Using CNN for solving two-player zero-sum games

Dawen Wu, Abdel Lisser

Expert Systems With Applications 204 (2022) 117545

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Using CNN for solving two-player zero-sum games

Dawen Wu *, Abdel Lisser

Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 3 rue Joliot Curie, 91190 Gif-sur-Yvette, France

ARTICLE INFO

Keywords:
Two-player zero-sum game
Saddle point
Convolutional neural network
Machine learning

ABSTRACT

We study a two-player zero-sum game (matrix game for short) with the objective of finding the saddle point and its value. We develop a novel convolutional neural network (CNN for short) approach to achieve the goal. We propose a complete training pipeline, including a specific CNN model structure to handle varying game sizes, generating training datasets, and model fitting. The experiment results show that our proposed method outperforms the traditional linear programming (LP for short) method and two regret minimization learning algorithms in terms of computational efforts.

1. Introduction

A two-player zero-sum game or matrix game is a game where there are only two players, and one player wins whatever the other player loses. It can be reduced to a matrix form $\mathbf{A} = (a_{i,j})_{n\times m}$, where the number of rows and columns represent the size of the action set of the row player and column player, respectively. The row player and the column player choose a pure strategy (i, j) will get a return $(a_{ij}, -a_{ij})$, respectively. $\mathbf{A}$ is the payoff function of the row player, and $-\mathbf{A}$ is the payoff function of the column player.

The saddle point in a two-player zero-sum game describes a situation when two players optimize their payoff functions simultaneously. The definitions of the saddle point and its value are

$$(\mathbf{x}^*, \mathbf{y}^*) = \arg\max_{\mathbf{x}} \left(\arg\min_{\mathbf{y}} \mathbf{x}^T \mathbf{A}\mathbf{y} \right), \tag{1}$$

$$v^* = \max_{\mathbf{x}} \left(\min_{\mathbf{y}} \mathbf{x}^T \mathbf{A}\mathbf{y} \right). \tag{2}$$

The saddle point equilibrium in (1) can be solved by linear programs (3) and (4). The minimax theorem states that the optimal objective values v in those two linear programs are equal (von Neumann, 1928).

$$\begin{aligned} &\text{(P1)} \quad \max\ v \\ &\quad \text{s.t. } \mathbf{A}^T\mathbf{x} \geq v\mathbf{e_m} \\ &\qquad \mathbf{e_n}^T\mathbf{x} = 1, \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{3}$$

$$\begin{aligned} &\text{(P2)} \quad \min\ v \\ &\quad \text{s.t. } \mathbf{A}\mathbf{y} \leq v\mathbf{e_n} \\ &\qquad \mathbf{e_m}^T\mathbf{y} = 1, \mathbf{y} \geq \mathbf{0}, \end{aligned} \tag{4}$$

where $\mathbf{e_k}$ is a k-dimensional vector with all elements equal to 1.

A neural network is a statistical model that can acquire predictive ability after learning from data. CNN represents a class of deep neural networks widely used in the computer vision area. A CNN model is a function (5) with the following components: model parameter θ, input $\mathbf{M}$ and predicted value $\hat{\mathbf{p}}$. The input $\mathbf{M}$ of a CNN model is usually a three-dimensional array with size (c, h, w), where c, h, and w represent the number of channels, the height, and the width, respectively. The output of a CNN model $\hat{\mathbf{p}}$ is a real vector or a value representing the model's prediction. The training for the CNN model aims at minimizing the expected risk (6) w.r.t parameters θ. However, due to its inaccessibility, we usually minimize the empirical risk (7). Our paper's objective is to use the current popular and powerful model CNN to solve two-player zero-sum games, i.e., the problems (1) and (2).

$$f_\theta(\mathbf{M}) = \hat{\mathbf{p}} \tag{5}$$

$$L_D(\theta) = \mathbb{E}_D \ell\left(f_\theta(\mathbf{M}), \mathbf{p}\right) \tag{6}$$

$$\hat{L}_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(f_\theta(\mathbf{M_i}), \mathbf{p_i}\right) \tag{7}$$

Our contributions are three-fold.

- We use a CNN approach to find the saddle point of the two-player zero-sum game. The CNN model gives the prediction of the optimal value, and the associated strategy profile is obtained by solving two linear systems.
- We design training algorithms to train our CNN models and discuss training data generation.
- We conduct numerical experiments to compare our CNN approach with the traditional LP method and learning algorithms FP and EXP3.

* Corresponding author.
E-mail addresses: dawen.wu@centralesupelec.fr (D. Wu), abdel.lisser@l2s.centralesupelec.fr (A. Lisser).

https://doi.org/10.1016/j.eswa.2022.117545
Received 20 April 2021; Received in revised form 23 March 2022; Accepted 7 May 2022
Available online 16 May 2022

The rest of the paper is organized as follows. In Section 2, we give a literature review for two-player zero-sum games, CNN, the recent progressing connection between these two areas, and two regret minimization learning algorithms. In Section 3, we present our CNN method for solving two-player zero-sum games. In Section 4, we provide the numerical results in terms of two aspects, computation speed and accuracy, and compare our CNN method with other approaches such as LP, FP, and EXP3. In Section 5, we sum up the paper and give directions for future works.

The notation used in the paper can be summarized as follows.

- $\mathbf{A}$ denotes the payoff matrix of a two-player zero-sum game.
- n and m denote the sizes of an action set of row player and column player, respectively.
- $(\mathbf{x}, \mathbf{y})$ denotes a strategy profile, where x and y denote a row player and column player strategies, respectively, either pure or mixed.
- $(\mathbf{x}^*, \mathbf{y}^*)$ denotes the optimal strategy profile of a saddle point.
- $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ denotes a predicted strategy profile obtained by solving two linear systems.
- v^* denotes the optimal value of a saddle point.
- $\hat{v}$ denotes a predicted value from a CNN model.
- $f_\theta(\cdot)$ denotes a CNN model, where θ is the model parameter.
- ℓ denotes a loss function used in training.
- $\mathbb{P}$ denotes a matrix game generating distribution.

