CCGnet: A deep learning approach to predict Nash equilibrium of chance-constrained games

Dawen Wu, Abdel Lisser

CCGnet: A deep learning approach to predict Nash equilibrium of chance-constrained games

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

CCGnet: A deep learning approach to predict Nash equilibrium of chance-constrained games

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 novel method for efficiently finding the Nash equilibrium in a chance-constrained games (CCG). Conventional numerical integration methods require significant computational time to solve multiple instances of CCG. We introduce CCGnet, a deep learning approach which can solve efficiently multiple instances of CCG in a one-shot manner. CCGnet uses a specialized network structure and training algorithm based on neurodynamic optimization. We present the strong performance of CCGnet in practice and show that our proposed method outperforms conventional methods.

Keywords: Chance-constrained game, Nash equilibrium, Neural network, Neurodynamic optimization

1. Introduction

Game theory analyzes the strategic interactions between rational individuals in situations involving conflict or cooperation [1]. A Nash equilibrium is a state in which no player can improve his payoff by changing his strategy unilaterally. von Neumann [2] demonstrated the existence of a saddle point for two-person zero-sum games through the minimax theorem. Nash [3] showed that an equilibrium also exists in multi-player non-zero-sum games commonly called Nash equilibrium.

The games mentioned above are all deterministic. However, many real-world situations involve games where the player's payoff function or strategy set contains randomness. Such a game with randomness is called a stochastic Nash game. Ravat & Shanbhag [4] characterized the solution set for various types of stochastic Nash games. If players are assumed to be risk-neutral, the expected payoff criterion can be used to handle the randomness in the game [4, 5].

When considering a risk-averse case, the randomness in a game can be addressed through the chance constraint programming approach, known as a chance constraint game [6, 7, 8]. In a chance constraint game (CCG), players are guaranteed to receive payoffs with a certain confidence level. For example, Singh & Lisser [7] studied a two-person zero-sum game with a random strategy set and characterized the saddle point as a primal-dual pair of second-order cone programs when the random variable follows an elliptical distribution.

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

Neurodynamic optimization uses ordinary differential equation (ODE) systems to solve optimization problems. Hopfield & Tank [9] proposed Hopfield networks for solving linear programming problems. Kennedy & Chua [10] developed an approach based on the penalty function method for solving nonlinear programming problems. However, this approach involves a penalty parameter and the optimal solution can only be obtained when the penalty term tends to infinity.. Since then, neurodynamic optimization has been well established for solving various optimization problems, such as convex optimization problems [11, 12], pseudoconvex problems [13, 14], distributed optimization problems [15, 16, 17, 18], and Nash equilibrium computation [19, 20].

Deep learning is a type of machine learning that involves the use of deep neural networks, which consist of multiple layers of interconnected nodes, to identify complex patterns and relationships in data. It has been applied successfully to various fields, including computer vision [21], natural language processing [22], bioinformatics [23], game theory [24, 25], and operation research [26, 27, 28]. However, deep learning also has limitations, and researchers are working to improve its performance and address challenges such as bias and interpretability.

Approximation methods using deep learning for differential equations were first studied in the 1990s. Dissanayake & Phan-Thien [29] used a neural network as an approximate solution to a differential equation, where the neural network was trained to satisfy the given differential equation and boundary conditions. Lagaris et al. [30] proposed a neural network model that can satisfy boundary conditions by construction. They discussed the use of this method on ODE and PDE problems, respectively. This method was extended to irregular boundaries [31]. In recent years, with the rapid development of deep learning, these methods have been further extended for solving high-dimensional PDEs [32, 33]. Flamant et al. took the parameters of the ODE system as the input variables of the neural network, allowing the neural network to be used as the solution for a group of ODE systems [34]. The rapid development of this research direction has been made possible by automatic differentiation tools, which facilitate the computation of derivatives [35, 36].

1.1. Contributions and paper outline

The main contributions of this paper can be summarized as follows:

- Our proposed CCGnet is able to receive instances of different parameters and solve them directly without any iterative process. In terms of computational time, CCGnet outperforms traditional solution approaches. This advantage becomes even more significant when solving multiple instances. For example, for 10,000 instances, the CCGnet model solves within 1.53 ms CPU time, while traditional numerical solvers take at least 22,500 ms CPU time.

- CCGnet transforms a CCG problem into a neural network training problem. Since our CCGnet model is based entirely on deep learning infrastructure, we can solve CCG without using any standard numerical solvers.

The remaining sections of this paper are organized as follows: Section 2 presents the background knowledge needed for understanding the paper, including the introduction of CCG and the neurodynamic optimization

approach. Section 3 presents our proposed CCGnet approach. Section 4 gives numerical results of using CCGnet for solving CCG. Section 5 summarizes the paper and gives future directions.

1.2. Notations

Notation	Definition
CCG	Chance-constrained game
NPE	Nonlinear projection equation
IVP	Initial value problem
CCG^θ, NPE^θ and IVP^θ	A CCG, NPE, IVP with parameter θ
$n \in \mathbb{N}$	The number of players
$x \in \mathbb{R}^n$	A strategy profile of a stochastic cournot competition
$y \in \mathbb{R}^{2n}$	Variable of a NPE
$\Phi(z) : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$	An ODE system, $\frac{dz}{dt} = \Phi(z)$
$z(t) : \mathbb{R} \to \mathbb{R}^{2n}$	A state solution
$\hat{z}(t, \theta; \mathbf{w})$	A CCGnet model with model weight $\mathbf{w}$
$(t_0, z_0) \in \mathbb{R}^{2n+1}$	An initial point
$[t_0, T] \subset \mathbb{R}$	A time range
$\theta \in \Theta$	CCG parameter

2. Preliminaries

Section 2.1.1 introduces the CCG problem, including the definition of a CCG and the existence theorem of Nash equilibria. Section 2.1.2 introduces a specific type of CCG, namely stochastic cournot games among electrical firms, and shows how to reformulate this CCG as an nonlinear projection equation (NPE). Section 2.2 introduces the neurodynamic optimization approach for solving the NPE.

2.1. Chance-constrained game

2.1.1. The model

Let an n-player game with continuous action and random payoffs be defined as a tuple $\left(I, \left(X^i\right)_{i\in I}, \left(r^i\right)_{i\in I}\right)$, where

- $I = \{1, 2, \ldots, n\}$ is a set of players.
- For each $i \in I$, let A_i be a finite action set of player i together with its generic element a_i. A vector $a = (a_1, a_2, \cdots, a_n)$ denotes an action profile of the game. Let $A = \times_{i=1}^n A_i$ be the set of all action profiles of the game. Denote, $A_{-i} = \times_{j=1;j\neq i}^n A_j$, and $a_{-i} \in A_{-i}$ is a vector of actions a_j, $j \neq i$. $x^i \in \mathbb{R}^{A_i}$ is a strategy of player i. $x \in \mathbb{R}^A$ is a strategy profile. $x^{-i} \in \mathbb{R}^{A_{-i}}$ is a strategy profile without x^i. X^i, X^{-i} and X are feasible set for x^i, x^{-i} and x, respectively.
- Let $(\Omega, \mathcal{F}, P)$ be a probability space. $\xi^i : \Omega \to \mathbb{R}^{l_i}$ is a random vector, and $f^i : X \to \mathbb{R}^{l_i}$ is a function determining player i's payoff. Consider a strategy profile $x \in X$ and an event $\omega \in \Omega$, the payoff of player i is

$$r^i(x, \omega) = f^i(x) \cdot \xi^i(\omega). \tag{1}$$

The CCG defines the payoff function of player i as

$$u_i^{\alpha_i}(x) = \sup\left\{\gamma \mid P\left(\left\{\omega \mid r^i(x,\omega) \geq \gamma\right\}\right) \geq \alpha_i\right\}, \tag{2}$$

where $\alpha_i \in [0,1]$ is a confidence level of player i, and $\alpha = (\alpha_i)_{i\in I} \in [0,1]^n$. A strategy profile x^* is a Nash equilibrium, if the following holds

$$u_i^{\alpha_i}\left(x^{i*}, x^{-i*}\right) \geq u_i^{\alpha_i}\left(x^i, x^{-i*}\right), \quad \forall x^i \in X^i. \tag{3}$$

We consider the case where each random vector $\xi^i,\ \forall i \in I$ follows an elliptically symmetric distribution, i.e., Ellip $(\mu_i, \Sigma_i, \varphi_i)$. μ_i is a location parameter. Σ_i is a positive definite matrix and φ_i is a characteristic generator function. Then, the payoff function of player i is

$$u_i^{\alpha_i}(x) = \mu_i^T f^i(x) + \left\|\Sigma_i^{1/2} f^i(x)\right\| \phi_{Z_i}^{-1}(1-\alpha_i), \tag{4}$$

where $\phi_{Z_i}^{-1}(\cdot)$ is the quantile function of the distribution Ellip $(\mu_i, \Sigma_i, \varphi_i)$.

