MG-CNN: A Deep CNN To Predict Saddle Points Of Matrix Games

Dawen Wu, Abdel Lisser

MG-CNN: A Deep CNN To Predict Saddle Points Of Matrix 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

MG-CNN: A Deep CNN To Predict Saddle Points Of Matrix Games

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

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

Abstract

Finding the saddle point of a matrix game is a classical problem that arises in various fields, e.g., economics, computer science, and engineering. The standard problem-solving methods consist of formulating the problem as a linear program (LP). However, this approach seems to be inappropriate, especially when multiple instances need to be solved. In this paper, we propose a Convolutional Neural Network based approach, which is able to predict both the strategy profile (x, y) and the optimal value v of the game. We call this approach Matrix Game-Conventional Neural Network or MG-CNN for short. Thanks to a global pooling technique, MG-CNN can solve matrix games with different shapes. We propose a specialized algorithm to train MG-CNN, which includes both data generation and model training. Our numerical experiments show that MG-CNN outperforms standard LP solvers in terms of computational CPU time and provides a high quality prediction.

Keywords: Convolutional neural network, Saddle point, Matrix game

1. Introduction

Game theory is a theoretical framework to study conflict and cooperation among rational agents in a strategic setting. The assumption behind game theory is that agents choose actions that maximize their own payoffs (Fudenberg & Tirole, 1991). In this paper, we call matrix game the two-player zero-sum game problem with the following characteristics. Firstly, the players' action sets are discrete and finite. Secondly, for a given choice of actions, the payoff for both players adds up to zero. Matrix games model many real-world problems to help decision-makers make the right decisions in a competitive environment (Dixit & Pindyck, 2012; Simmons, 1998; Vega-Redondo, 2003; Singh, 1999). There are also studies on the situation where games contain random variables (Cheng et al., 2016; Singh & Lisser, 2019; Wu & Lisser, 2022a).

The saddle point or Nash equilibrium is the most common way to define the solution concept of a non-cooperative game. These solutions describe a situation where no player can improve his payoff by unilaterally changing his strategy. A saddle point refers to the solution of a two-player zero-sum game. von Neumann (1928) proved the minimax theorem, which states that there exists a saddle point for any finite two-person zero-sum game. A Nash equilibrium refers to the solution of a n-player game. Nash (1950) proved that there

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

always exists an equilibrium for any finite n-player games called Nash equilibrium.

With the rapid growth of available data and computing power, deep learning in the form of deep neural networks is used in many areas, such as image processing (Krizhevsky et al., 2017; Goodfellow et al., 2016), natural language processing (Devlin et al., 2018), recommender system(Zhang et al., 2019), see also (Silver et al., 2016; Jumper et al., 2021) for additional application examples. Convolutional neural networks (CNNs) are deep neural networks that are frequently used in computer vision (LeCun & Bengio, 1995; Courville et al., 2016). A CNN uses a shared weight convolutional kernel that slides along the input feature maps. The architecture of a neural network model is crucial for the performance enhancement, and there are specialized network structures for various situations (Krizhevsky et al., 2017; He et al., 2016; Huang et al., 2017; Tan & Le, 2019; Alzubaidi et al., 2021; Khan et al., 2020; Xu & Zhang, 2022). PyTorch is an open-source library for building neural network models provided by CUDA support to speed up processing by harnessing the power of GPUs (Paszke et al., 2019).

Generative Adversarial Networks (GANs) is a novel deep learning method, widely used for image generation and text generation(Goodfellow et al., 2020; Zhou et al., 2019; Dasgupta & Collins, 2019; Tembine, 2020). A GAN contains two neural networks, a generator, and a discriminator. These two neural networks are trained by competing against each other. We would like to emphasize the relationship and difference between this paper and GAN. The latter is a two-person zero-sum game with continuous actions. The generator and the discriminator are two players whose model parameters are viewed as actions and have opposite payoff functions. The training purpose for GANs is to find a saddle point $(\mathbf{w}_1, \mathbf{w}_2)$ of the two neural networks. Our paper considers a two-player zero-sum game with finite actions, and our goal is to use a CNN to predict the saddle point of such a game.

1.1. Related works

We present here two approaches for solving matrix games, i.e., the LP approach and the WL-CNN approach (Wu & Lisser, 2022b).

The LP approach. The conventional way to solve a matrix game is to convert it into a linear programming (LP) problem. The LP can then be solved by the simplex method (Dantzig, 1963; Bertsimas & Tsitsiklis, 1997; Vanderbei, 2014), or interior-point methods(Wright, 1997; Karmarkar, 1984). Here, We introduce four optimization solvers, i.e., ECOS, GLPK, SCS, and Gurobi. ECOS, embedded conic solver, is an efficient implementation of an interior-point method for second-order cone programming (Domahidi et al., 2013) designed for embedded applications. GLPK is a classical solver first released in 2000, which is dedicated only to LP problems (GLPK, 2012). SCS, splitting conic solver, is based on a first-order method to solve convex cone programs, which can scale to large problems and solve many types of convex nonlinear problems (Brendan et al., 2016). Gurobi is a well-established commercial solver with a large number of applications in the industry, it is known for its good performances in many benchmarks (Gurobi Optimization, LLC, 2021; Mittelmann, 2022). These four solvers represent a large part of the state-of-the-art (SOTA) for solving LPs.

	Optimal value v^*	Strategy profile (x^*, y^*)
MG-CNN	**CNN model**	**CNN model**
WL-CNN	CNN model	Convex optimization solver
LP	Convex optimization solver	Convex optimization solver

Table 1: **Three approaches to solve a matrix game** The saddle point is divided into two components, the optimal value v^*, and the strategy profile (x^*, y^*). Our MG-CNN approach predicts both the optimal value and strategy profile by a CNN model. The WL-CNN approach predicts the optimal value by a CNN model and then obtains the strategy profile by calling an optimization solver. The LP approach solves both the optimal value and strategy profile by an optimization solver.

Their computational performance varies depending on the specific problem instances, experimental environment, and parameter choices. Mittelmann (2022) provides a platform for testing the performance of different optimization solvers (Jared L, Adair, Kristin L, Detry, Richard J, Durfee, Justin D, Jones & A, Martin, 2012). CVXPY is a modeling package for convex optimization problems, which translates the problem into a standard form and can call sereval optimization solvers (Diamond & Boyd, 2016).

