Improved Saddle Point Prediction in Stochastic Two-Player Zero-Sum Games with a Deep Learning Approach

Dawen Wu, Abdel Lisser

Improved Saddle Point Prediction in Stochastic Two-Player Zero-Sum Games with a Deep Learning Approach

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

Improved Saddle Point Prediction in Stochastic Two-Player Zero-Sum Games with a Deep Learning Approach

Dawen Wu[a,*], Abdel Lisser[a]

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

Abstract

In this paper, we propose a novel deep learning approach for predicting saddle points in stochastic two-player zero-sum games. Our method combines neurodynamic optimization and deep neural networks. First, we model the stochastic two-player zero-sum game as an ordinary differential equation (ODE) system using neurodynamic optimization. Second, we develop a neural network to approximate the solution to the ODE system, which includes the saddle point prediction for the game problem. Third, we introduce a specialized algorithm for training the neural network to enhance the accuracy of the saddle point prediction. Our experiments demonstrate that our model outperforms existing approaches, yielding faster convergence and more accurate saddle point predictions.

Keywords: Stochastic two-player zero-sum games, Saddle points, Neurodynamic optimization, Deep learning, Ordinary differential equations

1. Introduction

In non-cooperative game theory, two-player zero-sum games are among the most fundamental models, featuring two players, each with finite actions, and their payoffs add up to zero. A saddle point of such a game represents a situation in which no player can increase his/her payoff by unilaterally changing his/her strategy. von Neumann (1928) proved that there always exists a saddle point for any two-player zero-sum game. Later, Nash (1950) extended this result and proved that there always exists a Nash equilibrium for any n-player general-sum game with finite actions. Charnes (1953) studied the two-player zero-sum games with linear constraints, which can be formulated as linear programming problems. Recently, Singh & Lisser (2019) studied a stochastic version of the two-player zero-sum games, namely stochastic two-player zero-sum games. They demonstrated that a saddle point exists if the random vectors defining stochastic linear constraints follow elliptically symmetric distributions.

Neurodynamic optimization refers to a method that uses a first-order ODE system to model a nonlinear optimization problem. Hopfield & Tank (1985) proposed the Hopfield network to solve the well-known traveling salesman problem. The Hopfield network was extended to solve nonlinear convex programming

*Corresponding author
Email address: dawen.wu@centralesupelec.fr, abdel.lisser@l2s.centralesupelec.fr (Abdel Lisser)

by introducing a penalty parameter (Kennedy & Chua, 1988). However, this method struggles to achieve an optimal solution because the true minimizer can only be achieved when the penalty parameter goes to infinity. Since then, various types of neurodynamic optimization methods have been proposed without using the penalty parameter, to solve various optimization problems, e.g., quadratic programming (Xia & Wang, 2000), nonlinear projection equations (Xia & Feng, 2007), second-order cone programming (Nazemi, 2019), non-smooth optimization problems (Qin & Xue, 2014), pseudoconvex optimization problems (Xu et al., 2020). Wu & Lisser (2022b) proposed a neurodynamic optimization approach to model the stochastic two-player zero-sum game problem using Karush–Kuhn–Tucker conditions.

Deep learning is a type of machine learning that uses deep neural networks, comprising many layers of interconnected nodes, to learn complex patterns and relationships in data. Thanks to the exponential growth of data and computing resources in recent years, deep learning has been applied successfully to a wide range of fields, including image processing (Raza et al., 2021), natural language processing (Tan & Zhang, 2020), bioinformatics (Hu et al., 2020), game theory (Wu & Lisser, 2022d,c), and operations research (Wu & Lisser, 2022a; Nair et al., 2020; Bengio et al., 2021).

Research on utilizing neural networks to solve differential equations dates back to the 1990s, with the initial concept involving training a neural network to satisfy both a given differential equation and its boundary conditions (Dissanayake & Phan-Thien, 1994). Advances in this field include the development of construction methods that inherently satisfy initial and boundary conditions, with applications in ordinary and partial differential equations (Lagaris et al., 1998, 2000; McFall & Mahan, 2009). The advent of deep learning has revitalized the use of neural networks for solving differential equations, enabling researchers to address the challenge of solving high-dimensional nonlinear PDEs (Han et al., 2017; Yu et al., 2018; Han et al., 2018) and leading to significant developments such as the introduction of physics-informed neural networks (PINNs) (Raissi et al., 2019a). PINNs have found successful applications across various fields, including computational mechanics (Anitescu et al., 2019; Samaniego et al., 2020), and have been modified to accommodate a range of problem scenarios (Wang et al., 2021; Lu et al., 2021b; Zhang et al., 2020). The rapid progress in this research direction has been facilitated by automatic differentiation tools that simplify the calculation of derivatives, gradients, and Jacobian matrices (Baydin et al., 2018; Paszke et al., 2019a), and it has been demonstrated that the neural network approximator converges to the PDE solution as the number of hidden units increases (Sirignano & Spiliopoulos, 2018). Currently, software packages are available that utilize deep learning methods to solve differential equations (Lu et al., 2021a; Chen et al., 2020).

This paper addresses the saddle point problem in stochastic two-player zero-sum games (Singh & Lisser, 2019). Previous research reformulated this problem as a problem of solving the state solution of an ODE system (Wu & Lisser, 2022b). However, the existing solution method demands solving all intermediate states in the ODE system to arrive at the final prediction, which can lead to computationally intensive and time-consuming processes. To overcome these challenges, this paper aims to enhance the saddle point prediction problem by incorporating advanced deep learning techniques for solving differential equations (Raissi et al.,

2019b; Sirignano & Spiliopoulos, 2018). By developing an efficient approach to the stochastic two-player zero-sum game problem, we address the current limitations and open new possibilities for solving various game-theoretic problems. The proposed approach can encourage researchers to explore the use of deep learning techniques in tackling other complex game-theoretic scenarios, contributing to advancements in the field of game theory and optimization.

It is important to note that the previous convolutional neural network (CNN) approach (Wu & Lisser, 2022d,c) employed to address two-player zero-sum games is not directly applicable to the stochastic two-player zero-sum game considered in this paper. The aforementioned papers focused on classical two-player zero-sum games without random variables, which could be represented by a matrix denoting payoffs and subsequently used as input for the CNN. In contrast, the stochastic two-player zero-sum game examined in this paper incorporates random variables within the constraints, preventing a direct input of the game into the CNN.

1.1. Key contributions

Our key contributions can be summarized as follows:

- We propose a novel method that combines neurodynamic optimization and deep learning to solve the stochastic two-player zero-sum game. To the best of our knowledge, this is the first time deep learning has been applied to this problem.
- Our work transforms the stochastic two-player zero-sum game into a neural network training problem, allowing us to solve the problem without the need for a standard numerical integration solver.
- Our method outperforms the state-of-the-art method presented in Wu & Lisser (2022b). In the specific game examples given in the experimental section, our method achieves an objective value of 5.48 and maximum violation of constraints of 0.0827, respectively, outperforming the method of Wu & Lisser (2022b).

1.2. Paper outline

The remaining sections are organized as follows. The background knowledge necessary to understand this paper is provided in Section 2, including an introduction to the stochastic two-player zero-sum game, neurodynamic optimization, and numerical integration methods. Section 3 describes our proposed method and how it solves the game problem. Section 4 presents experimental results and a discussion in which we compare our method with the numerical integration methods. Section 5 summarizes this paper and outlines future directions.

1.3. Notation

Table 1 presents the notations list of this paper.

Notation	Description
$M = \{1, \ldots, m\}$	Action set of player 1; m denotes the number of pure strategies.
$N = \{1, \ldots, n\}$	Action set of player 2; n denotes the number of pure strategies.
$x \in \mathbb{R}^m$	Mixed strategy of player 1.
$y \in \mathbb{R}^n$	Mixed strategy of player 2.
$\mathcal{J}_1$	Index set of constraints for player 1.
$\mathcal{J}_2$	Index set of constraints for player 2.
$\alpha_1 = \left(\alpha_k^1\right)_{k \in \mathcal{J}_1}$	Confidence levels for player 1's constraints.
$\alpha_2 = \left(\alpha_l^2\right)_{l \in \mathcal{J}_2}$	Confidence levels for player 2's constraints.
$G(\alpha)$	Stochastic two-player zero-sum game with confidence level α; $\alpha = (\alpha_1, \alpha_2)$.
$(x^* \in \mathbb{R}^m, y^* \in \mathbb{R}^n)$	Saddle point of $G(\alpha)$.
$(\hat{x} \in \mathbb{R}^m, \hat{y} \in \mathbb{R}^n)$	Predicted saddle point.
$\text{Ellip}_d(\mu, \Sigma, \varphi)$	d-dimensional elliptically symmetric distribution.
$s \in \mathbb{R}^{ns}$	Decision variables; ns denotes the number of decision variables.
$u \in \mathbb{R}^{nu}$	Dual variables; nu denotes the number of dual variables.
$\Phi(\cdot)$	ODE system.
$[0, T]$	Time range.
$r : [0, T] \to \mathbb{R}^{nr}$	State solution of the ODE system.
$\hat{r} : [0, T] \to \mathbb{R}^{nr}$	Approximate state solution.
$\|\cdot\|$	Euclidean norm.
$\|\cdot\|_\infty$	Infinity norm.

Table 1: Summary of mathematical notations

2. Preliminaries

The stochastic two-player zero-sum game is introduced in Section 2.1. The neurodynamic optimization method, which models the stochastic two-player zero-sum game as an ODE system, is presented in Section 2.2. Numerical integration methods, which are existing approaches for solving the game problem, are described in Section 2.3.

