CentraleSupélec | université PARIS-SACLAY

POLYTECH PARIS-SACLAY

CHINA SCHOLARSHIP COUNCIL

Using CNN to solve two-player zero-sum games

Dawen Wu

2022 /05/20

Summary

- Two-player Zero-sum Games
- Convolutional neural networks
-
- Prediction of the saddle point value
- Solution of the saddle point strategy
- Numerical results
- Conclusion

Two-player Zero-sum Games

Preliminary 1: Problem formulation

Two-player Zero-sum games

Matrix games

A two-player zero-sum game has the form:

- Two player : player 1 and player 2
- Finite actions : $A_1 = \{1, \dots, n\}, A_2 = \{1, \dots, m\}$
- Payoff : When player 1 chooses action i and player 2 chooses action j, players 1 and 2 receive payoffs a_{ij} and $-a_{ij}$

The player 1's payoff can be simplified as a matrix with shape (n, m). $A = (a_{ij})$
The player 2's payoff can be simplified as a matrix with shape (n, m). $-A = (-a_{ij})$

Since Player 2's payoff is just the negative of the Player 1, we can represent a two-player zero-sum game by the player 1's matrix $A = (a_{ij})$.

The payoff matrix A contain all the information of the game, including both player action set and payoff.

The general form

Player 2

Player 1			
	a_{11}	a_{12}	a_{13}
	a_{21}	a_{22}	a_{23}
	a_{31}	a_{32}	a_{33}

Player 2

Player 1			
	$-a_{11}$	$-a_{12}$	$-a_{13}$
	$-a_{21}$	$-a_{22}$	$-a_{23}$
	$-a_{31}$	$-a_{32}$	$-a_{33}$

An example

Player 2

Player 1			
	3	5	7
	6	1	2
	-2	1	4

Player 2

Player 1			
	-3	-5	-7
	-6	-1	-2
	2	-1	-4

Two-player Zero-sum games

Solution concept

Mixed Strategy profile (x, y)

Let x, y be player 1 and player 2 strategies. x and y should be a discrete probability distribution, and satisfy

$e_n^T x = 1, x \geq 0.\ e_m^T y = 1, y \geq 0,$

Where e_n is a n-dimensional all-ones vector .

Saddle point

Solving a two-player zero-sum game mean solve the following equation

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

$$v^* = x^* A y^*$$

(x^*, y^*) is called the saddle point strategy (Nash equilibrium).
v^* is called the saddle point value.

Minimax theorem

The minimax Theorem states that the saddle point always exist for any two-player zero-sum games. (von Neumann, 1928)

Linear programming

The traditional and still state-of-the-art approach is to use linear programming to find (x^*, y^*)

The following primal-dual pair of linear programs can find the saddle point of the game (Dantzig, 1963).

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

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

Convolutional neural network

Preliminary 2: Machine learning approach

CentraleSupélec | université PARIS-SACLAY

Convolutional neural network (CNN)

CNN Model

A CNN model

Can be represented as a function $f_\theta(A) = \hat{v}$

Components:

- Input A, is an array.
- Parameters θ, are learnable parameters inside the CNN model.
- Output $\hat{v}$, is the CNN prediction. In most cases, $\hat{v}$ is a vector representing the probability corresponding to each category.
- True v, is the true value related to input A.

$\ell(f_\theta(A), v) = \ell(\hat{v}, v)$

measures the difference between the CNN prediction $\hat{v}$ and the true value v.