2. Literature review

As a mathematical model of conflict and cooperation, game theory studies the situations where a set of self-motivated players act to maximize its own profit. Since the pioneering results of von Neumann (1928), game theory has been widely developed both from theoretical and practical points of view (Charilas & Panagopoulos, 2010; Dixit, Skeath, & McAdams, 2020; Gibbons, 2019; Hauert & Szabó, 2005; Myerson, 1997; von Neumann & Morgenstern, 2007). In a two-player zero-sum context, saddle point states a situation where the outcome is maximum for one player and is minimum for the other. Later, in a finite multiple players general-sum context, Nash (1950) proved that there is at least one mixed strategy profile where no player can improve his/her payoff by changing his strategy unilaterally, namely Nash equilibrium.

Two-player Zero-sum games or matrix games, as a basic type of game, plays a central role in game theory development. A fundamental useful mathematical theorem for zero-sum games is the Minimax Theorem (Fan, 1953; von Neumann, 1928), which guarantees that the interchange of the orders max-min and min-max in (2) would not affect the result. Two-player Zero-sum games model many real-world situations in order to help decision-makers to make the good decisions in a competitive environment, including business (Dixit & Pindyck, 2012; Simmons, 1998), economics (Vega-Redondo, 2003), and engineering (Singh, 1999). Dantzig (2018) shows that solving any matrix game is equivalent to a linear program. Most commercial or academic software such as Gurobi, CPLEX, Matlab,Scipy (Cplex, 2009; Gearhart et al., 2012; Gurobi Optimization, LLC, 2021; Matlab, 2017; Virtanen et al., 2020) provide tools for linear programming based on interior point methods and simplex method (Nocedal & Wright, 2006; Vanderbei, 2014). Besides, there are also some studies on the situation where the game contains randomness (Cheng, Leung, & Lisser, 2016; Singh & Lisser, 2019).

CNN (LeCun & Bengio, 1995) is a kind of Deep neural networks (Courville, Bengio, & Aaron, 2016). As a type of Feedforward neural network, it uses the backpropagation algorithm for the training step (Rumelhart, Hinton, & Williams, 1986). The main characteristic of CNN is its use of shared parameter filters to scan the previous feature maps, which can significantly reduce the size of the parameter space. Since the CNN model Alexnet (Krizhevsky, Sutskever, & Hinton, 2017) won the ImageNet challenge in 2012 (Deng et al., 2010), there is a tremendous amount of research on this topic (Li, Liu, Yang, Peng, & Zhou, 2021; Wiatowski & Bolcskei, 2018; Zhou, 2020), and more sophisticated CNN structures have been proposed (He, Zhang, Ren, & Sun, 2016; Howard et al., 2017; Huang, Liu, Van Der Maaten, & Weinberger, 2017). CNN has applications in many fields, e.g., image classification (Rawat & Wang, 2017), medical image analysis (Tajbakhsh et al., 2016), video recognition (Karpathy et al., 2014), natural language processing (Duque, Santos, Macêdo, & Zanchettin, 2019). It is worth noting that the problem we are dealing with in this paper has two characteristics different from most situations: regression rather than classification and varying input size. For regression, it can be viewed as a prediction of the rotation angle of the image (Fischer, Dosovitskiy, & Brox, 2015; Mahendran, Ali, & Vidal, 2017). For varying input sizes, there are three approaches for solving game problems: global pooling, variable-sized pooling, and padding input images (Yamashita, Nishio, Do, & Togashi, 2018). PyTorch is an open-source machine learning library used for building neural network models, and it also provides GPU-CUDA support to accelerate the training step (Paszke et al., 2019)

More recently, two-player zero-sum games built many connections with deep learning, such as Generative adversarial networks(GAN) (Dasgupta & Collins, 2019; Goodfellow et al., 2020; Tembine, 2020; Zhou, Kantarcioglu, & Xi, 2019). In GAN, there are two neural network models, generator and discriminator, which can be viewed as two players in a zero-sum game, and the objectives of the two players are opposite. In Adversarial learning, another topic related to game theory besides GAN, the two players are the model parameter and input data, and the objective of this training is to push the neural network model to become more robust (Chivukula & Liu, 2017; Zhu, Li, Wang, Gong, & Yang, 2020). Moreover, some research work uses the zero-sum game theory framework to promote or understand machine learning algorithms (Farnia & Ozdaglar, 2020; Ganapathiraman, Zhang, Yu, & Wen, 2016). On the contrary, some research work uses machine learning methods to solve zero-sum games with incomplete observations (Ling, Fang, & Zico Kolter, 2018, 2019).

A large number of learning algorithms belong to the regret minimization methods family amongst all Fictitious play (FP for short) and Exp3. These algorithms are generally used to solve stochastic games and are often used in multi-armed bandits topic (Auer, Cesa-Bianchi, Freund, & Schapire, 2003; Berger, 2007; O'Donoghue, Lattimore, & Osband, 2020).

3. Methodology

In this section, we give a detailed presentation of our CNN approach. Section 3.1 introduces our CNN model, and how it predicts $\hat{v}$. Section 3.2 introduces the concrete training algorithms. Section 3.3 describes how to obtain the associated strategy profile when the predicted saddle point value $\hat{v}$ is known. Section 3.4 states the advantages and disadvantages of the CNN method.

3.1. The CNN model

The CNN model maps a matrix game $\mathbf{A}$ to a predicted saddle point value $\hat{v}$. As shown in Fig. 1, the input matrix game on the left-hand side goes through the CNN model to get the predicted value.

Given different sizes of matrix games, the convolutional layer will lead to different sizes of the feature maps. This might lead to an issue as the afterward fully-connected layer requires a fixed input size. To overcome this issue, we insert either a maximum or an average global pooling layer at the end of convolutional layers. A global pooling layer down-samples an entire 2-d feature map to a single value. For example, consider two different input matrix games with sizes $10 * 10$ and $50 * 50$, respectively. After going through a padding convolutional layer with 6 filters and kernel size $3 * 3$, the feature map sizes are $6 * 10 * 10$ and $6 * 50 * 50$, respectively. After crossing a maximum pooling layer that follows the previous convolutional layer and has a kernel size $2 * 2$ and stride 2, the feature map sizes are $6 * 5 * 5$ and $6 * 25 * 25$, respectively. A global pooling layer can compress these two different sizes of feature maps to vectors with the same size of 6.