Assumption 1. *The following conditions hold for each player i.*

- $X^i \subset \mathbb{R}^{j_i}$ *is a non-empty, convex and compact set.*
- $f_k^i : \mathbb{R}^j \to \mathbb{R}$ *is a continuous function, for all* $k = 1, 2, \ldots, l_i$.
- *For a given* $x^{-i} \in X^{-i}$, $f_k^i\left(\cdot, x^{-i}\right)$ *is an affine function, for all* $k = 1, 2, \ldots, l_i$. *Or, for a given* $x^{-i} \in X^{-i}$, $f_k^i\left(\cdot, x^{-i}\right)$ *is an non-positive and concave function, for all* $k = 1, 2, \ldots, l_i$, *and all elements of* μ_i *and* Σ_i *are non-negative.*

Theorem 1 (Singh & Lisser [37], Theorem 1)**.** *Consider a chance constrained game* $\left(I, \left(X^i\right)_{i\in I}, \left(r^i\right)_{i\in I}\right)$. *For each player* $i \in I$, *the random vector* ξ^i *follows an elliptical distribution* Ellip $(\mu_i, \Sigma_i, \varphi_i)$. *Let Assumption 1 holds. There exists a Nash equilibrium for this chance constrained game with any* $\alpha \in (0.5, 1]^n$.

2.1.2. Stochastic cournot competition

We now consider an example of CCG, namely stochastic cournot competitions among electricity firms. We show how this CCG can be reformulated as an nonlinear projection equation.

Consider an electricity market with n competing firms. $x^i \in X^i \subset \mathbb{R}_+$ denote an amount of electricity generated by firm i. Each firm i has a finite capacity C^i, i.e., $X^i = \left[0, C^i\right]$. $x = \left(x^1, x^2, \ldots, x^n\right) \in \mathbb{R}^n$ denote a strategy profile. Let $(\Omega, \mathcal{F}, P)$ be a probability space. The unit market price is determined by x and an event ω,

$$P(x,\omega) = a - b \cdot \sum_{i=1}^{n} x^i + \zeta(\omega), \tag{5}$$

where $\zeta : \Omega \to \mathbb{R}$ is a random variable, and $a \in \mathbb{R}$ and $b \in \mathbb{R}_+$ are two market price factors.

The payoff function of firm i is

$$r^i(x,\omega) = x^i \cdot P(x,\omega) - c_i\left(x^i\right), \tag{6}$$

where $c_i\left(x^i\right)$ is the cost of firm i to produce x^i amount of electricity, and $c_i(\cdot)$ is assumed to be differentiable and convex.

The chance-constraint payoff function for player i with confidence level α_i is defined as

$$u_i^{\alpha_i}(x) = \sup\left\{\gamma \mid P\left(\left\{\omega \mid x^i\left(a - b\cdot\sum_{j=1}^n x^j\right) + x^i\cdot\zeta(\omega) - c_i\left(x^i\right) \geq \gamma\right\}\right) \geq \alpha_i\right\}. \tag{7}$$

If $x^i > 0, \forall i \in I$, we have

$$\begin{aligned} u_i^{\alpha_i}(x) &= \sup\left\{\gamma \mid P\left(\left\{\omega \mid \zeta(\omega) \leq \frac{\gamma - x^i\left(a - b\cdot\sum_{j=1}^n x^j\right) + c_i\left(x^i\right)}{x^i}\right\}\right) \leq 1-\alpha_i\right\} \\ &= \sup\left\{\gamma \mid \gamma \leq x^i\left(a - b\cdot\sum_{j=1}^n x^j\right) - c_i\left(x^i\right) + x^i\phi_\zeta^{-1}(1-\alpha_i)\right\} \\ &= x^i\left(a - b\cdot\sum_{j=1}^n x^j\right) - c_i\left(x^i\right) + x^i\phi_\zeta^{-1}(1-\alpha_i). \end{aligned} \tag{8}$$

If $x^i = 0, \forall i \in I$, we have

$$u_i^{\alpha_i}(x) = -c_i\left(x^i\right). \tag{9}$$

Therefore, for a given x and α_i , the payoff of firm i is

$$u_i^{\alpha_i}(x) = x^i\left(a - b\cdot\sum_{j=1}^n x^j\right) - c_i\left(x^i\right) + x^i\phi_\zeta^{-1}(1-\alpha_i). \tag{10}$$

Definition 1. *The nonlinear complementarity problem $NCP(F)$ is to find a vector $y^* \in \mathbb{R}^m$ such that*

$$0 \leq y^* \perp F(y^*) \geq 0, \tag{11}$$

where $F : \mathbb{R}^m \to \mathbb{R}^m$.

Theorem 2 ([37])**.** *Denote a decision vector $y = \left(x^1, \ldots, x^n, \lambda^1, \ldots, \lambda^n\right) \in \mathbb{R}^{2n}$, and let $F(y) = (F_1(y), \ldots, F_{2n}(y))$, where*

$$F_i(y) = \begin{cases} -\left(a - b\sum_{j=1;j\neq i}^n x^j\right) + 2bx^i + \frac{dc_i\left(x^i\right)}{dx^i} - \phi_\zeta^{-1}(1-\alpha_i) + \lambda^i, & \text{if } i = 1,\ldots,n \\ C^{i-n} - x^{i-n}, & \text{if } i = n+1,\ldots,2n. \end{cases} \tag{12}$$

The strategy profile x^* of $y^* = (x^*, \lambda^*)$ is a Nash equilibrium of the CCG if and only if y^* is a solution of the $NCP(F)$.

Proposition 1 ([38, 39])**.** *The nonlinear projection equation $NPE(F)$ is to find a vector $y^* \in \mathbb{R}^m$ such that*

$$(y^* - F(y^*))^+ = y^*, \tag{13}$$

where $F : \mathbb{R}^m \to \mathbb{R}^m$ is a continuous function, $(y)^+ = \max(0, y)$. y^* is the solution of $NCP(F)$ if and only if y^* is the solution of $NPE(F)$.

By Theorem 1, a Nash equilibrium exists for this CCG with any $\alpha \in (0.5, 1]^n$. By Theorem 2 and Proposition 1, the Nash equilibrium of this CCG can be obtained by solving the $NPE(F)$. Note that NPEs are equivalent to variational inequalities and generalized complementarity problems, the proof is given by Browder fixed-point theorem [38, 39]. Here, Proposition 1 considers a simpler case, i.e., the equivalence between NPE and NCP.

2.2. Neurodynamic optimization

Xia & Feng [11] proposes the following ODE system to solve the $NPE(F)$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = -F\left((z)^+\right) + (z)^+ - z. \tag{14}$$

The ODE system (14) can be simplified as $\frac{\mathrm{d}z}{\mathrm{d}t} = \Phi(z)$. Let z^* be an equilibrium point of the ODE system, i.e., $\Phi(z^*) = 0$. Then, we have

$$z^* = -F\left((z^*)^+\right) + (z^*)^+ \tag{15}$$

Applying the projection operator $(\cdot)^+$ on both side, we have

$$(z^*)^+ = \left(-F\left((z^*)^+\right) + (z^*)^+\right)^+, \tag{16}$$

and hence the point $(z^*)^+$ is a solution of $NPE(F)$.

