Using CNN for solving two-player zero-sum games

Dawen Wu, Abdel Lisser

Using CNN for solving two-player zero-sum games

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

[a] *Centralesupelec, Université Paris-Saclay*

Abstract

We study a two-player zero-sum game (matrix game for short) with the objective to find 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 size, generating training dataset and model fitting. The experiment results show that our proposed method outperforms the traditional linear programming (LP for short) method in terms of computational efforts.

Keywords: Two-player Zero-sum Game, Saddle point, Convolutional neural network, Machine learning

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 games describes a situation when two players optimize their payoff functions simultaneously. The definitions of

*Corresponding author

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

the saddle point and its value are

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

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

The saddle point equilibrium in (1) can be solved by linear programs (3) and (4). The minimax Theorem [1] states that the optimal objective values v in those two linear programs are equal.

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

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

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 components: model parameters θ, input $\mathbf{A}$, true value v and predicted value $\hat{v}$. The input $\mathbf{A}$ 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 $\hat{v}$ is a real value or a vector 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).

$$f_\theta(\mathbf{A}) = \hat{v} \tag{5}$$

$$L_{\mathcal{D}}(\theta) = \mathbb{E}_{\mathcal{D}}\ell\left(f_\theta(\mathbf{A}), v\right) \tag{6}$$

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

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 for using CNN in two aspects, computation speed and accuracy, and compare it to linear programming, and two learning algorithms. 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 the two-player zero-sum game.
- x and y denote a row player and column player strategies, respectively, either pure or mixed.
- n and m denote the sizes of the action set of row player and column player, respectively.
- $(\mathbf{x}^*, \mathbf{y}^*)$ denotes the strategy profile of the saddle point.
- v^* denotes the value of the saddle point.
- $f_\theta(\cdot)$ denotes a CNN model, where θ is the model parameter.
- ℓ denotes a loss function used in training.
- $\mathcal{D}$ 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 John von Neumann and Oskar Morgenstern [1, 2], game theory has been widely developed both from theoretical and practical points of view [3, 4, 5, 6, 7]. In a two-player zero-sum context, saddle point [1] 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 [8] 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 are 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 [1, 9], 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 take the good decisions in a competitive environment, including business [10, 11], economics [12], and engineering [13]. Dantzig [14] shows that solving any matrix game is equivalent to a linear program. Most commercial or academic softwares such as Gurobi, CPLEX, Matlab, Scipy [15], provide tools for linear programming based on interior point method and simplex method [16, 17]. Besides, there are also some studies on the situation where the game contain randomness [18, 19].

CNN proposed by LeCun [20] is a kind of Deep neural networks [21]. As a type of Feedforward neural networks, it uses the back propagation algorithm for the training step [22]. 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 [23] won the ImageNet challenge in 2012 [24], there a tremendous amount of research on this topic [25, 26, 27], and more sophisticated CNN structures [28, 29, 30] have

been proposed. CNN has applications in many fields, e.g., image classification [31], medical image analysis [32], video recognition [33], natural language processing [34]. 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 [35, 36]. For varying input size, there are three approaches for solving game problems: global pooling, variable sized pooling, and padding input images [37].

More recently, two-player zero-sum games built many connections with deep learning, such as Generative adversarial networks(GAN) [38, 39, 40], with applications in Cybersecurity [41, 42]. In GAN [43], 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 [44, 45]. Moreover, some research work uses the zero-sum game theory framework to promote or understand machine learning algorithms [46, 47]. On the contrary, some research work uses machine learning methods to solve zero-sum games with incomplete observations [48, 49].

A large number of learning algorithms belongs to 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, see [50, 51, 52] and the references therein.

3. Methodology

In this section, we give a detailed presentation of our CNN model . Subsection 3.1 introduces the overall pipeline and the specific CNN model. Subsection 3.2 introduces the concrete training algorithm. Subsection 3.3 describes how to solve the corresponding saddle point strategy when the predicted saddle point

value v is known. Subsection 3.4 states the advantages and disadvantages of the CNN method compared to linear programming.