2.1. Stochastic two-player zero-sum game

A two-player zero-sum game problem with linear constraints introduced by Charnes (1953) is characterized as follows.

- The game has two players in total, and players 1 and 2 have pure strategy sets of $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$, respectively. m and n denote the number of pure strategies of players 1 and 2, respectively.
- The payoff of the game is represented by a matrix $A = (a_{ij}) \in \mathbb{R}^{m \times n}$. When player 1 chooses action $i \in M$ and player 2 chooses action $j \in N$, players 1 and 2 receive payoffs of a_{ij} and $-a_{ij}$, respectively.
- $x \in X$ and $y \in Y$ are mixed strategies of players 1 and 2, respectively. $X = \left\{x \in \mathbb{R}^m \mid Bx \leq b, \mathbf{1}_m^T x = 1, x \geq \mathbf{0}\right\}$ and $Y = \left\{y \in \mathbb{R}^n \mid Dy \leq d, \mathbf{1}_n^T y = 1, y \geq \mathbf{0}\right\}$ denote the feasible sets, where $B \in \mathbb{R}^{p \times m}$, $b \in \mathbb{R}^p$, $D \in \mathbb{R}^{q \times n}$, $d \in \mathbb{R}^q$, $\mathbf{1}_m^T = [1, 1, \ldots, 1] \in \mathbb{R}^m$, and $\mathbf{1}_n^T = [1, 1, \ldots, 1] \in \mathbb{R}^n$. Let $\mathcal{J}_1 = \{1, 2 \ldots, p\}$ and $\mathcal{J}_2 = \{1, 2 \ldots, q\}$ be the index sets of constraints of player 1 and 2, respectively.

Given a mixed strategy y of player 2, the objective of player 1 is to find a mixed strategy x that solves the following linear programming problem,

$$\begin{cases} \max\limits_{x} x^T A y \\ \text{s.t.} \\ \quad Bx \leq b, \\ \quad \mathbf{1}_m^T x = 1, \\ \quad x \geq 0. \end{cases} \tag{1}$$

Similarly, for a given strategy x of player 1, the aim of player 2 is to find a strategy y that solves the following linear programming problem.

$$\begin{cases} \min\limits_{y} x^T A y \\ \text{s.t.} \\ \quad Dy \geq d, \\ \quad \mathbf{1}_n^T y = 1, \\ \quad y \geq 0. \end{cases} \tag{2}$$

A strategy profile (x, y) is called a saddle point of this game if x and y are optimal solutions of (1) and (2), respectively. Charnes (1953) proved that there is always a saddle point for such a game.

Definition 1 (Elliptically symmetric distribution)**.** *A d-dimensional random vector ξ follows an elliptically symmetric distribution* $\text{Ellip}_d(\mu, \Sigma, \varphi)$*, i.e., $\xi \sim \text{Ellip}_d(\mu, \Sigma, \varphi)$ if its characteristic function is given by $\mathbb{E}e^{iz^T\xi} = e^{iz^T\mu}\varphi\left(z^T\Sigma z\right)$, where $z \in \mathbb{R}^d$ is the argument of the characteristic function, μ is the location parameter, Σ is the scale matrix, and φ is the characteristic generator function.*

In this paper, we aim at solving the stochastic two-player zero-sum game introduced by Singh & Lisser (2019), in which the matrices B and D are treated as random variables. A stochastic two player zero sum game is characterized as follows.

- Let B^ω and D^ω denote random matrices that defines the constraints of player 1 and 2, respectively, and ω denotes some uncertainty parameter.
- We denote the k-th row vector of B^ω as B_k^ω and the l-th row vector of D^ω as D_l^ω. Both B_k^ω and D_l^ω follow elliptical distributions, i.e., $B_k^\omega \sim \text{Ellip}_m\left(\mu_k^1, \Sigma_k^1, \varphi_k^1\right)$ and $D_l^\omega \sim \text{Ellip}_n\left(\mu_l^2, \Sigma_l^2, \varphi_l^2\right)$. The k-th element of b is denoted as b_k, and the l-th element of d is denoted as d_l.
- $\alpha_k^1 \in [0, 1]$ denotes the confidence level for the k-th constraint of player 1, and $\alpha_l^2 \in [0, 1]$ denotes the confidence level for the l-th constraint of player 2. Let $\alpha^1 = \left(\alpha_k^1\right)_{k \in \mathcal{J}_1}$, and $\alpha^2 = \left(\alpha_l^2\right)_{l \in \mathcal{J}_2}$, and $\alpha = \left(\alpha^1, \alpha^2\right)$.

Therefore, the stochastic two-player zero-sum game can be written as:

$$\begin{cases} \max\limits_{x} x^T A y \\ \text{s.t.} \\ \quad P\left\{B_k^{\omega} x \leq b_k\right\} \geq \alpha_k^1, \quad \forall k \in \mathcal{J}_1 \\ \quad \mathbf{1}_m^T x = 1 \\ \quad x \geq \mathbf{0}, \end{cases} \tag{3}$$

and

$$\begin{cases} \min\limits_{y} x^T A y \\ \text{s.t.} \\ \quad P\left\{D_l^{\omega} y \geq d_l\right\} \geq \alpha_l^2, \quad \forall l \in \mathcal{J}_2 \\ \quad \mathbf{1}_n^T y = 1 \\ \quad y \geq \mathbf{0}. \end{cases} \tag{4}$$

A mixed strategy (x^*, y^*) is said to be a saddle point of the stochastic two-player zero-sum game if it simultaneously solves for both players' optimization problems, as described in equations (3) and (4). In other words, (x^*, y^*) represents the optimal strategies for each player, such that neither player can unilaterally improve their outcome by deviating from these strategies.

By the second-order cone constraint reformulation (Henrion, 2007; Van de Panne & Popp, 1963; Kataoka, 1963), the feasible sets of (3) and (4) are reformulated as

$$S_1\left(\alpha^1\right) = \left\{x \in \mathbb{R}^m \mid \mathbf{1}_m^T x = 1, x \geq \mathbf{0}, x^T \mu_k^1 + \Psi_{\xi_k^1}^{-1}(\alpha_k^1) \|(\Sigma_k^1)^{\frac{1}{2}} x\| \leq b_k, \quad \forall k \in \mathcal{J}_1\right\}, \tag{5}$$

and

$$S_2\left(\alpha^2\right) = \left\{y \in \mathbb{R}^n \mid \mathbf{1}_n^T y = 1, y \geq \mathbf{0}, -y^T \mu_l^2 + \Psi_{\xi_l^2}^{-1}(\alpha_l^2) \|(\Sigma_l^2)^{\frac{1}{2}} y\| \leq -d_l, \quad \forall l \in \mathcal{J}_2\right\}, \tag{6}$$

respectively, where $\xi_k^1 = \frac{B_k^{\omega} x - x^T \mu_k^1}{\left\|(\Sigma_k^1)^{\frac{1}{2}} x\right\|}, k \in \mathcal{J}_1$, and $\xi_l^2 = \frac{-D_l^{\omega} y + y^T \mu_l^2}{\left\|(\Sigma_l^2)^{\frac{1}{2}} y\right\|}, l \in \mathcal{J}_2$ follow univariate standard elliptical distributions, i.e., $\xi_k^1 \sim \text{Ellip}\left(0, 1, \varphi_k^1\right)$, and $\xi_l^2 \sim \text{Ellip}\left(0, 1, \varphi_l^2\right)$. $\Psi_{\xi_k^1}^{-1}\left(\alpha_k^1\right)$ and $\Psi_{\xi_l^2}^{-1}\left(\alpha_l^2\right)$ are the quantile functions of 1-dimensional distribution functions induced by characteristic functions φ_k^1 and φ_l^2, respectively. We denote a stochastic two-player zero-sum game with a confidence level α as $G(\alpha)$.

Assumption 1.

- *$S_1(\alpha^1)$ is strictly feasible, i.e., there exists an $x \in \mathbb{R}^m$ which is a feasible point of $S_1(\alpha^1)$ and the inequality constraints of $S_1(\alpha^1)$ are strictly satisfied by x.*

- *$S_2(\alpha^2)$ is strictly feasible, i.e., there exists an $y \in \mathbb{R}^n$ which is a feasible point of $S_2(\alpha^2)$ and the*

inequality constraints of $S_2(\alpha^2)$ are strictly satisfied by x.