Definition 2. *Consider an ODE system $\frac{dz}{dt} = \Phi(z)$, $\Phi(z) : \mathbb{R}^m \to \mathbb{R}^m$, and a given initial point $(t_0, z_0) \in \mathbb{R}^{m+1}$. A vector value function $z(t) : \mathbb{R} \to \mathbb{R}^m$ is called a state solution, if it satisfies the initial condition $z(t_0) = z_0$ and the ODE system $\frac{dz}{dt} = \Phi(z)$.*

Definition 3. *An ODE system $\frac{dz}{dt} = \Phi(z)$ converges globally to a solution set $\mathcal{Z}^*$ if for any given initial point, the state solution $z(t)$ satisfies*

$$\lim_{t \to \infty} \operatorname{dist}(z(t), \mathcal{Z}^*) = 0, \tag{17}$$

where $\operatorname{dist}(z(t), \mathcal{Z}^*) = \inf_{z^* \in \mathcal{Z}^*} \|z(t) - z^*\|$, and $\|\cdot\|$ is the euclidean norm. In particular, if the set $\mathcal{Z}^*$ contains only one point z^*, then $\lim_{t\to\infty} z(t) = z^*$, and the ODE system is globally asymptotically stable at z^*.

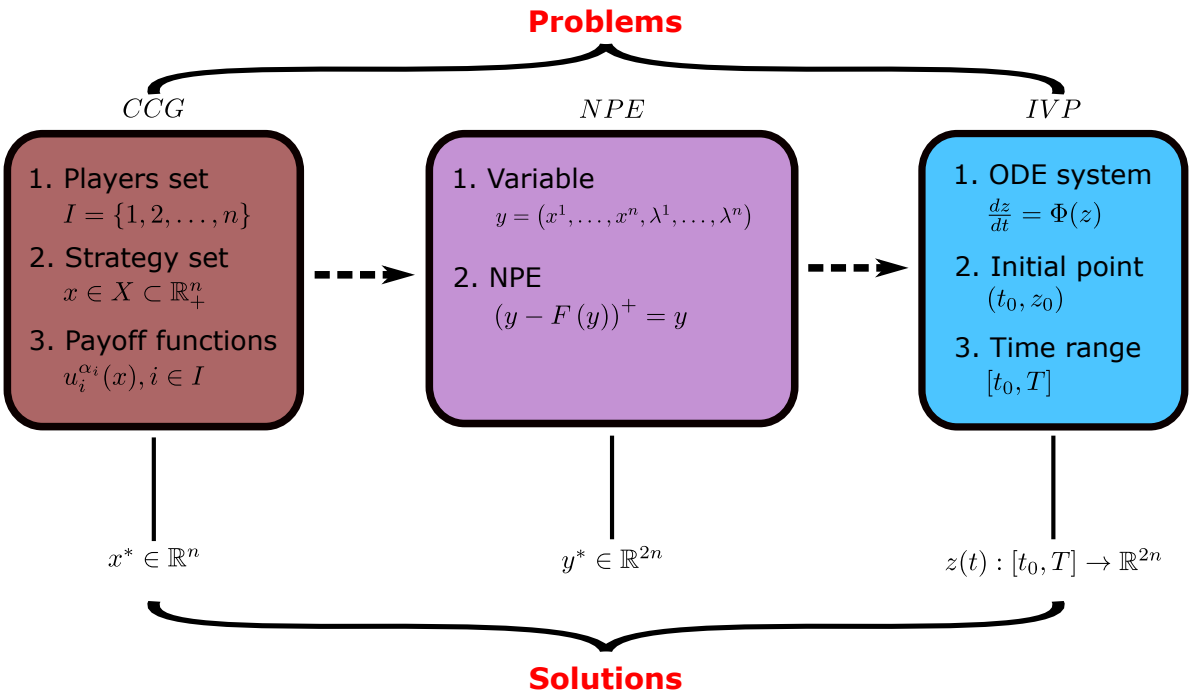

Figure 1: **The problem reformulation from a CCG to an IVP via an NPE.** We use the stochastic cournot competition for illustration. In the left box of CCG, x^* represents the Nash equilibrium, and the payoff function $u_i^{\alpha_i}(x)$ is given by (10). In the middle box of NPE, y^* represents the solution of the NPE, and the function $F(y)$ is given by (12). In the right box of IVP, $z(t), t \in [t_0, T]$ represents the state solution, and the ODE system $\Phi(z)$ is given by (14).

Theorem 3 (Xia & Feng [11], Theorem 1)**.** *If $\nabla G(z)$ is symmetrical and positive semi-definite, then the ODE system* (14) *converges globally to the solution set of $NPE(K, G)$. In particular, if $NPE(K, G)$ has only one solution z^*, then z^* is globally asymptotically stable.*

3. CCGnet

In Section 3.1, we summarize the reformulation of a CCG to an initial value problem (IVP) and present a method for parametrizing CCG instances using θ. Section 3.2 introduces the CCGnet model and its associated loss function. Section 3.3 presents the training algorithm and discusses newly introduced hyperparameters.

3.1. Problem setup

In this work, we consider the stochastic cournot game as introduced in Section 2.1.2. The CCG problem can be reformulated as an NPE where the solution $y^* = (x^*, \lambda^*)$ includes the Nash equilibrium x^* of the CCG. The neurodynamic approach, introduced in Section 2.2, models this NPE as an IVP, resulting in a state solution $z(t)$ for $t \in [t_0, T]$. According to the global convergence theorem, as T approaches infinity, $z(T)$ converges to y^*. Figure 1 summarizes this reformulation from a CCG to an IVP.

Next, we consider the case of multiple instances. We parameterize a CCG instance by $\theta \in \Theta$, where a different θ leads to a different CCG instance. The set Θ is typically an uncountably infinite set that represents a range of possible values for θ. For example, in the stochastic cournot game, θ can be the market price

factor a or b, and Θ can be $[1, 5]$. We denote the CCG instance for θ as CCG^{θ}, the corresponding NPE as NPE^{θ}, and the IVP as IVP^{θ}.

3.2. CCGnet framework

The CCGnet model is defined by the following equation:

$$\hat{z}\left(t, \theta; \mathbf{w}\right) = z_0 + (1 - e^{-(t-t_0)})\mathbf{N}\left(t, \theta; \mathbf{w}\right), \tag{18}$$

where t is an input time that falls within the time range $[t_0, T] \subset \mathbb{R}$. (t_0, z_0) is an initial point. $\mathbf{N}\left(t, \theta; \mathbf{w}\right)$ is a fully-connected neural network with weight $\mathbf{w}$. We put $\theta \in \Theta$ as an input to the neural network, allowing the CCGnet model to solve multiple CCG instances. The terms z_0 and $\left(1 - e^{-(t-t_0)}\right)$ in (18) ensure that the CCGnet model satisfies the initial condition (t_0, z_0) by construction, i.e., $\hat{z}\left(t_0, \theta; \mathbf{w}\right) = z_0$. This construction method for handling initial conditions was introduced by Lagaris et al [30]. We use an exponential multiplier of $1 - e^{-(t-t_0)}$, which has been shown to achieve better convergence than the Lagaris method [40].

The CCGnet model solves CCG^{θ}, NPE^{θ} and IVP^{θ}, for any $\theta \in \Theta$, as shown in Figure 2-(A). For a given instance of parameter θ, the CCGnet model's predicted state solution for the IVP^{θ} is $\hat{z}\left(t, \theta; \mathbf{w}\right)$, where $t \in [t_0, T]$. This is obtained by using t as a variable and keeping θ constant. The predicted solution for the NPE^{θ} is $\hat{z}\left(T, \theta; \mathbf{w}\right) = [\hat{x}_1, \ldots, \hat{x}_n, \hat{\lambda}^1, \ldots, \hat{\lambda}^n]$, which is obtained by using a constant value of T and a constant value of θ. The predicted Nash equilibrium for the CCG^{θ} is $[\hat{x}_1, \ldots, \hat{x}_n]$. Figure 2-(B) illustrates how the CCGnet model gives predictions for these three problems.