3.1. The CNN model

A CNN model, viewed as a function (5), maps a matrix game $\mathbf{A}$ to the predicted saddle point value $\hat{v}$. As shown in figure 1, the input matrix game at left-hand side goes through the CNN model to get the predicted value.

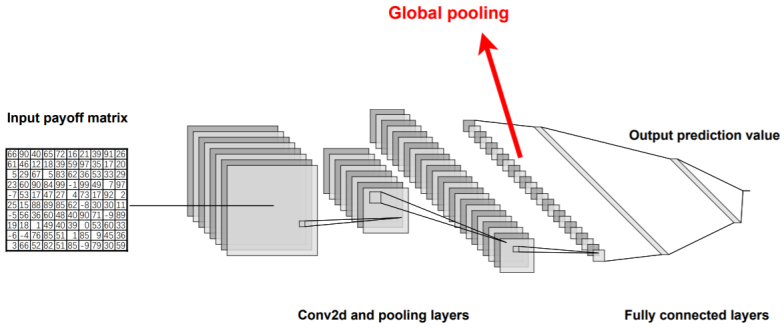


Figure 1: A CNN model

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 maps sizes are $6 * 10 * 10$ and $6 * 50 * 50$, respectively. After crossing a maximum pooling layer which follows the previous convolutional layer, and has a kernel size $2 * 2$ and stride 2, the feature maps 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 6.

3.2. The training algorithm

Figure 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 Figure 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.

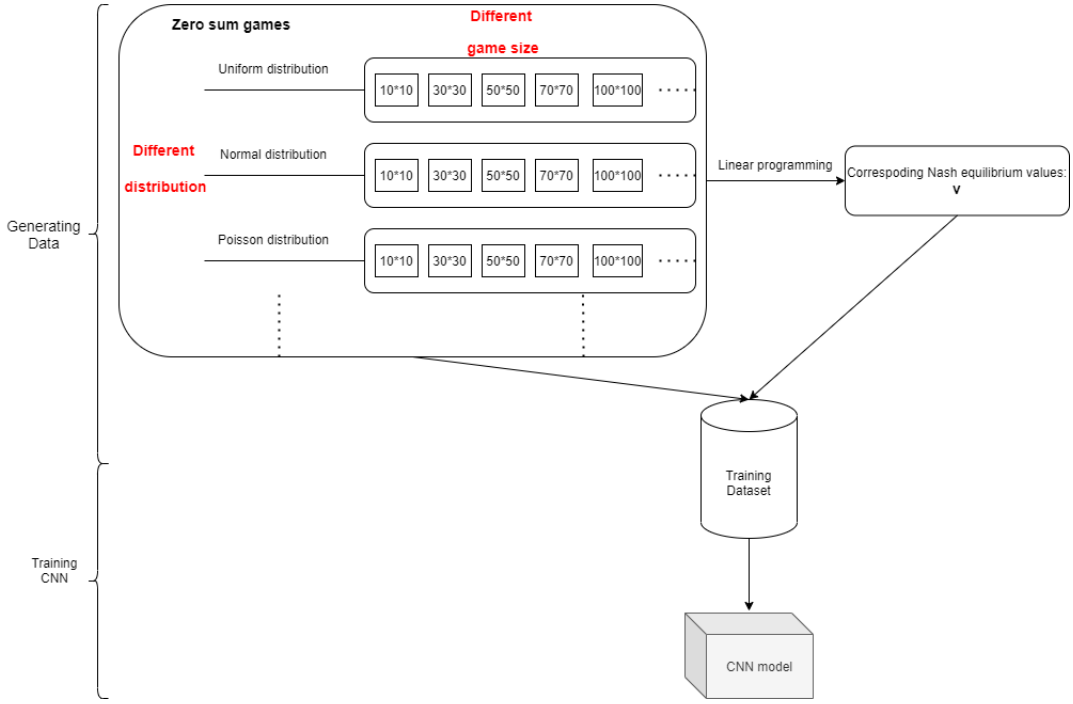


Figure 2: The training procedure for the CNN model

Algorithms 1 and 2 are the concrete training methods for the CNN model. Algorithm 1 shows the procedures to generates 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 train 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 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.