The WL-CNN approach. Wu & Lisser (2022b) proposed a CNN-based method to predict saddle points, which we call in this paper WL-CNN. This approach predicts the saddle point in two steps. First, it uses a CNN model to predict the optimal value v. Then, after getting the prediction v, the related strategy profile (x, y) is obtained by solving a linear system of inequalities.

The LP approach is time-consuming, especially when solving several instances. Although the WL-CNN method shows some computational speed advantages, its CNN model can only predict the optimal value. It still requires solving an inequalities system to obtain a strategy profile. Unlike WL-CNN, our MG-CNN model can directly predict both the optimal value and strategy profile. Table 1 summarizes how these three approaches obtain the saddle point of a matrix game.

1.2. Contributions

The contributions of this paper can be summarized as follows

- Our proposed MG-CNN outperforms both WL-CNN and LP approaches in terms of computational speed. For example, when solving one matrix game of size $(10, 10)$, MG-CNN takes only 5.1 ms, while WL-CNN takes 5.8 ms and LP takes 12.6 ms. When solving multiple instances, the computational advantage becomes much more significant. MG-CNN solves 1000 instances of size $(10, 10)$ in 73.2 ms, WL-CNN in 4520 ms, and LP in 18300 ms.

- Our proposed MG-CNN outperforms WL-CNN in terms of prediction accuracy. For a matrix game of size $(20, 20)$ and generated by $U(0, 100)$, MG-CNN has a mean absolute percentage error of 1.73%, lower than 2.83% of WL-CNN. Additionally, MG-CNN is more reliable across a range of probability distributions and game sizes, whereas WL-CNN fails in many situations.

The remaining of this paper is organized as follows. Section 2 sketches the minimum knowledge to understand the problem, including the matrix game and the LP formulation. Section 3 presents our proposed

MG-CNN approach. Section 4 gives the comparative experimental results among the three approaches. Section 5 discusses the pros and cons of MG-CNN. Section 6 summarizes the whole paper and gives future directions.

1.3. Notations

Table 2 presents the notations list of this paper.

Notation	Definition
$n \in \mathbb{N}$	The number of actions for player 1.
$m \in \mathbb{N}$	The number of actions for player 2.
$\mathbf{A} \in \mathbb{R}^{n \times m}$	A matrix game represented by the payoff matrix of player 1.
(x^*, y^*), $x^* \in \mathbb{R}^n, y^* \in \mathbb{R}^m$	The strategy profile of the saddle point.
$v^* \in \mathbb{R}$	The optimal value of the saddle point.
MG-CNN$(\mathbf{A}; \mathbf{w})$	The MG-CNN model with parameters $\mathbf{w}$.
(x, y), $x \in \mathbb{R}^n, y \in \mathbb{R}^m$	The strategy profile prediction from the MG-CNN model.
$v \in \mathbb{R}$	The optimal value prediction from the MG-CNN model.
$\mathcal{L}(\mathbf{A}; \mathbf{w})$	The loss function of the MG-CNN model.
P	The probability distribution used to generate $\mathbf{A}$.
$\min \mathop{\mathbb{E}}_{\mathbf{A} \sim P}[\mathcal{L}(\mathbf{A}; \mathbf{w})]$	The objective function of the MG-CNN model.

Table 2: Notations

2. Preliminaries

2.1. Matrix game

We denote the two players in a matrix game as players 1 and 2, respectively. Players 1 and 2 have n and m discrete actions, respectively. The action sets of the two players can be written as $\mathcal{X} = \{1, \ldots, n\}$, $\mathcal{Y} = \{1, \ldots, m\}$, respectively. When the row player selects action $i \in \mathcal{X}$, the column player selects action $j \in \mathcal{Y}$, the payoff for the players 1 and 2 are a_{ij} and $-a_{ij}$, respectively. Thus, player 1's payoff can be represented by the matrix $\mathbf{A}$ (1), while player 2's payoff can be represented by the matrix $-\mathbf{A}$.

Throughout this paper, a matrix game is represented by the payoff matrix $\mathbf{A}$, where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nm} \end{bmatrix}. \tag{1}$$

We denote x and y as mixed strategies for players 1 and 2, respectively. A mixed strategy is a discrete probability distribution over the player's action set. The vectors x and y should be within probability simplices, i.e., $x \in \{x \in \mathbb{R}^n : \mathbf{e}_\mathbf{n}^T x = 1, x \geq \mathbf{0}\}$, $y \in \{y \in \mathbb{R}^m : \mathbf{e}_\mathbf{m}^T y = 1, y \geq \mathbf{0}\}$, where $\mathbf{e}_k = [1, \ldots, 1]^T \in \mathbb{R}^k$.

Theorem 1 (von Neumann (1928)). *Let $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$ be compact convex sets. If $f : X \times Y \to \mathbb{R}$ is a continuous function that is concave-convex.*

Then, the following holds

$$\max_{x \in X} \min_{y \in Y} f(x, y) = \min_{y \in Y} \max_{x \in X} f(x, y). \tag{2}$$

The saddle point existence of a matrix game is guaranteed by Theorem 1. The optimal strategy profile of a matrix game $\mathbf{A}$ is defined as follows,

$$(x^*, y^*) = \arg\max_x \left(\arg\min_y x^T \mathbf{A} y \right), \tag{3}$$

and the corresponding optimal value is

$$v^* = \max_x \left(\min_y x^T \mathbf{A} y \right). \tag{4}$$

2.2. LP to solve Matrix game

The saddle point of a matrix game $\mathbf{A}$ can be obtained by solving the primal-dual LP pair (5)-(6),

$$\begin{aligned} &\text{(P)} \ \max \ v \\ &\quad \text{s.t.} \ \mathbf{A}^T x \geq v \mathbf{e_m} \\ &\qquad \mathbf{e_n}^T x = 1, x \geq \mathbf{0}, \end{aligned} \tag{5}$$

$$\begin{aligned} &\text{(D)} \ \min \ v \\ &\quad \text{s.t.} \ \mathbf{A} y \leq v \mathbf{e_n} \\ &\qquad \mathbf{e_m}^T y = 1, y \geq \mathbf{0}. \end{aligned} \tag{6}$$

The primal LP (5) has decision variable $(x, v) \in \mathbb{R}^{n+1}$, one equality constraint and m inequality constraints. The dual LP (6) has decision variable $(y, v) \in \mathbb{R}^{m+1}$, one equality constraint and n inequality constraints. The optimal solutions for these two LP problems are (x^*, v^*) and (y^*, v^*), where (x^*, y^*) is the desired optimal strategy profile, and v^* is the desired optimal value. For the construction and related details, we refer the reader to Dantzig (1963).