The loss function is defined by the following equation:

$$\mathcal{L}(t, \theta; \mathbf{w}) = e^{(-\tau(t-t_0))}\ell\left(\frac{\partial \hat{z}(t, \theta; \mathbf{w})}{\partial t}, \Phi^{\theta}(\hat{z}(t, \theta; \mathbf{w}))\right). \tag{19}$$

Φ^{θ} is the ODE system corresponding to CCG^{θ}. $\frac{\partial \hat{z}(t,\theta;\mathbf{w})}{\partial t}$ is the partial derivative of the CCGnet model with respect to time t. $\ell(\cdot, \cdot)$ is an error metric, e.g., mean square error. The term $\ell\left(\frac{\partial \hat{z}(t,\theta;\mathbf{w})}{\partial t}, \Phi^{\theta}(\hat{z}(t, \theta; \mathbf{w}))\right)$ represents how well the CGGnet model solves the ODE system Φ^{θ} at time t. The weighting function $e^{(-\tau(t-t_0))}$ is an exponentially decaying function with respect to time t, with $\tau \in \mathbb{R}$ as a hyperparameter. We include this weighting function to avoid initial errors which might increase the global errors exponentially [34]. For a given instance of parameter $\theta \in \Theta$ and a time $t \in [t_0, T]$, the computational flow of the loss value $\mathcal{L}(t, \theta; \mathbf{w})$ is shown in Figure 2-(C). The batch loss is defined as follows:

$$\mathcal{L}(\mathbb{T}, \theta; \mathbf{w}) = \frac{1}{|\mathbb{T}|}\sum_{t \in \mathbb{T}} \mathcal{L}(t, \theta; \mathbf{w}), \tag{20}$$

where $\mathbb{T} \subset [t_0, T]$ is the batch of t, and $|\mathbb{T}|$ represents the batch size.

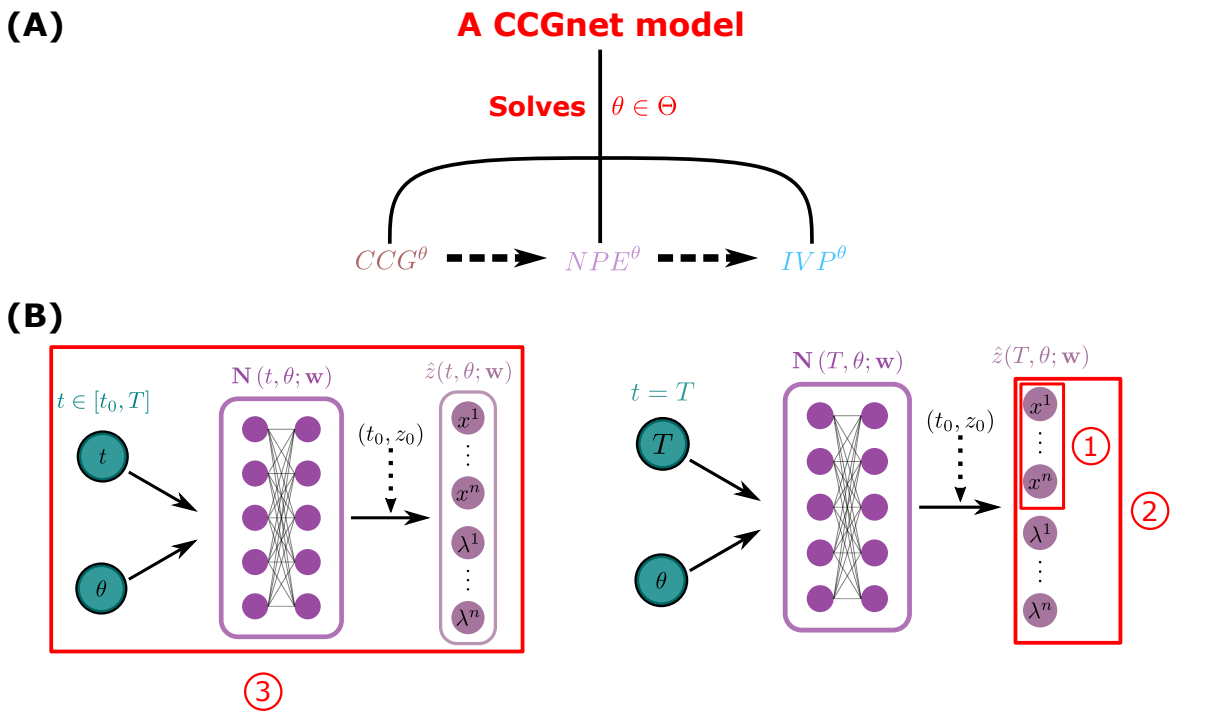

Figure 2: **CCGnet framework. (A) One CCGnet model to solve multiple CCG instances. (B) The CCGnet predictions for the CCG^θ, NPE^θ and IVP^θ. (C) The computation flow of the CCGnet loss.**

Algorithm 1: Training of a CCGnet model for solving $CCG^{\theta}\ \forall\theta \in \Theta$

Input : The CCG problem; Confidence level set Θ
Hyperparameters: Time range $[t_0, T]$; Initial point (t_0, y_0)
Output : The CCGnet model after training

Function Main():
  Initialize a CCGnet model, $\hat{y}(t, \theta; w)$.
  while iter $\leq$ Max iteration **do**
    $\theta \sim \Theta$: Uniformly sample a confidence level $\alpha \in \Theta$
    $\mathbb{T} \sim [t_0, T]$: Uniformly sample a batch of times $\mathbb{T}$ from the time range $[t_0, T]$
    Φ^{θ}: Derive the ODE system Φ^{θ} associated with the instance CCG^{θ}
    Forward propagation: Compute the batch loss $\mathcal{L}(\mathbb{T}, \theta; \mathbf{w})$
    Backward propagation: Update the weight $\mathbf{w}$ of the CCGnet model by $\nabla_{\mathbf{w}}\mathcal{L}(\mathbb{T}, \theta; \mathbf{w})$
  end
end

Algorithm 2: Training of a CCGnet model for solving $CCG^{\theta}\ \forall\theta \in \Theta$

Hyperparameters: Time range $[t_0, T]$; Initial point (t_0, z_0)
Result : The CCGnet model after training

Function Main():
  Initialize a CCGnet model.
  while iter $\leq$ Max iteration **do**
    $\theta \sim \Theta$: Uniformly sample a θ from the set Θ
    $\mathbb{T} \sim [t_0, T]$: Uniformly sample a batch of time $\mathbb{T}$ from the time range $[t_0, T]$
    Φ^{θ}: Derive the ODE system Φ^{θ} related to the instance CCG^{θ}
    Forward propagation: Compute the batch loss $\mathcal{L}(\mathbb{T}, \theta; \mathbf{w})$
    Backward propagation: Update the weight $\mathbf{w}$ of the CCGnet model by $\nabla_{\mathbf{w}}\mathcal{L}(\mathbb{T}, \theta; \mathbf{w})$
  end
end

The objective function of the CCGnet model is given by the following equation:

$$E(\mathbf{w}) = \int_{\theta\in\Theta}\int_{t\in[t_0,T]} \mathcal{L}(t, \theta; \mathbf{w}), dt, d\theta. \tag{21}$$

The goal of training the CCGnet model is to minimize the objective function, i.e.,

$$\min_{\mathbf{w}} E(\mathbf{w}). \tag{22}$$

The loss value $\mathcal{L}(t, \theta; \mathbf{w})$ measures the error of the instance with θ at time t. The objective value $E(\mathbf{w})$ measures the overall error for all instances of $\theta \in \Theta$ over the time range $[t_0, T]$.

3.3. CCGnet training

Algorithm 2 presents the training of a CCGnet model for solving $CCG^{\theta}\ \forall\theta \in \Theta$. At each training iteration, a value of θ is randomly sampled from the set Θ, and a batch of time points $\mathbb{T}$ is randomly sampled from the time range $[t_0, T]$, forming a training data $(\theta, \mathbb{T})$. The CCGnet model is then trained using this batch of data, and once the iteration finishes, the batch is discarded. The goal of the algorithm is to minimize

the objective function $E(\mathbf{w})$, and the batch loss $\mathcal{L}(\mathbb{T}, \theta; \mathbf{w})$ is an estimate of $E(\mathbf{w})$.