Theorem 1 (Singh & Lisser (2019), Theorem 3.5)**.** *Consider a stochastic two player zero sum game $G(\alpha)$, where the matrices B^w and D^w defining the constraints of both the players, respectively, are random. Let the row vectors $B_k^w \sim Ellip_m\left(\mu_k^1, \Sigma_k^1, \varphi_k^1\right), k \in \mathcal{J}_1$, and $D_l^w \sim \text{Ellip}_n\left(\mu_l^2, \Sigma_l^2, \varphi_l^2\right), l \in \mathcal{J}_2$. If all $k \in \mathcal{J}_1$ and $l \in \mathcal{J}_2$, Σ_k^1 and Σ_l^2 are positive definite matrices. Then, there exists a saddle point equilibrium for the game $G(\alpha)$ for all $\alpha \in (0.5, 1]^p \times (0.5, 1]^q$.*

Theorem 2 (Singh & Lisser (2019), Theorem 3.7)**.** *Let Assumption 1 hold. (x^*, y^*) is the saddle point of the stochastic two-player zero-sum game $G(\alpha)$ if and only if there exist $\left(v^{1^*}, \left(\delta_k^{1^*}\right)_{k\in\mathcal{J}_1}, \lambda^{1^*}\right)$ and $\left(v^{2^*}, \left(\delta_l^{2^*}\right)_{l\in\mathcal{J}_2}, \lambda^{2^*}\right)$ such that $\left(y^*, v^{1^*}, \left(\delta_k^{1^*}\right)_{k\in\mathcal{J}_1}, \lambda^{1^*}\right)$ and $\left(x^*, v^{2^*}, \left(\delta_l^{2^*}\right)_{l\in\mathcal{J}_2}, \lambda^{2^*}\right)$ are optimal solutions of the following primal-dual pair of nonlinear optimization problems.*

$$
\left.
\begin{aligned}
&\min_{y, v^1, \left(\delta_k^1\right)_{k\in\mathcal{J}_1}, \lambda^1} v^1 + \textstyle\sum_{k\in\mathcal{J}_1} \lambda_k^1 b_k \\
&s.t. \\
&\textit{(i)} Ay - \textstyle\sum_{k\in\mathcal{J}_1} \lambda_k^1 \mu_k^1 - \sum_{k\in\mathcal{J}_1} \left(\Sigma_k^1\right)^{\frac{1}{2}} \delta_k^1 \leq v^1 \mathbf{1}_m \\
&\textit{(ii)} - y^T \mu_l^2 + \Psi_{\xi_l^2}^{-1}\left(\alpha_l^2\right) \left\| \left(\Sigma_l^2\right)^{\frac{1}{2}} y \right\| \leq -d_l, \quad \forall l \in \mathcal{J}_2 \\
&\textit{(iii)} \left\|\delta_k^1\right\| \leq \lambda_k^1 \Psi_{\xi_k^1}^{-1}\left(\alpha_k^1\right), \quad \forall k \in \mathcal{J}_1 \\
&\textit{(iv)} \mathbf{1}^T y = 1 \\
&\textit{(v)} y \geq 0 \\
&\textit{(vi)} \lambda_k^1 \geq 0, \quad \forall k \in \mathcal{J}_1
\end{aligned}
\right\} \qquad (\mathcal{P})
$$

$$
\left.
\begin{aligned}
&\max_{x, v^2, \left(\delta_l^2\right)_{l\in\mathcal{J}_2}, \lambda^2} v^2 + \textstyle\sum_{l\in\mathcal{J}_2} \lambda_l^2 d_l \\
&s.t. \\
&\textit{(i)} A^T x - \textstyle\sum_{l\in\mathcal{J}_2} \lambda_l^2 \mu_l^2 - \sum_{l\in\mathcal{J}_2} \left(\Sigma_l^2\right)^{\frac{1}{2}} \delta_l^2 \geq v^2 \mathbf{1}_n \\
&\textit{(ii)} x^T \mu_k^1 + \Psi_{\xi_k^1}^{-1}\left(\alpha_k^1\right) \left\| \left(\Sigma_k^1\right)^{\frac{1}{2}} x \right\| \leq b_k, \quad \forall k \in \mathcal{J}_1 \\
&\textit{(iii)} \left\|\delta_l^2\right\| \leq \lambda_l^2 \Psi_{\xi_l^2}^{-1}\left(\alpha_l^2\right), \quad \forall l \in \mathcal{J}_2 \\
&\textit{(iv)} \mathbf{1}^T x = 1 \\
&\textit{(v)} x \geq 0 \\
&\textit{(vi)} \lambda_l^2 \geq 0, \quad \forall l \in \mathcal{J}_2
\end{aligned}
\right\} \qquad (\mathcal{D})
$$

We refer readers to Singh & Lisser (2019) for the proof and other details related to Theorems 1 and 2. Additionally, we refer readers to Wu & Lisser (2022b) (Proposition 1) for the reformulation of a stochastic two-player zero-sum game $G(\alpha)$ to $(\mathcal{P})$ and $(\mathcal{D})$.

2.2. Neurodynamic optimization

The neurodynamic optimization method introduced by Wu & Lisser (2022b) is used to model the stochastic two-player zero-sum game by a ODE system. Because $(\mathcal{P})$ and $(\mathcal{D})$ are a primal-dual pair, it is only necessary to model $(\mathcal{P})$. The optimal solution for $(\mathcal{D})$ can be found in the dual solution of $(\mathcal{P})$.

2.2.1. Reformulation of $(\mathcal{P})$.

Let us denote the decision variables of $(\mathcal{P})$ as $s = (y \in \mathbb{R}^n, v^1 \in \mathbb{R}, \delta \in \mathbb{R}^{n*p}, \lambda \in \mathbb{R}^p)$, where $\delta = [{\delta_1^1}^T, \ldots, {\delta_p^1}^T]^T$, and $\lambda = [\lambda_1^1, \ldots, \lambda_p^1]^T$. Define $ns = n + 1 + n * p + p$ as the number of decision variables, and $s \in \mathbb{R}^{ns}$. We reformulate the equality constraint $\mathbf{1}_n^T y = 1$ in $(\mathcal{P})$ as two inequalities, i.e., $\mathbf{1}_n^T y - 1 \leq 0$ and $1 - \mathbf{1}_n^T y \leq 0$. The second-order cone constraints in $(\mathcal{P})$ are not differentiable at the origin, so a smoothness technique must be used to transform them into smooth functions. This technique is given by the following expression

$$\|s\|_{\text{smooth}} = \sqrt{\|s\|^2 + \epsilon^2}, \tag{7}$$

where ϵ is a small positive constant, typically chosen to be on the order of 10^{-6} or 10^{-8}, and is set to $\epsilon = 10^{-6}$ in this paper. Then, the optimization problem $(\mathcal{P})$ can be rewritten as

$$\begin{aligned} &\min_s f(s) \\ &\text{s.t.} \\ &\quad g(s) \leq 0, \end{aligned} \tag{8}$$

where the objective function $f : \mathbb{R}^{ns} \to \mathbb{R}$ is denoted as

$$f(s) = v^1 + \sum_{k \in \mathcal{J}_1} \lambda_k^1 b_k, \tag{9}$$

and the constraints are denoted as

$$g(s) = \begin{bmatrix} g_1(s) \\ g_2(s) \\ g_3(s) \\ g_{41}(s) \\ g_{42}(s) \\ g_5(s) \\ g_6(s) \end{bmatrix} = \begin{bmatrix} Ay - v\mathbf{1}_m - \sum_{k \in \mathcal{J}_1} \left(\Sigma_k^1\right)^{\frac{1}{2}} \delta_k - \sum_{k \in \mathcal{J}_1} \lambda_k \mu_k^1 \\ (-y^T \mu_l^2 + \Psi_{\xi_l^2}^{-1}\left(\alpha_l^2\right) \left\| \left(\Sigma_l^2\right)^{\frac{1}{2}} y \right\|_{\text{smooth}} + d_l)_{l \in \mathcal{J}_2} \\ (\|\delta_k\|_{\text{smooth}} - \Psi_{\xi_k^1}^{-1}\left(\alpha_k^1\right) \lambda_k)_{k \in \mathcal{J}_1} \\ \mathbf{1}_n^T y - 1 \\ -\mathbf{1}_n^T y + 1 \\ -y \\ -\lambda \end{bmatrix}, \tag{10}$$

where $g_1 : \mathbb{R}^{n+1+n*p+p} \to \mathbb{R}^n$, $g_2 : \mathbb{R}^{n+1+n*p+p} \to \mathbb{R}^q$, $g_3 : \mathbb{R}^{n+1+n*p+p} \to \mathbb{R}^p$, $g_{41} : \mathbb{R}^{n+1+n*p+p} \to \mathbb{R}$, $g_{42} : \mathbb{R}^{n+1+n*p+p} \to \mathbb{R}$, $g_5 : \mathbb{R}^{n+1+n*p+p} \to \mathbb{R}^n$, and $g_6 : \mathbb{R}^{n+1+n*p+p} \to \mathbb{R}^q$. Let define $nu = n+q+p+1+1+n+q$ as the number of constraints, and $g : \mathbb{R}^{ns} \to \mathbb{R}^{nu}$.