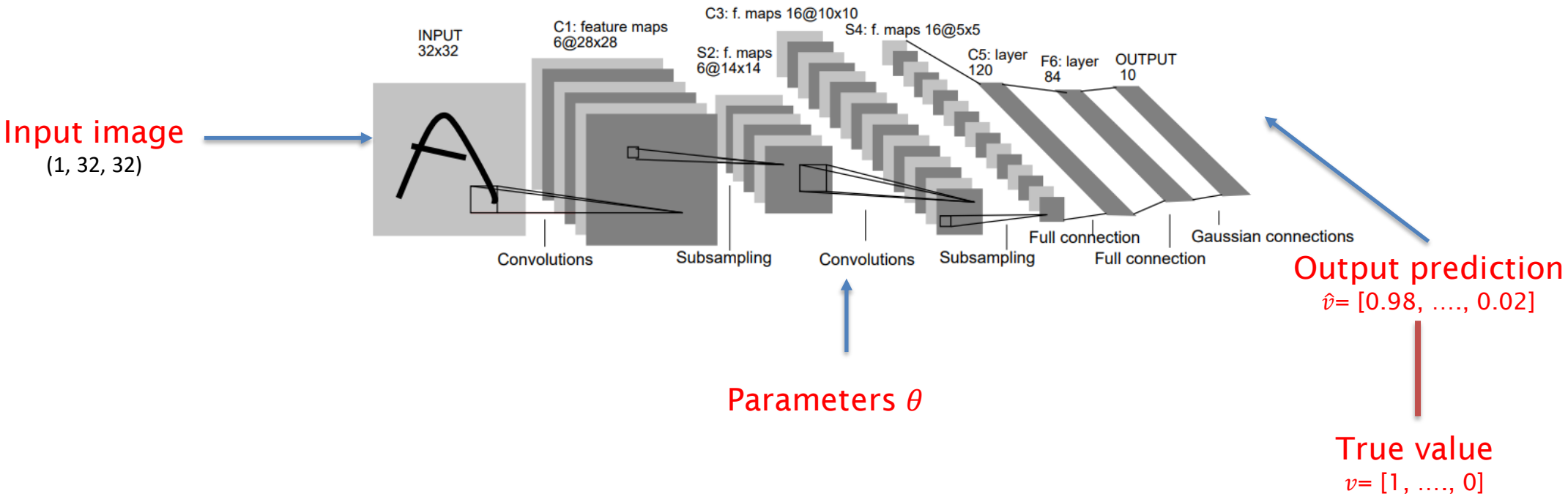
Application

CNNs have a large number of applications in the field of computer vision, where the input is an image, represented by a three-dimensional array $(channel, height, width)$. When the image is in black and white, channel=1. When the image is in color, channel=3 (RGB).

Convolutional neural network (CNN)

CNN Model-example

The following example shows how the CNN model recognizes that the input image is the letter "A".



Convolutional neural network (CNN)

Training

The objective of the training is to minimize the loss function or objective function w.r.t parameters θ.

The expected risk

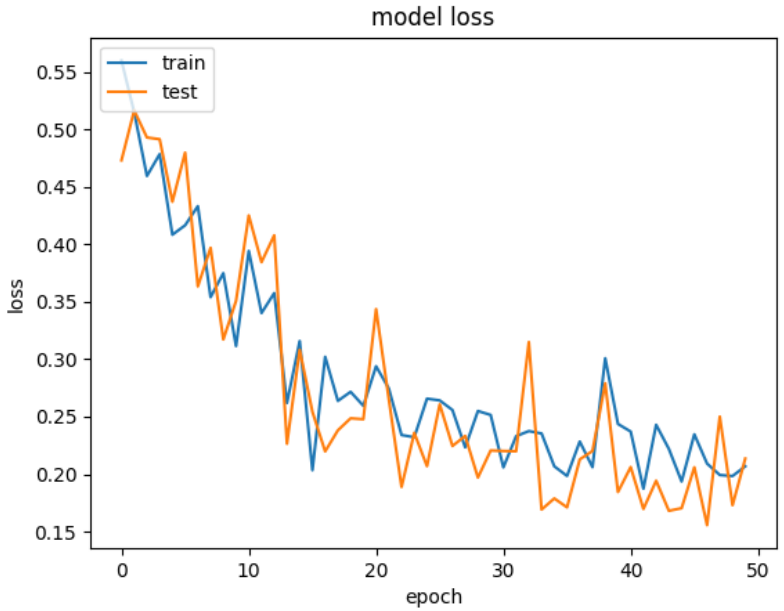
$$L_{\mathcal{D}}(\theta) = \mathbb{E}_{\mathcal{D}}\ell\left(f_\theta(A), v\right)$$

The empirical risk

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

When the training set is ready, we can optimize the empirical risk. This minimization of the empirical risk is an unconstrainted non-convex problem, where we can use gradient descent to optimize

$$\theta_{k+1} = \theta_k - \alpha\nabla\hat{L}_n(\theta_k)$$



Prediction of the saddle point value

Method

Solve the game problem by CNN

Model

- **Input A**

The input A is a two-player zero-sum game represented with the shape (1, n, m).