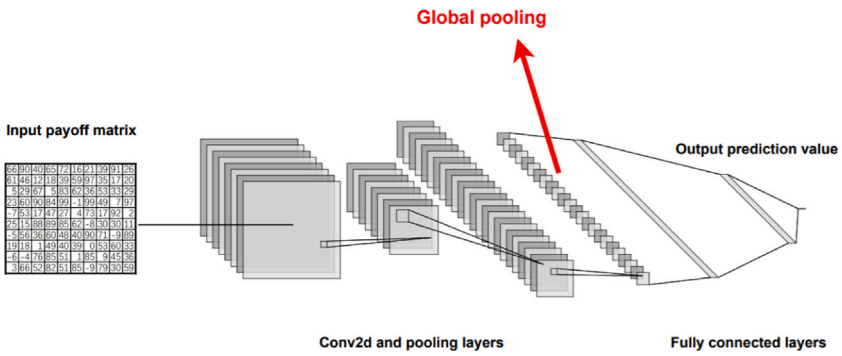


Fig. 1. A CNN model.

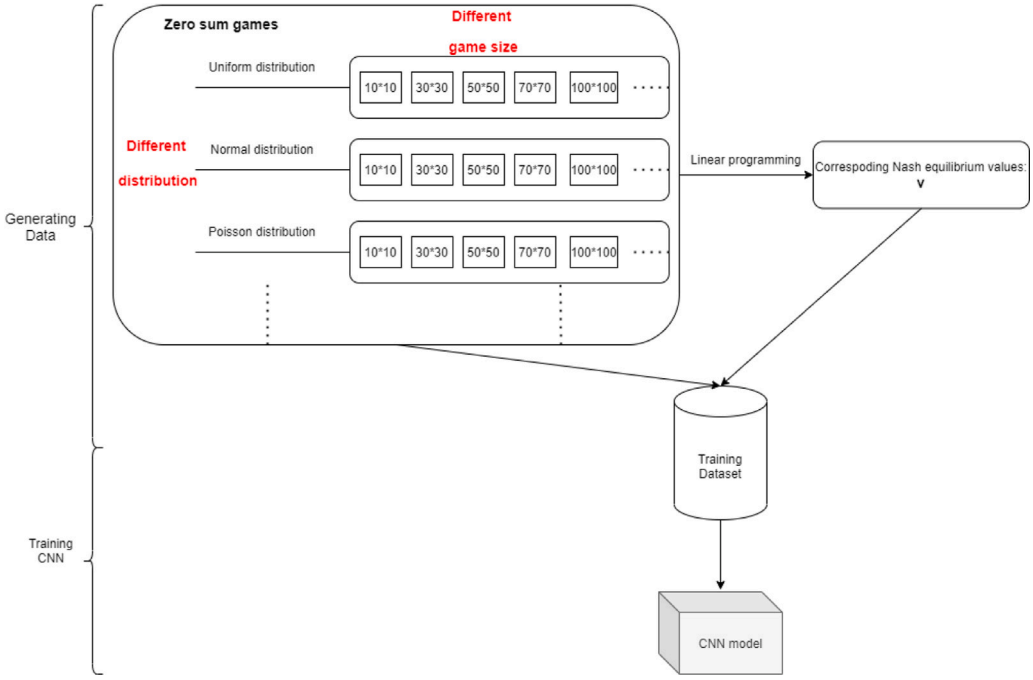


Fig. 2. The training procedure for the CNN model.

Remark 3.1. Our CNN method addresses the problem in a different way than the traditional LP method. The LP method first solves a pair of primal-dual linear programs to obtain the optimal strategy profile $(\mathbf{x}^*, \mathbf{y}^*)$ together with v^*. Our CNN model directly predicts $\hat{v}$ without any knowledge of the strategy profile. This is because the output of the CNN model is only the prediction $hat v$ to the optimal value v^*. And the whole CNN predicting process does not involve any knowledge of the strategy profile.

3.2. The training algorithm

Fig. 2 shows the training procedure for the CNN model. The game sizes pool and the distributions pool represent several game sizes and distributions as highlighted in red in Fig. 2. A training data sample has the form $(\mathbf{A}, v^*)$, where the matrix $\mathbf{A}$ is sampled from a given probability distribution, and the corresponding true value v is obtained by solving a linear program.

Algorithms 1 and 2 are the concrete training methods for the CNN model. Algorithm 1 shows the procedures to generate one batch of data and trains the CNN model for one iteration. Algorithm 2 presents the main procedure to train the CNN model. We provide two training options in Algorithm 2, namely the separated training and the joint training. The separated training is the same as most machine learning training procedures where the model weight training occurs after the complete dataset is created. The joint training generates data and trains the model parameters at the same time. At each iteration, joint training first generates a batch of data in order to train the model weight and then discards this data.

From the perspective of minimizing the objective function, the separated training minimizes the following empirical risk at each iteration,

$$E_N(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\left(f_\theta(\mathbf{A_i}), v_i^*\right), \tag{8}$$

where N is the number of data samples. The joint training minimizes the following empirical risk at iteration i,

$$E_{(i)}(\theta) = \ell\left(f_\theta(\mathbf{A_i}), v_i^*\right). \tag{9}$$

At each iteration, the separated training considers the same empirical risk, while the joint training considers different empirical risks.

Different hyperparameter settings can affect the performance in different ways. The learning rate is set between 10^{-3} and 10^{-6} in our case. In practice, we generate and use data in batches instead of just one sample. Additionally, we introduce a dedicated hyperparameter for the joint training to reuse data, namely the training round, which indicates how many times a sample will be used repeatedly.