3. MG-CNN

3.1. MG-CNN model

The MG-CNN model can be represented by the following mathematical expressions

$$\text{MG-CNN}(\mathbf{A}; \mathbf{w}) = (x, y), \tag{7}$$

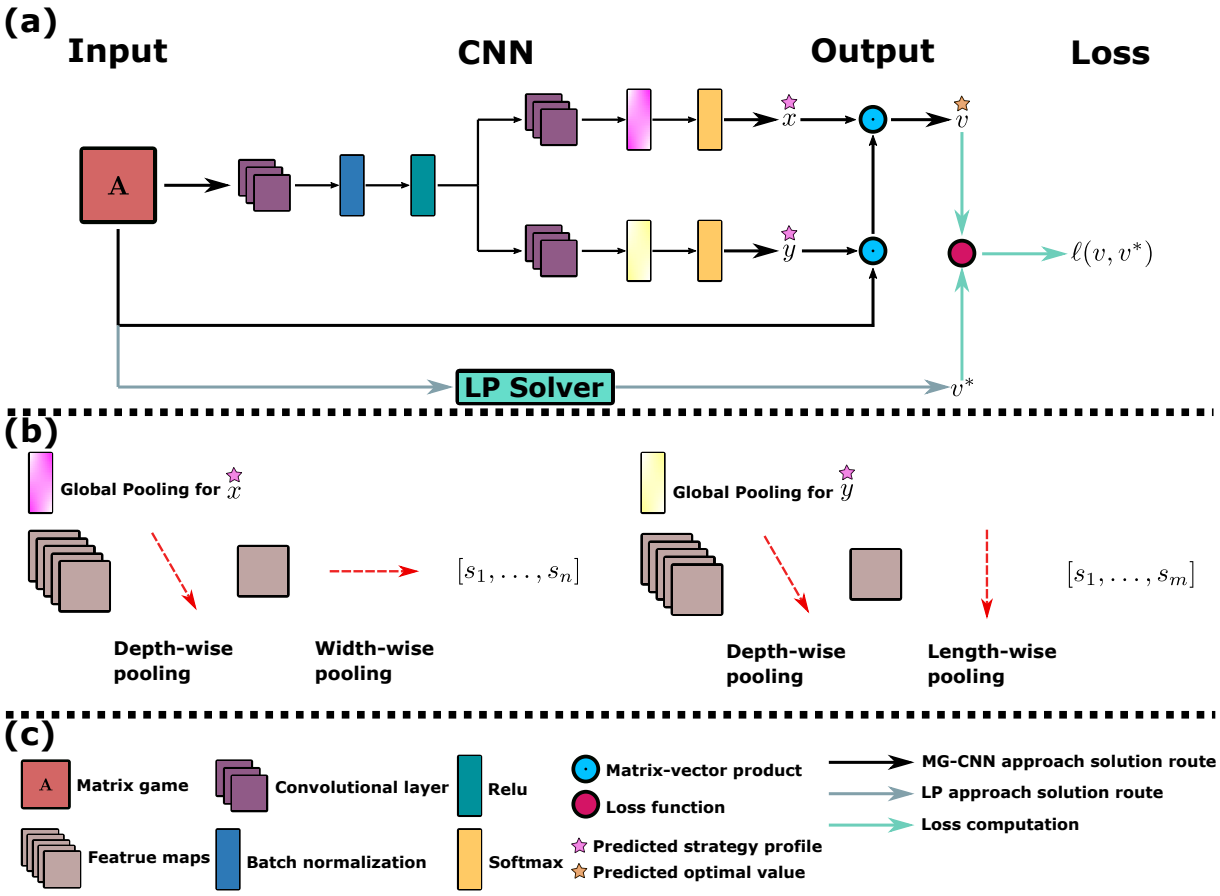

Figure 1: **MG-CNN framework (a) The MG-CNN model structure** The input of the model is a matrix game $\mathbf{A}$, and the output is a predicted saddle point (x, y). The predicted optimal value v is computed by $v = x^T \mathbf{A} y$. The computation of loss is based on the predicted optimal value v. **(b) The global pooling technique for x and y.** Consider a feature map of shape (c, n, m). The global pooling for x only pools the depth and the width dimension and generates a vector of n-dimensional. The global pooling for y only pools the depth and the length dimension and generates a vector of m-dimensional. **(c) Explanation of each icon**

where $\mathbf{A} \in \mathbb{R}^{n \times m}$ represents a matrix game to be solved, $\mathbf{w}$ refers to learnable parameters of the model, (x, y) refers to a strategy profile prediction, with $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. The corresponding optimal value prediction is computed by

$$v = x^T \mathbf{A} y. \tag{8}$$

Figure 1-(a) shows how MG-CNN solves a matrix game $\mathbf{A}$ where the black lines represent the computation of (x, y) and v.

MG-CNN model structure As shown in Figure 1-(a), the MG-CNN model has a single-input, double-output structure. The model is divided into one main part and two subparts. The main part processes the input matrix game into common hidden feature maps with multiple channels. The two subparts process the common feature maps independently and output x and y respectively. The main part is responsible for extracting the general knowledge of the problem. The subpart is responsible for generating the two strategies. The subparts end with the softmax activation function to ensure that x and y are discrete

probability distributions.

Fully convolutional layers The MG-CNN model uses only convolutional layers and does not contain dense connected layers. By appropriately setting the convolutional kernel size, padding, and stride, the convolutional layers do not change the shape of the input feature map. Other types of operations can be added, as long as they do not affect the shape of the feature map, e.g., pooling layers, batch normalization, or residual connections. MG-CNN does not change the shape of all feature maps between input and output.

Global pooling technique We use a novel pooling technique in order to generate output (x, y) of the desired shape. This pooling technique pools only two dimensions of 3D feature maps, as shown in Figure 1-(b). For example, consider the 3D feature maps of shape (c, n, m). For generating x of shape $(n,)$, it pools the channel and width dimensions and keeps the length dimension. For generating y of shape $(m,)$, it pools the channel and length dimensions and keeps the width dimension. The pooling can be average or maximum pooling.