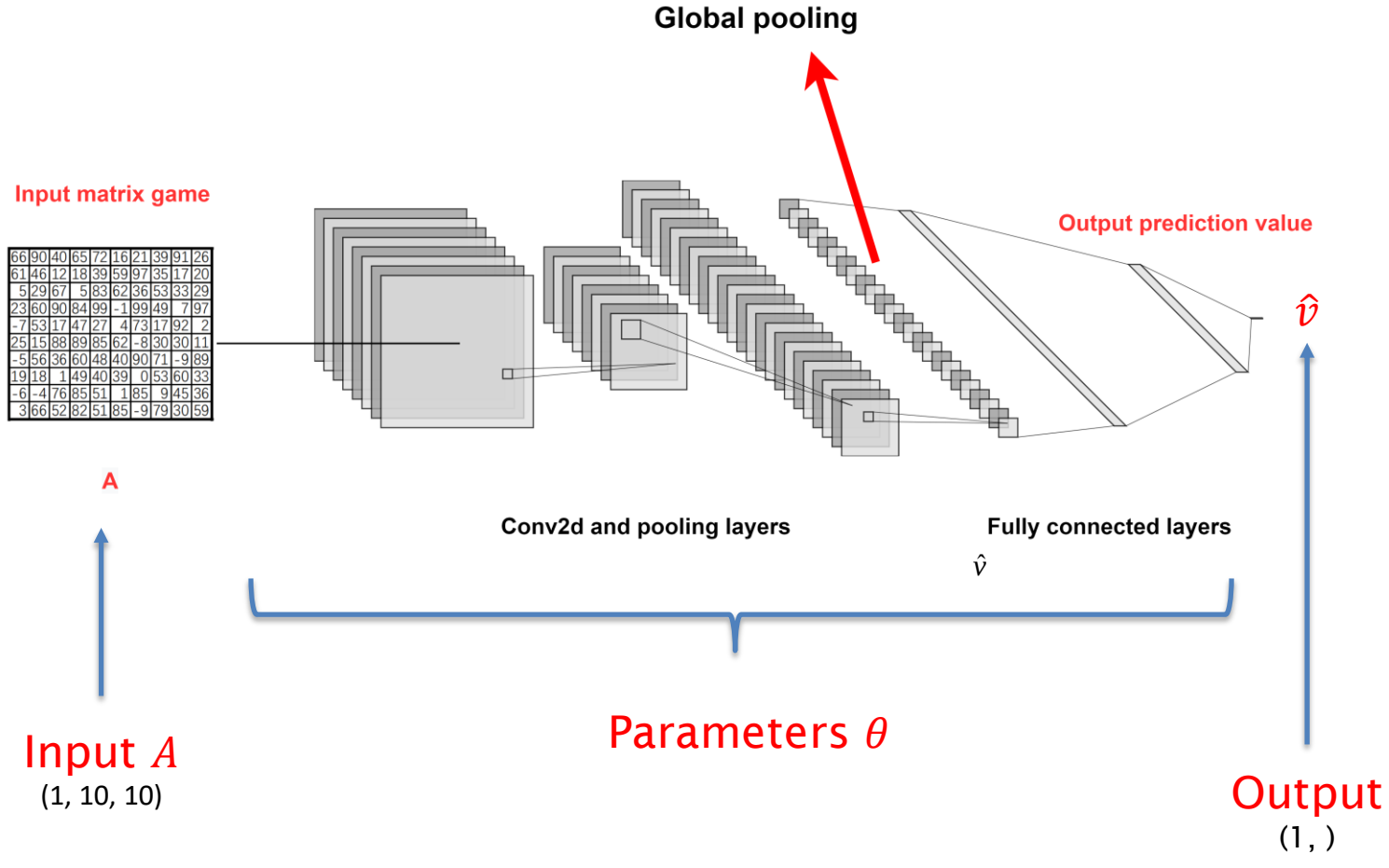
- **Output $\hat{v}$**

The output represent the CNN prediction to the saddle point value.

- **True v**

v is the saddle point value of the input matrix game.

