Theorem 1, indicating that the true end state is the predicted solution of the NCOP.

163 4. Model training

164 4.1. Training Objective

Loss Function. We define the loss function of the proposed model (10) as follows:

$$\mathcal{L}(t, w) = \left\| \frac{\partial \hat{y}(t; w)}{\partial t} - \Phi(\hat{y}(t; w)) \right\|, \quad (13)$$

165 where $\Phi(\cdot)$ refers to the ODE system (3), which corresponds to the NCOP to be solved. $\|\cdot\|$ is the Euclidean
 166 norm. $\Phi(\hat{y}(t; w))$ is the expected derivative according to the ODE system. $\frac{\partial \hat{y}(t; w)}{\partial t}$ is the actual derivative

167 of the model, which can be computed using automatic differentiation tools such as PyTorch or JAX (Paszke
 168 et al., 2019; Bradbury et al., 2018). $\mathcal{L}(t, w)$ represents the difference between the two at time t and with
 169 network parameters w .

170 **Embedding the NCOP into the Loss Function.** The NCOP is integrated into the loss computation
 171 process through the ODE system rather than as a component of the neural network. A neural network is
 172 created as an empty framework without a specific goal to solve a particular NCOP. Instead, by reformulating
 173 the NCOP as an ODE system and embedding it into the loss function, the neural network is trained toward
 174 solving the NCOP.

Objective Function. The goal of training the proposed model is to minimize the following objective
 function:

$$J(w) = \int_{t_0}^T \mathcal{L}(t, w) dt, \quad (14)$$

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

Batch Loss. However, the objective function $J(w)$ is computationally intractable to compute due to
 its integral part. Therefore, in practice, we train the model by minimizing the following batch loss:

$$\mathcal{L}(\mathbb{T}, w) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \mathcal{L}(t, w), \quad (15)$$

178 where \mathbb{T} is a set of randomly sampled time points from the interval $[t_0, T]$, and $|\mathbb{T}|$ denotes the size of this set.
 179 In this way, we can approximate the integral in the objective function $J(w)$ by a sum of loss values over the
 180 set of sampled times. By minimizing the batch loss, we can effectively train the model to solve the NCOP.

181 4.2. Algorithm Design

Objective value under Constraints (OuC) Metric. We introduce an evaluation metric called OuC
 to measure how well a predicted solution, x_{pred} , solves the NCOP:

$$\text{OuC}(x_{\text{pred}}) = \begin{cases} f(x_{\text{pred}}) & \text{if } x_{\text{pred}} \in \Omega, \\ +\infty & \text{otherwise,} \end{cases} \quad (16)$$

182 where Ω is the feasible set defined as $\Omega = \{x | x \leq g(x), Ax = b\}$. The OuC metric evaluates a predicted
 183 solution by setting OuC to positive infinity or a very large real number when the predicted solution is not
 184 feasible, and to the objective value when it is feasible.

Projection Mapping onto Equality Constraints. To increase the likelihood of $\text{OuC}(x_{\text{pred}})$ being a
 real value instead of infinite, we employ the following projection function to map x_{pred} to the set of equality
 constraints,

$$P_{eq}(x_{\text{pred}}) = x_{\text{pred}} - A^T (AA^T)^{-1} (Ax_{\text{pred}} - b). \quad (17)$$

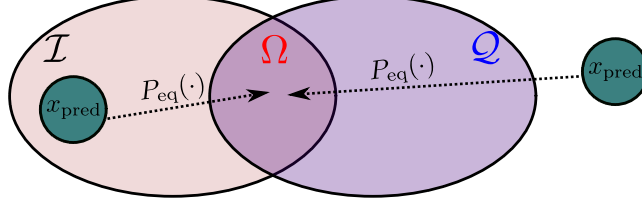


Figure 2: \mathcal{I} and \mathcal{Q} denote the feasible set of inequality constraints and equality constraints, respectively. $\Omega = \mathcal{I} \cap \mathcal{Q}$ denotes the general feasible set of the problem. P_{eq} is a projection function that maps x_{pred} onto \mathcal{Q} .

185 By definition, the evaluation metric $\text{OuC}(x_{\text{pred}})$ attains a real value only when x_{pred} lies within the feasible
 186 set Ω . As illustrated in Figure 2, there are two circumstances when P_{eq} aids in projecting x_{pred} onto the
 187 feasible set:

- 188 • x_{pred} satisfies the inequality constraints but fails to meet the equality constraints, i.e., $x_{\text{pred}} \in \mathcal{I} - \mathcal{Q}$.
- 189 • x_{pred} does not satisfy both the inequality and equality constraints, i.e., $x_{\text{pred}} \notin \mathcal{I} \cup \mathcal{Q}$.

190 In both of the above scenarios, P_{eq} has a reasonable chance of mapping x_{pred} onto the feasible set Ω , resulting
 191 in the value of OuC being a real number rather than infinite.

Algorithm 1: Deep learning solver for convex optimization problems

Input : A COP; A time range $[t_0, T]$; An initial condition (t_0, y_0)

Output: Predicted primal and dual solutions \hat{x}_{best} and \hat{u}_{best} .

```

1 Function Main:
2   ODE system  $\Phi(\cdot) \leftarrow$  the COP
3   Construct a NN model  $\hat{y}(t; w)$ .
4    $(\hat{x}_{\text{best}}, \hat{u}_{\text{best}}) \leftarrow \hat{y}(t = T; w)$ 
5    $\epsilon_{\text{best}} = \text{Evaluate}(\hat{x}_{\text{best}})$ 
6   while  $\text{iter} \leq \text{max iter}$  do
7      $\mathbb{T} \sim U(t_0, T)$ ; ▷ Sample a batch of times,  $\mathbb{T}$ .
8      $\mathcal{L}(\mathbb{T}, w)$ ; ▷ Forward propagation.
9      $w = w - \nabla_w \mathcal{L}(\mathbb{T}, w)$ ; ▷ Update  $w$  through backward propagation.
10     $(\hat{x}_{\text{curr}}, \hat{u}_{\text{curr}}) = \hat{y}(t = T; w)$ ; ▷ Extract the predicted solution from the model.
11     $\hat{x}_{\text{curr}} = P_{eq}(\hat{x}_{\text{curr}})$ ; ▷ Project  $\hat{x}_{\text{curr}}$  onto equality constraints.
12     $\text{OuC}_{\text{curr}} = \text{OuC}(\hat{x}_{\text{curr}})$ ; ▷ Calculate the  $\text{OuC}$  value of  $\hat{x}_{\text{curr}}$ .
13    if  $\text{OuC}_{\text{curr}} < \text{OuC}_{\text{best}}$  then
14       $\text{OuC}_{\text{best}} = \text{OuC}_{\text{curr}}$ ; ▷ Update the best  $\text{OuC}$  value.
15       $(\hat{x}_{\text{best}}, \hat{u}_{\text{best}}) = (\hat{x}_{\text{curr}}, \hat{u}_{\text{curr}})$ ; ▷ Update the best prediction.
16    end
17    return  $(\hat{x}_{\text{best}}, \hat{u}_{\text{best}})$ 
18 end

```

192 **Pipeline.** Algorithm 1 summarizes how to use our proposed method to solve the NCOP. First, we need
 193 to specify an initial condition $y(t_0) = y_0$ and a time range $[t_0, T]$ to construct the IVP. Then, we instantiate
 194 the proposed model (10), which serves as an approximate state solution for this IVP. The model is trained
 195 by performing gradient descent on the batch loss (15) to improve the approximation. Note that our solver is
 196 completely based on the deep learning infrastructure and does not require any standard optimization solver
 197 or numerical integration solver.