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($\mathbf{b}$, net)**:**
    net $\leftarrow \mathbf{b}$: Train the CNN model by the sample $\mathbf{b}$.
end

Different hyperparameters 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 training round, which indicates how many times a sample will be used repeatedly.

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($\mathbf{b}$)
  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($\mathbf{b}, net$)
    end
  end
  return net
end

3.3. Saddle point strategy

Although the CNN model can predict the saddle point's value after the training step, it cannot estimate the corresponding mixed strategies. Since it will be too complicated to modify the CNN model to be able to predict the mixed strategies, we compute the related mixed strategies of the predicted value by solving the following system of equations where $\mathbf{A}$ and $\hat{v}$ are known.

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

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

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

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

Notice that the system (10) can be reduced to the linear systems (12) or (13), which can be solved by any linear programming solver. If the predicted value is larger than the true value, (12) will have a feasible solution. Otherwise, (13) will have a feasible solution. For (12), the related y is a unit vector with only one element of 1 and all other elements of 0, and the position of one is the same as the position of the maximum value in the vector $\mathbf{A^T}x$. Similarly, for (13), the position of one in the related unit vector x is the same as the position of the maximum value in the vector $\mathbf{A}y$. We only need to solve one of the linear systems in order to get the desired strategy profile. However, since the true value is unknown, it is necessary to consider both of them to get the strategy profile.

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

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

3.4. Pros and cons of our CNN method

We give the advantages and disadvantages of the CNN method in comparison with linear programming. The most significant advantage of the CNN method is the computational performances. That is because it directly solves the problems 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, our CNN model can also deal with out of the pool game sizes and generating distributions.

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 which proceeds in rounds manner. In each round, the players play a best response to mixed strategy obtained by previous rounds empirical frequencies of actions. FP was originally introduced by Brown, see [50] and references therein. Exp3 is a popular adversarial multiarmed bandits algorithm suggested and studied in this setting by [51]. 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.

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 an eight virtual N2D CPU, 64GB of memory, 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.

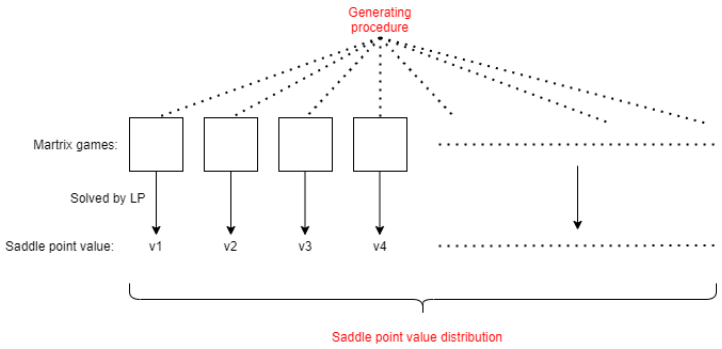


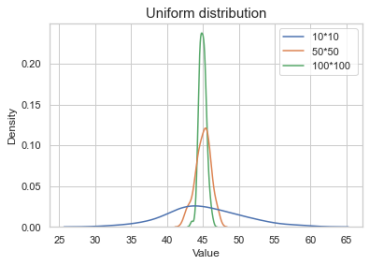
Figure 3: Saddle point value distribution

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 linear programming to get the saddle point value. Figure 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.