Algorithm 1: Generate one matrix game and train

Input: Game size (m, n); Probability distribution $\mathbb{P}$; CNN model net

Function Generate(m, n, $\mathbb{P}$):
  $\mathbf{A} \sim \mathbb{P}$: sample a matrix game $\mathbf{A}$ with shape (m, n) from distribution $\mathbb{P}$
  $v^* = LP(\mathbf{A})$: Find v^* by solving the LP
  $\mathbf{b} = (\mathbf{A}, v^*)$
  return $\mathbf{b}$
end
Function Train(**b**, net):
  net $\leftarrow$ **b**: Train the CNN model by the sample **b**.
end

Algorithm 2: Main procedure: Training for the CNN model

Hyperparameters: CNN structure Net; Learning rate α; Training rounds K; Sample size N; Iterations number T
Input : Game sizes pool; Probability distributions pool
Output : The CNN model
Initialize : net = Net(), $\mathbf{B}$ = [ ]

Function Separated(net):
  for n in N **do**
    randomly select a game size (m, n) from the game sizes pool
    randomly select a distribution $\mathbb{P}$ from the probability distributions pool
    $\mathbf{b}$ = Generate$(m, n, \mathbb{P})$
    B.append(**b**)
  end
  for t in T **do**
    for **b** in **B** **do**
      Train(**b**, net)
    end
  end
  return net
end
Function Joint(net):
  for t in T **do**
    randomly select a game size (m, n) from the game sizes pool
    randomly select a distribution $\mathbb{P}$ from the probability distributions pool
    $\mathbf{b}$ = Generate$(m, n, \mathbb{P})$
    for k in K **do**
      Train(**b**, *net*)
    end
  end
  return net
end

3.3. Saddle point strategy

Although the CNN model can predict the saddle point's value $\hat{v}$ after training, it cannot obtain the mixed strategy profile $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ directly. In order to obtain the strategy profile $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ associated with the predicted values $\hat{v}$, we need to solve the following linear system,

$$\hat{\mathbf{x}}^T \mathbf{A} \hat{\mathbf{y}} = \hat{v}, \tag{10}$$

where $\mathbf{A}$ and $\hat{v}$ are known.

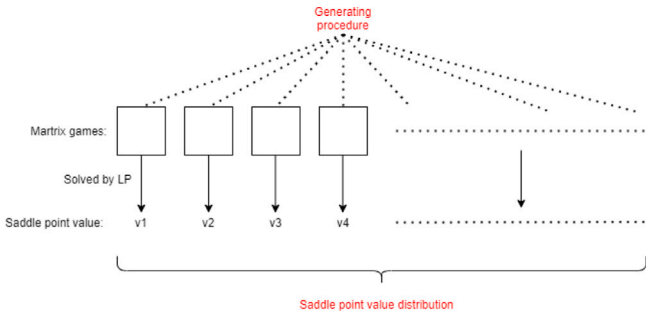


Fig. 3. Saddle point value distribution.

Getting only one feasible strategy profile is sufficient, though several feasible solutions might exist. For example, let

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \hat{v} = 0.7. \tag{11}$$

There are infinitely many feasible strategy profiles, such as $\hat{\mathbf{x}} = [1, 0, 0]^T, \hat{\mathbf{y}} = [0.7, 0.1, 0.2]^T$ and $\hat{\mathbf{x}} = [1, 0, 0]^T, \hat{\mathbf{y}} = [0.7, 0.2, 0.1]^T$.

The linear system (10) can be reduced to (12) or (13). If the predicted value is less than the true value, i.e., $\hat{v} < v^*$, the strategy $\hat{\mathbf{x}}$ can be obtained by (12), and the strategy $\hat{\mathbf{y}}$ is the best response of $\hat{\mathbf{x}}$. Similarly, If $\hat{v} > v^*$, the strategy $\hat{\mathbf{y}}$ can be obtained by (13), and the strategy $\hat{\mathbf{x}}$ is the best response of $\hat{\mathbf{y}}$. If $\hat{v} = v^*$, this indicates that our CNN model successfully predicts the optimal value, (12) and (13) will both have solutions.

$$\begin{aligned} &\mathbf{A}^T \hat{\mathbf{x}} \geq \hat{v} \mathbf{e_m} \\ &\hat{\mathbf{x}}^T \mathbf{e_n} = 1, \hat{\mathbf{x}} \geq 0, \end{aligned} \tag{12}$$

$$\begin{aligned} &\mathbf{A}\hat{\mathbf{y}} \leq \hat{v} \mathbf{e_n} \\ &\hat{\mathbf{y}}^T \mathbf{e_m} = 1, \hat{\mathbf{y}} \geq 0. \end{aligned} \tag{13}$$

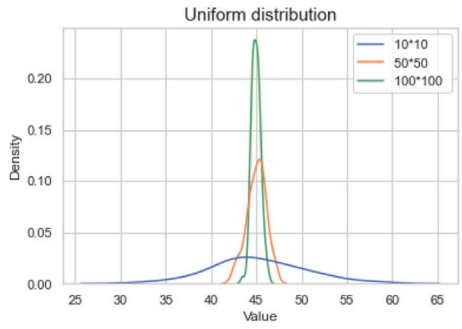
Remark 3.2. When $\hat{v}$ is between the minimum and maximum values of $\mathbf{A}$, the strategy profile $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ corresponding to $\hat{v}$ can always be found by the equations system (10). However, when the predicted value $\hat{v}$ is greater than the maximum value of $\mathbf{A}$, or $\hat{v}$ is less than the minimum value of $\mathbf{A}$, (10) will not have a solution. For such cases, the predicted value $\hat{v}$ can be taken as the maximum value of $\mathbf{A}$ (if $\hat{v}$ is greater than the maximum value of $\mathbf{A}$), or the minimum value of $\mathbf{A}$, (if $\hat{v}$ is less than the minimum value of $\mathbf{A}$).

3.4. Pros and cons of our CNN method

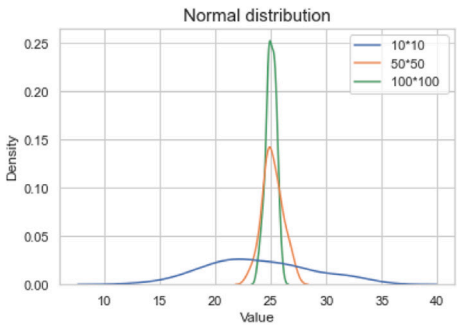
We give the advantages and disadvantages of the CNN method in comparison with LP. The most significant advantage of the CNN approach is the computational performance. Our CNN approach can solve these problems directly without using any optimization solver. However, the disadvantage is that the solution is approximate, and the model requires training time before use.