198 **Optimal Result Retention (ORR) Mechanism.** A key to Algorithm 1 is that we use an ORR
 199 mechanism based on the OuC metric (16). Specifically, at each iteration, the algorithm compares the OuC
 200 value at the current iteration, denoted as OuC_{curr} , with the best OuC value found so far, denoted as OuC_{best} .
 201 $(\hat{x}_{\text{curr}}, \hat{u}_{\text{curr}})$ and $(\hat{x}_{\text{best}}, \hat{u}_{\text{best}})$ represent the current prediction and the best prediction found so far, respec-
 202 tively. If OuC_{curr} is less than OuC_{best} , it means that the model found a better prediction in this iteration. The
 203 algorithm updates OuC_{best} to equal OuC_{curr} and stores the best prediction as $(\hat{x}_{\text{best}}, \hat{u}_{\text{best}}) = (\hat{x}_{\text{curr}}, \hat{u}_{\text{curr}})$.
 204 This mechanism ensures that the best prediction obtained by the model is maintained throughout the training
 205 process, improving the overall performance of the algorithm.

206 **Neural Network Solution Procedure.** Our approach employs gradient descent on the batch loss (15)
 207 to improve the neural network prediction at each iteration. Assume that the maximum number of training
 208 iterations is M . The evolution of the neural network is represented by $\hat{y}(t; w_1), \hat{y}(t; w_2), \dots, \hat{y}(t; w_M)$, where
 209 w_i and $\hat{y}(t; w_i)$ denote the network parameters and the approximate state solution at the i -th iteration,
 210 respectively. The predicted end states are $\hat{y}(t = T; w_1), \hat{y}(t = T; w_2), \dots, \hat{y}(t = T; w_M)$, where $\hat{y}(t = T; w_i)$
 211 represents the prediction to the NCOP at the i -th iteration.

212 5. Experiments

213 **Neural Network Setup.** To implement our proposed model and the ODE system, we used PyTorch
 214 1.12.1 with CUDA 11.2 (Paszke et al., 2019) and JAX 0.4.1 (Bradbury et al., 2018). Our neural network
 215 architecture consisted of a single layer fully connected network with 100 neurons and a Tanh activation
 216 function. For training, we used the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.001, and
 217 a batch size of 128.

218 **Evaluation metrics.** We used two metrics to measure the performance of the model: 1) The OuC
 219 metric, as defined in equation (16). 2) Computational time, which measures the time required to achieve a
 220 desired accuracy.

221 5.1. Comparisons with Numerical Integration Methods

222 In this subsection, we compare our proposed method with six classical numerical integration methods:
 223 Runge-Kutta 45 (RK45), Runge-Kutta 23 (RK23), Dormand-Prince 853 (DOP853), Backward Differentiation
 224 Formula (BDF), Radau, and LSODA. All of these methods are available in the Scipy library (Virtanen et al.,
 225 2020). The RK45, RK23, and DOP853 are explicit Runge-Kutta methods, while the BDF and Radau are
 226 implicit methods. LSODA is an adaptive method that automatically switches between explicit and implicit
 227 methods depending on the stiffness of the ODE system.

Example 1: We aim to solve the following NCOP:

$$\begin{aligned} \min_x f(x) &= 10(x_1 + x_2)^2 + (x_1 - 2)^2 + 20|x_3 - 3| + e^{x_3} \\ \text{s.t.} \quad & \\ g(x) &= (x_1 + 3)^2 + x_2 - 36 \leq 0 \\ h(x) &= 2x_1 + 3x_2 + 5x_3 - 7 = 0. \end{aligned} \tag{18}$$

228 The feasible set of this problem is convex, and the objective function is convex but non-smooth due to its
 229 inclusion of absolute values.

230 **Construction of IVPs.** We model the problem (18) by the ODE system (3) and set the time range
 231 as $[t_0, T] = [0, 10]$. We choose three initial points to study, namely $[0, 0, 0, 0]$, $[1, 0, -2, 3]$, and $[-1, 1, -1, 1]$,
 232 which result in three IVPs. Based on these three initial points, we construct each of the three proposed
 233 neural networks (10) as approximate state solutions to the IVPs.

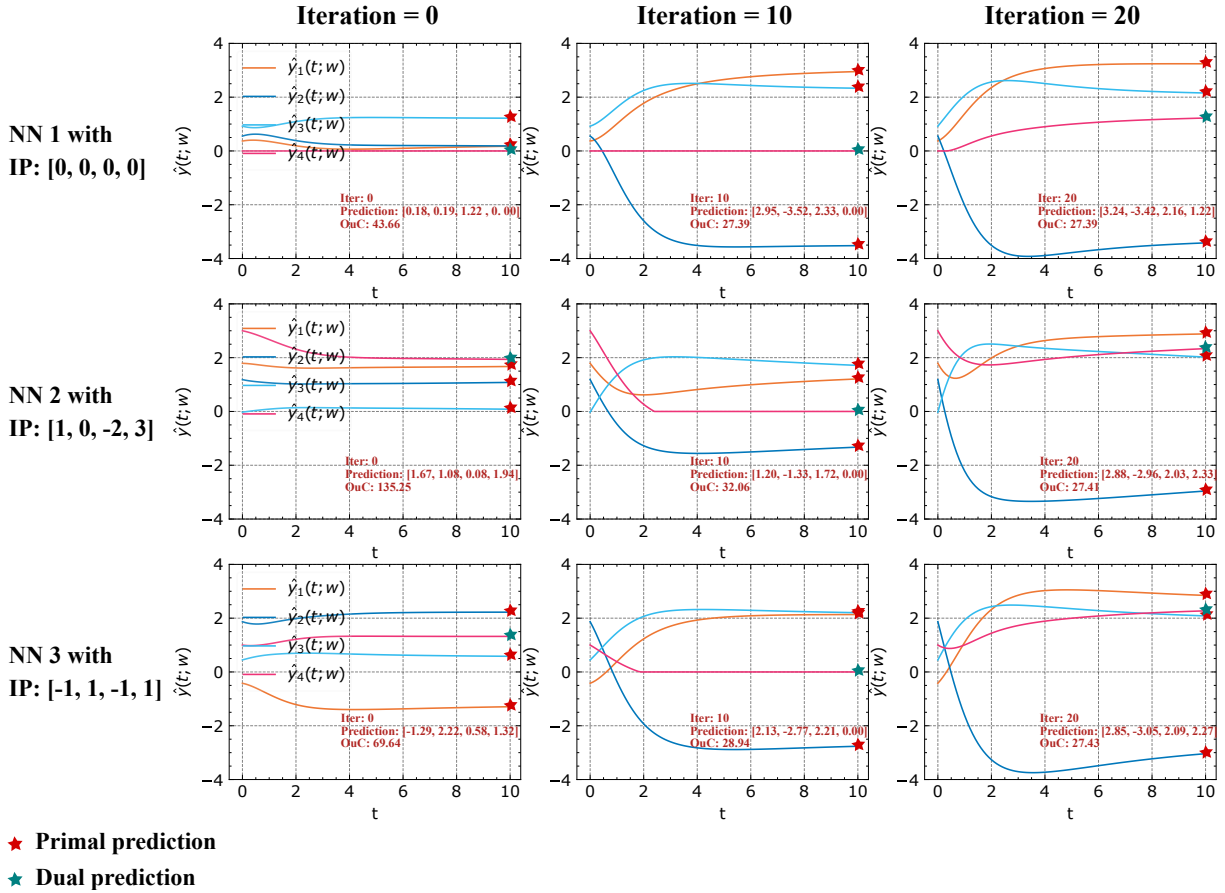


Figure 3: Neural network solutions to problem (18). The three neural networks are initialized with three different initial points (IPs). Each row shows a neural network as a function at selected training iterations.