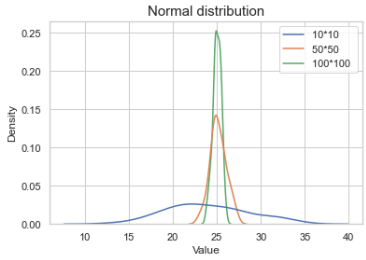
We consider three game sizes, namely $10 * 10$, $50 * 50$ and $100 * 100$. We generate 100 instances for each game size and for each one of the following three distributions as generating procedures: Uniform distribution with interval

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

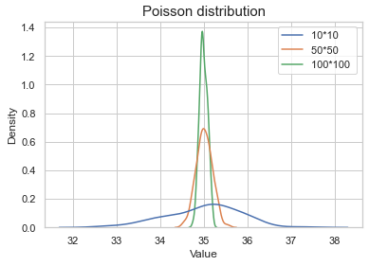
$[-10, 100]$, Normal distribution with mean 25 and standard deviation 3, Poisson distribution with $\lambda = 35$, i.e., $U(-10, 100)$, $\mathcal{N}(25, 3)$, $P(35)$. Besides, we study a more complex generating procedure with two uniform distributions denoted as UU. The UU generating procedure starts with sampling two points l_1, l_2 from $U(0, 75)$ and $U(75, 150)$ respectively, and a matrix game is generated by $U(l_1, l_2)$. Figure 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 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 $10 * 10$.



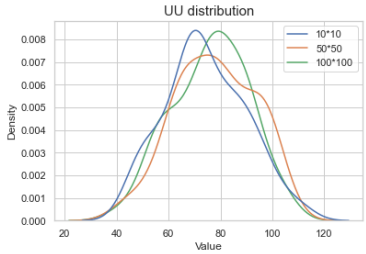
(a) From Uniform distribution $U(-10, 100)$



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



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



(d) From UU generating procedure

Figure 4: Saddle point value distribution

Figure 4a's mean value is 45, which is the median number of the Uniform distribution. Figure 4b's mean value is 25, which is the μ of the Normal distribution. Figure 4c'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

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

Table 1: Moments of saddle point value distributions

increases, and the distributions are sharper in figure 4a, 4b, and 4c. 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 are generally symmetric and taildness 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. The structure of the CNN model is given in Table 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 activation function is rectified linear activation unit generally noted "leaky relu".
- For the hyperparameters, the learning rate is 0.00001. The batch size is 90. The training rounds is 10.

layer	Detail	Output size
Conv2d	Filter size: 3*3 Filters number: 16 Padding: 1 Activation: leaky relu	(16, 100, 100)
Conv2d	Filter size: 3*3 Filters number: 32 Padding: 1 Activation: leaky relu Pooling: 2*2 max pooling	(32, 50, 50)
Conv2d	Filter size: 3*3 Filters number: 64 Padding: 1 Activation: leaky relu	(64, 50, 50)
Conv2d	Filter size: 3*3 Filters number: 64 Padding: 1 Activation: leaky relu Pooling: 2*2 max pooling global pooling	(64,)
fc1	Neurons number: 32 Activation: leaky relu	(32,)
fc2	Neurons number: 16 Activation: leaky relu	(16,)
fc3	Neurons number: 10 Activation: leaky relu	(10,)
fc4	Neurons number: 10 Activation: leaky relu	(10,)
fc5	Neurons number: 1 Activation: leaky relu	(1,)

Table 2: the structure of the CNN model

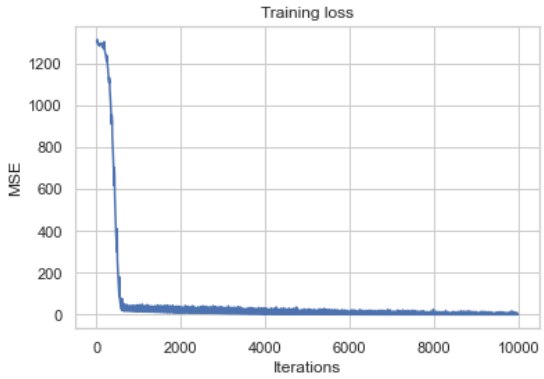


Figure 5: Training Loss of the CNN model

Figure 5 shows the training loss using the above mentioned setting and joint training. The loss function value is around 1200 at the starting 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 10000 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.