The time range $[t_0, T]$ is a hyperparameter that affects both the prediction accuracy and training difficulty of the CCGnet model. Given a CCG^θ and its corresponding NPE^θ with solution y^*, the initial value problem IVP^θ with time range $[t_0, T]$ has state solution $z(t)$, where $z(T) \approx y^*$. The predicted state solution $\hat{z}(t, \theta; \mathbf{w})$ approximates $z(t)$ on $[t_0, T]$, such that $\hat{z}(T, \theta; \mathbf{w}) \approx z(T) \approx y^*$. Increasing the time range $[t_0, T]$ leads to higher accuracy for $z(T)$ and, subsequently, a higher accuracy limit for $\hat{z}(T, \theta; \mathbf{w})$. However, a larger time range also makes training more challenging as the model has a larger input space to learn.

The size of the time range is a trade-off that must be considered carefully. If the time range is too small, the model will have a lower accuracy limit and will not be able to surpass it, regardless of the number of training iterations. If the time range is too large, the model will require more iterations to reach its accuracy limit. Thus, it is important to choose a time range that is appropriate for the number of training iterations.

4. Numerical results

We conducted our experiments using the Google Colab platform and built the neural network with Pytorch 1.9.1 and the ODE system with JAX 0.3.13 [41]. The hyperparameters for training are as follows:

- The ADAM optimizer [42] was used for training with a learning rate of 0.001 and a batch size of 512. The maximum number of iterations was set to 10,000.
- The CCGnet model consists of a fully connected neural network with three hidden layers, each containing 100 neurons and using the tanh activation function.
- The time range for the model was set to the interval $[0, 1]$ with an initial point of $(0, \mathbf{0})$.
- The mean squared error (MSE) was used as the error metric and the weighting hyperparameter was set to $\tau = 0.5$.

The CCGnet model was compared to four numerical integration methods: RK45, LSODA, BDF, and DOP853 [43, 44, 45, 46]. These four methods can be accessed using Scipy [47].

We consider a concrete example of stochastic cournot competitions as introduced in Section 2.1.2. The number of electricity firms are $n = 5$, and the cost function of firm i is defined as $c_i\left(x^i\right) = \left(x^i\right)^2$. The random variable follows the normal distribution $\zeta \sim N\left(\mu, \sigma^2\right)$. The Nash equilibrium of this game can be reformulated as the following NPE

$$P_Y\left(y - (My + q)\right) = y, \tag{23}$$

where $Y = \{y \in \mathbb{R}^{10} \mid y \geq 0\}$, $y = \left(x^1, x^2, x^3, x^4, x^5, \lambda^1, \lambda^2, \lambda^3, \lambda^4, \lambda^5\right)^T$,

$$M = \begin{pmatrix} 2b+2 & b & b & b & b & 1 & 0 & 0 & 0 & 0 \\ b & 2b+2 & b & b & b & 0 & 1 & 0 & 0 & 0 \\ b & b & 2b+2 & b & b & 0 & 0 & 1 & 0 & 0 \\ b & b & b & 2b+2 & b & 0 & 0 & 0 & 1 & 0 \\ b & b & b & b & 2b+2 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad q = \begin{pmatrix} -a - \phi_\zeta^{-1}(1-\alpha_1) \\ -a - \phi_\zeta^{-1}(1-\alpha_2) \\ -a - \phi_\zeta^{-1}(1-\alpha_3) \\ -a - \phi_\zeta^{-1}(1-\alpha_4) \\ -a - \phi_\zeta^{-1}(1-\alpha_5) \\ C^1 \\ C^2 \\ C^3 \\ C^4 \\ C^5 \end{pmatrix}.$$

The two market price factors are $a = 1$ and $b = 2$, The capacity of each firm is $C^1 = C^2 = C^3 = C^4 = C^5 = 5$. The confidence level of each firm are $\alpha^1 = \alpha^2 = \alpha^3 = \alpha^4 = \alpha^5 = 0.6$. The mean and the variance of the normal distribution is $\mu = 1$ and $\sigma^2 = 2$.

We use the following metric to evaluate the accuracy of the prediction $\hat{y}$

$$\epsilon = \|P_Y(\hat{y} - (M\hat{y} + q)) - \hat{y}\|. \tag{24}$$

We consider three different ways to parameterize this CCG problem. Subsection 4.1 parameterize the market price factor a as a variable. Subsection 4.2 parameterize the market price factor b as a variable. Subsection 4.3 parameterize the confidence level $\bar{\alpha}$ as a variable. We construct three independent CCGnet models, each corresponding to one subsection, then train and test these three models separately.

The experimental setup for Sections 4.1, 4.2, and 4.3 is the same and each subsection includes the following results: (1) the training loss of the CCGnet model, (2) the predicted state solutions of four IVPs corresponding to four CCG instances, (3) the predicted solutions to the four CCG instances, and (4) a comparison of CPU time between the CCGnet model and the numerical integration methods. Finally, Section 4.4 compares the advantages and limitations of our method to the numerical integration methods.

4.1. Case 1: a as variable

Index	a	CCGnet prediction	Nash equilibrium
1	1.37	[0.16, 0.16, 0.16, 0.16, 0.16]	[0.14, 0.14, 0.14, 0.14, 0.14]
2	3.27	[0.29, 0.29, 0.29, 0.29, 0.29]	[0.28, 0.28, 0.28, 0.28, 0.28]
3	2.53	[0.24, 0.24, 0.24, 0.24, 0.24]	[0.23, 0.23, 0.23, 0.23, 0.23]
4	4.20	[0.36, 0.35, 0.36, 0.35, 0.36]	[0.35, 0.35, 0.35, 0.35, 0.35]

Table 1: **Case 1: a as variable. The predicted Nash equilibrium for the four example instances.** CCGnet prediction refers to the predicted Nash equilibrium from the CCGnet model. Nash equilibrium refers to the true value.

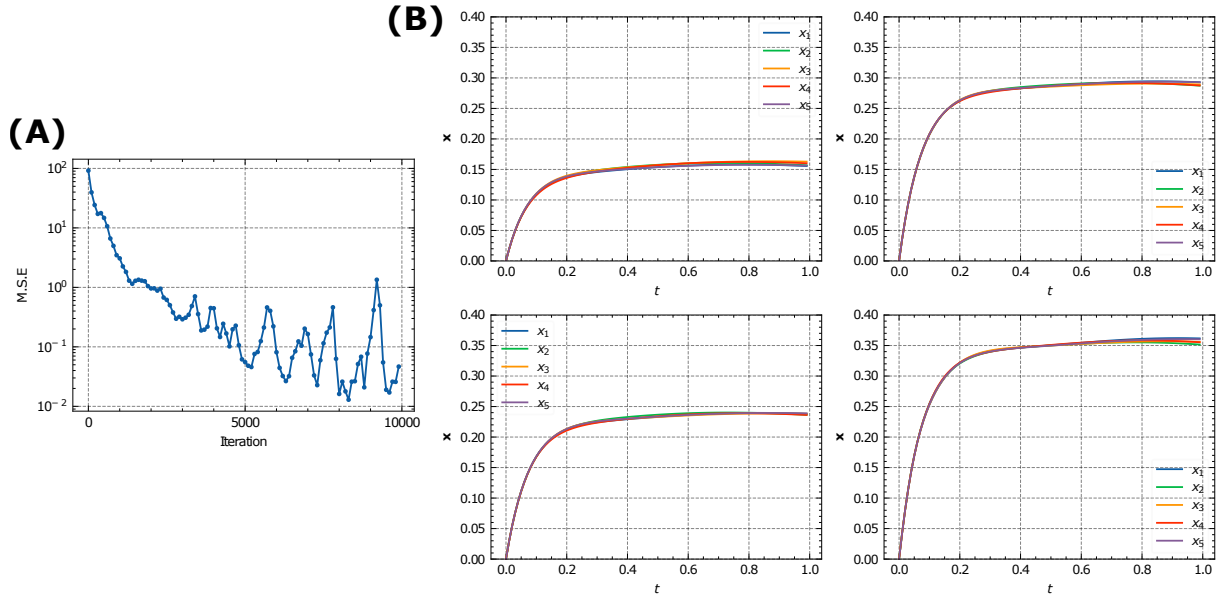

Figure 3: **Case 1: a as variable. (A) The training loss versus the number of iterations. (B) The predicted state solutions for the four example instances.** The top-left, top-right, bottom-left, and bottom-right show the results of $a = 1.37$, $a = 3.27$, $a = 2.53$, and $a = 4.20$, respectively.