Neural Network Solution. Figure 3 shows the solution process of our approach. Each horizontal row represents the evolution of a neural network trained by Algorithm 1. Each sub-figure shows $\hat{y}(t; w)$

for $t \in [0, 10]$, which represents the approximate state solution of the IVP at a given training iteration. Here, we must emphasize that $\hat{y}(t; w)$ is implemented by one neural network with four output units, and $\hat{y}(t; w) = (\hat{y}_1(t; w), \hat{y}_2(t; w), \hat{y}_3(t; w), \hat{y}_4(t; w))$. In particular, the end state has

$$\hat{y}(t = 10; w) = \left(\underbrace{\hat{y}_1(t = 10; w), \hat{y}_2(t = 10; w), \hat{y}_3(t = 10; w)}_{=\hat{x}}, \underbrace{\hat{y}_4(t = 10; w)}_{=\hat{u}} \right), \quad (19)$$

where \hat{x} and \hat{u} represent the predicted primal and dual solutions, respectively, of the problem (18).

Evolution of the Predictions. As discussed in Section 4.2, the end state prediction $\hat{y}(t = 10; w)$ is improved by training on the entire approximate state solution. At training iteration 0, the approximate state solution is far from the true state solution, resulting in a high OuC value of the endpoint prediction. After 20 training iterations, the approximate state solution gets closer to the true state solution, and the endpoint prediction improves significantly. Notably, as these networks are constructed with different initial conditions, they have varying initial OuC values. Nonetheless, our proposed algorithm ensures that all networks converge to a prediction with an OuC value of less than 28 after 20 training iterations.

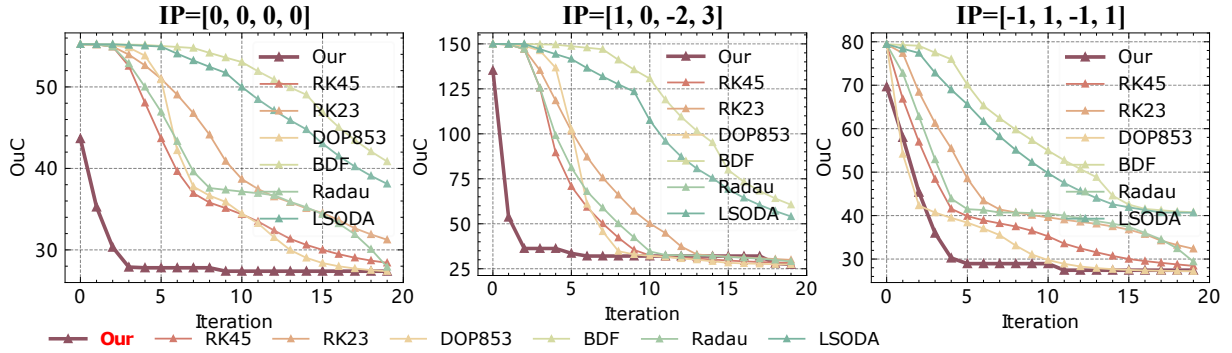


Figure 4: Comparison of our proposed method with the numerical integration methods on the OuC metric. The OuC metric is defined in Equation (16).

OuC Performance. Figure 4 presents a comparative analysis of the OuC drop rates for our proposed method and six traditional numerical integration methods over three different initial points. The following observations can be made:

- Our method outperforms traditional numerical integration methods in terms of OuC reduction, as evidenced by the lower OuC values achieved in fewer iterations. For example, for the initial point (IP) $[0, 0, 0, 0]$, our method reduces OuC from 43.66 to 28 in just five iterations, whereas the best-performing numerical integration methods, namely RK45, DOP853, and Radau, require 20 iterations to achieve comparable results. Similar results are observed for the other two initial points.
- The speed of convergence varies for different initial points, with our method showing greater robustness to different initial settings compared to numerical integration methods. While most numerical integration methods converge faster for the IP $[1, 0, -2, 3]$ and slower for $[0, 0, 0, 0]$, our method converges at

253 approximately the same rate for both.

- 254 • The starting OuC values differ for various initial points, with the IP $[1, 0, -2, 3]$ having the highest
 255 starting OuC value of around 135, and $[0, 0, 0, 0]$ having the lowest starting OuC of approximately
 256 43. However, the results demonstrate that the OuC values do not significantly affect the speed of
 257 convergence.

258 **Why Our Approach has Better OuC than RK45.** It is important to emphasize that our approach
 259 has no advantage over classical numerical integrators such as RK45 when it comes to solving for the full
 260 solution on the IVP, i.e. the function y . As shown in Figure 4, our method outperforms these numerical
 261 integrators in the OuC metric because our approach focuses on improving the end point $\hat{y}(t = T; w)$, rather
 262 than the entirety of the function \hat{y} . It is also worth noting that the OuC performance metric only considers
 263 the endpoint $\hat{y}(t = T; w)$, not the entire function \hat{y} . The methodology we propose, described in sections 3
 264 and 4, is deliberately designed with this specific goal in mind.

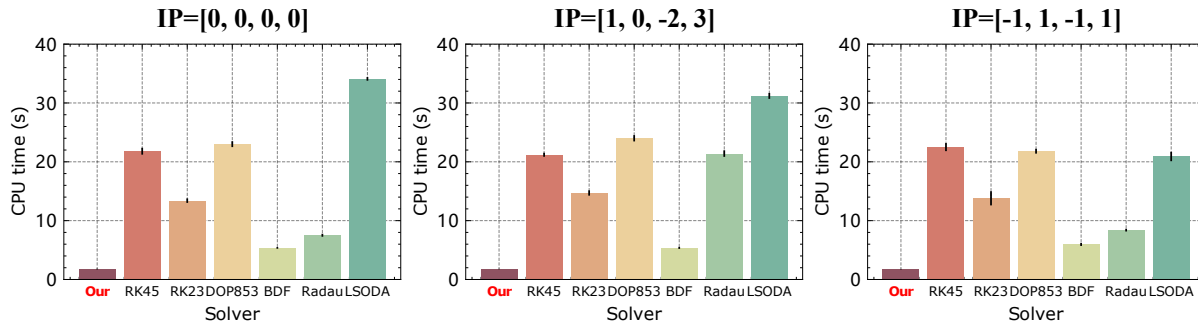


Figure 5: Comparison of our method with the numerical integration methods in terms of computational efficiency. Time is measured in seconds.