Thanks to the fully convolutional layers and Global pooling technique, the MG-CNN model can accept matrix games $\mathbf{A}$ of varied sizes and output (x, y) with desired shapes. This framework is generic and is not specific to the matrix game problem. It can be used in numerous situations when the input shape is not fixed, and a corresponding output shape is needed.

3.2. MG-CNN training

The loss of the MG-CNN model is defined as

$$\mathcal{L}(\mathbf{A}; \mathbf{w}) = \ell(v, v^*). \tag{9}$$

The optimal value prediction v is computed by (8). $\ell(\cdot, \cdot)$ stands for an error function, e.g., mean square error. The loss function is based on the optimal value instead of the strategy profile for two reasons: 1) v and v^* are two scalars, while (x, y) and (x^*, y^*) are two vectors with possibly high dimensions. 2) A matrix game may have multiple optimal (x^*, y^*) values, whereas the optimal v^* is unique. Predicting one saddle point among many is inefficient. In contrast, it is more reasonable to predict the unique optimal point.

The training objective for the MG-CNN model is defined as

$$\min_{\mathbf{w}} \mathop{\mathbb{E}}_{\mathbf{A} \sim P} [\mathcal{L}(\mathbf{A}; \mathbf{w})], \tag{10}$$

where $\mathbf{A} \sim P$ represents that each element of the matrix $\mathbf{A}$ follows the probability distribution P.

Training data generation The generation of a training matrix game $\mathbf{A}$ is governed by two factors: game size (n, m) and probability distribution P, which must be specified prior to training. The game size (n, m) decides the shape of $\mathbf{A}$. The probability distribution P decides how $\mathbf{A}$ is sampled. Specifically, each element a_{ij} of $\mathbf{A}$ is sampled by the probability distribution P in an i.i.d way. These two factors govern the generation of training data and, consequently, the predictive ability of the MG-CNN model. The optimal

Algorithm 1: MG-CNN training

Required: Game size (n, m); Probability distribution P; A LP solver
Result : The MG-CNN model
Function Main():
  Initialize a MG-CNN model.
  while iter $\leq$ Max iteration **do**
    $\mathbf{A} \sim P$: sample a matrix game $\mathbf{A}$ with shape (n, m) from distribution P
    $(x, y) = \text{CNN}(\mathbf{A}; \mathbf{w})$: Predict the saddle point by the MG-CNN model
    $v = x^T \mathbf{A} y$: Compute the optimal value
    $v^* = LP(\mathbf{A})$: Obtain v^* by using the LP solver
    Forward propagation: Compute the loss $\mathcal{L}(\mathbf{A}; \mathbf{w}) = \ell(v, v^*)$.
    Backward propagation: Update $\mathbf{w}$ by $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{A}; \mathbf{w})$
  end
end

value v^* of $\mathbf{A}$ is provided by an LP solver. A training data sample has the form $(\mathbf{A}, v^*)$.

Training algorithm The training of MG-CNN is shown in Algorithm 1, which is essentially an optimization process for (10). During each iteration, a training batch is created and trained by the model. The training batch will be used only once and discarded immediately after use. The algorithm trains the model in an unsupervised manner. Instead of training on a fixed dataset, the algorithm incorporates the data generation and model training into the same loop, similar to the meshfree method in Sirignano & Spiliopoulos (2018). Since it is trained on brand-new data each time, the model does not overfit any dataset. Training is over when a certain metric is satisfied. For example, when the loss drops to a threshold or the number of iterations exceeds a given value.

4. Results

We use the Google Colab Pro+ platform to conduct our experiments. We build and train the neural network model by Pytorch 1.9.1. We use CVXPY to model the LP and call various solvers therein (Diamond & Boyd, 2016). We compare MG-CNN with the WL-CNN approach (Wu & Lisser, 2022b) and the LP approach. The LP approach is represented by four optimization solvers, ECOS, GLPK, SCS, and Gurobi (Domahidi et al., 2013; GLPK, 2012; Brendan et al., 2016; Gurobi Optimization, LLC, 2021).

In Section 4.1, We show how to train an MG-CNN model, including the hyperparameter setting, the model structure, and the training loss. In Section 4.2, we demonstrate the computational speed advantage of the MG-CNN model versus WL-CNN and LP approaches. In Section 4.3, we show the prediction accuracy of the MG-CNN model versus the WL-CNN approach.

4.1. Model training

The hyperparameters for training are as follows

- The optimizer for training the neural network is ADAM (Kingma & Ba, 2014), with a learning rate of 0.001.

Group	Layer Type	Layer description		Input size	Output size
		Description	Activation		
Main part	Conv2d	16 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(1, n, m)	(16, n, m)
	Conv2d	32 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(16, n, m)	(32, n, m)
	Conv2d	64 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(32, n, m)	(64, n, m)
	Conv2d	64 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(64, n, m)	(64, n, m)
Subpart for x	Conv2d	64 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(64, n, m)	(64, n, m)
	Conv2d	32 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(64, n, m)	(32, n, m)
	Conv2d	16 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(32, n, m)	(16, n, m)
	Conv2d	8 filters, 3*3 kernal size, 1 padding, 1 stride	None	(16, n, m)	(8, n, m)
	Globaly polling	Depth-wise and width-wise polling	Softmax	(8, n, m)	(n,)
Subpart for y	Conv2d	16 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(64, n, m)	(64, n, m)
	Conv2d	16 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(64, n, m)	(32, n, m)
	Conv2d	16 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	Leaky relu	(32, n, m)	(16, n, m)
	Conv2d	16 filters, 3*3 kernal size, 1 padding, 1 stride, with BN	None	(16, n, m)	(8, n, m)
	Globaly polling	Depth-wise and length-wise polling	Softmax	(8, n, m)	(m,)

Table 3: **The MG-CNN model details.** The MG-CNN model is dived into one main part and two subparts. BN refers to batch normalization. (n, m) refers to the game size of the input matrix game. The final feature maps of the main part is passed to the two subparts.

- The batch size is 128, and the maximum number of iterations is $30,000$.
- The neural network structure is shown by Table 3.

We initialize an MG-CNN model and train it according to Algorithm 1. We use the game size $(20, 20)$ and the uniform distribution $U(0, 100)$ to generate the matrix games. Both training and testing data are randomly generated in the same way. At each iteration, a training batch is generated to train the model. Every 50 iterations, a testing batch is generated to test the model. We use the same training scheme and a similar network structure to train the WL-CNN model, which is used hereafter for comparison purposes.