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 3: Difference between the separated and the joint training

Table 3 compares the two training options provided in Algorithm 2. We can see that the loss function of the separated training is smaller than the one in the joint training during the first 2000 iterations whilst the loss function of the joint training is smaller than the separated training counterpart when the number of iterations is beyond 2000. Notice that the loss function of the joint training is less than one after 5000 iterations.

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

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

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.

	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%

Table 4: CNN for trained game sizes and distributions

	Mixed [0.3, 0.5, 0.2]*		Mixed [1, 1, 1]**		UU generating procedure***	
	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%

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

Table 5: CNN for untrained game sizes and distributions

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.

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%

Table 6: Comparison between LP and CNN

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 game uniformly generated in interval $[-10, 100]$. We set the number of rounds for FP and Exp3 to 10000. 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 value of the game which make them less competitive for large size games. Within 10000 rounds Exp3 shows the lowest gap whilst FP requires the highest computing time.

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%

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

5. Conclusion

In this paper, we study a two-players zero-sum games with the aim 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 as well as with 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 show 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.

Bibliography

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

[2] J. von Neumann, O. Morgenstern, Theory of games and economic behavior, Princeton University Press, 1947.

[3] R. B. Myerson, Game theory, Harvard university press, 2013.

[4] A. K. Dixit, S. Skeath, D. McAdams, Games of Strategy: Fifth International Student Edition, WW Norton & Company, 2020.

[5] C. Hauert, G. Szabó, Game theory and physics, American Journal of Physics 73 (5) (2005) 405–414.

[6] D. E. Charilas, A. D. Panagopoulos, A survey on game theory applications in wireless networks, Computer Networks 54 (18) (2010) 3421–3430.

[7] R. S. Gibbons, Game theory for applied economists, Princeton University Press, 1992.

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

[9] K. Fan, Minimax theorems, Proceedings of the National Academy of Sciences of the United States of America 39 (1) (1953) 42.

[10] A. K. Dixit, R. K. Dixit, R. S. Pindyck, Investment under uncertainty, Princeton university press, 1994.

[11] D. G. Luenberger, et al., Investment science, OUP Catalogue.

[12] M. Bacharach, Economics and the Theory of Games, CRC Press, 2019.

[13] H. Singh, Introduction to game theory and its application in electric power markets, IEEE Computer Applications in Power 12 (4) (1999) 18–20.

[14] G. B. Dantzig, Linear programming and extensions, Vol. 48, Princeton university press, 1998.

[15] J. L. Gearhart, K. L. Adair, R. J. Detry, J. D. Durfee, K. A. Jones, N. Martin, Comparison of open-source linear programming solvers, Sandia National Laboratories, SAND2013-8847.

[16] R. J. Vanderbei, et al., Linear programming, Vol. 3, Springer, 2015.

[17] J. Nocedal, S. J. Wright, Numerical Optimization, 2nd Edition, Springer, New York, NY, USA, 2006.

[18] J. Cheng, J. Leung, A. Lisser, Random-payoff two-person zero-sum game with joint chance constraints, European Journal of Operational Research 252 (1) (2016) 213–219.

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

[20] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, The handbook of brain theory and neural networks 3361 (10) (1995) 1995.

[21] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.

[22] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning Representations by Back-propagating Errors, Nature 323 (6088) (1986) 533–536.

[23] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012) 1097–1105.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE conference on computer vision and pattern recognition, IEEE, 2009, pp. 248–255.