265 **Computational Time Performance.** Figure 5 shows the time needed by different solution methods
 266 to obtain an acceptable solution (i.e., $\text{OuC} \leq 28$) to Problem (18). Our proposed method outperforms all
 267 considered numerical integration methods in terms of computational efficiency. The most efficient numerical
 268 integration method, BDF, still requires 10 times more computational time than our method to obtain a
 269 satisfactory solution. Moreover, the computational time of our method is less affected by different IP settings,
 270 requiring about 1.78 seconds for all three initial points. In contrast, some numerical integration methods,
 271 such as Radau and LSODA, show significant variations in computation time at different initial points.

272 5.2. Comparisons with PINN

273 In this subsection, we perform an ablation study comparing the proposed method with PINN (Raissi
 274 et al., 2019) and Lagaris method (Lagaris et al., 1998) to validate the effectiveness of our method.

Example 2:

$$\begin{aligned} \min_x f(x) &= |2.3x_1 + x_3 - 3.5| + |x_2 + 2x_3 - 1.8| + |1.3x_1 + x_2 + x_3 + 3| \\ \text{s.t.} \\ g(x) &= x_1^2 - x_2 + x_3 + 3 \leq 0 \\ h_1(x) &= x_1 + x_2 + x_3 = 0, \\ h_2(x) &= 2x_2 + x_3 = 0. \end{aligned} \tag{20}$$

275 We aim to solve the NCOP (20). The problem has a convex feasible set and a convex but non-smooth
 276 objective function due to its inclusion of absolute values.

277 **Construction of IVPs.** We model problem (20) as an ODE system (3) and set the time range as
 278 $[t_0, T] = [0, 10]$. To construct three IVPs, we choose three initial points, namely $[1, -1, 0, 3]$, $[2, 3, -2, 1]$, and
 279 $[2, -2, 1, -2]$. Based on these initial points, we instantiate three proposed neural networks (10) as approximate
 280 state solutions.

281 **Experimental Setup.** We compare our proposed approach with two methods: vanilla PINN and PINN
 282 with Lagaris construction method (PINN+Lagaris). Our approach can be regarded as the PINN+Lagaris
 283 method enhanced by the ORR mechanism (PINN+Lagaris+ORR). The hyperparameters and training details
 284 are the same as those in Section 5.1.

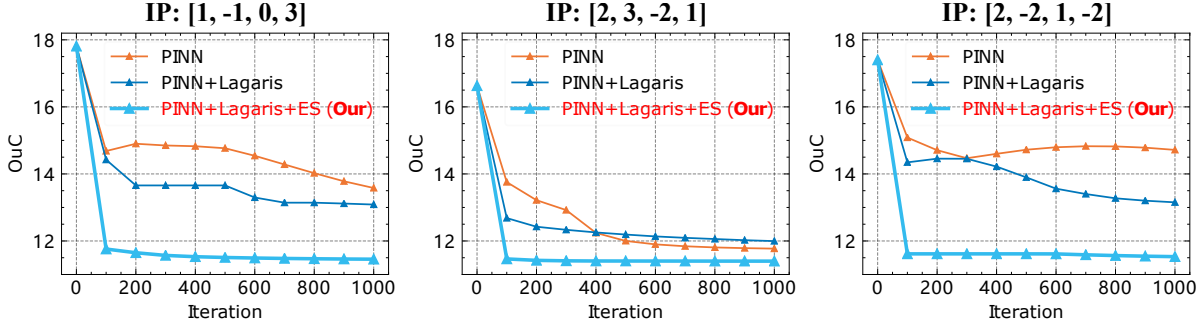


Figure 6: Comparison of our proposed method with PINN and the PINN+Lagaris method on the OuC metric. The experiment is conducted on Problem (20)

285 Table 1 shows the performance of the three methods over the first one hundred iterations, while Figure 6
 286 shows their convergence behavior over the first one thousand iterations. The results reveal the following key
 287 observations:

- 288 • Our proposed approach yields an excellent predicted solution within the first 20 training iterations,
 289 consistent with the results presented in Section 5.1. In contrast, even after 1000 iterations, neither the
 290 PINN nor the PINN+Lagaris methods achieves a predicted solution that compares favorably to that
 291 of our approach.
- 292 • Our method has a higher probability of obtaining feasible solutions. As shown in Table 1, the PINN

Initial point: [1, -1, 0, 3]						
Iteration	PINN		PINN+Lagaris		PINN+Lagaris+EC	
	Predicted solution	OuC	Predicted solution	OuC	Predicted solution	OuC
0	[-0.64 1.36 -0.71 0.47]	inf	[-1.00 1.00 -0.00 3.09]	inf	[-1.00 1.00 -0.00 3.09]	inf
5	[-0.40 1.60 -1.19 1.10]	inf	[-1.42 0.58 0.84 7.02]	inf	[0.05 2.05 -2.09 4.85]	12.44
10	[-0.31 1.69 -1.37 2.17]	inf	[0.16 2.16 -2.31 9.50]	12.77	[-0.18 1.82 -1.64 9.25]	11.76
20	[0.81 2.81 -3.63 3.53]	14.74	[1.43 3.43 -4.86 6.63]	16.60	[-0.18 1.82 -1.64 9.25]	11.76
40	[0.72 2.72 -3.44 5.03]	14.45	[0.29 2.29 -2.58 1.91]	13.17	[-0.18 1.82 -1.64 9.25]	11.76
60	[0.74 2.74 -3.48 4.80]	14.52	[1.12 3.12 -4.25 2.18]	15.67	[-0.18 1.82 -1.64 9.25]	11.76
80	[0.84 2.84 -3.68 4.35]	14.82	[0.45 2.45 -2.90 1.48]	13.65	[-0.18 1.82 -1.64 9.25]	11.76
100	[0.82 2.82 -3.64 3.96]	14.76	[0.19 2.19 -2.39 0.72]	12.88	[-0.18 1.82 -1.64 9.25]	11.76

Initial point: [2, 3, -2, 1]						
Iteration	PINN		PINN+Lagaris		PINN+Lagaris+EC	
	Predicted solution	OuC	Predicted solution	OuC	Predicted solution	OuC
0	[-1.00 1.00 0.00 0.00]	inf	[0.41 2.41 -2.82 0.23]	13.54	[0.41 2.41 -2.82 0.23]	13.54
5	[-1.07 0.93 0.13 0.64]	inf	[-0.77 1.23 -0.46 1.52]	inf	[-0.29 1.71 -1.41 1.77]	11.42
10	[-1.42 0.58 0.85 2.04]	inf	[0.04 2.04 -2.08 1.39]	12.42	[-0.29 1.71 -1.41 1.77]	11.42
20	[-1.44 0.56 0.87 4.04]	inf	[0.85 2.85 -3.69 2.26]	14.84	[-0.29 1.71 -1.41 1.77]	11.42
40	[0.89 2.89 -3.77 3.24]	14.96	[0.26 2.26 -2.53 1.21]	13.09	[-0.29 1.71 -1.41 1.77]	11.42
60	[0.56 2.56 -3.11 1.69]	13.97	[-0.29 1.71 -1.43 0.25]	11.44	[-0.29 1.71 -1.41 1.77]	11.42
80	[0.44 2.44 -2.88 1.79]	13.63	[-0.62 1.38 -0.75 0.71]	inf	[-0.30 1.70 -1.40 1.19]	11.40
100	[0.01 2.01 -2.02 1.05]	12.33	[-0.09 1.91 -1.83 0.88]	12.04	[-0.30 1.70 -1.40 1.19]	11.40