2.2.2. Karush–Kuhn–Tucker conditions

The Karush–Kuhn–Tucker conditions (KKT conditions) for the optimization problem (8) are

$$\begin{aligned} \nabla f(s) + \left(\frac{\partial g}{\partial s}(s)\right)^T u = 0, \\ g(s) \leq 0, \quad u^T \geq 0, \quad u^T g(s) = 0, \end{aligned} \tag{11}$$

where $u \in \mathbb{R}^{nu}$ denotes the dual variables. $\nabla f(s) \in \mathbb{R}^{ns}$ denotes the gradient of the objective function at point s. $\frac{\partial g}{\partial s}(s) \in \mathbb{R}^{nu \times ns}$ denotes the Jacobian matrix of the constraints at point s. The details of $u \in \mathbb{R}^{nu}$, $\nabla f(s) \in \mathbb{R}^{ns}$, and $\frac{\partial g}{\partial s}(s) \in \mathbb{R}^{nu \times ns}$ are given as follows

$$u = \begin{bmatrix} u_1 \in \mathbb{R}^n \\ u_2 \in \mathbb{R}^q \\ u_3 \in \mathbb{R}^p \\ u_{41} \in \mathbb{R} \\ u_{42} \in \mathbb{R} \\ u_5 \in \mathbb{R}^n \\ u_6 \in \mathbb{R}^q \end{bmatrix}, \quad \nabla f(s) = \begin{bmatrix} \nabla f_y(s) \in \mathbb{R}^n \\ \nabla f_v(s) \in \mathbb{R} \\ \nabla f_\delta(s) \in \mathbb{R}^{p*n} \\ \nabla f_\lambda(s) \in \mathbb{R}^p \end{bmatrix}, \tag{12}$$

$$\begin{aligned} \frac{\partial g}{\partial s}(s) &= \begin{bmatrix} \frac{\partial g}{\partial y}(s) \in \mathbb{R}^{nu \times n} & \frac{\partial g}{\partial v}(s) \in \mathbb{R}^{nu \times 1} & \frac{\partial g}{\partial \delta}(s) \in \mathbb{R}^{nu \times n*p} & \frac{\partial g}{\partial \lambda}(s) \in \mathbb{R}^{nu \times p} \end{bmatrix}, \\ &= \begin{bmatrix} \frac{\partial g_1}{\partial y}(s) \in \mathbb{R}^{n \times n} & \frac{\partial g_1}{\partial v}(s) \in \mathbb{R}^{n \times 1} & \frac{\partial g_1}{\partial \delta}(s) \in \mathbb{R}^{n \times n*p} & \frac{\partial g_1}{\partial \lambda}(s) \in \mathbb{R}^{n \times p} \\ \frac{\partial g_2}{\partial y}(s) \in \mathbb{R}^{q \times n} & \frac{\partial g_2}{\partial v}(s) \in \mathbb{R}^{q \times 1} & \frac{\partial g_2}{\partial \delta}(s) \in \mathbb{R}^{q \times n*p} & \frac{\partial g_2}{\partial \lambda}(s) \in \mathbb{R}^{q \times p} \\ \frac{\partial g_3}{\partial y}(s) \in \mathbb{R}^{p \times n} & \frac{\partial g_3}{\partial v}(s) \in \mathbb{R}^{p \times 1} & \frac{\partial g_3}{\partial \delta}(s) \in \mathbb{R}^{p \times n*p} & \frac{\partial g_3}{\partial \lambda}(s) \in \mathbb{R}^{p \times p} \\ \frac{\partial g_{41}}{\partial y}(s) \in \mathbb{R}^{1 \times n} & \frac{\partial g_{41}}{\partial v}(s) \in \mathbb{R}^{1 \times 1} & \frac{\partial g_{41}}{\partial \delta}(s) \in \mathbb{R}^{1 \times n*p} & \frac{\partial g_{41}}{\partial \lambda}(s) \in \mathbb{R}^{1 \times p} \\ \frac{\partial g_{42}}{\partial y}(s) \in \mathbb{R}^{1 \times n} & \frac{\partial g_{42}}{\partial v}(s) \in \mathbb{R}^{1 \times 1} & \frac{\partial g_{42}}{\partial \delta}(s) \in \mathbb{R}^{1 \times n*p} & \frac{\partial g_{42}}{\partial \lambda}(s) \in \mathbb{R}^{1 \times p} \\ \frac{\partial g_5}{\partial y}(s) \in \mathbb{R}^{n \times n} & \frac{\partial g_5}{\partial v}(s) \in \mathbb{R}^{n \times 1} & \frac{\partial g_5}{\partial \delta}(s) \in \mathbb{R}^{n \times n*p} & \frac{\partial g_5}{\partial \lambda}(s) \in \mathbb{R}^{n \times p} \\ \frac{\partial g_6}{\partial y}(s) \in \mathbb{R}^{q \times n} & \frac{\partial g_6}{\partial v}(s) \in \mathbb{R}^{q \times 1} & \frac{\partial g_6}{\partial \delta}(s) \in \mathbb{R}^{q \times n*p} & \frac{\partial g_6}{\partial \lambda}(s) \in \mathbb{R}^{q \times p} \end{bmatrix}. \end{aligned} \tag{13}$$

2.2.3. ODE system

Now, let $s(\cdot), y(\cdot), v(\cdot), \delta(\cdot), \lambda(\cdot)$ and $u(\cdot)$ be time dependent functions, i.e., $s : \mathbb{R} \to \mathbb{R}^{ns}$, $y : \mathbb{R} \to \mathbb{R}^n$, $v : \mathbb{R} \to \mathbb{R}$, $\delta : \mathbb{R} \to \mathbb{R}^{n*p}$, and $\lambda : \mathbb{R} \to \mathbb{R}^p$. Denote $r(t) = (s^T(t), u^T(t))^T = (y^T(t), v^T(t), \delta^T(t), \lambda^T(t), u^T(t))^T$,

and $r : \mathbb{R} \to \mathbb{R}^{nr}$, where $nr = ns + nu$. The following ODE system models the KKT conditions (11)

$$\frac{dr}{dt} = \Phi(r) = \begin{bmatrix} \frac{dy}{dt} \\ \frac{dv}{dt} \\ \frac{d\delta}{dt} \\ \frac{d\lambda}{dt} \\ \frac{du}{dt} \end{bmatrix} = \begin{bmatrix} -\left(\nabla f_y(s) + \left(\frac{\partial g}{\partial y}(s)\right)^T (u + g(s))^+\right) \\ -\left(\nabla f_v(s) + \left(\frac{\partial g}{\partial v}(s)\right)^T (u + g(s))^+\right) \\ -\left(\nabla f_\delta(s) + \left(\frac{\partial g}{\partial \delta}(s)\right)^T (u + g(s))^+\right) \\ -\left(\nabla f_\lambda(s) + \left(\frac{\partial g}{\partial \lambda}(s)\right)^T (u + g(s))^+\right) \\ (u + g(s))^+ - u \end{bmatrix}, \tag{14}$$

where $(u + g(s))^+ = \max\{\mathbf{0}, u + g(s)\}$.

Definition 2 (State solution). *Consider an ODE system $\frac{dr}{dt} = \Phi(r)$, where $\Phi(r) : \mathbb{R}^{nr} \to \mathbb{R}^{nr}$. Given an initial point $r_0 \in \mathbb{R}^{nr}$, a vector value function $r(t) : \mathbb{R} \to \mathbb{R}^{nr}$ is called a state solution, if it satisfies the ODE system $\frac{dr}{dt} = \Phi(r)$ and the initial condition $r(0) = r_0$.*

Definition 3 (Equilibrium point). *Consider an ODE system $\frac{dr}{dt} = \Phi(r)$. A point r^* is called an equilibrium point if it satisfies $\Phi(r^*) = \mathbf{0}$.*

Theorem 3 (Wu & Lisser (2022b)). *Let $r^* = (s^*, u^*)^T$. The point r^* is an equilibrium point of the ODE system (14) if and only if (s^*, u^*) satisfy the KKT conditions (11).*

Lemma 4 (Wu & Lisser (2022b)). *The equilibrium point r^* of the proposed ODE system (14) is unique.*

Theorem 5 (Wu & Lisser (2022b)). *The equilibrium point r^* of the proposed ODE system (14) is globally asymptotically stable, i.e., given any initial point r_0, the state solution has $\lim_{t\to\infty} r(t) = r^*$.*

Remark 1. *The primary objective of the ODE system presented in (14) is to address the KKT conditions given in (11). As stated in Theorem 3, the equilibrium point in the ODE system coincides with to the KKT conditions outlined in (11). As a result, by determining the equilibrium point of the ODE system, we can effectively solve the KKT conditions and subsequently identify the saddle point of the stochastic two-player zero-sum game.*

Lemma 4 establishes the uniqueness of the equilibrium point in the ODE system, which is consistent with the notion that the stochastic two-player zero-sum game has a unique saddle point. Furthermore, Theorem 5 ensures that the state solution of the ODE system converges to the equilibrium point of the system as t approaches infinity, implying that it converges to the saddle point of the stochastic two-player zero-sum game. This means that we can solve the saddle point of the corresponding game by simply solving the state solution of the ODE system.

2.3. Numerical integration method

In this subsection, we briefly describe how existing methods solve for the state solution of an ODE system, which will be compared with our proposed method in the experimental section. Since the considered ODE system (14) is nonlinear, it cannot be solved analytically.

The existing methods for solving the state solution of an ODE system use numerical integration techniques to approximate the solution by discretizing the domain (Butcher, 2016). These methods, such as the Runge-Kutta method, choose multiple time points in the domain, known as collocation points, and find a solution that satisfies the ODE system at these points. However, these methods can be inefficient when only the final state is of interest because the final state is only obtained after calculating all the intermediate collocation points.

Numerical integration methods are further divided into two categories: explicit and implicit methods. Explicit methods, such as RK45, RK23, and DOP853, determine the state at a later time based on the current state (Dormand & Prince, 1980; Bogacki & Shampine, 1989; Hairer et al., 1993). Implicit methods, such as Radau and BDF, find the solution by solving equations involving the current and later states (Wanner & Hairer, 1996; Shampine & Reichelt, 1997). Additionally, LSODA can switch automatically between stiff and nonstiff methods (Petzold, 1983). *Scipy* provides software implementations of these methods to make them easier to use (Virtanen et al., 2020).

3. Method

In Section 3.1, we describe the problem setup and use an initial value problem to approximate the stochastic two-player zero-sum game. Section 3.2 presents our proposed approach, which uses a neural network to solve the initial value problem and subsequently predict the saddle point of the game. The loss function for training the neural network is discussed in Section 3.3, and an evaluation metric is given in Section 3.4 to assess the model's effectiveness at predicting saddle points. Finally, Section 3.5 provides a complete pipeline for solving the game problem using our proposed neural network.

