Learning Successful Strategy in Adversarial Games

Aditya Sharma, Allard Dupuis, Christopher Dunkers, Mitchell Adam Kosowski IV
Robotics Institute
Carnegie Mellon University

Abstract

One of the earliest problems in Artificial Intelligence has been to make computers become good at playing games. Most of the early work in this area was centred around traditional AI search algorithms like minimax search with alpha beta pruning. This was followed by more "machine learning" centric approaches which enabled agents to learn from experience. In this current work, we propose a technique that first learns the rules of a game and then learns a strategy on top of this by experiencing the game itself, thereby emulating a more "human-learning" centred behaviour. We have used Connect-4 as the game of our choice. This is an interesting technique because (1) it is similar to how humans learn to play games and (2) this method should be generalizable to different games including ones where search and other techniques perform poorly. Hence, with this approach it should be easy to extend the same system to learn to play and win a new game (with a grid-based structure) like Tic-Tac-Toe, Connect 5 etc.

1. Introduction

One of the most studied areas in the field of Artificial Intelligence is teaching computers to play games. Recently, DeepMind has trained a network to successfully play classic Atari video games using high-dimensional sensory inputs with end-to-end reinforcement learning, which is human-like experience-based strategy.[1][2]

Our team was interested in expanding on the work of DeepMind by developing a network that would learn to play a game that is adversarial, specifically playing against an opponent that has the same capabilities as the network and has a winning/losing condition, as opposed to simply trying to maximize a running score.

For our approach, we attempted to create a network that learns to play games similar to how humans learn to play games.[3] That is, first, play the game in a way to best figure out the rules of the game. Next, learn the optimal strategy to become good at playing the game.

We chose Connect Four as the game for this project as it meets both these conditions and is relatively easy to model. Although we have chosen Connect Four as our game, this method should be generalizable to many different types of games including ones where search and other techniques perform poorly.

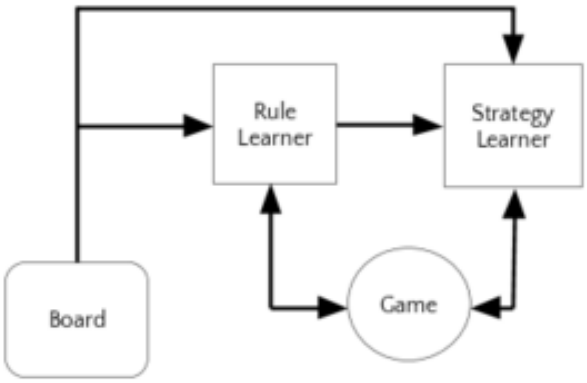

Figure 1. The system architecture with the computational flow.

2. Methodology

The system is split into 2 main phases: The Rule Learner (RL) and the Strategy Learner (SL).This is to encapsulate the idea of human-inspired learning that our technique focuses on. Figure 1 outlines the System Architecture. The RL receives the game board (in this case, Connect4) as an input and starts playing randomly against the game. In this case, the learner is only concerned about making valid moves and not actually winning the game. This again, is how a human would start learning Connect4 when given the game to play for the first time without any prior instructions. Once the system learns the rules of the game, they are given to the SL along with the board as input. The SL then starts playing against the game adhering to the given rules and tries to learn the optimal winning strategy. The following subsections go into further details on how the RL and SL are implemented.

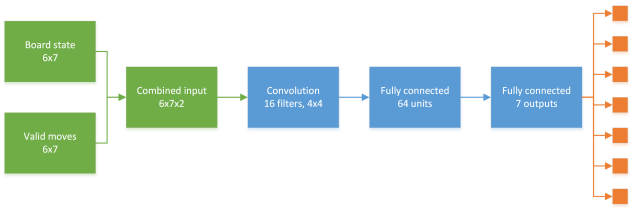

Figure 2. Strategy Learner network structure

2.1. Rule Learner

The Rule Learners task is to return the valid moves given a certain board state. The board is represented by a HXWX3 tensor, where H and W are the board height and width while the third dimension is used to represent the state of each grid cell. This tensor encodes the state of each of the h x w board cells, which can be:

- Empty
- Occupied by player 1
- Occupied by player 2

These states are encoded by the vectors [1 0 0], [0 1 0] and [0 0 1] respectively.

The output is represented by a HXW matrix. Valid moves are indicated by the respective elements in the matrix having been set to 1. Non-valid moves are represented by 0s. The RL network maps the input tensor to the output matrix. The network has an input layer with HXWX3 units, a single 50-unit fully-connected hidden layer with a $tanh$ activation function, and a fully connected $tanh$ output layer with HXW units. More information about the train and test process is given in section 3.