Initial point: [2, -2, 1, -2]						
Iteration	PINN		PINN+Lagaris		PINN+Lagaris+EC	
	Predicted solution	OuC	Predicted solution	OuC	Predicted solution	OuC
0	[-0.65 1.35 -0.7 0.]	inf	[-1.05 0.95 0.1 0.]	inf	[-1.05 0.95 0.1 0.]	inf
5	[-0.68 1.32 -0.65 0.44]	inf	[-2.78 -0.78 3.56 1.67]	inf	[-0.10 1.90 -1.80 6.38]	12.01
10	[-1.53 0.47 1.05 2.08]	inf	[-1.21 0.79 0.41 5.23]	inf	[-0.10 1.90 -1.80 6.38]	12.01
20	[-1.66 0.34 1.32 5.54]	inf	[2.03 4.03 -6.06 9.]	18.39	[-0.23 1.77 -1.54 6.37]	11.62
40	[0.42 2.42 -2.84 6.84]	13.56	[0.53 2.53 -3.05 3.46]	13.88	[-0.23 1.77 -1.54 6.37]	11.62
60	[1.24 3.24 -4.47 6.27]	16.01	[0.67 2.67 -3.34 5.02]	14.31	[-0.23 1.77 -1.54 6.37]	11.62
80	[1.3 3.3 -4.6 5.42]	16.2	[0.63 2.63 -3.25 3.59]	14.18	[-0.23 1.77 -1.54 6.37]	11.62
100	[1.27 3.27 -4.54 4.92]	16.11	[0.59 2.59 -3.19 3.69]	14.08	[-0.23 1.77 -1.54 6.37]	11.62

Table 1: Comparison of PINN, PINN+Lagaris, and PINN+Lagaris+EC (Our approach) for three different initial points with predicted solutions and corresponding OuC values. inf indicates that the predicted solution is not in the feasible set.

method returns 'inf' 11 times for the three IP configurations, while the Lagaris method reduces this occurrence to 7. In contrast, our method returns 'inf' only twice, both times in the first round, indicating that it can reach a feasible solution more quickly.

- Neither the PINN method nor the PINN+Lagaris method maintains an optimal solution during the optimization process. As shown in Table 1, the PINN method and the PINN+Lagaris method achieve good OuC values of 12.33 and 12.04, respectively, for the IP $[2, 3, -2, 1]$ at the 100th iteration. However, neither method maintains this level of performance, and their OuC values increase in subsequent iterations.
- The Lagaris method can improve the performance of vanilla PINN, as shown in Figure 6. However, this improvement is not substantial and varies depending on the IP configuration. For example, the improvement is significant for the first and third initial points but negligible for the second IP.

5.3. Hyperparameter Study

In this subsection, we perform a hyperparameter study on the following NCOP.

Example 3:

$$\begin{aligned}
 \min_x f(x) &= \|Cx - d\|_1 \\
 \text{s.t.} & \\
 g_1(x) &= x_1^2 - x_2 + x_3 + x_5 - x_8 - 10 \leq 0, \\
 g_2(x) &= |x_1 - x_3 + x_4 + x_7| - 4.8 \leq 0, \\
 h(x) &= x_1 + x_3 + x_5 + x_7 - 1 = 0,
 \end{aligned} \tag{21}$$

where $\|\cdot\|_1$ denotes the L^1 norm, and

$$C = \begin{pmatrix} 1 & 4 & 2 & 2 & 1.3 & 4 & 2 & 1 \\ 2.8 & 2 & 1.6 & 3.2 & 0 & 2 & 1 & 1 \\ 1 & 4 & 2.3 & 2 & 2.5 & 0 & 5 & 1 \\ 1 & 1 & 1 & 3.1 & 2.3 & 0 & 0.8 & 1 \end{pmatrix}, \quad d = \begin{pmatrix} 1.5 \\ -3.8 \\ 6.2 \\ 7.5 \end{pmatrix}. \tag{22}$$

Example 3 involves a nonsmooth objective function and a nonsmooth inequality constraint $g_2(x)$.

To set up the algorithm, we choose the IP as an all-ones vector and the time range as $[0, 10]$. In the following, we discuss the computational performance for different neural network sizes and different learning rates.

Model Size. In Figures 7 (A) and (B), we investigate the computational performance of neural networks with various widths and depths, and the optimal result is obtained with a 700-neuron-wide, single-layer structure. In (A), with a maximum of 1000 training iterations, networks with fewer neurons (such as 100, 300, and 500) underperform due to their model capacity, limiting further improvements in OuC even with

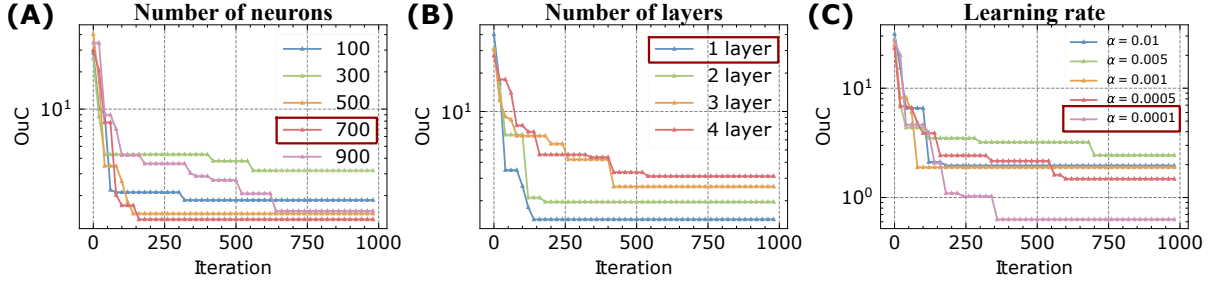


Figure 7: Hyperparameter study. (A): OuC performance on different numbers of neurons in a single layer neural network with a learning rate of 0.01. (B): OuC performance of different layers in a multilayer neural network with 500 neurons per layer and a learning rate of 0.01. (C): OuC performance at different learning rates in a two-layer neural network with 500 neurons per layer.

314 more training. Conversely, a network with more neurons (such as 900) shows underperformance, likely due
 315 to insufficient model training, and its OuC would potentially improve with additional training. In (B),
 316 a single layer neural network is shown to outperform other configurations. Taken together, these results
 317 underscore the need to find the most appropriate network structure for a given NCOP problem. An overly
 318 complex network would require an excessive amount of computing resources for optimization that may not
 319 be necessary. Conversely, a network that is too simple would not find the appropriate solution, regardless of
 320 the amount of training. Thus, the size of the neural network should be determined by factors related to the
 321 NCOP being solved. These include the number of decision variables, the constraints, and the complexity of
 322 both the objective and constraint functions.

323 **Learning Rate.** Figure 7(C) shows the performance of neural networks trained with different learning
 324 rates. At iteration 1000, the optimal performance is observed at the learning rate of $\alpha = 0.0001$. It is
 325 important to note that if we zoom into the first 100 iterations, a larger learning rate $\alpha = 0.001$ is more
 326 effective. This suggests that the choice of learning rate should depend on the actual preferences of the user.
 327 The advantage of a large learning rate is that it can find a better prediction for the NCOP in a short time,
 328 while the disadvantage is that it performs poorly in the long run. In contrast, a small learning rate finds
 329 better solutions in the long run. Overall, the choice of learning rate should be determined by the user's
 330 specific requirements for speed and accuracy.