We developed a CCGnet model, denoted as:

$$\hat{z}(t, a; \mathbf{w}), \quad t \in [0, 1], a \in [1, 5], \tag{25}$$

to solve for the case when the market price factor a is a variable with a parameter set of $\Theta = [1, 5]$. At each iteration, a market price factor a is uniformly sampled from the interval $[1, 5]$, and a batch of time $\mathbb{T}$ is uniformly sampled from the time range $[0, 1]$, together forming the batch $(a, \mathbb{T})$ to train the model. Figure 3-(A) shows the loss value during training, which decreases from an initial value of 91.75 to 0.05.

We selected four different values of a to represent four different instances and used the CCGnet model to solve them. Denote $\hat{z}(t, a; \mathbf{w}) = \Big(\hat{x}(t, a; \mathbf{w}), \hat{\lambda}(t, a; \mathbf{w})\Big)$. In the following results, we only present the results of $\hat{x}(t, a; \mathbf{w})$. Figure 3-(B) shows the predicted state solutions for $\hat{x}(t, a = 1.37; \mathbf{w})$, $\hat{x}(t, a = 3.27; \mathbf{w})$, $\hat{x}(t, a = 2.53; \mathbf{w})$, and $\hat{x}(t, a = 4.20; \mathbf{w})$. Table 1 shows the predicted Nash equilibria for these four instances at $t = 1$, i.e., $\hat{x}(t = 1, a = 1.37; \mathbf{w})$, $\hat{x}(t = 1, a = 3.27; \mathbf{w})$, $\hat{x}(t = 1, a = 2.53; \mathbf{w})$, and $\hat{x}(t = 1, a = 4.20; \mathbf{w})$.

Table 2 compares the computational performance of the CCGnet model, $\hat{z}(t, a; \mathbf{w})$, and the neurodynamic approach when solving multiple instances. When solving a single instance, the CCGnet model has a CPU time of less than 1 ms, which is faster than the best result of 2.23 ms for the neurodynamic approach. When solving 10,000 instances, the CCGnet model takes only 1.53 ms of CPU time, significantly faster than the best result of 22,500 ms for the neurodynamic approach. On average, the CCGnet model has an error of $\epsilon = 0.2$.

Instance number	CCGnet			Neurodynamic approach			
	CPU time (without GPU) (ms)	CPU time (with GPU) (ms)	ϵ error	RK45 CPU time (ms)	LSODA CPU time (ms)	BDF CPU time (ms)	DOP853 CPU time (ms)
1	< 1	< 1	0.27	2.51	2.23	8.94	3.41
100	< 1	< 1	0.21	250	217	852	341
500	1.07	< 1	0.20	1260	1120	4340	1760
1000	2.18	< 1	0.20	2590	2280	8830	3510
5000	8.94	1.1	0.20	12800	11200	43900	17400
10000	17.5	1.53	0.20	25500	22500	84000	34500

Table 2: **Case 1: a as variable. The computational performance of the CCGnet model and the neurodynamic approach.** Each row represents a test batch of many different instances. With or without GPU, refers to whether the CCGnet model uses CUDA. RK45, LSODA, BDF, and DOP853 are four numerical integration methods.

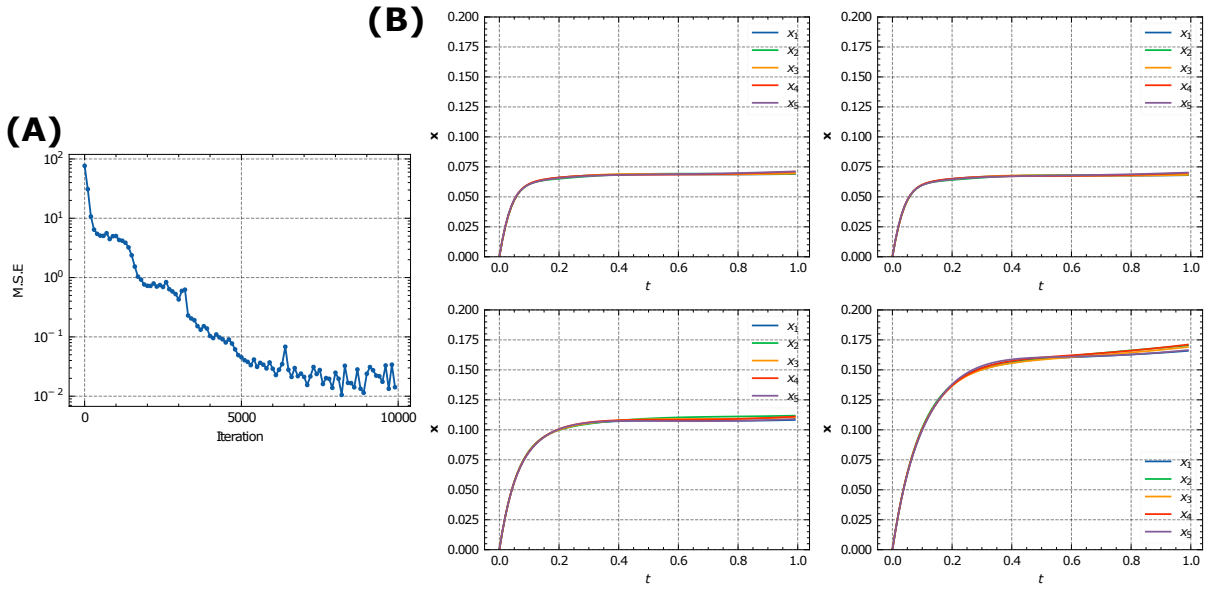

Figure 4: **Case 2: b as variable. (A) The training loss versus the number of iterations. (B) The predicted state solutions for the four example instances.** The top-left, top-right, bottom-left, and bottom-right show the results of $b = 3.83$, $b = 3.90$, $b = 2.34$, and $b = 1.46$, respectively.

Index	b	CCGnet prediction	Nash equilibrium
1	3.83	[0.07, 0.07, 0.07, 0.07, 0.07]	[0.07, 0.07, 0.07, 0.07, 0.07]
2	3.90	[0.07, 0.07, 0.07, 0.07, 0.07]	[0.06, 0.06, 0.06, 0.06, 0.06]
3	2.34	[0.11, 0.11, 0.11, 0.11, 0.11]	[0.10, 0.10, 0.10, 0.10, 0.10]
4	1.46	[0.17, 0.17, 0.17, 0.17, 0.17]	[0.15, 0.15, 0.15, 0.15, 0.15]

Table 3: **Case 2: b as variable. The predicted Nash equilibrium for the four example instances.** CCGnet prediction refers to the predicted Nash equilibrium from the CCGnet model. Nash equilibrium refers to the true value.

4.2. Case 2: b as variable

We developed a CCGnet model, denoted as:

$$\hat{z}(t, b; \mathbf{w}), \quad t \in [0, 1], b \in [1, 5], \tag{26}$$

to solve for the case when the market price factor b is a variable with a parameter set of $\Theta = [1, 5]$. At each iteration, a market price factor b is uniformly sampled from the interval $[1, 5]$ and a batch of time points, denoted as $\mathbb{T}$, is uniformly sampled from the time range $[0, 1]$. These two samples, $(b, \mathbb{T})$, form a batch used to train the model. Figure 4-(A) shows the loss value during training, which decreases from an initial value of 76.51 to 0.01.

We tested this CCGnet model on four example instances with different values of b, specifically 3.83, 3.90, 2.34, and 1.46. Figure 4-(B) shows the predicted state solutions for $\hat{x}(t, b = 3.83; \mathbf{w})$, $\hat{x}(t, b = 3.90; \mathbf{w})$, $\hat{x}(t, b = 2.34; \mathbf{w})$, and $\hat{x}(t, b = 1.46; \mathbf{w})$. Table 3 shows the predicted Nash equilibria for these four instances at $t = 1$, i.e., $\hat{x}(t = 1, b = 3.83; \mathbf{w})$, $\hat{x}(t = 1, b = 3.90; \mathbf{w})$, $\hat{x}(t = 1, b = 2.34; \mathbf{w})$, and $\hat{x}(t = 1, b = 1.46; \mathbf{w})$.