3.1. Problem setup

This subsection provides a brief summary of how to reformulate a stochastic two-player zero-sum game as an initial value problem based on Sections 2.1 and 2.2. As presented in Figure 1, the reformulation procedure is as follows:

1. Reformulate the original game problem $G(\alpha)$, i.e., (3)-(4), as a primal-dual pair of nonlinear optimization problems $(\mathcal{P})$ and $(\mathcal{D})$.
2. Apply the smoothness technique (7) to derive the KKT conditions (11) of the primal problem $(\mathcal{P})$.
3. Represent the KKT conditions using the ODE system (14), where the equilibrium point is unique and globally asymptotically stable.
4. The desired saddle point (x^*, y^*) of the game is included in the equilibrium point r^* of the ODE system.

In order to obtain the equilibrium point, we must construct an initial value problem (IVP), which consists of (a) the ODE system (14), (b) an initial point r_0, and (c) a time range $[0, T]$. The state solution $r(t)$ of the IVP satisfies the ODE system on the domain $t \in [0, T]$ and the initial condition $r(0) = r_0$. By Theorem 5,

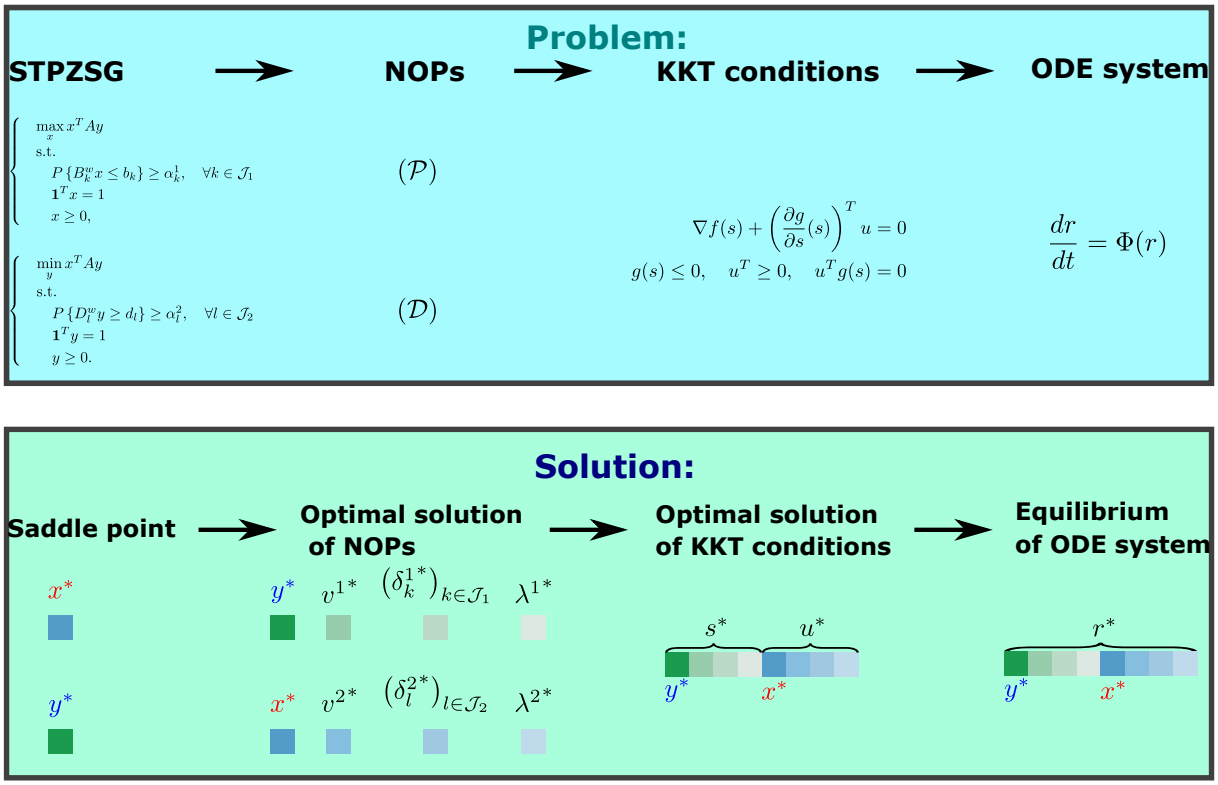

Figure 1: **The upper part** illustrates the workflow of modeling a stochastic two-player zero-sum game (STPZSG) using an ODE system. The STPZSG is represented by equations (3) and (4), and the ODE system is represented by equation (14). The nonlinear optimization problems (NOPs) in equations $(\mathcal{P})$ and $(\mathcal{D})$ and the KKT conditions in equation (11) are also shown. **The lower part** demonstrates that the saddle point of the STPZSG is incorporated in the optimal solution of the NOPs and KKT conditions, as well as the equilibrium point of the ODE system.

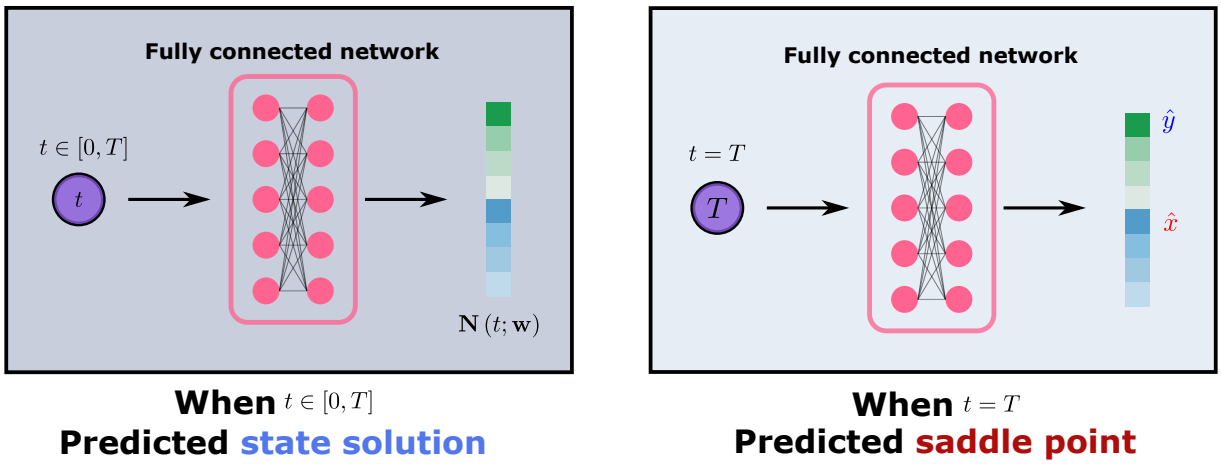

Figure 2: **The left part** of the figure shows the neural network model, $\hat{r}(t; \mathbf{w}) = \left(1 - e^{-t}\right) \mathbf{N}(t; \mathbf{w}), t \in [0, T]$, acting as a predicted state solution to the IVP. **The right part** of the figure illustrates that when the final time is selected (i.e., $t = T$), $\hat{r}(t; \mathbf{w})$ contains the predicted saddle point.

given any initial point r_0, the corresponding state solution $r(t)$ will converge to the same equilibrium r^* as time t goes to infinity. Therefore, we set the initial point of the ODE system to all zeros, i.e.,

$$r_0 = \mathbf{0}. \tag{15}$$

The final state $r(T)$ approximates the equilibrium point r^*, i.e.,

$$r(T) \approx r^*. \tag{16}$$

According to Theorem 5, the larger the provided time range $[0, T]$, the closer the final state is to r^*. Our goal is to solve for the final state $r(T)$ of the state solution $r(t), t \in [0, T]$, given an adequate size of the time range.

3.2. Neural network model

In this subsection, we present a model based on a fully connected neural network that is designed to predict the state solution $r(t)$ and its final state $r(T)$. The proposed model is given as follows

$$\hat{r}(t; \mathbf{w}) = \left(1 - e^{-t}\right) \mathbf{N}(t; \mathbf{w}), \quad t \in [0, T], \tag{17}$$

where $\mathbf{N}(t; \mathbf{w})$ is a fully connected neural network with trainable parameters $\mathbf{w}$. $(1 - e^{-t})$ is an auxiliary function to ensure the satisfaction of the initial point, i.e., $\hat{r}(t = 0; \mathbf{w}) = \mathbf{0}$.

As shown in Figure 2(Left), the model $\hat{r}(t; \mathbf{w})$ itself is used as a predicted state solution to the IVP, i.e.,

$$\hat{r}(t; \mathbf{w}) \approx r(t), \quad t \in [0, T]. \tag{18}$$

By choosing the time $t = T$, we have

$$\hat{r}(t = T; \mathbf{w}) \approx r(T). \tag{19}$$

Combining (16) with (19), we have

$$\hat{r}(t = T; \mathbf{w}) \approx r^*. \tag{20}$$

As shown in Figure 2(Right), $\hat{r}(t = T; \mathbf{w})$ contains the predicted saddle point of the considered game problem.

The effectiveness of the proposed neural network in solving stochastic two-player zero-sum game problems is supported by two theorems: 1) Theorem 5, which ensures the convergence of the state solutions of the ODE system to the saddle point of the game. 2) The universal approximation theorem for neural networks (Hornik, 1991), which states that there always exists a neural network as a state solution for solving the ODE system.