Theoretically, a predicting model should only be able to handle a given pool of game sizes and generating distributions known in advance. However, based on the experimental results, we are surprised to find that our CNN model can also handle game sizes and generative distributions outside the pool, and it is worth exploring the reason behind in the future.

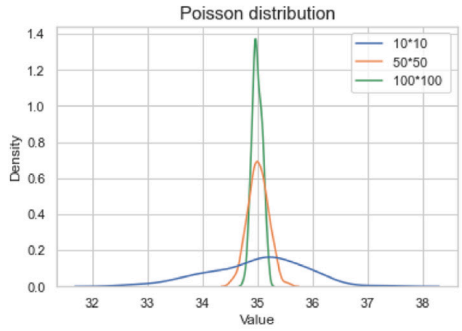
Additionally, in order to compare the performances of our CNN model with existing learning algorithms from the literature, we test two algorithms, namely FP and Exp3. FP is a strategic game learning algorithm that proceeds in an iterative manner. In each round, the players play the best response to mixed strategy obtained by previous rounds' empirical frequencies of actions. FP was originally introduced



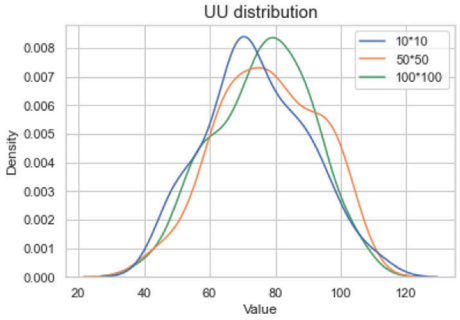
(a) From Uniform distribution $U(-10, 100)$



(b) From Normal distribution $\mathcal{N}(25, 3)$



(c) From Poisson distribution $P(35)$



(d) From UU generating procedure

Fig. 4. Saddle point value distribution.

by Brown (Berger, 2007). Exp3 is a popular adversarial multiarmed bandits algorithm suggested and studied in this setting by Auer et al. (2003). Exp3 stands for Exponential-weight algorithm for Exploration and Exploitation. It is based on a list of weights for each of the actions in order to choose randomly the action to be taken next. Exp3 increases the relevant weights in case of good payoff and decreases them otherwise.

Although FP and EXP3 are shown to have convergence properties, they require a large number of iterations, especially when the problem size is large. In contrast, CNN models do not require iterations to achieve predictions, but the convergence property is not guaranteed.

4. Numerical experiments

In this section, we provide numerical results for solving zero-sum games in order to investigate the performances of our algorithms. Our CNN algorithms are implemented under the Google cloud platform for training and testing tasks. We use eight virtual N2D CPUs, 64 GB of memory, and one P100 Nvidia Tesla GPU computer. We use Python 3.8 language for our codes, Gurobi for solving linear programs, Pytorch 1.7.1 as the neural network library to build up our neural network model, and CUDA 10.2 as the GPU computation platform.

4.1. Games distribution

Definition 4.1 (*Saddle Point Value Distribution*)**.** Given a generating procedure, generate a number n of instances from it, and solve them to get n saddle point values.[1] A saddle point value distribution of the generating procedure is the distribution of these n values.

[1] The saddle point value v^* instead of saddle point strategy x^* and y^*.

Table 1
Moments of saddle point value distributions.

Generating procedure	Mean	Variance	Skewness	Kurtosis
U-10*10	45.32	22.32	−0.01	3.28
U-50*50	44.81	0.83	0.15	2.25
U-100*100	44.96	0.25	−0.23	2.67
N-10*10	23.96	17.19	−0.32	3.09
N-50*50	25.16	0.88	−0.35	3.36
N-100*100	25.03	0.25	−0.21	3.24
P-10*10	34.91	0.85	−0.11	4.33
P-50*50	34.99	0.03	0.17	2.66
P-100*100	34.99	0.01	−0.18	2.39
UU-10*10	74.23	239.38	0.18	2.50
UU-50*50	77.83	236.19	−0.14	2.31
UU-100*100	76.15	215.93	−0.20	2.46

A matrix game generating procedure in Definition 4.1 usually contains one or more probability distribution. It decides how each matrix components are sampled. The generated matrix game will be solved by LP to get the saddle point value. Fig. 3 shows the connection from the generating procedure to the saddle point values distributions. The generating procedures studied in this subsection will be used in the following subsection either for training or testing purposes.

We consider three game sizes, namely $10 * 10$, $50 * 50$ and $100 * 100$. We generate 100 instances for each game size and for each one of the following three distributions as generating procedures: Uniform distribution with interval $[-10, 100]$, Normal distribution with mean 25 and standard deviation 3, Poisson distribution with $\lambda = 35$, i.e., $U(-10, 100)$, $\mathcal{N}(25, 3)$, $P(35)$. Besides, we study a more complex generating procedure with two uniform distributions denoted as UU. The UU generating procedure starts with sampling two points l_1, l_2 from $U(0, 75)$ and $U(75, 150)$ respectively, and a matrix game is generated by $U(l_1, l_2)$. Fig. 4 shows the obtained saddle point value distributions. Table 1 gives the four first moments of the saddle point value distributions. The first column shows the saddle point value

Table 2
The structure of the CNN model.

Layer	Type	Detail	Output size
1	Conv2d	Kernal size: 3*3, Filters number: 16, Padding: 1, Stride: 1 Activation: leaky relu	(16, 100, 100)
2	Conv2d	Kernal size: 3*3, Filters number: 32, Padding: 1, Stride: 1 Activation: leaky relu Pooling: 2*2 max pooling	(32, 50, 50)
3	Conv2d	Kernal size: 3*3, Filters number: 64, Padding: 1, Stride: 1 Activation: leaky relu	(64, 50, 50)
4	Conv2d	Kernal size: 3*3, Filters number: 64, Padding: 1, Stride: 1 Activation: leaky relu Pooling: 2*2 max pooling Global mean pooling	(64,)
5	Fully connected	Neurons number: 32 Activation: leaky relu	(32,)
6	Fully connected	Neurons number: 16 Activation: leaky relu	(16,)
7	Fully connected	Neurons number: 10 Activation: leaky relu	(10,)
8	Fully connected	Neurons number: 10 Activation: leaky relu	(10,)
9	Fully connected	Neurons number: 1 Activation: leaky relu	(1,)

Table 3
Difference between the separated and the joint training.