We want the CNN model to be able to predict the saddle point value for a given two-player zero-sum game



Solve the game problem by CNN

Training-Data set

66	90	40	65	72	16	21	39	91	26
61	46	12	18	39	59	97	35	17	20
5	29	67	5	83	62	36	53	33	29
23	60	90	84	99	-1	99	49	7	97
-7	53	17	47	27	4	73	17	92	2
25	15	88	89	85	62	-8	30	30	11
-5	56	36	60	48	40	90	71	-9	89
19	18	1	49	40	39	0	53	60	33
-6	-4	76	85	51	1	85	9	45	36
3	66	52	82	51	85	-9	79	30	59

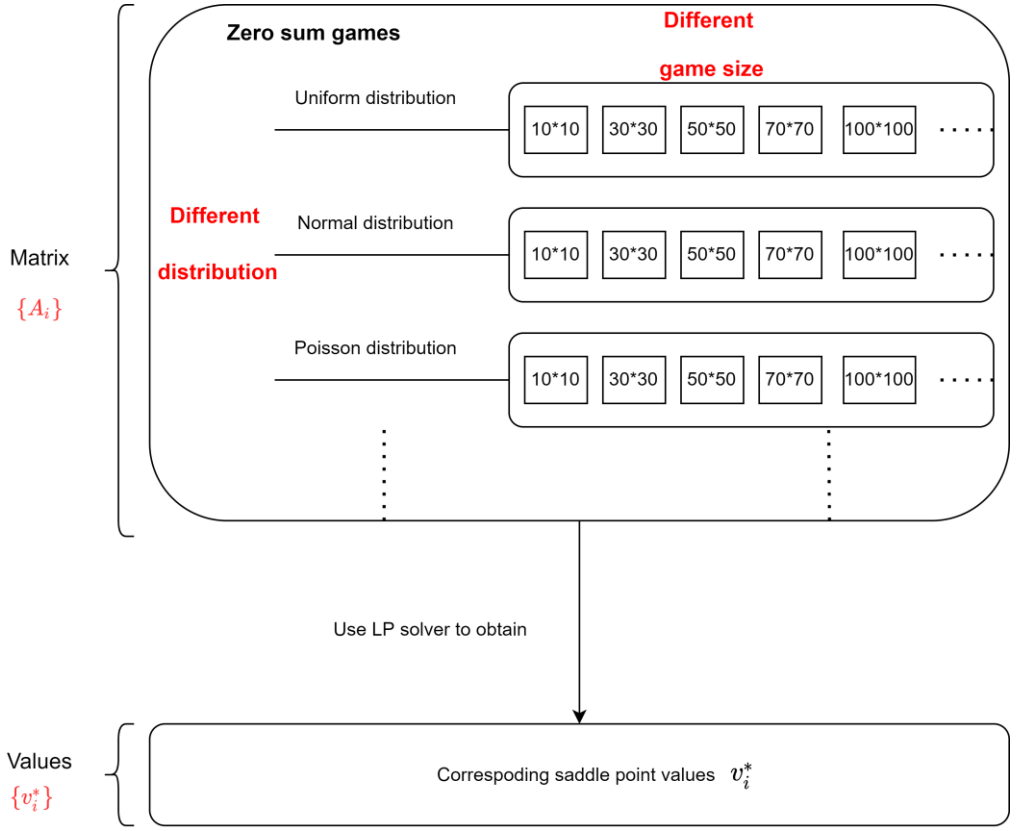
- **Matrix $\{A_i\}$**

Each A_i represents a two-player zero-sum game. A_i is sampled according to a probability distribution and a game size. For example, we can use the uniform distribution U(-10, 100) with game size (10, 10)

- **True $\{v_i^*\}$**

For each A_i, we obtain its true saddle point value v_i^* by using the linear programming solver.

A training sample has the form (A_i, v_i^*), where A_i represent the input game and v_i^* is its saddle point value. $\{(A_i, v_i^*)\}$ represents the whole training dataset.