2.2. Strategy Learner

Similiar to DeepMind we used a convolutional neural network to train the Strategy Learner. This network consists of an input layer which takes in the state of the board and the valid moves from the rule learner. This then goes to a single convolution layer with filters of size 4 by 4. From there we enter into a fully connected layer with 64 nodes and finally to the output layer with 7 nodes, 1 for each column. A graphical representation of the network can be seen in Figure 2.

3. Experimental Results and Conclusions

3.1. Rule Learner

We used 2000 randomly initialized board states each for our training, validation and testing datasets. All of these were valid board states and there were no repetitions.

We were able to achieve 100% accuracy (0 Error) on the test set, where the error is the Euclidean distance between the predicted valid moves and actual valid moves (represented as cells containing either a 0 or a 1) averaged over all test set samples.

Figure 3. A state in a game between minimax 3 and our trained network

3.2. Strategy Learner

We trained this network over 150 epochs with 25000 episodes per epoch. This resulted in about 90000 games to train on. We were able to see the strategy network understand the valid moves input from the rule learner. To test the network we played against a random AI and a minimax algorithm with varying depths. The results can be seen in Table 1.

opponent	result
random	95%
minimax 1	win
minimax 3	win
minimax 5	lose

Table 1. Performance of the Strategy Learner against different opponents.

For the random result we would expect that there are situations that would arise that the network was not trained for and would lose. For the other results, against the minimax algorithm, only one game was played because both our network and the minimax algorithm are deterministic.

A minimax at a depth of 10 is considered optimal, as it balances searching the tree and speed of the algorithm, however we found that the network was not able to learn a strategy that could beat the minimax at a depth of 5 let alone 10.

The strategy that the network learned does not always pick the optimal move. For example, in the situations below, it should have picked the winning move, but the network does not seem to learn how to win in all situations. This is demonstrated in Figures 3 and 4.

Looking at these states we can see that the network could have won, however it did not pick the best result, confirming that currently the network is not optimal. However, we were

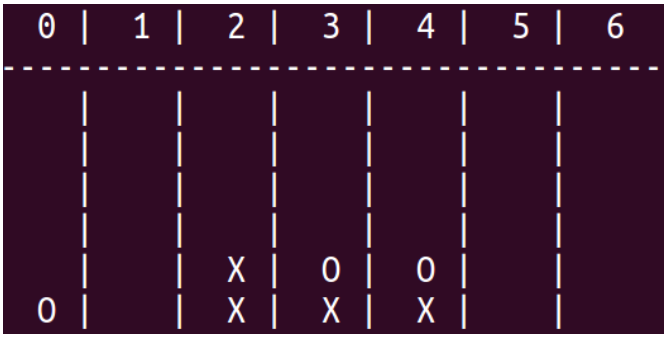

Figure 4. A second state 2 moves later in a game between minimax 3 and our trained network

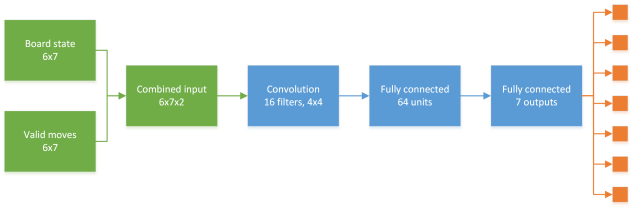

Figure 5. Possible future network structure

able to improve upon random suggest future improvements could yield better results.

4. Future Work

In the future we would like to attempt 3 things, the first being to train on even more games. There are approximately 4.5 trillion board states and we trained over at most 3.78 million. We think that the filters may not have been fully trained resulting in worse performance.

Another step we would like to try is to adjust the network structure. Currently we are passing the valid moves to the convolution layer. We think that merging the valid moves with the output of the dense layer $lout$ allow it to pick better moves faster. This change would make the network structure look like Figure **??**.

This is just another structure that we could try. Additional structures with more layers could be used to create a better network.

Finally, we would like to attempt to incorporate memory. Just like a person will try to manipulate the game to get to a state where they know they can win, we want to add that ability to the network. By giving a board state to the network which is close to our current board state and a state from which we have won before we could train the network to use the memory of past games to improve future games performance just like a person would do.

5. REFERENCES

[1] Mnih, Volodymyr et al. Playing Atari With Deep Reinforcement Learning. 2014.

[2] Mnih, Volodymyr et al. 'Human-Level Control Through Deep Reinforcement Learning'. Nature 518.7540 (2015): 529-533.

[3] Whitebread, David. 'The Importance Of Play'. N.p., 2012.