[25] Z. Li, F. Liu, W. Yang, S. Peng, J. Zhou, A survey of convolutional neural networks: analysis, applications, and prospects, IEEE Transactions on Neural Networks and Learning Systems.

[26] T. Wiatowski, H. Bölcskei, A mathematical theory of deep convolutional neural networks for feature extraction, IEEE Transactions on Information Theory 64 (3) (2017) 1845–1866.

[27] D.-X. Zhou, Universality of deep convolutional neural networks, Applied and computational harmonic analysis 48 (2) (2020) 787–794.

[28] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[29] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[30] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861.

[31] W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural computation 29 (9) (2017) 2352–2449.

[32] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, J. Liang, Convolutional neural networks for medical image analysis: Full training or fine tuning?, IEEE transactions on medical imaging 35 (5) (2016) 1299–1312.

[33] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.

[34] A. Conneau, H. Schwenk, L. Barrault, Y. Lecun, Very deep convolutional networks for text classification, arXiv preprint arXiv:1606.01781.

[35] P. Fischer, A. Dosovitskiy, T. Brox, Image orientation estimation with convolutional networks, in: German Conference on Pattern Recognition, Springer, 2015, pp. 368–378.

[36] S. Mahendran, H. Ali, R. Vidal, 3d pose regression using convolutional neural networks, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 2174–2182.

[37] R. Yamashita, M. Nishio, R. K. G. Do, K. Togashi, Convolutional neural networks: an overview and application in radiology, Insights into imaging 9 (4) (2018) 611–629.

[38] Y. Zhou, M. Kantarcioglu, B. Xi, A survey of game theoretic approach for adversarial machine learning, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 9 (3) (2019) e1259.

[39] P. Dasgupta, J. Collins, A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks, AI Magazine 40 (2) (2019) 31–43.

[40] H. Tembine, Deep learning meets game theory: Bregman-based algorithms for interactive deep generative adversarial networks, IEEE Transactions on Cybernetics 50 (3) (2020) 1132–1145.

[41] S. Mahdavifar, A. A. Ghorbani, Application of deep learning to cybersecurity: A survey, Neurocomputing 347 (2019) 149–176.

[42] C. Yinka-Banjo, O.-A. Ugot, A review of generative adversarial networks and its application in cybersecurity, Artificial Intelligence Review (2019) 1–16.

[43] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, Advances in neural information processing systems 27.

[44] A. S. Chivukula, W. Liu, Adversarial learning games with deep learning models, in: 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2758–2767.

[45] D. Zhu, Z. Li, X. Wang, B. Gong, T. Yang, A robust zero-sum game framework for pool-based active learning, in: K. Chaudhuri, M. Sugiyama (Eds.), Proceedings of Machine Learning Research, Vol. 89 of Proceedings of Machine Learning Research, PMLR, 2019, pp. 517–526.

[46] D. Schuurmans, M. A. Zinkevich, Deep learning games, in: D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 29, Curran Associates, Inc., 2016.

[47] F. Farnia, A. Ozdaglar, Do gans always have nash equilibria?, in: International Conference on Machine Learning, PMLR, 2020, pp. 3029–3039.

[48] C. K. Ling, F. Fang, J. Z. Kolter, What game are we playing? end-to-end learning in normal and extensive form games, arXiv preprint arXiv:1805.02777.

[49] C. K. Ling, F. Fang, J. Z. Kolter, Large scale learning of agent rationality in two-player zero-sum games, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 6104–6111.

[50] U. Berger, Brown's original fictitious play, Journal of Economic Theory 135 (2007) 572–578.

[51] P. Auer, N. Cesa-Bianchi, Y. Freund, R. Schapire, The nonstochastic multiarmed bandit problem, SIAM Journal of Computing 32 (1) (2007) 48–77.

[52] B. O'Donoghue, T. Lattimore, I. Osband, Stochastic matrix games with bandit feedback, arXiv preprint arXiv:2006.05145.