Training options	Iterations					
	0	1000	2000	3000	4000	5000
Separated training	1293.70	19.64	13.59	11.12	7.50	4.44
Joint training	1293.70	24.02	15.53	6.55	1.47	0.55

Table 4
CNN for trained game sizes and distributions.

Game sizes	Uniform		Normal		Poisson	
	Mean value	Gap	Mean value	Gap	Mean value	Gap
10*10	45.77	5.88%	25.38	1.70%	35.59	2.30%
50*50	46.50	3.31%	25.75	2.74%	36.14	2.90%
100*100	45.37	0.90%	25.13	0.54%	35.20	0.63%

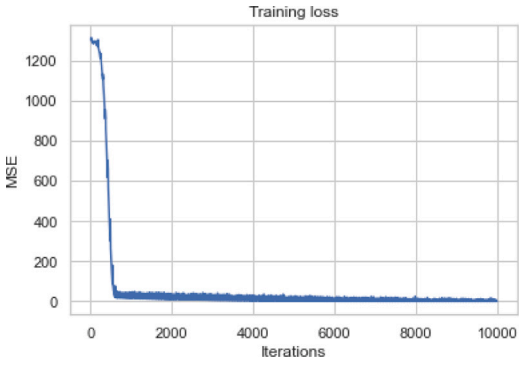


Fig. 5. Training loss of the CNN model.

distribution obtained from each generating procedure, e.g., $U-10*10$ represents the saddle point value distribution generated by the uniform distribution for the game size $10*10$.

Fig. 4(a)'s mean value is 45, which is the median number of the Uniform distribution. Fig. 4(b)'s mean value is 25, which is the μ of the Normal distribution. Fig. 4(c)'s mean value is 35, which is the parameter λ of the Poisson distribution. From Table 1, we can see that the variances of the distributions from these three generating procedures are getting smaller when the game size increases, and the distributions are sharper in Figs. 4(a), 4(b), and 4(c). The reducing variance will make the learning method trivial because simply setting the predicted value to the average value can get satisfying accuracy. The UU generating procedure would not occur in such a situation, and the variance remains high in larger game sizes. The generated distribution is generally symmetric and tailedness as shown by the Skewness and Kurtosis values which are generally close to zero and three.

4.2. Accuracy of CNN

As for the CNN training, we use the joint one described in Algorithm 2. We use the following setting:

- The considered game sizes are: $10*10$, $30*30$, $50*50$, $70*70$, $100*100$, and $200*200$. The considered probability distributions are: $U(-10,100)$, $\mathcal{N}(25,3)$, $P(35)$.
- For the CNN model, the loss function is mean square error, and the structure of the CNN model is given in Table 2..
- For the hyperparameters, the learning rate is 0.00001. The batch size is 90. The training rounds is 10.

Fig. 5 shows the training loss using the above-mentioned setting and joint training. The loss function value is around 1200 at the start of the training. Then, it drops off quickly in the first 500 iterations. The loss function gradually converges to a single-digit value by our proposed method after 10 000 iterations. It means that the CNN model successfully acquires the ability to predict the matrix game problems under the given selected game sizes and the distributions.

Table 3 compares the two training options provided in Algorithm 2 under the same setting. We can see that the loss function of the separated training is mostly smaller than the one in the joint training during the first 2000 iterations whilst the loss function of the joint training is mostly smaller than the separated training counterpart when the number of iterations is beyond 2000. That is, at shorter training times, the separated training is more advantageous, but in the long run, the joint training can reduce the loss function to a lower level.

Table 5
CNN for untrained game sizes and distributions.

Game sizes	Mixed [0.3, 0.5, 0.2][a]		Mixed [1, 1, 1][b]		UU generating procedure[c]	
	Mean value	Gap	Mean value	Gap	Mean value	Gap
10*10	34.03	3.14%	106.42	3.23%	77.32	3.74%
25*25	32.57	1.60%	102.83	2.32%	74.75	2.35%
75*75	34.25	3.65%	108.12	2.91%	76.61	2.92%
150*150	33.82	2.44%	106.80	1.66%	77.00	1.61%

[a]The mixed distribution $\mathcal{P} = 0.3 * U(-10, 100) + 0.5 * \mathcal{N}(25, 3) + 0.2 * Pois(35)$.
[b]The mixed distribution $\mathcal{P} = 1.0 * U(-10, 100) + 1.0 * \mathcal{N}(25, 3) + 1.0 * Pois(35)$.
[c]Described in Section 4.1, a generating procedure with high variance.

Table 6
Comparison between LP and CNN.

Game sizes	LP		CNN		
	CPU time	Value	CPU time	Value	Gap
10*10	0.0002	44.95	0.0011	45.70	6.79%
50*50	0.0016	45.02	0.0011	46.47	3.15%
100*100	0.0066	44.92	0.0011	45.32	1.06%
500*500	0.2537	45.00	0.0013	45.58	1.27%
1000*1000	1.3562	44.97	0.0090	45.63	1.45%
2000*2000	6.4688	45.04	0.0341	45.63	1.27%
3000*3000	19.2352	45.01	0.0789	45.63	1.36%

We use GAP as the evaluation metric for our model accuracy,

$$\text{GAP} = \left| \frac{\text{True value} - \text{Predicted value}}{\text{Predicted value}} \right| * 100\%. \quad (14)$$

Tables 4 and 5 show the model accuracy results after training. Each entry is averaged from 100 untrained test samples, which can be viewed as a test set. Table 4 presents the results for game sizes and distributions in the training candidate pool. Table 5 show the results when game sizes and generating distributions are not in the training candidate pool. For example, the game size $10 * 10$ and the uniform distribution $U(-10, 100)$ in Table 4 are considered in the training candidate pool, while the game size $25 * 25$ and the mixed distributions in Table 5 are not.