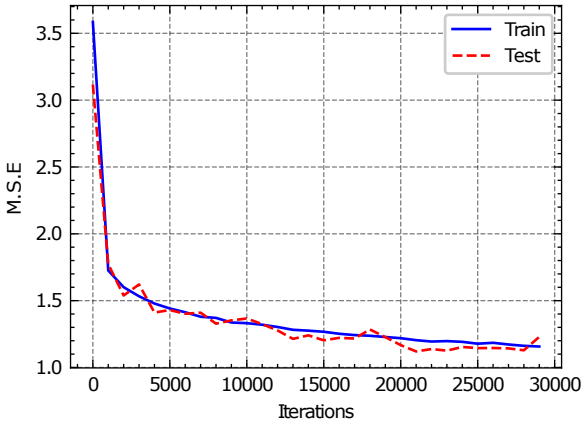

Figure 2: **The training and testing loss versus the number of iterations.** M.S.E refers to mean square error.

Figure 2 shows the loss computed on the training and testing batch during the training step. The MG-CNN model does not overfit any training data. The training loss and the testing loss have almost the same

rate of decline. This is due to the fact that the training data is not fixed but rather generated randomly and only used once. Our algorithm is not concentrated on any specific dataset but rather on minimizing the overall generalization error.

4.2. Computational performance

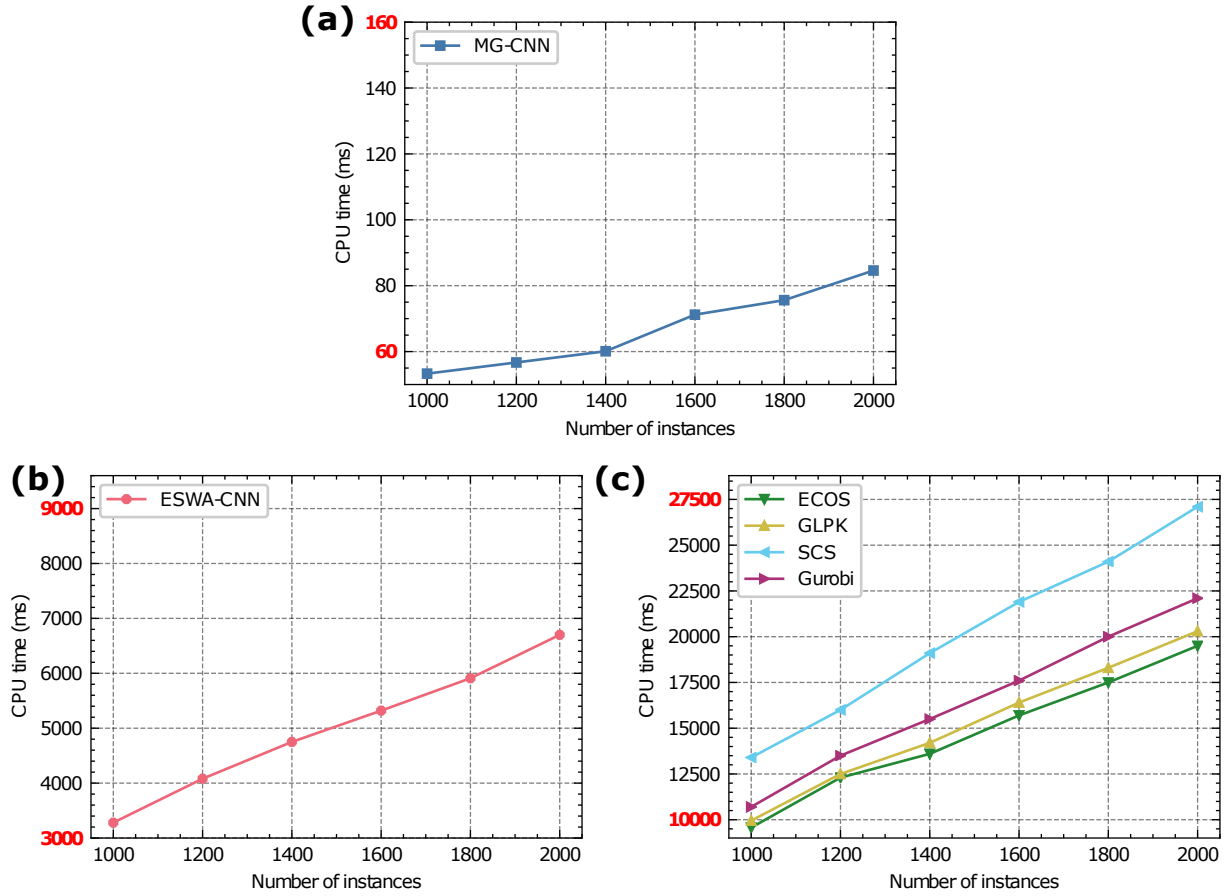

Figure 3: **The CPU times versus the number of instances.** The vertical axis indicates the total CPU time required to solve a certain number of instances. CPU time is shown in milliseconds (ms). **(a) The MG-CNN approach. (b) The WL-CNN approach. (c) The LP approach**, represented by the four optimization solvers, ECOS, GLPK, SCS, and Gurobi.

We now test the computational performance of MG-CNN and compare it with the WL-CNN and LP approaches. The MG-CNN computation time is the CPU time of the CNN model for both v and (x, y). The WL-CNN computation time includes the CPU time of the CNN model for v and the CPU time of the optimization solver for (x, y). The LP computation time is the CPU time of the optimization solver for both v^* and (x^*, y^*).