331 5.4. L^1 Norm Minimization Problem

332 Consider the following NCOP problem:

Example 4:

$$\begin{aligned}
 \min_x f(x) &= \|x\|_1 \\
 \text{s.t.} & \\
 g_i(x) &= x_{10*(i-1)+1}^2 + x_{10*(i-1)+2}^2 + \dots + x_{10*(i-1)+10}^2 - 20 \leq 0, \quad i = 1, 2, \dots, 300 \\
 h(x) &= Ax - b = 0,
 \end{aligned} \tag{23}$$

Description	Initial $f(x)$	Initial $\max_{i=1,2,\dots,300} (g_i(x))$	Initial $h(x)$	Initial OuC	OuC at iteration 100	OuC at iteration 1000	OuC at iteration 3000	OuC at iteration 10000
IP1 All-ones vector: (1, 1, ..., 1)	3000.0	-10.0	5984.	inf	133.92	7.08	6.31	5.81
IP2 All-threes vector: (3, 3, ..., 3)	9000.0	70	17984	inf	202	118	25	6.00
IP3 All-negative ones vector: (-1, -1, ..., -1)	3000.0	-10.0	-6016	inf	166.10	8.65	6.23	6.23
IP4 Alternating sequence of 2 and -2: (2, -2, ..., 2, -2)	6000.0	20	-16	inf	643	16.62	16.62	16.62
IP5 First half entries are 1 and the rest are 3: (1, 1, ..., 3)	5700.0	70	14084	inf	946	380	344	95

Table 2: Description of IPs and their OuC performance at different algorithm iterations. Columns 2 to 6 describe the IPs and the their initial information, and columns 7 to 10 describe their OuC values at different training iterations.

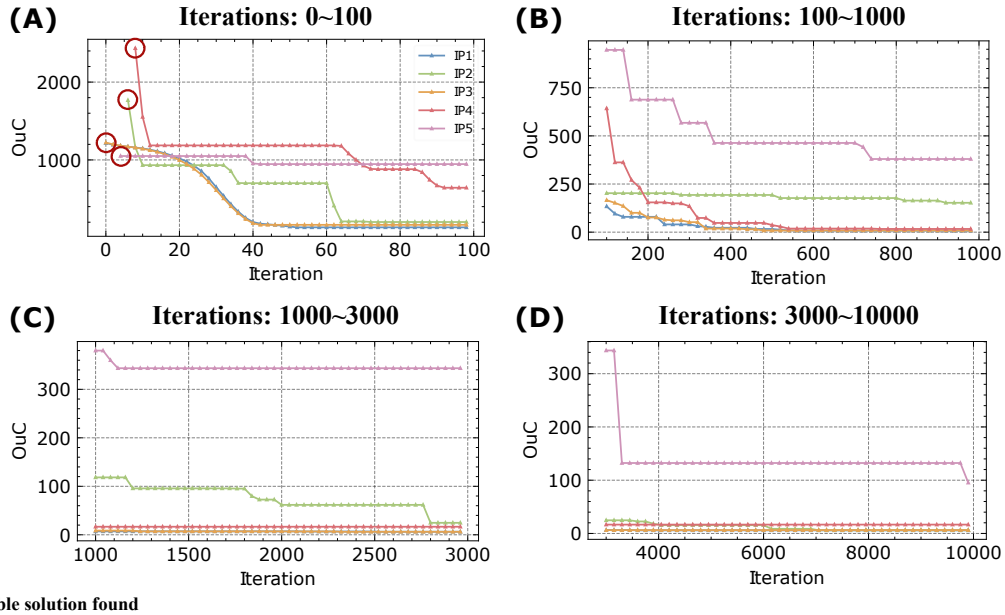


Figure 8: OuC performance with various IP configurations. The detailed descriptions for the five IPs are given in Table 2. In (A), the red circle indicates the first time a feasible solution is found. (A), (B), (C), and (D) show the results of the algorithm iterations 0~100, 100~1000, 1000~3000, and 3000~10000, respectively.

333 where $x \in \mathbb{R}^{3000}$, $A \in \mathbb{R}^{1 \times 3000}$, with the first half entries of A being 1 and the rest 3, and $b = 16$.

334 **IPs Description.** We examine five different IPs, which are listed in Table 2. The IPs are used to
335 configure Algorithm 1 to solve problem (23). As shown in the table, all the five IPs have large initial
336 objective values, with IP2, IP4, and IP5 failing to satisfy the inequality constraint, i.e., $\max_{i=1,2,\dots,300} (g_i(x)) \geq 0$,
337 and all IPs failing to meet the equality constraint, i.e., $h(x) \neq 0$. These observations indicate that the IPs
338 initially do not solve Example 4 well and are far from the optimal solution.

339 Based on the five IPs, the OuC values at different algorithm iterations are shown in Table 2 and Figure
340 8. We observe that:

- 341 • As shown in Figure 8-(A), our algorithm quickly finds a feasible solution that satisfies both the inequality
342 and equality constraints. Moreover, once the first feasible solution is found, the subsequent solutions
343 given by the algorithm are all within the feasible set.

- The final solutions given by the algorithm are acceptable. After going through the entire solution process, the OuC values associated with the IPs decrease significantly. For IP1, the OuC decreases from 3000 to 5.81 (100% \rightarrow 0.2%), and similar results can be found for other IPs. Given the fact that problem (23) has a known lower bound 0 for the objective value. This indicates that the final solutions produced by the proposed algorithm are already very close to the optimal solution.
- The OuC decreasing speed or convergence rate varies under different IP configuration. The convergence rate and final result of IP1~IP4 are significantly better than those of IP5. This may be because IP5 is the farthest from the optimal solution, and thus requires a larger time range and more model training. Nevertheless, the proposed algorithm still significantly improves the OuC performance of IP5 (100% \rightarrow 1.6%) with the given experimental setup.
- Most of the decrease in OuC values occurs in the first 1000 iterations. In particular, IP1, IP2, and IP4 reduce the OuC values to about 10 within only 1000 iterations, which is already very close to the final result, demonstrating the efficiency of the algorithm.

5.5. NCOP Problem Set

Problem Set Description. We construct a set of NCOPs based on Example 4 (23), where each NCOP problem takes the following form:

$$\begin{aligned}
 \min_x f(x) &= \|x\|_1 \\
 \text{s.t.} & \\
 g_i(x) &= x_{10*(i-1)+1}^2 + x_{10*(i-1)+2}^2 + \dots + x_{10*(i-1)+10}^2 - c^{(k)} \leq 0, \quad i = 1, 2, \dots, 100 \\
 h(x) &= A^{(k)}x - b^{(k)} = 0,
 \end{aligned} \tag{24}$$

where $x \in \mathbb{R}^{1000}$, $A^{(k)} \in \mathbb{R}^{1000}$, $b^{(k)} \in \mathbb{R}$, and $c^{(k)} \in \mathbb{R}$. $A^{(k)}$, $b^{(k)}$, $c^{(k)}$ are sampled from uniform distributions $U(1, 5)$, $U(10, 20)$, $U(20, 30)$, respectively. We randomly generate 100 different problem data $\{(A^{(k)}, b^{(k)}, c^{(k)})\}$ to form 100 different NCOPs. These problem datasets $\{(A^{(k)}, b^{(k)}, c^{(k)})\}_{k=1}^{100}$ can be accessed from the link¹. Consistent with previous experimental subsections, we set the time range for all NCOPs to $[0, 10]$ and the IP y_0 as an all-one vector.