Instance number	CCGnet			Neurodynamic approach			
	CPU time (ms)	CPU time (with GPU) (ms)	ϵ error	RK45 CPU time (ms)	LSODA CPU time (ms)	BDF CPU time (ms)	DOP853 CPU time (ms)
1	< 1	< 1	0.04	3.09	2.64	8.70	3.68
100	< 1	< 1	0.07	282	247	841	355
500	1.03	< 1	0.07	1510	1410	4290	1840
1000	2.02	< 1	0.07	3000	2620	8550	3740
5000	8.61	1.07	0.07	15000	13000	42700	18500
10000	17.1	1.50	0.07	30200	26100	84000	36900

Table 4: **Case 2: b as variable. The computational performance of the CCGnet model and the neurodynamic approach.** Each row represents a test batch of many different instances. With or without GPU, refers to whether the CCGnet model uses CUDA. RK45, LSODA, BDF, and DOP853 are four numerical integration methods.

Table 4 compares the computational performance of the CCGnet model, $\hat{z}(t, b; \mathbf{w})$, and the neurodynamic approach. When solving a single instance, the CCGnet model has a CPU time of less than 1 ms, outperforming the best result of 2.64 ms for the neurodynamic approach. When solving 10,000 instances, the CCGnet model takes only 1.50 ms of CPU time, faster than the best result of 26,100 ms for the neurodynamic approach. On average, the CCGnet model has an error of $\epsilon = 0.07$.

4.3. Case 3: $\bar{\alpha}$ as variable

Index	$\bar{\alpha}$	CCGnet endpoint	Nash equilibrium
1	0.60	[0.11, 0.11, 0.11, 0.11, 0.11]	[0.12, 0.12, 0.12, 0.12, 0.12]
2	0.90	[0.01, 0.01, 0.01, 0.01, 0.01]	[0.01, 0.01, 0.01, 0.01, 0.01]
3	0.67	[0.09, 0.09, 0.09, 0.09, 0.09]	[0.10, 0.10, 0.10, 0.10, 0.10]
4	0.75	[0.06, 0.06, 0.06, 0.07, 0.06]	[0.08, 0.08, 0.08, 0.08, 0.08]

Table 5: **Case 3: $\bar{\alpha}$ as variable. The predicted Nash equilibrium for the four example instances.** CCGnet prediction refers to the predicted Nash equilibrium from the CCGnet model. Nash equilibrium refers to the true value.

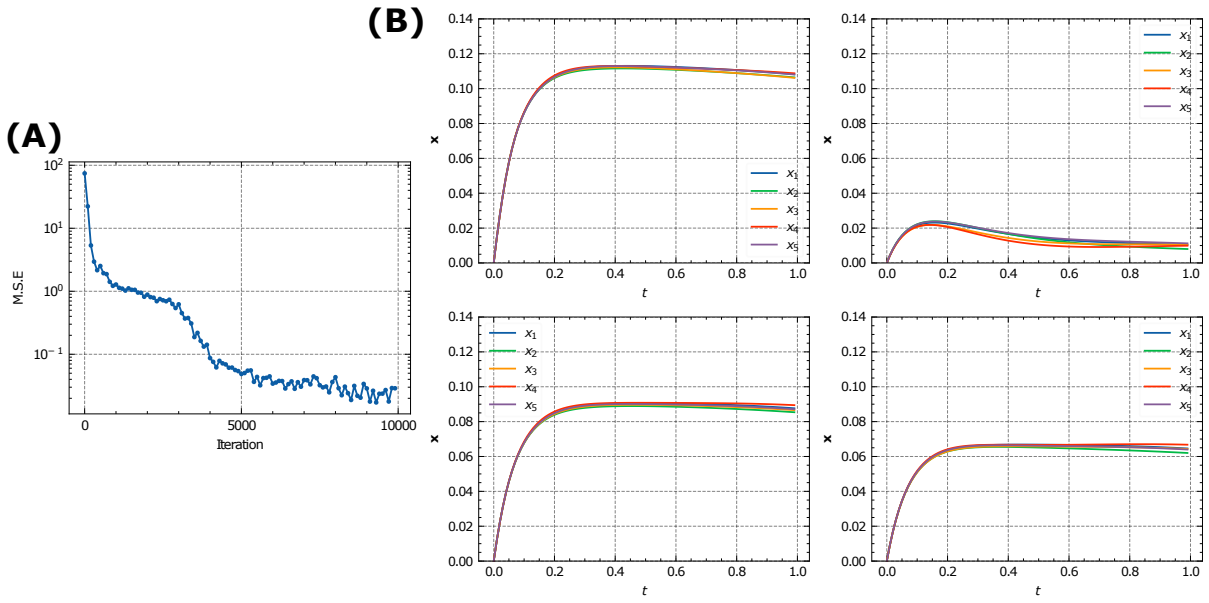

Figure 5: **Case 3: $\bar{\alpha}$ as variable. (A) The training loss versus the number of iterations (B) The predicted state solutions for the four instances** The top-left, top-right, bottom-left, and bottom-right show the results of $\bar{\alpha} = 0.60$, $al\bar{p}ha = 0.90$, $\bar{\alpha} = 0.67$, and $\bar{\alpha} = 0.75$, respectively.

We studied the case where the confidence level is a variable and assumed that $\alpha^1 = \alpha^2 = \alpha^3 = \alpha^4 = \alpha^5$. We denote this common value as $\bar{\alpha}$ and build a CCGnet model, denoted as:

$$\hat{z}(t, \alpha; \mathbf{w}), \quad t \in [0, 1], \alpha \in [0.5, 0.9], \tag{27}$$

to solve for this case when $\bar{\alpha}$ is a variable with a parameter set of $\Theta = [0.5, 0.9]$. At each iteration, a value of $\bar{\alpha}$ is uniformly sampled from the interval $[0.5, 0.9]$ and a batch of time points, denoted as $\mathbb{T}$, is uniformly sampled from the time range $[0, 1]$. These two samples, $(\bar{\alpha}, \mathbb{T})$, form a batch used to train the model. Figure 5-(A) shows the loss value during training, which decreases from an initial value of 74.39 to 0.03.

We tested this CCGnet model on four example instances with different values of $\bar{\alpha}$, specifically 0.60, 0.90, 0.67, and 0.75. Figure 5-(B) shows the predicted state solutions for $\hat{x}(t, \bar{\alpha} = 0.60; \mathbf{w})$, $\hat{x}(t, \bar{\alpha} = 0.90; \mathbf{w})$, $\hat{x}(t, \bar{\alpha} = 0.67; \mathbf{w})$, and $\hat{x}(t, \bar{\alpha} = 0.75; \mathbf{w})$. Table 5 shows the predicted Nash equilibria for these four instances at $t = 1$, i.e., $\hat{x}(t = 1, \bar{\alpha} = 0.60; \mathbf{w})$, $\hat{x}(t = 1, \bar{\alpha} = 0.90; \mathbf{w})$, $\hat{x}(t = 1, \bar{\alpha} = 0.67; \mathbf{w})$, and $\hat{x}(t = 1, \bar{\alpha} = 0.75; \mathbf{w})$.

Table 6 compares the computational performance of the CCGnet model, $\hat{z}(t, \alpha; \mathbf{w})$, and the neurodynamic approach. When solving a single instance, the CCGnet model has a CPU time of less than 1 ms, faster than the best result of 2.27 ms for the neurodynamic approach. When solving 10,000 instances, the CCGnet model takes only 1.14 ms of CPU time, significantly faster than the best result of 24,300 ms for the neurodynamic approach. On average, the CCGnet model has an error of $\epsilon = 0.15$.

Instance number	CCGnet			Neurodynamic approach			
	CPU time (ms)	CPU time (with GPU) (ms)	ϵ error	RK45 CPU time (ms)	LSODA CPU time (ms)	BDF CPU time (ms)	DOP853 CPU time (ms)
1	< 1	< 1	0.16	2.57	2.27	8.02	3.47
100	< 1	< 1	0.15	267	225	838	347
500	1.02	< 1	0.16	1320	1180	4230	1770
1000	2.05	< 1	0.15	2640	2420	8440	3540
5000	8.52	1.08	0.15	13200	12200	42300	17700
10000	17.9	1.41	0.15	26600	24300	87000	35700

Table 6: **Case 3: $\bar{\alpha}$ as variable. The computational performance of the CCGnet model and the neurodynamic approach.** Each row represents a test batch of many different instances. With or without GPU, refers to whether the CCGnet model uses CUDA. RK45, LSODA, BDF, and DOP853 are four numerical integration methods.