We randomly generate six test batches with 1000, 1200, 1400, 1600, 1800, and 2000 instances, respectively. Each matrix game within the test batch has size $(20, 20)$ and is sampled by the uniform distribution $U(0, 100)$. Figure 3 shows the CPU time of the three approaches to solve these test batches. When solving 1000 samples, MG-CNN takes 53.3ms, WL-CNN takes 3280ms, and the LP approach takes at least 9570ms. When solving

Instance number	Probability distribution	Game size	CPU time (ms)					
			MG-CNN	WL-CNN	LP			
					ECOS	GLPK	SCS	Gurobi
1	U(0, 100)	(10*10)	**5.1**	5.8	13.6	12.6	12.7	12.9
10			**8.6**	27.3	102	106	96.7	114
100			**9.2**	228	934	982	1040	1080
1000			**18.5**	2220	9170	9570	9700	10200
3000			**73.2**	4520	18300	19100	20300	20900
1	U(0, 100)	(15*15)	**5.5**	6.1	12.9	11.7	13.7	12.9
10			**8.5**	27.8	104	104	112	109
100			**7.5**	289	975	1020	1140	1090
1000			**24.3**	2780	9390	9820	11000	10400
3000			**99.6**	5530	19200	19600	23600	21600
1	U(0, 100)	(20*20)	**5.8**	6.91	13.1	12.3	14.2	16.2
10			**9.5**	42.2	105	102	136	119
100			**15.2**	351	1010	1040	1300	1150
1000			**53.3**	3280	9570	9950	13400	10700
3000			**156**	7170	20200	20700	29700	22200
1	U(0, 100)	(25*25)	**6.2**	7.3	13.2	14.0	15.5	15.0
10			**6.9**	45.1	104	110	145	117
100			**15.6**	377	1020	1050	1590	1130
1000			**73.7**	3910	9950	10500	16400	11100
3000			**230**	8570	21100	21800	38800	23400
1	U(0, 100)	(30*30)	**7.3**	10.2	14.3	13.2	15.7	14.2
10			**8.2**	46.5	109	117	220	121
100			**17.2**	472	1110	1140	2090	1180
1000			**76.3**	9000	10700	11000	21300	11500
3000			**237**	10300	22300	23100	53200	24200

Table 4: **The CPU time comparison of the three approaches.** CPU time is represented as milliseconds (ms). Instance number refers to how many instances a test batch contains. Probability distribution refers to how each instance is sampled. Game size refers to the size of each instance.

2000 samples, MG-CNN takes 84.6ms, WL-CNN takes 6700ms, and the LP approach takes at least 19500ms. In addition, the CPU time of MG-CNN does not grow as fast as WL-CNN and LP.

Table 4 presents the CPU time for different numbers of instances and different game sizes. In all cases, MG-CNN outperforms WL-CNN and is much faster than the four LP solvers. The advantage becomes more significant when dealing with a large number of instances.

4.3. Prediction accuracy

We evaluate the accuracy on the basis of the prediction v and the true v^*. We use the mean absolute error (MAE), the mean squared error (MSE), and the mean absolute percentage error (MAPE) as metrics, defined by

$$\text{MAE} = \frac{1}{B}\sum_{i=1}^{B}\left|v_i^* - v_i\right|, \tag{11}$$

$$\text{MSE} = \frac{1}{B}\sum_{i=1}^{B}(v_i^* - v_i)^2, \tag{12}$$

$$\text{MAPE} = \frac{1}{B}\sum_{i=1}^{B}\left|\frac{v_i^* - v_i}{v_i}\right| * 100\%, \tag{13}$$

where B refers to the number of instances in a test batch, v_i^* refers to the true optimal value of a matrix game, which is obtained by the LP approach. v_i refers to the prediction of MG-CNN or WL-CNN. We also provide "Mean_v^*" and "Mean_v" used to describe a test batch, defined as

$$\text{Mean\_}v^* = \frac{1}{B}\sum_{i=1}^{B}v_i^*, \tag{14}$$

$$\text{Mean\_}v = \frac{1}{B}\sum_{i=1}^{B}v_i. \tag{15}$$

Probability distribution	Game size	MG-CNN				WL-CNN				LP
		MAE	MSE	MAPE	Mean_v	MAE	MSE	MAPE	Mean_v	Mean_v^*
$U(0,100)$	$(20,20)$	0.87	1.18	1.73%	50.07	1.41	3.11	2.83%	49.90	50.07

Table 5: **Prediction accuracy on training probability distribution and game size.** The test batch is generated by $U(0,100)$ with size $(20,20)$ but is not trained.

Table 5 shows the prediction accuaracy on the training distribution and the game size, i.e., $U(0,100)$ and $(20,20)$, respectively. Table 6 and 7 study various probability distributions and game sizes that the models have not trained. Each row in the tables shows the results of a test batch containing 1000 matrix games. In all cases, MG-CNN has a lower error than WL-CNN one, and its predicted Mean_v is closer to the Mean_v^*.

Probability distribution	Game size	MG-CNN				WL-CNN				LP
		MAE	MSE	MAPE	Mean_v	MAE	MSE	MAPE	Mean_v	Mean_v^*
$U(0, 100)$	$(10, 10)$	1.39	3.23	**2.80%**	49.86	2.89	14.83	5.82%	49.75	49.90
	$(15, 15)$	0.97	1.51	**1.95%**	50.04	1.67	4.52	3.35%	49.84	50.00
	$(25, 25)$	0.74	0.86	**1.47%**	49.96	1.24	2.45	2.50%	49.88	50.00
	$(30, 30)$	0.62	0.62	**1.24%**	50.05	1.19	2.30	2.41%	49.89	50.07
	$(15, 25)$	2.41	7.16	**4.97%**	48.67	3.71	17.23	7.41%	49.94	46.26
	$(25, 15)$	2.50	7.51	**4.88%**	51.28	3.92	18.62	7.88%	49.88	53.77
$N(50, 50)$	$(10, 10)$	2.35	9.17	**4.86%**	49.47	5.13	43.63	10.36%	49.64	49.55
	$(15, 15)$	1.75	5.05	**3.56%**	49.76	3.21	17.02	6.45%	49.74	49.76
	$(25, 25)$	1.24	2.45	**2.49%**	50.05	1.90	5.62	3.81%	49.89	50.03
	$(30, 30)$	1.05	1.82	**2.10%**	49.98	1.63	4.16	3.28%	49.80	49.96
	$(15, 25)$	4.35	22.86	**9.16%**	47.82	6.42	50.52	12.95%	49.85	43.48
	$(25, 15)$	4.43	23.54	**8.54%**	52.08	6.64	53.52	13.25%	49.91	56.50
$Pois(50)$	$(10, 10)$	0.44	0.33	**0.88%**	49.89	0.93	1.38	1.87%	49.92	49.87
	$(15, 15)$	0.31	0.15	**0.62%**	49.99	1.06	1.81	2.15%	49.88	49.97
	$(25, 25)$	0.20	0.07	**0.41%**	49.97	1.30	2.74	2.65%	49.86	50.00
	$(30, 30)$	0.17	0.04	**0.33%**	49.97	1.42	3.28	2.89%	49.83	49.98
	$(15, 25)$	0.61	0.46	**1.22%**	49.66	1.48	2.96	2.95%	49.88	49.06
	$(25, 15)$	0.63	0.50	**1.26%**	50.26	1.27	2.77	2.60%	49.89	50.88

Table 6: **Prediction accuracy on untrained probability distributions and game sizes i.** $U(a, b)$ refers to the uniform distribution on the interval $[a, b]$. $N(\mu, \sigma^2)$ refers to the normal distribution with mean μ and variance σ^2. $Pois(\lambda)$ refers to the Poisson distribution with parameter λ.