Figure 9 shows the resolution of these 100 NCOPs using our neural network approach, and Table 3 shows the statistical information for these OuC results at different iterations. A clear trend can be seen is that all the OuC values decrease as training progresses. Starting from a mean of 372.79, the OuC value drops to 24.00 after 1000 iterations (a reduction from 100% to 6.4%). This trend of decreasing OuC values is not only observed at the mean, but also consistently observed at the 25%, 50%, 75%, and 90% quantiles. Of particular note is the impressive magnitude of this reduction. The significant reduction in OuC values

¹https://drive.google.com/drive/folders/1D_3HP-fBp9tew4IgdRoIQtgVb8vU-vG0?usp=drive_link

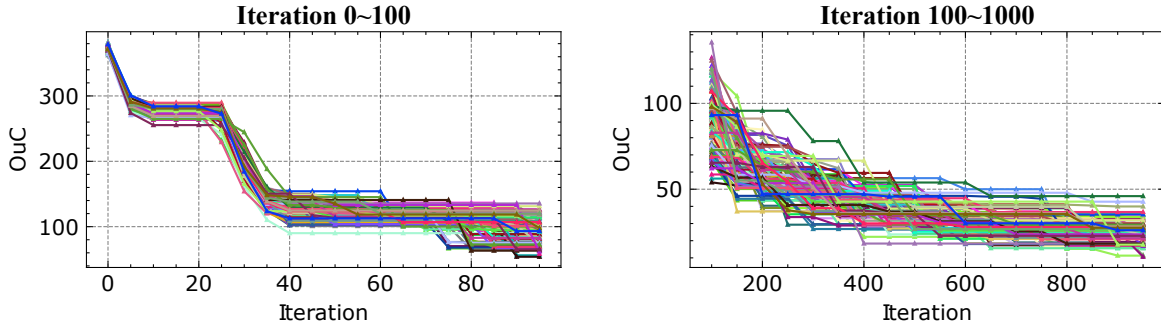


Figure 9: OuC values of the 100 NCOPs at different training iterations. Left and right show the results of iterations 0~100 and 100~1000, respectively.

Iteration	Mean	STD	25% quantile	50% quantile	75% quantile	90% quantile
0	372.79	3.93	370.11	372.46	375.05	378.15
20	277.87	6.21	274.28	278.69	282.21	284.90
40	130.84	14.12	121.48	130.32	142.55	148.95
60	120.27	9.91	114.34	120.55	126.57	129.90
80	108.60	18.59	102.61	113.15	121.67	127.45
100	89.10	18.81	72.78	88.25	101.73	117.89
300	49.08	9.55	43.14	48.21	53.61	61.63
500	37.23	7.61	32.01	36.37	42.91	46.19
700	30.13	6.95	25.86	29.85	34.31	39.05
999	24.00	6.17	20.73	23.64	27.64	30.35

Table 3: Statistical data of OuC values for 100 NCOPs at different training iterations. The table shows the mean, standard deviation (STD), and values at the 25%, 50% (median), 75%, and 90% quantiles of the OuC distribution at each iteration.

369 indicates that our method efficiently navigates the solution space, making steady progress towards optimality.
 370 This highlights the potential of our approach for tackling a wide variety of NCOPs. In summary, these results
 371 provide a strong indication of the effectiveness of our proposed method for solving NCOPs, demonstrating
 372 its robustness and efficiency over a wide range of problem datasets.

373 6. Conclusion

374 In this study, we present a deep learning-based methodology for solving NCOPs. The proposed method-
 375 ology is a fruitful fusion of neurodynamic optimization and PINNs. Methodologically, we have extended
 376 the PINN approach to accommodate neurodynamic optimization. In addition, we have developed a novel
 377 training algorithm that increases computational efficiency by exploiting the problem structure of NCOPs.
 378 Experimental results have demonstrated the effectiveness of the proposed method on a number of NCOPs.
 379 The computational performance can be further improved by tuning the hyperparameters and refining the
 380 training details.

381 In addition, our results have identified several avenues for future research. Specifically, we recommend
 382 investigating better methods for selecting initial points and time ranges, exploring different network ar-
 383 chitectures and advanced neurodynamic optimization techniques. Further development in these areas will
 384 undoubtedly improve the effectiveness and robustness of our approach, making it an important tool for

385 addressing NCOPs in diverse applications.

386 **Bibliography**

387 Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in
388 machine learning: a survey. *Journal of machine learning research*, *18*.

389 Bian, W., Ma, L., Qin, S., & Xue, X. (2018). Neural network for nonsmooth pseudoconvex optimization with
390 general convex constraints. *Neural Networks*, *101*, 1–14.

391 Biloš, M., Sommer, J., Rangapuram, S. S., Januschowski, T., & Günnemann, S. (2021). Neural flows: Efficient
392 alternative to neural odes. *Advances in Neural Information Processing Systems*, *34*, 21325–21337.

393 Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke,
394 A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). JAX: composable transformations of
395 Python+NumPy programs. URL: <http://github.com/google/jax>.

396 Burden, R. L., Faires, J. D., & Burden, A. M. (2015). *Numerical analysis*. Cengage learning.

397 Che, H., & Wang, J. (2018). A two-timescale duplex neurodynamic approach to biconvex optimization. *IEEE*
398 *Transactions on Neural Networks and Learning Systems*, *30*, 2503–2514.

399 Che, H., & Wang, J. (2019). A collaborative neurodynamic approach to global and combinatorial optimiza-
400 tion. *Neural Networks*, *114*, 15–27.

401 Chen, F., Sondak, D., Protopapas, P., Mattheakis, M., Liu, S., Agarwal, D., & Giovanni, M. D. (2020).
402 NeuroDiffEq: A Python package for solving differential equations with neural networks. *Journal of Open*
403 *Source Software*, *5*, 1931. URL: <https://doi.org/10.21105/joss.01931>. doi:10.21105/joss.01931.

404 Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equa-
405 tions. *Advances in neural information processing systems*, *31*.

406 De Ryck, T., Jagtap, A. D., & Mishra, S. (2022). Error estimates for physics informed neural networks
407 approximating the navier-stokes equations. *arXiv preprint arXiv:2203.09346*, .

408 Guo, Z., Liu, Q., & Wang, J. (2011). A one-layer recurrent neural network for pseudoconvex optimization
409 subject to linear equality constraints. *IEEE Transactions on Neural Networks*, *22*, 1892–1900.

410 Hopfield, J. J., & Tank, D. W. (1985). “Neural” computation of decisions in optimization problems. *Biological*
411 *cybernetics*, *52*, 141–152.

412 Hou, Z.-G., Cheng, L., Tan, M., & Wang, X. (2010). Distributed adaptive coordinated control of multi-
413 manipulator systems using neural networks. *Robot intelligence: An advanced knowledge processing ap-
414 proach*, (pp. 49–69).