4.4. Discussion

The main advantage of the CCGnet model is its computational performance. It can directly predict the Nash equilibrium without any iterative process, making it much faster than numerical integration methods. This advantage becomes even more significant when there are a large number of instances to solve. For example, when solving 10,000 different instances, the CCGnet model can predict all the Nash equilibria in a one-shot manner with only 1.5 ms, while numerical methods require more than 20,000 ms to solve each instance one after another. Additionally, the CCGnet model can utilize a GPU to further accelerate the solution process, while GPU-based numerical methods are still under development.

One limitation of the CCGnet model is its prediction accuracy compared to exact solutions obtained through numerical integration methods. While numerical integration methods can provide exact solutions given sufficient computational time, the CCGnet model can only provide predictions. The accuracy of these predictions depends on technical details such as the neural network structure, training algorithm, and hyperparameter settings, which are active areas of research in machine learning and deep learning. As these areas progress, the CCGnet model has the potential to improve its prediction accuracy.

5. Conclusion

This paper presents a deep learning approach called CCGnet for solving chance-constrained games. CCGnet is based on neurodynamic optimization, which models a chance-constrained game as an ODE system. One of the key benefits of CCGnet is its ability to solve multiple instances in a very short amount of CPU time, significantly faster than traditional methods. The paper provides a detailed description of the proposed method, including the parametrization of CCG instances, the model framework, the training algorithm, and a discussion of hyperparameters.

However, it is important to note that the proposed method should not be considered a replacement for standard solvers like RK45 and BDF. These methods have been well-developed over many years. Our purpose is to link the machine learning community and CCG. We believe that with the rapid growth of research on

machine learning, both methodologically and experimentally, this paper will continue to contribute to the efficient solution of CCG problems.

There are many potential avenues for future research. Some examples include: 1) Choosing the hyperparameter initial point to be all zero may not always lead to the best computational performance. Is it possible to find other choices that lead to better results? 2) We used a uniform distribution to sample the dataset. Could other sampling methods lead to better results? 3) We used a fully-connected network structure. How can we design a more suitable neural network structure and activation function?

Bibliography

[1] Algorithmic Game Theory, Cambridge University Press, 2007. doi:10.1017/CBO9780511800481.

[2] J. von Neumann, Zur theorie der gesellschaftsspiele, Mathematische annalen 100 (1) (1928) 295–320.

[3] J. F. Nash, et al., Equilibrium points in n-person games, Proceedings of the national academy of sciences 36 (1) (1950) 48–49.

[4] U. Ravat, U. V. Shanbhag, On the characterization of solution sets of smooth and nonsmooth convex stochastic nash games, SIAM Journal on Optimization 21 (3) (2011) 1168–1199.

[5] H. Jiang, U. V. Shanbhag, S. P. Meyn, Distributed computation of equilibria in misspecified convex stochastic nash games, IEEE Transactions on Automatic Control 63 (2) (2017) 360–371.

[6] V. V. Singh, A. Lisser, A characterization of nash equilibrium for the games with random payoffs, Journal of Optimization Theory and Applications 178 (3) (2018) 998–1013.

[7] V. V. Singh, A. Lisser, A second-order cone programming formulation for two player zero-sum games with chance constraints, European Journal of Operational Research 275 (3) (2019) 839–845.

[8] H. N. Nguyen, A. Lisser, V. V. Singh, Random games under elliptically distributed dependent joint chance constraints, Journal of Optimization Theory and Applications 195 (1) (2022) 249–264.

[9] J. J. Hopfield, D. W. Tank, "neural" computation of decisions in optimization problems, Biological cybernetics 52 (3) (1985) 141–152.

[10] M. P. Kennedy, L. O. Chua, Neural networks for nonlinear programming, IEEE Transactions on Circuits and Systems 35 (5) (1988) 554–562.

[11] Y. Xia, G. Feng, A new neural network for solving nonlinear projection equations, Neural Networks 20 (5) (2007) 577–589.

[12] A. Nazemi, A. Sabeghi, A new neural network framework for solving convex second-order cone constrained variational inequality problems with an application in multi-finger robot hands, Journal of Experimental & Theoretical Artificial Intelligence 32 (2) (2020) 181–203.

[13] J. Liu, X. Liao, A projection neural network to nonsmooth constrained pseudoconvex optimization, IEEE Transactions on Neural Networks and Learning Systems (2021).

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

[15] Y.-W. Lv, G.-H. Yang, C.-X. Shi, Differentially private distributed optimization for multi-agent systems via the augmented lagrangian algorithm, Information Sciences 538 (2020) 39–53.

[16] J. Zou, R. Sun, S. Yang, J. Zheng, A dual-population algorithm based on alternative evolution and degeneration for solving constrained multi-objective optimization problems, Information Sciences 579 (2021) 89–102.

[17] Z. Wang, J. Liu, D. Wang, W. Wang, Distributed cooperative optimization for multiple heterogeneous euler-lagrangian systems under global equality and inequality constraints, Information Sciences 577 (2021) 449–466.

[18] C. Xu, Q. Liu, T. Huang, Resilient penalty function method for distributed constrained optimization under byzantine attack, Information Sciences 596 (2022) 362–379.

[19] C.-X. Shi, G.-H. Yang, Distributed nash equilibrium computation in aggregative games: An event-triggered algorithm, Information Sciences 489 (2019) 289–302.

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

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

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

[23] Y. Hu, H. Chen, F. Zhuang, Deep learning in bioinformatics: A comprehensive survey, Briefings in Bioinformatics 21 (3) (2020) 742–761.

[24] D. Wu, A. Lisser, Using cnn for solving two-player zero-sum games, Expert Systems with Applications (2022) 117545.

[25] D. Wu, A. Lisser, Mg-cnn: A deep cnn to predict saddle points of matrix games, Neural Networks (2022).

[26] D. Wu, A. Lisser, A deep learning approach for solving linear programming problems, Neurocomputing (2022).

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

[28] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, European Journal of Operational Research 290 (2) (2021) 405–421.

[29] M. W. M. G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Communications in Numerical Methods in Engineering 10 (3) (1994) 195–201. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.1640100303, doi:https://doi.org/10.1002/cnm.1640100303.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303

[30] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (5) (1998) 987–1000. doi:10.1109/72.712178.

[31] K. S. McFall, J. R. Mahan, Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions, IEEE Transactions on Neural Networks 20 (8) (2009) 1221–1233.

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

[33] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences 115 (34) (2018) 8505–8510.

[34] C. Flamant, P. Protopapas, D. Sondak, Solving differential equations using neural network solution bundles (2020). arXiv:2006.14372.

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

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

[37] V. V. Singh, A. Lisser, Variational inequality formulation for the games with random payoffs, Journal of Global Optimization 72 (4) (2018) 743–760.

[38] P. T. Harker, J.-S. Pang, Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications, Mathematical programming 48 (1) (1990) 161–220.

[39] B. C. Eaves, On the basic theorem of complementarity, Mathematical Programming 1 (1) (1971) 68–75. doi:10.1007/BF01584073.
URL https://doi.org/10.1007/BF01584073

[40] M. Mattheakis, D. Sondak, P. Protopapas, Hamiltonian neural networks for solving equations of motion, arXiv preprint arXiv:2001.11107 (2020).

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

[42] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2014). doi:10.48550/ARXIV.1412.6980.
URL https://arxiv.org/abs/1412.6980

[43] J. R. Dormand, P. J. Prince, A family of embedded runge-kutta formulae, Journal of computational and applied mathematics 6 (1) (1980) 19–26.

[44] L. Petzold, Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations, SIAM journal on scientific and statistical computing 4 (1) (1983) 136–148.

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

[46] E. Hairer, S. P. Nørsett, G. Wanner, Solving ordinary differential equations. 1, Nonstiff problems, Springer-Vlg, 1993.

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