Using CNN to solve two-player zero-sum games

Dawen Wu
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**: $A_1 = \{1, \ldots, n\}, A_2 = \{1, \ldots, m\}$
- **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.
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
\[
e_n^T x = 1, x \geq 0, e_n^T y = 1, y \geq 0,
\]
Where \(e_n\) is a \(n\)-dimensional all-ones vector.

Saddle point
Solving a two-player zero-sum game mean solve the following equation
\[
(x^*, y^*) = \arg \max_x \left( \arg \min_y x^T A y \right)
\]
\[
v^* = x^* A y^*
\]
\((x^*, y^*)\) 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).

\[
\text{(P1)} \quad \max v \\
\text{s.t.} \quad A^T x \geq v e_m \\
e_n^T x = 1, x \geq 0,
\]

\[
\text{(P2)} \quad \min v \\
\text{s.t.} \quad A y \leq v e_n \\
e_m^T y = 1, y \geq 0,
\]
Convolutional neural network

Preliminary 2: Machine learning approach
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 $\theta$, 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).
The following example shows how the CNN model recognizes that the input image is the letter "A".

Parameters $\theta$

Input image $(1, 32, 32)$

Output prediction $\phi = [0.98, \ldots, 0.02]$  

True value $v = [1, \ldots, 0]$
**Convolutional neural network (CNN)**

### Training

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

The expected risk

$$L_D(\theta) = \mathbb{E}_D \ell(f_\theta(A), v)$$

The empirical risk

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

When the training set is ready, we can optimize the empirical risk. This minimization of the empirical risk is an unconstrained 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
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**

- **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 divided 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

<table>
<thead>
<tr>
<th>Input: Game size ((m, n)); Probability distribution (P); CNN model net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Generate((m, n, P)):</td>
</tr>
<tr>
<td>1. (A \sim P): sample a matrix game (A) with shape ((m, n)) from distribution (P)</td>
</tr>
<tr>
<td>2. (v^* = LP(A)): Find (v^*) by solving the LP</td>
</tr>
<tr>
<td>3. (b = (A, v^*))</td>
</tr>
<tr>
<td>4. return (b)</td>
</tr>
<tr>
<td>5. end</td>
</tr>
<tr>
<td>Function Train((b, net)):</td>
</tr>
<tr>
<td>6. net (\leftarrow b): Train the CNN model by the sample (b).</td>
</tr>
<tr>
<td>7. end</td>
</tr>
</tbody>
</table>
**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 $\alpha$; Training rounds $K$; Sample size $N$; Iterations number $T$

  **Input**:
  - Game sizes pool: Probability distributions pool

  **Output**:
  - The CNN model

  **Initialize**:
  - net = Net(); $B = []$

```python
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
The CNN model can only give prediction $\hat{v}$ for $v^\star$. 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 corresponding $(\hat{x}, \hat{y})$, according to the $\hat{v}$ and $A$.

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

It can be reduced to **the convex Linear system**

$$A^t \hat{x} \geq \hat{v} e_m$$

$$\hat{y}^t e_m = 1, \hat{y} \geq 0,$$

$$A \hat{y} \leq \hat{v} e_n$$

$$\hat{y}^t e_m = 1, \hat{y} \geq 0.$$
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
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.

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

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:

E-mail : dawen.wu@centralsupelec.fr