3.3. Loss function

We define the loss function of the proposed model as

$$\mathcal{L}(t, \mathbf{w}) = \left\| \frac{\partial \hat{r}(t; \mathbf{w})}{\partial t} - \Phi(\hat{r}(t; \mathbf{w})) \right\|, \tag{21}$$

where $\Phi(\cdot)$ refers to the ODE system of (14), which corresponds to the game problem being studied. $\frac{\partial \hat{r}(t;\mathbf{w})}{\partial t}$ is the derivative of the model output $\hat{r}(t; \mathbf{w})$ with respect to the input time t, which can be computed analytically by automatic differentiation tools, e.g., *PyTorch* or *JAX* (Paszke et al., 2019b; Bradbury et al., 2018).

We denote the objective function of the proposed model as

$$E(\mathbf{w}) = \int_0^T \mathcal{L}(t, \mathbf{w}) dt. \tag{22}$$

The objective function $E(\mathbf{w})$ is an integral of the loss function over the pre-given time range $[0, T]$. The loss value $\mathcal{L}(t, \mathbf{w})$ denotes the error of the model at the time t, and the objective function $E(\mathbf{w})$ denotes the overall error of the proposed model over the time range $[0, T]$.

However, calculating $E(\mathbf{w})$ is computationally difficult because of its integral part. In practice, we train the model by minimizing the following batch loss

$$\mathcal{L}(\mathbb{T}, \mathbf{w}) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \mathcal{L}(t, \mathbf{w}), \tag{23}$$

where $\mathbb{T}$ contains randomly sampled time points from the interval $[0, T]$, and $|\mathbb{T}|$ denotes the size of the set. Following the approach used in previous studies (Sirignano & Spiliopoulos, 2018), we sample the time points in $\mathbb{T}$ with a uniform distribution over the interval $[0, T]$. These time points are sampled in an independent

and identically distributed manner. The choice of probability distribution for sampling time points can potentially affect the training performance, which warrants further investigation in future studies.

3.4. Evaluation metric

To evaluate the performance of our model in predicting the final state $\hat{r}(t = T; \mathbf{w})$, we propose the following evaluation metric:

$$\epsilon(\hat{r}(T; \mathbf{w})) = \begin{cases} f(\hat{s}) & \text{if } \|g(\hat{s})\|_\infty \leq \tau, \\ +\infty & \text{otherwise }, \end{cases} \tag{24}$$

where $\epsilon(\hat{r}(T; \mathbf{w}))$ is called the epsilon value of $\hat{r}(T; \mathbf{w})$. $\hat{s}$ is retrieved from the predicted final state, i.e., $\hat{r}(T; \mathbf{w}) = [\hat{s}, \hat{u}]$. $f(\cdot)$ and $g(\cdot)$ are defined in (9) and (10), respectively. τ is a positive constant that represents a threshold parameter. When $\|g(\hat{s})\|_\infty \leq \tau$, the epsilon value is set to the objective value $f(\hat{s})$; otherwise, it is set to $+\infty$.

3.5. Pipeline

Algorithm 1: Enhancing neurodynamic optimization with deep learning for solving stochastic two-player zero-sum games

Input : A stochastic two player zero sum game $G(\alpha)$, A time range $[0, T]$
Output: Predicted saddle point $(\hat{x}, \hat{y})$

Function Main**:**
  Derive the ODE system $\Phi(\cdot)$ corresponding to the game $G(\alpha)$.
  Initialize a neural network model $\hat{r}(t; \mathbf{w})$.
  Initialize $\epsilon_{\text{best}} = \epsilon(\hat{r}(T; \mathbf{w}))$.
  while iter $\leq$ Max iteration **do**
    $\mathbb{T} \sim U(0, T)$: Uniformly sample a batch of t from the interval $[0, T]$.
    Forward propagation: Compute the batch loss $\mathcal{L}(\mathbb{T}, \mathbf{w})$.
    Backward propagation: Update $\mathbf{w}$ by $\nabla_{\mathbf{w}} \mathcal{L}(\mathbb{T}, \mathbf{w})$.
    Compute the epsilon value: $\epsilon_{\text{temp}} = \epsilon(r(T; \mathbf{w}))$.
    if $\epsilon_{\text{temp}} < \epsilon_{\text{best}}$ **then**
      $\epsilon_{\text{best}} = \epsilon_{\text{temp}}$
      Save the model with parameters $\mathbf{w}$
  end
  Extract $\hat{x}$ and $\hat{y}$ from $r(T; \mathbf{w})$.
  $\hat{x} \leftarrow \text{softmax}(\hat{x})$, and $\hat{y} \leftarrow \text{softmax}(\hat{y})$
  return $(\hat{x}, \hat{y})$
end

Algorithm 1 integrates all the methods introduced in this section. This algorithm is an optimization procedure that minimizes the objective function $E(\mathbf{w})$ while considering the proposed evaluation metric (24). First, the algorithm reformulates the game problem as an IVP for a given time range $[0, T]$. Next, it initializes a neural network to solve this problem and trains the network to improve its accuracy. In training, each iteration samples a batch of t uniformly from the time range $[0, T]$ as a training dataset. The neural network then performs gradient descent on the batch loss $\mathcal{L}(\mathbb{T}, \mathbf{w})$ to update its parameters $\mathbf{w}$. Finally, $\hat{x}$ and $\hat{y}$, which

are obtained from the predicted final state, need to be transformed into probability distributions using the softmax function.

ϵ_{best} represents the lowest epsilon value that the model has achieved so far. ϵ_{temp} represents the epsilon value that the model obtained at the most recent iteration. After each update of the model's parameters, the algorithm calculates ϵ_{temp} and compares it with ϵ_{best}. If ϵ_{temp} is lower than ϵ_{best}, it indicates that the model has found a better prediction after this iteration of parameter updates. In this case, the algorithm sets $\epsilon_{\text{best}} = \epsilon_{\text{temp}}$ and saves the current model. This idea is similar to the concept of *early-stopping* in deep learning, but here, we consider the epsilon value rather than the loss value.

Algorithm 1 requires specifying a time range as a hyperparameter. According to Theorem 5, the larger the chosen time range, the closer the corresponding final state will be to the optimal solution. In practice, however, a large time range can lead to training difficulties because the neural network has to approximate the state solution over a larger time range. Therefore, the choice of the time range involves a trade-off. A larger time range offers a higher maximum achievable accuracy but requires more training resources, whereas a smaller time range allows for faster convergence with fewer training iterations but results in a weaker achievable accuracy. One should determine the most appropriate time range that strikes a balance between accuracy and available training resources for their specific application.

In addition, the design of the neural network structure should be guided by two factors: 1) the chosen time range and 2) the size of the stochastic two-player zero-sum game to be solved. The size of the stochastic two-player zero-sum game is determined by the sizes of the action sets for both players and the number of constraints in their respective sets. The complexity of the neural network structure, which includes the number of neurons and the number of hidden layers, should be tailored to accommodate these two factors.

4. Numerical results

In this section, we conduct experiments to evaluate the performance of our proposed neural network model for solving the stochastic two-person zero-sum game. We first present the problem instance and describe the training process of our proposed model in Sections 4.1 and 4.2. Next, we compare our proposed method with the numerical integration method in Section 4.3, and discuss the advantages and limitations of our method in Section 4.4.

We use the *Google Colab Pro+* platform to perform our experiments. The neural network model is implemented using *Pytorch* 1.12.1 with *CUDA* 11.2 (Paszke et al., 2019b), and the ODE system is modelled with *JAX* (Bradbury et al., 2018). Our proposed method is compared with six numerical integration solvers, namely RK45, RK23, DOP853, Radau, BDF, and LSODA (Dormand & Prince, 1980; Bogacki & Shampine, 1989; Hairer et al., 1993; Wanner & Hairer, 1996; Shampine & Reichelt, 1997; Petzold, 1983), which can be accessed using *Scipy* (Virtanen et al., 2020).

4.1. Problem instance

We consider a specific instance of the stochastic two-player zero-sum game as shown in (3) and (4). Both players 1 and 2 have four pure strategies and one probabilistic constraint. B^ω and D^ω are 4-dimensional random vectors following normal distributions, i.e., $B^\omega \sim \mathcal{N}\left(\mu^1, \Sigma^1\right)$, $D^\omega \sim \mathcal{N}\left(\mu^2, \Sigma^2\right)$, where μ^1 and μ^2 are the means and Σ^1 and Σ^2 are the variances of the normal distributions. The problem data $(A, \Sigma^1, \Sigma^2, \mu^1, \mu^2, b, d, \alpha^1, \alpha^2)$ are given as follows:

$$A = \begin{bmatrix} 8.67 & 0.17 & 6.04 & 3.81 \\ 7.09 & 0.37 & 9.83 & 6.89 \\ 4.75 & 7.32 & 4.06 & 6.52 \\ 2.20 & 2.22 & 4.76 & 3.10 \end{bmatrix}, \quad \Sigma^1 = \begin{bmatrix} 1.10 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.40 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.38 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.43 \end{bmatrix}, \tag{25}$$

$$\Sigma^2 = \begin{bmatrix} 1.70 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.35 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.69 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.28 \end{bmatrix}, \quad \mu^1 = \begin{bmatrix} 4.53 \\ 0.06 \\ 2.53 \\ 3.66 \end{bmatrix}, \quad \mu^2 = \begin{bmatrix} 1.74 \\ 3.83 \\ 7.37 \\ 8.77 \end{bmatrix}, \tag{26}$$

$b = 7.03$, and $d = 4.53$. The confidence levels of both players are set to $\alpha^1 = \alpha^2 = 80\%$.