Solve the game problem by CNN

Training-the overall procedure

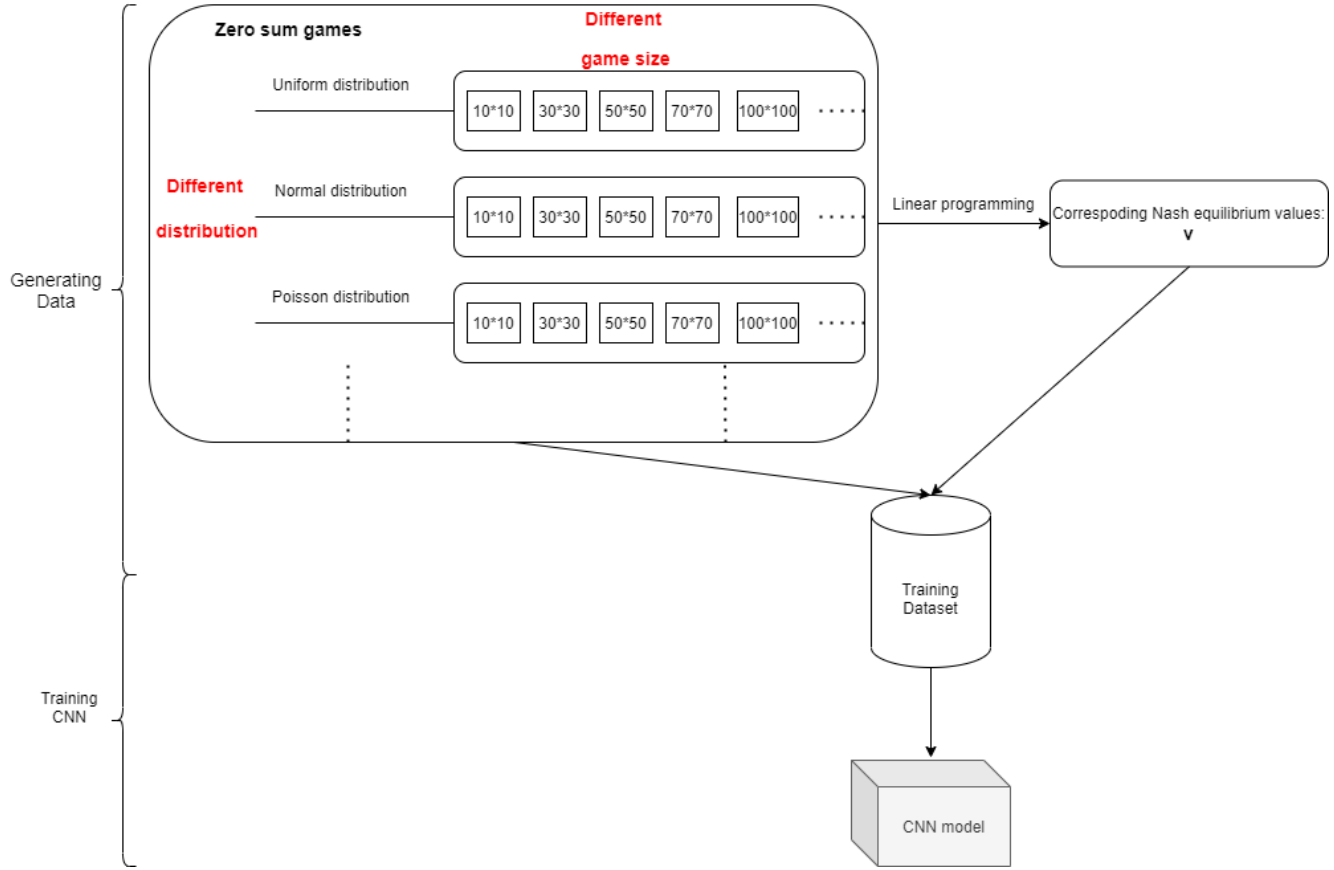
The overall training procedure can be dived as two parts.

- **Generating data**

Is to obtain the training data set $\{(A_i, v_i^*)\}$.

- **Training CNN**

Is to train the CNN model using the just generated dataset $\{(A_i, v_i^*)\}$.



Solve the game problem by CNN

Training-Algorithm-1

- **Function Generate**

Is used to generate a training sample (A_i, v_i^*). We need to provide this function with the game size (n, m) and probability distribution, and the matrix game will be generated according to these given conditions.

- **Function Train**

Is used to train the CNN model by the training sample (A_i, v_i^*).

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 $\mathbf{b}$.
end

Solve the game problem by CNN

Training-Algorithm-2

- **Separated training**

Is the traditional way to train a model, in machine learning study. The training dataset is created first and then the model will be trained on the dataset.

- **Joint training**

combines generating data and training model.