Tables 4 and 5 show that the CNN model receives an excellent predictive ability with a satisfying gap error after training. Moreover, Table 5 shows that the CNN model can even solve a matrix game from an untrained distribution and untrained game size.

4.3. Computational performances of CNN

Table 6 compares the computational performances of CNN and LP. Each row entry representing a game size is averaged from 100 instances. It goes from a small game size $10 * 10$ to a large game size $3000 * 3000$, and gives the mean values of two approaches and gap error of the CNN method. We use GPU for both training and predicting phrases for the CNN model. The computational speed increases by more than 100 times for our case by utilizing GPU.

The difference is not significant for small game sizes since LP is efficient enough to solve small-size linear programs. When the game size is large, the advantage of CNN becomes notable. It is much faster than LP, and the gap error is relatively small. For example, for a 3000*3000 size matrix game, the CNN approach is 200 times faster than LP with a 1.36% gap loss.

Table 7 shows the simulation results of LP, CNN, FP, and Exp3 for 1000*1000 games uniformly generated in the interval $[-10, 100]$. We set the number of rounds for FP and Exp3 to 10 000. We can see that our CNN model outperforms LP, FP, and Exp3 in terms of CPU time. Notice that FP and Exp3 require a high number of rounds to provide a good approximation of the game's value, which makes them less competitive for large size games. Within 10 000 rounds, Exp3 shows the lowest gap whilst FP requires the highest computing time.

Table 7
Comparison between LP, CNN and two learning algorithms.

Algorithms	1000*1000		
	CPU time	Value	Gap (%)
LP	1.3035	45.0293	–
CNN	0.00086	45.6481	1.3553%
FP	259.3845	47.6020	5.4043%
Exp3	0.7100	45.0431	0.0304%

5. Conclusion

In this paper, we study a two-player zero-sum game to find the saddle point for any given matrix game. We use a novel machine learning method CNN to achieve the goal and compare it with the traditional linear programming method and two learning algorithms, namely FP and Exp3. We design a specific CNN structure containing a global pooling layer capable of handling varying input game sizes. Hence, we develop a complete pipeline, including data generation and model fitting. We study saddle point value distributions for different matrix game generating procedures. Our numerical experiment shows that the CNN method outperforms the traditional linear programming method and the two learning algorithms with a reasonable loss. Furthermore, the CNN method can take advantage of parallel computing, which provides high computing performance when solving multiple games simultaneously. Our approach can be extended to other game theory problems, namely n-player games.

CRediT authorship contribution statement

Dawen Wu: Writing – original draft, Writing – review & editing. **Abdel Lisser:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

All of the sources of funding for the work described in this publication are acknowledged below:
The first author is supported by a Chinese Scholarship Council grant No. CSC202006010086.

References

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (2003). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, *32*(1), 48–77. http://dx.doi.org/10.1137/S0097539701398375.
Berger, U. (2007). Brown's original fictitious play. *Journal of Economic Theory*, *135*(1), 572–578. http://dx.doi.org/10.1016/j.jet.2005.12.010.

Charilas, D. E., & Panagopoulos, A. D. (2010). A survey on game theory applications in wireless networks. *Computer Networks*, *54*(18), 3421–3430. http://dx.doi.org/10.1016/j.comnet.2010.06.020.

Cheng, J., Leung, J., & Lisser, A. (2016). Random-payoff two-person zero-sum game with joint chance constraints. *European Journal of Operational Research*, *252*(1), 213–219. http://dx.doi.org/10.1016/j.ejor.2015.12.024.

Chivukula, A. S., & Liu, W. (2017). Adversarial learning games with deep learning models. In *2017 international joint conference on neural networks (IJCNN)* (pp. 2758–2767). http://dx.doi.org/10.1109/IJCNN.2017.7966196.

Courville, I. G., Bengio, Y., & Aaron (2016). Deep learning. *Nature*, *29*(7553), 1–73, URL: http://www.deeplearningbook.org.

Cplex, I. I. (2009). V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, *46*(53), 157.

Dantzig, G. (2018). *Linear programming and extensions*. Santa Monica, CA: RAND Corporation, http://dx.doi.org/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*(2), 31–43. http://dx.doi.org/10.1609/aimag.v40i2.2847, arXiv:1912.02258.

Deng, J., Dong, W., Socher, R., Li, L.-J., Kai Li, & Li Fei-Fei (2010). ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255). http://dx.doi.org/10.1109/cvpr.2009.5206848.

Dixit, A. K., & Pindyck, R. S. (2012). *Investment under Uncertainty* (pp. 1–468). Princeton University Press, http://dx.doi.org/10.2307/2329279.

Dixit, A. K., Skeath, S., & McAdams, D. (2020). *Games of strategy: Fifth edition*. WW Norton & Company.

Duque, A. B., Santos, L. L. J., Macêdo, D., & Zanchettin, C. (2019). Squeezed very deep convolutional neural networks for text classification. In *LNCS*: *Vol. 11727*, *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (pp. 193–207). http://dx.doi.org/10.1007/978-3-030-30487-4_16, arXiv:1901.09821.

Fan, K. (1953). Minimax theorems. *Proceedings of the National Academy of Sciences*, *39*(1), 42–47. http://dx.doi.org/10.1073/pnas.39.1.42.

Farnia, F., & Ozdaglar, A. (2020). Do GANs always have Nash equilibria? In *Proceedings of machine learning research*: *Vol. PartF16814*, *37th international conference on machine learning, ICML 2020* (pp. 3010–3020). PMLR, URL: https://proceedings.mlr.press/v119/farnia20a.html.

Fischer, P., Dosovitskiy, A., & Brox, T. (2015). Image orientation estimation with convolutional networks. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), Vol. 9358* (pp. 368–378). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-24947-6_30.

Ganapathiraman, V., Zhang, X., Yu, Y., & Wen, J. (2016). Convex two-layer modeling with latent structure. In *Advances in neural information processing systems, Vol. 29* (pp. 1288–1296). Curran Associates, Inc., URL: https://proceedings.neurips.cc/paper/2016/file/5487315b1286f907165907aa8fc96619-Paper.pdf.