4.2. Model training

We construct the proposed model to solve the game instance, $\hat{r}(t; \mathbf{w}) = (1 - e^{-t})\,\mathbf{N}(t; \mathbf{w})$, where $\mathbf{N}(t; \mathbf{w})$ is a fully connected neural network with a hidden layer of 200 neurons and the activation function is *Tanh*. The training hyperparameters are as follows: the maximum number of iterations is $100,000$, the optimizer is *Adam* (Kingma & Ba, 2014) with a learning rate of 0.001 and batch size of 128.

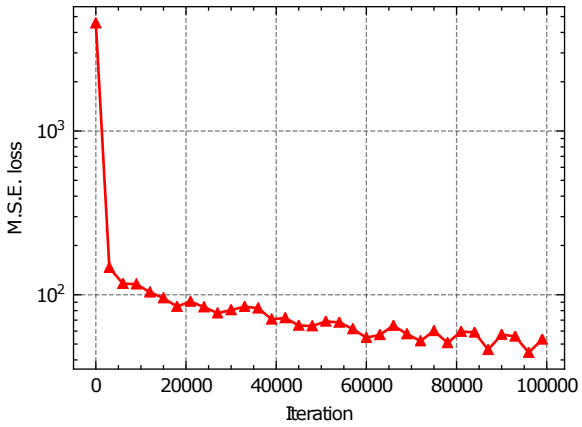

Figure 3: The mean square error (M.S.E) loss plotted against the number of iterations.

We apply the ODE system (14) to model the game instance given in Section 4.1. The neural network model is trained using Algorithm 1, with the time range set to $[0, 10]$ and the threshold τ in equation (24)

set to 0.1. We use the MSE loss as a performance metric to evaluate the model's ability to solve the derived ODE system. Figure 3 shows the evolution of the MSE loss throughout the training process. The MSE loss decreases from an initial value of 4,557 to 53 after 100,000 iterations, indicating the transition of the neural network from a naive approximation to an accurate state solution of the ODE system. It is important to note that the reported loss values do not distinguish between training and testing losses, as the model trains on a completely new randomly generated time set $\mathcal{T}$ in each iteration. This training strategy is consistent with a range of deep learning methods for solving differential equations (Raissi et al., 2019b; Sirignano & Spiliopoulos, 2018).

4.3. Comparison with numerical integration methods

In this subsection, we compare the performance of our proposed method with the numerical integration solvers on the stochastic two-player zero-sum game instance. We first show the predicted saddle points given by the two types of methods and then evaluate the performance of these saddle points on two evaluation metrics.

The setup of the numerical solver involves dividing the time range $[0, 10]$ into $100,000$ collocation points, which corresponds to the number of training iterations of our method.

Our method			Numerical integration method (RK45)		
Iteration	$\hat{x}$	$\hat{y}$	Collocation point	$\hat{x}$	$\hat{y}$
1	[0.225 0.41 0.366 0.]	[0. 0.358 0.459 0.183]	1	[0.218 0.324 0.283 0.175]	[0.078 0.182 0.327 0.412]
1000	[0. 0.384 0.566 0.05]	[0. 0.272 0.005 0.724]	1000	[0.019 0.3 0.404 0.278]	[0. 0.234 0.297 0.469]
10000	[0. 0.384 0.566 0.05]	[0. 0.272 0.005 0.724]	10000	[0.002 0.087 0.31 0.601]	[0. 0.096 0.186 0.718]
30000	[0.087 0.387 0.526 0.]	[0.105 0.397 0.303 0.195]	30000	[0. 0.163 0.461 0.376]	[0. 0.123 0.216 0.661]
50000	[0.087 0.387 0.526 0.]	[0.105 0.397 0.303 0.195]	50000	[0. 0.274 0.63 0.096]	[0. 0.208 0.359 0.434]
100000	[0.023 0.41 0.567 0.]	[0.002 0.405 0.484 0.109]	100000	[0. 0.334 0.666 0.001]	[0.303 0.278 0.258 0.16]

Table 2: The predicted saddle points $(\hat{x}, \hat{y})$ produced by our method and the RK45 method.

Table 2(left) lists the predicted saddle points given by our proposed method at the 1st, 1000th, 10,000th, 30,000th, 50,000th, and 100,000th training iterations. Table 2(right) lists the predicted saddle points given by the RK45 method, a typical numerical integration solver, at the 1st, 1000th, 10,000th, 30,000th, 50,000th, and 100,000th collocation points.

Our method		Numerical integration methods						
Iteration	OBJ ↓	Collocation point	RK45	RK23	DOP853	Radau	BDF	LSODA
			OBJ ↓	OBJ ↓	OBJ ↓	OBJ ↓	OBJ ↓	OBJ ↓
1	-1.6222	1	-0.0048	-0.0048	-0.0048	-0.0048	-0.0048	-0.0048
1000	17.7826	1000	-0.2288	-0.2293	-0.2318	-0.2289	-0.2289	-0.2289
10000	17.7826	10000	-2.6507	-2.6482	-2.6515	-2.6505	-2.6495	-2.6489
30000	5.7754	30000	0.317	0.317	0.3171	0.3171	0.3156	0.316
50000	5.7754	50000	1.6081	1.6081	1.6086	1.608	1.6074	1.6061
100000	5.4862	100000	5.5497	5.5496	5.55	5.5496	5.5488	5.5496

Table 3: Comparison of the objective (OBJ) value produced by our method and various numerical integration methods, including RK45, RK23, DOP853, Radau, BDF, and LSODA. The table shows the OBJ values for different iterations and collocation points.

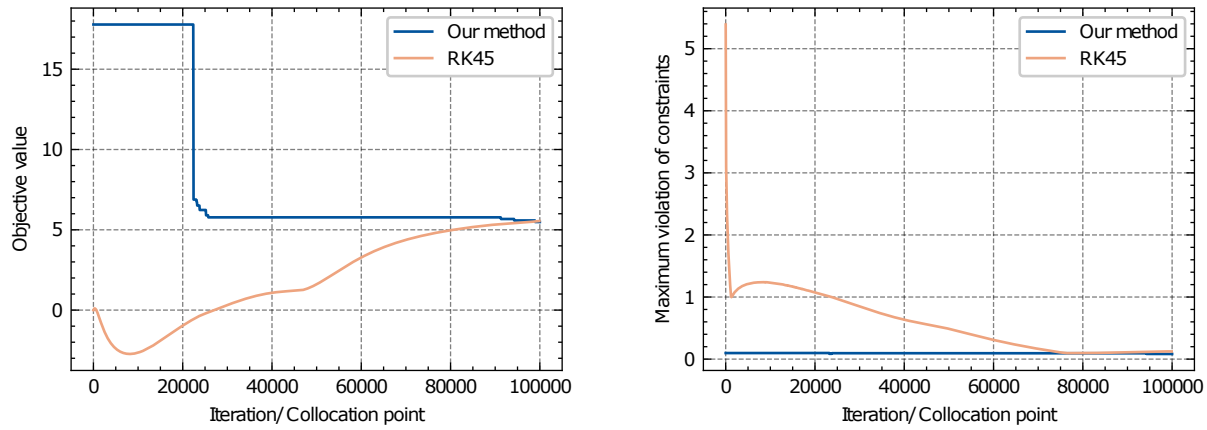

Figure 4: The objective value plotted against the number of iterations (left figure). The maximum violation of constraints plotted against the number of iterations (right figure).

Our method		Neurodynamic optimization						
Iteration	MVC ↓	Collocation point	RK45	RK23	DOP853	Radau	BDF	LSODA
			MVC ↓	MVC ↓	MVC ↓	MVC ↓	MVC ↓	MVC ↓
1	8.6806	1	5.2964	5.297	5.2968	5.2973	5.2973	5.2973
1000	0.0976	1000	1.1898	1.1785	1.1364	1.1875	1.1874	1.1874
10000	0.0976	10000	1.2311	1.2374	1.229	1.2316	1.2315	1.2315
30000	0.0951	30000	0.8481	0.8481	0.848	0.848	0.8481	0.8481
50000	0.0951	50000	0.4881	0.4881	0.4881	0.4881	0.4883	0.4882
100000	0.0827	100000	0.1239	0.1239	0.1238	0.1239	0.1235	0.1238

Table 4: Comparison of the maximum violation of constraints (MVC) produced by our method and various numerical integration methods, including RK45, RK23, DOP853, Radau, BDF, and LSODA. The table shows the MVC values for different iterations and collocation points.

We evaluate the performance of our model using two metrics: the objective value (OBJ, lower is better) and the maximum violation of constraints (MVC, lower is better). OBJ is defined as $f(s)$, where $f(\cdot)$ is given by equation (9) and here s represents the prediction from our method or a numerical integration method. MVC is defined as $|g(s)|_\infty$, where $g(\cdot)$ is given by equation (10).

Figure 4, Table 3, and Table 4 compare our method with numerical integration methods using the OBJ and MVC metrics. Figure 4 shows the results in a continuous manner, while Tables 3 and 4 show the results in a discrete manner at selected iterations or collocation points. Figure 4 only compares our method with the RK45 method, while Tables 3 and 4 compare six numerical integration methods.

The comparison results yield the following insights:

- Our method outperforms all numerical integration methods in terms of evaluation metrics OBJ and MVC, as shown in Tables 3 and 4. Ultimately, our method achieves an OBJ of 5.48 and an MVC of 0.0827. Among the numerical integration methods, BDF attains the best results, with an OBJ of 5.54 and an MVC of 0.1235.