Table 6-7 demonstrates the ability of our MG-CNN to predict untrained probability distributions and game sizes. However, MG-CNN is less accurate in solving non-square matrix games than square matrix games. This is because the neural network model was trained only for square games and not for non-square games.

One drawback of WL-CNN is that it only works when Mean_v and Mean_v^* are close to each other. For example, WL-CNN has been trained only on $U(0, 100)$ with Mean $\_v \approx 50$, such that it can only predict matrix games generated by a probability distribution with Mean_$v \approx 50$ like $U(20, 80)$, $N(50, 50)$ and $Pois(50)$. For the games generated by other distributions, WL-CNN needs to be retrained before prediction. Our MG-CNN does not have this drawback and is robust with a respect to different distributions. As shown in Table 7, for the distribution $U(50, 100)$ and size $(10, 10)$, MG-CNN has Mean_$v = 74.78$ with MAPE of 1.15%, while WL-CNN has Mean_$v = 49.89$ with MAPE of 50.09%.

5. Discussion

The MG-CNN approach solves a matrix game differently from WL-CNN and LP approaches. The LP approach formalizes a matrix game problem as an LP problem which is then solved by an optimization solver. WL-CNN solves a matrix game in two steps. First, it uses a CNN model to predict the optimal value v, then obtains the related strategy profile (x, y) by solving a system of linear inequalities. The MG-CNN model solves a matrix game by directly predicting (x, y) and v.

Probability distribution	Game size	MG-CNN				WL-CNN				LP
		MAE	MSE	MAPE	Mean_v	MAE	MSE	MAPE	Mean_v	Mean_v^*
U(0, 50)		0.68	0.78	**2.77%**	24.93	24.85	620.94	49.91%	49.82	24.98
U(50, 100)		0.86	1.26	**1.15%**	74.78	24.97	626.90	50.09%	49.89	74.86
U(20, 80)		0.89	1.28	**1.79%**	49.99	1.72	4.93	3.45%	49.85	50.10
N(25, 50)		2.37	9.26	**10.62%**	24.92	24.66	648.84	50.03%	49.63	24.97
N(50, 50)	(10, 10)	2.35	9.31	**4.79%**	49.79	4.98	41.82	10.04%	49.81	49.96
N(75, 50)		2.31	9.07	**3.11%**	74.98	25.47	695.00	51.00%	49.78	75.26
Pois(25)		0.29	0.14	**1.16%**	24.94	24.92	622.20	49.95%	49.86	24.95
Pois(50)		0.44	0.31	**0.88%**	49.96	0.98	1.57	1.97%	49.92	50.01
Pois(75)		0.60	0.58	**0.80%**	74.91	25.08	630.43	50.33%	49.92	75.00
U(0, 50)		0.40	0.25	**1.61%**	24.98	24.88	621.19	49.88%	49.86	24.98
U(50, 100)		0.45	0.34	**0.60%**	74.99	25.17	635.40	50.61%	49.87	75.04
U(20, 80)		0.52	0.43	**1.03%**	49.99	1.16	2.18	2.35%	49.86	49.99
N(25, 50)		1.45	3.36	**5.98%**	24.93	24.99	634.13	50.22%	49.89	24.90
N(50, 50)	(20, 20)	1.42	3.28	**2.86%**	49.89	2.26	8.41	4.53%	49.89	49.89
N(75, 50)		1.47	3.46	**1.97%**	75.00	25.21	644.26	50.69%	49.77	74.99
Pois(25)		0.16	0.04	**0.65%**	24.97	24.93	623.67	49.93%	49.90	24.96
Pois(50)		0.23	0.09	**0.46%**	49.96	0.99	1.63	2.01%	49.88	49.95
Pois(75)		0.30	0.15	**0.40%**	74.94	25.09	631.75	50.49%	49.86	74.95
U(0, 50)		0.31	0.15	**1.25%**	24.99	24.93	623.74	49.90%	49.92	24.99
U(50, 100)		0.32	0.17	**0.43%**	74.99	25.21	638.27	50.83%	49.82	75.03
U(20, 80)		0.38	0.23	**0.75%**	49.98	1.22	2.39	2.47%	49.84	49.98
N(25, 50)		1.08	1.88	**4.34%**	25.03	24.83	620.63	49.87%	49.86	25.03
N(50, 50)	(30, 30)	1.07	1.79	**2.14%**	49.94	1.61	4.07	3.24%	49.88	49.86
N(75, 50)		1.09	1.90	**1.45%**	75.13	25.25	641.92	50.78%	49.84	75.09
Pois(25)		0.12	0.02	**0.47%**	24.98	24.87	621.99	49.83%	49.84	24.97
Pois(50)		0.17	0.05	**0.34%**	49.97	1.40	3.32	2.86%	49.79	49.98
Pois(75)		0.21	0.07	**0.28%**	74.95	25.14	635.20	50.69%	49.82	74.96

Table 7: **Prediction accuracy on untrained probability distribution and game size ii** $U(a, b)$ refers to the uniform distribution on the interval $[a, b]$. $N(\mu, \sigma^2)$ refers to the normal distribution with mean μ and variance σ^2. $Pois(\lambda)$ refers to the Poisson distribution with parameter λ.

Pros MG-CNN is faster than both the WL-CNN and LP approaches, which is mainly due to the following reasons:

- MG-CNN can predict the saddle point directly without calling any optimization solver. The LP approach needs to call an optimization solver to obtain both (x^*, y^*) and v^*, and WL-CNN needs to call an optimization solver to obtain the strategy profile (x, y).
- When dealing with a batch of instances, the inherent parallel solving capabilities allow MG-CNN to solve multiple instances in one shot. In comparison, the LP approach solves sequentially the different instances, and WL-CNN solves sequentially the linear systems to obtain (x, y).

MG-CNN shows acceptable accuracy, which outperforms WL-CNN. MG-CNN also exhibits robustness against untrained game sizes and probability distributions.