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(), 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 ℙ from the probability
         distributions pool
        b = Generate(m, n, 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 ℙ from the probability
         distributions pool
        b = Generate(m, n, ℙ)
        for k in K do
            Train(b, net)
        end
    end
    return net
end
```

Loop 1 for generating dataset

Loop 2 for training the CNN model

One Loop contains both generating data and training the CNN model

Solution of the saddle point strategy

Obtain the saddle point strategy

The CNN model can only give prediction $\hat{v}$ for v^*. we need to solve a linear system to obtain the corresponding saddle point strategy $(\hat{x}, \hat{y})$.

Solve the following Non-convex equality system to obtain the corrsponding $(\hat{x}, \hat{y})$, according to the $\hat{v}$ and A.

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

This can be solved directly By Gurobi or other non-convex solver.

It can be reduced to the convex Linear system

$$\mathbf{A}^T \hat{\mathbf{x}} \geq \hat{v} \mathbf{e_m}$$
$$\hat{\mathbf{x}}^T \mathbf{e_n} = 1, \hat{\mathbf{x}} \geq 0,$$
$$\mathbf{A} \hat{\mathbf{y}} \leq \hat{v} \mathbf{e_n}$$
$$\hat{\mathbf{y}}^T \mathbf{e_m} = 1, \hat{\mathbf{y}} \geq 0.$$

This can be solved all convex solver, such as cvxpy or scipy.

Example

A =

53	86	67	43	67
33	40	96	29	69
44	24	22	28	26
81	18	63	36	55
78	87	51	30	36

$\hat{v}$ = 35.6

$\hat{x}$ = [0. 06, 0.10, 0.62, 0.14, 0.08]

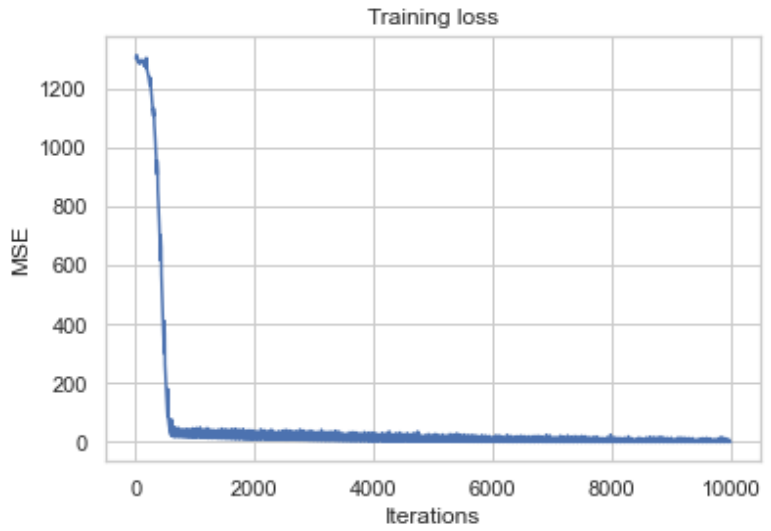
$\hat{y}$ = [0. 10, 0.23, 0.14, 0.36, 0.17]

Numerical results

Numerical results

Training loss

- The training loss is computed on the untrained test data.
- The loss function decreases from the initial 1300 to less than 1
- The joint training way can converge to a lower loss



	Iterations					
Training options	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

Numerical results

Comparison between LP and CNN

- CNN is much faster than LP in solving two-player zero-sum games
- CNN can take advantage of the GPU computation and parallel computation
- CNNs can compute the solution in constant time O(1), while LP requires complexity O(n). This is because CNNs are essentially a function.

	LP		CNN		
Game sizes	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

Conclusion

Conclusion

Comparison between LP and CNN

1. A brief review of two-person zero-sum games, saddle points, and linear programming solution methods.
2. A brief introduction of CNNs
3. How to use CNNs to predict saddle point values $\hat{v}$ of two-player zero-sum games.
4. How to find the corresponding strategy profile $(\hat{x}, \hat{y})$ according to the value $\hat{v}$.
5. Numerical results, comparison between our CNN approach and LP.

« Thank you »

Reference:
1. Wu, D., & Lisser, A. (2022). Using CNN for solving two-player zero-sum games. Expert Systems with Applications, (p. 117545).
2. Wu, D., & Lisser, A. (2022). A dynamical neural network approach for solving stochastic two-player zero-sum games. Neural Networks, 152, 140-149.
3. Dantzig, G. (2018). Linear Programming and Extensions. Santa Monica, CA: RAND Corporation. doi:10.7249/r366.
4. Fan, K. (1953). Minimax Theorems. Proceedings of the National Academy of Sciences, 39 , 42–47. doi:10.1073/pnas.39.1.42.
5. Courville, I. G., Bengio, Y., & Aaron (2016). Deep learning. Nature, 29 , 1–73. URL: http://www.deeplearningbook.org.

E-mail : dawen.wu@centralsupelec.fr