- Our method converges faster than the numerical integration methods. Figure 4 shows that our method obtains an almost exact solution at the 30,000th iteration, with an OBJ of 5.48 and an MVC of 0.0827. In comparison, the results of the numerical integration methods at the 30,000th collocation point show OBJ and MVC values of 0.317 and 0.8481, respectively, indicating a significant deviation from accuracy.

- Our proposed method can identify a solution that nearly satisfies the constraints in the early stages of solving. Table 4 shows that at the 1000th iteration, our method finds a solution with an MVC of less than 0.1. In contrast, the numerical integration method fails to find solutions with MVC less than 0.1 even after 100,000 collocation points.

- Figure 3 shows a significant decrease in the MSE loss value during the first 20,000 iterations, with a subsequent levelling off. Figure 4 shows that OBJ and MVC still improve significantly after 20,000 iterations, although the decrease in loss value has become less pronounced. This suggests that the MSE loss of the model is not entirely decisive for the OBJ and MVC indicators, although they are correlated.

- The performance of the six numerical integration methods does not differ significantly from each other.

4.4. Discussion

In Sections 4.2 and 4.3, we present experimental results that demonstrate the superiority of our proposed deep learning approach. The observed advantages are mainly due to three key aspects:

(i) During each iteration, the neural network can directly predict the saddle point of the game without having to calculate all the intermediate states in the ODE system. This is in contrast to traditional numerical integration methods, which require all intermediate states to be calculated sequentially in order to make a prediction.

(ii) In the algorithm 1, we incorporate several important designs that exploit the structure of the problem. These include the use of softmax functions to ensure the feasibility of the mixed strategy and the implementation of a mechanism that retains the best results from each iteration.

(iii) Our approach transforms the stochastic game problem into a neural network training problem, eliminating the dependence on numerical integration solvers. This allows us to solve the problem using only deep learning infrastructure, taking advantage of the latest advances such as GPU parallel computing.

While our method demonstrates significant performance advantages, we must acknowledge its limitations:

- Our approach requires careful hyperparameter tuning of the neural network to achieve optimal performance. In contrast, numerical integration methods are more straightforward and do not require numerous algorithmic settings.

- Our approach relies on training neural networks, which can be unstable. In some cases this instability may lead to rapid convergence to a satisfactory solution, while in other cases it may take many iterations to find an appropriate solution. Possible factors contributing to this inconsistency include the initial state of the neural network and the learning rate setting. The underlying cause is due to the black-box nature of neural networks and deserves further investigation.

5. Conclusion

In this paper, we presented a novel approach to find the saddle point of stochastic two-player zero-sum games by combining neurodynamic optimization and neural networks. This results in faster convergence and superior solutions compared to traditional methods. Our work establishes a link between stochastic games and deep learning, which can benefit the stochastic game theory given the rapid growth of the deep learning community.

However, there are limitations to our work, such as the stability and robustness of the method. Future research should focus on: (a) using state-of-the-art machine learning and deep learning techniques to improve our method, (b) exploring advanced neurodynamic optimization approaches for problem modelling, and (c) extending our method to other stochastic game problems.

Bibliography

Anitescu, C., Atroshchenko, E., Alajlan, N., & Rabczuk, T. (2019). Artificial neural network methods for the solution of second order boundary value problems. *Computers, Materials and Continua*, *59*, 345–359.

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

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, *290*, 405–421.

Bogacki, P., & Shampine, L. F. (1989). A 3 (2) pair of runge-kutta formulas. *Applied Mathematics Letters*, *2*, 321–325.

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

Butcher, J. C. (2016). *Numerical methods for ordinary differential equations*. John Wiley & Sons.

Charnes, A. (1953). Constrained games and linear programming. *Proceedings of the National Academy of Sciences of the United States of America*, *39*, 639.

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

Dissanayake, M. W. M. G., & Phan-Thien, N. (1994). Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, *10*, 195–201. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303`. doi:`https://doi.org/10.1002/cnm.1640100303`. `arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.1640100303`.

Dormand, J. R., & Prince, P. J. (1980). A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, *6*, 19–26.

Hairer, E., Nørsett, S. P., & Wanner, G. (1993). *Solving ordinary differential equations. 1, Nonstiff problems*. Springer-Vlg.

Han, J., Jentzen, A., & E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, *115*, 8505–8510.

Han, J., Jentzen, A. et al. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in mathematics and statistics*, *5*, 349–380.

Henrion, R. (2007). Structural properties of linear probabilistic constraints. *Optimization*, *56*, 425–440.

Hopfield, J. J., & Tank, D. W. (1985). "neural" computation of decisions in optimization problems. *Biological cybernetics*, *52*, 141–152.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4*, 251–257.

Hu, Y., Chen, H., & Zhuang, F. (2020). Deep learning in bioinformatics: A comprehensive survey. *Briefings in Bioinformatics*, *21*, 742–761.

Kataoka, S. (1963). A stochastic programming model. *Econometrica: Journal of the Econometric Society*, (pp. 181–196).

Kennedy, M. P., & Chua, L. O. (1988). Neural networks for nonlinear programming. *IEEE Transactions on Circuits and Systems*, *35*, 554–562.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, .

Lagaris, I., Likas, A., & Fotiadis, D. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*, 987–1000. doi:10.1109/72.712178.

Lagaris, I. E., Likas, A. C., & Papageorgiou, D. G. (2000). Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, *11*, 1041–1049.

Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021a). Deepxde: A deep learning library for solving differential equations. *SIAM Review*, *63*, 208–228. URL: http://dx.doi.org/10.1137/19M1274067. doi:10.1137/19m1274067.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., & Johnson, S. G. (2021b). Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, *43*, B1105–B1132. URL: https://doi.org/10.1137/21M1397908. doi:10.1137/21M1397908. arXiv:https://doi.org/10.1137/21M1397908.

McFall, K. S., & Mahan, J. R. (2009). Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, *20*, 1221–1233.

Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P. et al. (2020). Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, .

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, *36*, 48–49. doi:10.1073/pnas.36.1.48.

Nazemi, A. (2019). A new collaborate neuro-dynamic framework for solving convex second order cone programming problems with an application in multi-fingered robotic hands. *Applied Intelligence*, *49*, 3512–3523.

von Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, *100*, 295–320. doi:10.1007/BF01448847.

Van de Panne, C., & Popp, W. (1963). Minimum-cost cattle feed under probabilistic protein constraints. *Management Science*, *9*, 405–430.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019a). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019b). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. volume 32. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf. arXiv:1912.01703.

Petzold, L. (1983). Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM journal on scientific and statistical computing*, *4*, 136–148.

Qin, S., & Xue, X. (2014). A two-layer recurrent neural network for nonsmooth convex optimization problems. *IEEE transactions on neural networks and learning systems*, *26*, 1149–1160.

Raissi, M., Perdikaris, P., & Karniadakis, G. (2019a). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125. doi:https://doi.org/10.1016/j.jcp.2018.10.045.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019b). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707.

Raza, M., Khan, S. S., & Ali, M. (2021). Deep learning for computer vision: A comprehensive review. *IEEE Access*, *9*, 62530–62558.

Samaniego, E., Anitescu, C., Goswami, S., Nguyen-Thanh, V. M., Guo, H., Hamdia, K., Zhuang, X., & Rabczuk, T. (2020). An energy approach to the solution of partial differential equations in computational

mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering*, *362*, 112790.

Shampine, L. F., & Reichelt, M. W. (1997). The matlab ode suite. *SIAM journal on scientific computing*, *18*, 1–22.

Singh, V. V., & Lisser, A. (2019). A second-order cone programming formulation for two player zero-sum games with chance constraints. *European Journal of Operational Research*, *275*, 839–845.

Sirignano, J., & Spiliopoulos, K. (2018). Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, *375*, 1339–1364.

Tan, L., & Zhang, X. (2020). Deep learning for natural language processing: A review. *IEEE Access*, *8*, 138913–138931.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., & SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi:10.1038/s41592-019-0686-2.

Wang, S., Teng, Y., & Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, *43*, A3055–A3081. URL: https://doi.org/10.1137/20M1318043. doi:10.1137/20M1318043. arXiv:https://doi.org/10.1137/20M1318043.

Wanner, G., & Hairer, E. (1996). *Solving ordinary differential equations II* volume 375. Springer Berlin Heidelberg New York.

Wu, D., & Lisser, A. (2022a). A deep learning approach for solving linear programming problems. *Neurocomputing*, .

Wu, D., & Lisser, A. (2022b). A dynamical neural network approach for solving stochastic two-player zero-sum games. *Neural Networks*, .

Wu, D., & Lisser, A. (2022c). Mg-cnn: A deep cnn to predict saddle points of matrix games. *Neural Networks*, .

Wu, D., & Lisser, A. (2022d). Using cnn for solving two-player zero-sum games. *Expert Systems with Applications*, (p. 117545).

Xia, Y., & Feng, G. (2007). A new neural network for solving nonlinear projection equations. *Neural Networks*, *20*, 577–589.

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

Xu, C., Chai, Y., Qin, S., Wang, Z., & Feng, J. (2020). A neurodynamic approach to nonsmooth constrained pseudoconvex optimization problem. *Neural Networks*, *124*, 180–192.

Yu, B. et al. (2018). The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, *6*, 1–12.

Zhang, D., Guo, L., & Karniadakis, G. E. (2020). Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks. *SIAM Journal on Scientific Computing*, *42*, A639–A665. URL: https://doi.org/10.1137/19M1260141. doi:10.1137/19M1260141. arXiv:https://doi.org/10.1137/19M1260141.