Cons The disadvantage of MG-CNN is that it requires training to achieve the desired prediction accuracy. MG-CNN also shows weaknesses in predicting non-square matrix games, while the LP approach is more reliable in this case. Prediction accuracy depends heavily on technical details, such as neural network structure and training methods, which fall into the general machine learning scope. The numerous ongoing research on these topics can directly benefit MG-CNN.

Compared to WL-CNN, MG-CNN can be viewed as an enhanced version, which is more accurate, faster, and robust against various untrained probability distributions and game sizes. Compared to LP, MG-CNN and LP have their pros and cons and are suitable for different scenarios. The LP approach is more suitable when requiring reliability and high accuracy. MG-CNN is more suitable to solve multiple instances with a remarkably reduced computational effort.

6. Conclusion

We propose a deep learning approach to predict the saddle point of a matrix game, i.e., MG-CNN. We presented MG-CNN in detail, including the neural network structure, loss function, objective function, training data generation, and training algorithm. We conducted experiments and compared MG-CNN with other solution methods.

We must emphasize that our proposed approach should not be considered as substitute for the classical LP approaches, which have been developed over the last decades and are well established. The traditional LP solution method satisfies the requirements of robustness and reliability needed in practice. Our main contribution is to provide a novel approach for solving matrix games, which shows a great computational potential. With the rapid development of machine learning or deep learning in methodology and implementation, we believe this work will lead to ongoing contributions to benefit a wide range of practitioners in game theory.

There are many future directions for this work, and we give some examples here. 1) How to design a better model structure to adapt to the matrix game problem? 2) How to migrate this approach to games with

continuous actions ? 3) How to use advanced machine learning methods to improve the prediction accuracy and robustness of MG-CNN? We can gradually consolidate MG-CNN by answering the above questions.

Bibliography

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, *8*, 1–74.

Bertsimas, D., & Tsitsiklis, J. N. (1997). *Introduction to linear optimization* volume 6. Athena Scientific Belmont, MA.

Brendan, O., Eric, C., Neal, P., & Stephen, B. (2016). Operator splitting for conic optimization via homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, *169*, 1042–1068. URL: https://doi.org/10.1007/s10957-016-0892-3. doi:10.1007/s10957-016-0892-3.

Cheng, J., Leung, J., & Lisser, A. (2016). Random-payoff two-person zero-sum game with joint chance constraints. *European Journal of Operational Research*, *252*, 213–219. doi:10.1016/j.ejor.2015.12.024.

Courville, I. G., Bengio, Y., & Aaron (2016). Deep learning. *Nature*, *29*, 1–73. URL: http://www.deeplearningbook.org.

Dantzig, G. B. (1963). *Linear Programming and Extensions*. Santa Monica, CA: RAND Corporation. doi:10.7249/R366.

Dasgupta, P., & Collins, J. B. (2019). A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks. *AI Magazine*, *40*, 31–43. doi:10.1609/aimag.v40i2.2847. arXiv:1912.02258.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. URL: http://arxiv.org/abs/1810.04805. arXiv:1810.04805.

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, *17*, 1–5.

Dixit, A. K., & Pindyck, R. S. (2012). *Investment under uncertainty*. Princeton University Press. doi:10.2307/2329279.

Domahidi, A., Chu, E., & Boyd, S. (2013). Ecos: An socp solver for embedded systems. In *2013 European Control Conference (ECC)* (pp. 3071–3076). IEEE.

Fudenberg, D., & Tirole, J. (1991). *Game theory*. MIT press.

GLPK (2012). GNU Linear Programming Kit. URL: http://www.gnu.org/software/glpk/glpk.html.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*, 139–144. doi:10.1145/3422622. arXiv:1406.2661.

Gurobi Optimization, LLC (2021). Gurobi Optimizer Reference Manual. URL: https://www.gurobi.com.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 770–778). volume 2016-Decem. doi:10.1109/CVPR.2016.90. arXiv:1512.03385.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* (pp. 2261–2269). volume 2017-Janua. doi:10.1109/CVPR.2017.243. arXiv:1608.06993.

Jared L, Adair, Kristin L, Detry, Richard J, Durfee, Justin D, Jones, K., & A, Martin, N. (2012). Comparison of Open-Source Linear Programming Solvers, . doi:10.2172/1104761.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A. et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, *596*, 583–589.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing* (pp. 302–311).

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*, 5455–5516. URL: https://doi.org/10.1007%2Fs10462-020-09825-6. doi:10.1007/s10462-020-09825-6.

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

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*, 84–90. doi:10.1145/3065386.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*, 255–258. URL: http://www.iro.umontreal.ca/$\sim$lisa/pointeurs/handbook-convo.pdf.

Mittelmann, H. D. (2022). Benchmarks of optimization software. URL: http://plato.asu.edu/bench.html.

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.

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

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. (2019). 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.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–489. URL: https://doi.org/10.1038/nature16961. doi:10.1038/nature16961.

Simmons, G. (1998). Investment science. URL: https://econpapers.repec.org/RePEc:oxp:obooks:9780195108095. doi:10.1108/md.1998.36.6.419.1.

Singh, H. (1999). Introduction to Game Theory and Its Application in Electric Power Markets. *IEEE Computer Applications in Power*, *12*, 18–20. doi:10.1109/67.795133.

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. doi:10.1016/j.ejor.2019.01.010.

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

Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks, . URL: https://arxiv.org/abs/1905.11946. doi:10.48550/ARXIV.1905.11946.

Tembine, H. (2020). Deep Learning Meets Game Theory: Bregman-Based Algorithms for Interactive Deep Generative Adversarial Networks. *IEEE Transactions on Cybernetics*, *50*, 1132–1145. doi:10.1109/TCYB.2018.2886238.

Vanderbei, R. J. (2014). Integer programming. In *Linear Programming* (pp. 345–362). Springer.

Vega-Redondo, F. (2003). *Economics and the theory of games*. CRC Press. doi:10.1017/CBO9780511753954.

Wright, S. J. (1997). *Primal-dual interior-point methods*. SIAM.

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

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

Xu, Y., & Zhang, H. (2022). Convergence of deep convolutional neural networks. *Neural Networks*, .

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, *52*, 1–38.

Zhou, Y., Kantarcioglu, M., & Xi, B. (2019). A survey of game theoretic approach for adversarial machine learning. doi:10.1002/widm.1259.