- 415 Hu, Z., Jagtap, A. D., Karniadakis, G. E., & Kawaguchi, K. (2022a). Augmented physics-informed neu-
416 ral networks (apinns): A gating network-based soft domain decomposition methodology. *arXiv preprint*
417 *arXiv:2211.08939*, .
- 418 Hu, Z., Jagtap, A. D., Karniadakis, G. E., & Kawaguchi, K. (2022b). When do extended physics-informed
419 neural networks (xpinns) improve generalization? *SIAM Journal on Scientific Computing*, *44*, A3158–
420 A3182.
- 421 Jagtap, A. D., & Karniadakis, G. E. (2023). How important are activation functions in regression and
422 classification? a survey, performance comparison, and future directions. *Journal of Machine Learning for*
423 *Modeling and Computing*, *4*.
- 424 Jagtap, A. D., Kawaguchi, K., & Em Karniadakis, G. (2020a). Locally adaptive activation functions with
425 slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A*, *476*,
426 20200334.
- 427 Jagtap, A. D., Kawaguchi, K., & Karniadakis, G. E. (2020b). Adaptive activation functions accelerate
428 convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, *404*, 109136.
- 429 Jagtap, A. D., Kharazmi, E., & Karniadakis, G. E. (2020c). Conservative physics-informed neural networks on
430 discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods*
431 *in Applied Mechanics and Engineering*, *365*, 113028.
- 432 Jagtap, A. D., Mao, Z., Adams, N., & Karniadakis, G. E. (2022a). Physics-informed neural networks for
433 inverse problems in supersonic flows. *Journal of Computational Physics*, *466*, 111402.
- 434 Jagtap, A. D., Shin, Y., Kawaguchi, K., & Karniadakis, G. E. (2022b). Deep kronecker neural networks: A
435 general framework for neural networks with adaptive activation functions. *Neurocomputing*, *468*, 165–180.
- 436 Jia, W., Liu, N., & Qin, S. (2021). An adaptive continuous-time algorithm for nonsmooth convex resource
437 allocation optimization. *IEEE Transactions on Automatic Control*, *67*, 6038–6044.
- 438 Kennedy, M. P., & Chua, L. O. (1988). Neural Networks for Nonlinear Programming. *IEEE Transactions*
439 *on Circuits and Systems*, *35*, 554–562. doi:10.1109/31.1783.
- 440 Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint*
441 *arXiv:1412.6980*, .
- 442 Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial
443 differential equations. *IEEE Transactions on Neural Networks*, *9*, 987–1000. doi:10.1109/72.712178.
- 444 Liu, N., Wang, J., & Qin, S. (2022). A one-layer recurrent neural network for nonsmooth pseudoconvex
445 optimization with quasiconvex inequality and affine equality constraints. *Neural Networks*, *147*, 1–9.

- 446 Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A Deep Learning Library for Solving
447 Differential Equations. *SIAM Review*, *63*, 208–228. URL: <http://dx.doi.org/10.1137/19M1274067>.
448 doi:10.1137/19m1274067.
- 449 Mao, Z., Jagtap, A. D., & Karniadakis, G. E. (2020). Physics-informed neural networks for high-speed flows.
450 *Computer Methods in Applied Mechanics and Engineering*, *360*, 112789.
- 451 Mattheakis, M., Sondak, D., Dogra, A. S., & Protopapas, P. (2020). Hamiltonian neural networks for solving
452 equations of motion. *arXiv preprint arXiv:2001.11107*, .
- 453 Mishra, S., & Molinaro, R. (2022). Estimates on the generalization error of physics-informed neural networks
454 for approximating a class of inverse problems for pdes. *IMA Journal of Numerical Analysis*, *42*, 981–1022.
- 455 Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein,
456 N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chil-
457 amkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An impera-
458 tive style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer,
459 F. d Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Sys-*
460 *tems*. Curran Associates, Inc. volume 32. URL: [https://proceedings.neurips.cc/paper/2019/file/](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf)
461 [bdbca288fee7f92f2bfa9f7012727740-Paper.pdf](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf). arXiv:1912.01703.
- 462 Penwarden, M., Jagtap, A. D., Zhe, S., Karniadakis, G. E., & Kirby, R. M. (2023). A unified scalable
463 framework for causal sweeping strategies for physics-informed neural networks (pinns) and their temporal
464 decompositions. *arXiv preprint arXiv:2302.14227*, .
- 465 Qin, S., & Xue, X. (2014). A two-layer recurrent neural network for nonsmooth convex optimization problems.
466 *IEEE transactions on neural networks and learning systems*, *26*, 1149–1160.
- 467 Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning
468 framework for solving forward and inverse problems involving nonlinear partial differential equations. *Jour-*
469 *nal of Computational Physics*, *378*, 686–707. URL: [https://www.sciencedirect.com/science/article/](https://www.sciencedirect.com/science/article/pii/S0021999118307125)
470 [pii/S0021999118307125](https://www.sciencedirect.com/science/article/pii/S0021999118307125). doi:<https://doi.org/10.1016/j.jcp.2018.10.045>.
- 471 Shukla, K., Jagtap, A. D., Blackshire, J. L., Sparkman, D., & Karniadakis, G. E. (2021a). A physics-informed
472 neural network for quantifying the microstructural properties of polycrystalline nickel using ultrasound
473 data: A promising approach for solving inverse problems. *IEEE Signal Processing Magazine*, *39*, 68–77.
- 474 Shukla, K., Jagtap, A. D., & Karniadakis, G. E. (2021b). Parallel physics-informed neural networks via
475 domain decomposition. *Journal of Computational Physics*, *447*, 110683.
- 476 Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E.,
477 Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J.,

478 Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, , Feng, Y., Moore,
479 E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris,
480 C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., Vijaykumar, A., Bardelli, A. P.,
481 Rothberg, A., Hilboll, A., Kloeckner, A., Scopatz, A., Lee, A., Rokem, A., Woods, C. N., Fulton, C.,
482 Masson, C., Häggström, C., Fitzgerald, C., Nicholson, D. A., Hagen, D. R., Pasechnik, D. V., Olivetti, E.,
483 Martin, E., Wieser, E., Silva, F., Lenders, F., Wilhelm, F., Young, G., Price, G. A., Ingold, G.-L., Allen,
484 G. E., Lee, G. R., Audren, H., Probst, I., Dietrich, J. P., Silterra, J., Webber, J. T., Slavič, J., Nothman,
485 J., Buchner, J., Kulick, J., Schönberger, J. L., de Miranda Cardoso, J. V., Reimer, J., Harrington, J.,
486 Rodríguez, J. L. C., Nunez-Iglesias, J., Kuczynski, J., Tritz, K., Thoma, M., Newville, M., Kümmerer, M.,
487 Bolingbroke, M., Tartre, M., Pak, M., Smith, N. J., Nowaczyk, N., Shebanov, N., Pavlyk, O., Brodtkorb,
488 P. A., Lee, P., McGibbon, R. T., Feldbauer, R., Lewis, S., Tygier, S., Sievert, S., Vigna, S., Peterson,
489 S., More, S., Pudlik, T. et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in
490 Python. *Nature Methods*, *17*, 261–272. URL: <http://www.nature.com/articles/s41592-019-0686-2>.
491 doi:10.1038/s41592-019-0686-2.

492 Wang, Y., Wang, J., & Tao, D. (2023). Neurodynamics-driven supervised feature selection. *Pattern Recog-*
493 *nition*, *136*, 109254.

494 Xia, Y., Feng, G., & Wang, J. (2008). A novel recurrent neural network for solving nonlinear optimization
495 problems with inequality constraints. *IEEE Transactions on neural networks*, *19*, 1340–1353.

496 Xia, Y., & Wang, J. (2000). A recurrent neural network for solving linear projection equations. *Neural*
497 *Networks*, *13*, 337–350.