Gearhart, J. L., Adair, K. L., Detry, R. J., Durfee, J. D., Jones, K. A., & Martin, N. (2012). Comparison of open-source linear programming solvers. http://dx.doi.org/10.2172/1104761.

Gibbons, R. S. (2019). *Game theory for applied economists*. Princeton University Press, http://dx.doi.org/10.1515/9781400835881.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144. http://dx.doi.org/10.1145/3422622, arXiv:1406.2661.

Gurobi Optimization, LLC (2021). Gurobi optimizer reference manual. URL: https://www.gurobi.com.

Hauert, C., & Szabó, G. (2005). Game theory and physics. *American Journal of Physics*, *73*(5), 405–414. http://dx.doi.org/10.1119/1.1848514.

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, Vol. 2016-Decem* (pp. 770–778). http://dx.doi.org/10.1109/CVPR.2016.90, arXiv:1512.03385.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.

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, Vol. 2017-Janua* (pp. 2261–2269). http://dx.doi.org/10.1109/CVPR.2017.243, arXiv:1608.06993.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Li, F. F. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition* (pp. 1725–1732). http://dx.doi.org/10.1109/CVPR.2014.223.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet Classification with deep convolutional neural networks. In *NIPS'12*: *Communications of the ACM*, In *NIPS'12*: *60*(6), 84–90.http://dx.doi.org/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/~lisa/pointeurs/handbook-convo.pdf.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21. http://dx.doi.org/10.1109/tnnls.2021.3084827, arXiv:2004.02806.

Ling, C. K., Fang, F., & Zico Kolter, J. (2018). What game are we playing? End-to-end learning in normal and extensive form games. In *IJCAI international joint conference on artificial intelligence, Vol. 2018-July* (pp. 396–402). http://dx.doi.org/10.24963/ijcai.2018/55, arXiv:1805.02777.

Ling, C. K., Fang, F., & Zico Kolter, J. (2019). Large scale learning of agent rationality in two-player zero-sum games. *33*, In *33rd AAAI conference on artificial intelligence, AAAI 2019, 31st innovative applications of artificial intelligence conference, IAAI 2019 and the 9th AAAI symposium on educational advances in artificial intelligence, EAAI 2019* (pp. 6104–6111). http://dx.doi.org/10.1609/aaai.v33i01.33016104, arXiv:1903.04101.

Mahendran, S., Ali, H., & Vidal, R. (2017). 3D pose regression using convolutional neural networks. In *Proceedings - 2017 IEEE international conference on computer vision workshops, ICCVW 2017, Vol. 2018-Janua* (pp. 2174–2182). http://dx.doi.org/10.1109/ICCVW.2017.254, arXiv:1708.05628.

Matlab (2017). *MATLAB version 9.3.0.713579 (R2017b)*. Natick, Massachusetts: The MathWorks Inc..

Myerson, R. B. (1997). *Game theory: an analysis of conflict*. Harvard University Press, http://dx.doi.org/10.2307/j.ctvjsf522.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, *36*(1), 48–49. http://dx.doi.org/10.1073/pnas.36.1.48.

Nocedal, J., & Wright, S. J. (2006). *Springer series in operations research and financial engineering*, *Numerical optimization* (pp. 1–664). Springer New York, http://dx.doi.org/10.1201/b19115-11.

O'Donoghue, B., Lattimore, T., & Osband, I. (2020). Matrix games with bandit feedback. arXiv:2006.05145.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (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, Vol. 32*. Curran Associates, Inc., arXiv:1912.01703.

Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. http://dx.doi.org/10.1162/NECO_a_00990.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. http://dx.doi.org/10.1038/323533a0.

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

Singh, H. (1999). Introduction to game theory and its application in electric power markets. *IEEE Computer Applications in Power*, *12*(4), 18–20. http://dx.doi.org/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*(3), 839–845. http://dx.doi.org/10.1016/j.ejor.2019.01.010.

Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., et al. (2016). Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, *35*(5), 1299–1312. http://dx.doi.org/10.1109/TMI.2016.2535302, arXiv:1706.00712.

Tembine, H. (2020). Deep learning meets game theory: Bregman-based algorithms for interactive deep generative adversarial networks. *IEEE Transactions on Cybernetics*, *50*(3), 1132–1145. http://dx.doi.org/10.1109/TCYB.2018.2886238.

Vanderbei, R. J. (2014). *Linear programming, Vol. 196*. Boston, MA: Springer US, http://dx.doi.org/10.1007/978-1-4614-7630-6.

Vega-Redondo, F. (2003). *Economics and the theory of games* (pp. 1–512). CRC Press, http://dx.doi.org/10.1017/CBO9780511753954.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al., SciPy 1. 0 Contributors (2020). SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, *17*, 261–272. http://dx.doi.org/10.1038/s41592-019-0686-2.

von Neumann, J. (1928). Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, *100*(1), 295–320. http://dx.doi.org/10.1007/BF01448847.

von Neumann, J., & Morgenstern, O. (2007). *Theory of games and economic behavior*. Princeton, NJ, US: Princeton University Press, http://dx.doi.org/10.2307/2981222.

Wiatowski, T., & Bolcskei, H. (2018). A mathematical theory of deep convolutional neural networks for feature extraction. *IEEE Transactions on Information Theory*, *64*(3), 1845–1866. http://dx.doi.org/10.1109/TIT.2017.2776228, arXiv:1512.06293.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. http://dx.doi.org/10.1007/s13244-018-0639-9.

Zhou, D. X. (2020). Universality of deep convolutional neural networks. http://dx.doi.org/10.1016/j.acha.2019.06.004, arXiv:1805.10769.

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

Zhu, D., Li, Z., Wang, X., Gong, B., & Yang, T. (2020). A robust zero-sum game framework for pool-based active learning. In *Proceedings of machine learning research*: *Vol. 89*, *AISTATS 2019 - 22nd international conference on artificial intelligence and statistics* (pp. 517–526). PMLR, URL: https://proceedings.mlr.press/v89/zhu19a.html.