Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

TOWARDS AI ROBUSTNESS MULTI-AGENT ADVERSARIAL PLANNING IN GAME PLAY

Yan Lu
Sachin Shetty

Virginia Modeling, Analysis and Simulation Center
Old Dominion University
Suffolk, VA, 23435, USA

ABSTRACT

This paper applied Monte Carlo Tree Search (MCTS) and its variant algorithm "Upper Confidence bounds applied to Trees" (UCT) to Chinese checker game and visualized it in the Ludii games portal. Since the MCTS approach cannot guarantee a finite game length in our simulation and the winning strategy is inefficient, we developed a Convolution Neural Network (CNN) model formulated on top of the MCTS algorithm to improve the performance of the Chinese checker game. Our experiment and simulations show that by using the weak labels generated by MCTS algorithm, the CNN-based model could learn without supervised signals from human interventions and execute the game strategy in a finite time. PyGame and the Ludii game portal are used in our simulation to visualize the game and show the final game results.

1 INTRODUCTION

Adversarial games have become widely used for developing and testing the performance of learning agents. They possess many of the same properties of real-world decision-making problems as we want to design the agents (Dignum et al. 2009). The Chinese checker differs from other traditional games in two main aspects: first, all checkers remain indefinitely in the game, and the branching factor of the search tree does not decrease as the game progresses; second, there are also no upper bounds on the depth of the search tree since repetitions and backward movements are allowed. As shown in Figure 1, there are at least ten rows of spaces a piece must traverse before landing at the goal. The board's geometry is not square but rather hexagonal, with six stars on each segment of the hexagon. The spaces are not perpendicular but at angles to each other, meaning movement to the final destination is not direct. The allowed moves include rolling and hopping. Rolling is moving one checker in any adjacent space, hopping is jumping over a neighbor piece and landing on a vacant space. Jumping over multiple neighbor pieces continuously is allowed. Due to the fact that the pieces can "jump" over one another, the positioning of the pieces is crucial for enabling friendly pieces to quickly traverse the board by hopping and obstructing the movement of the opponent's pieces. Even though the moves and interactions among the players are simple, the game's strategy is still complex. Monte-Carlo Tree Search (MCTS) (Liu et al. 2019), Upper Confidence bounds Trees (UCT) (Wang et al. 2018), and Reinforcement Learning (RL) (He et al. 2016) are popular algorithms had been applied to solve this game to determine the most optimal solution. Due to the high computational complexity of the Chinese Checker Game, no established record shows that this game is "resolved" where the game's outcome can be predicted if all players play correctly to achieve their winning goals (Yisi et al. 2020).

In this paper, we applied Monte Carlo Tree Search (MCTS) and its variant algorithm Upper Confidence bounds applied to Trees (UCT) to Chinese checker game and visualized it in the Ludii games portal (Piette et al. 2020). To improve the game results, a Convolutional Neural Network (CNN) in conjunction with MCTS and UCT approach is introduced in our paper. This research aims to develop an AI model which

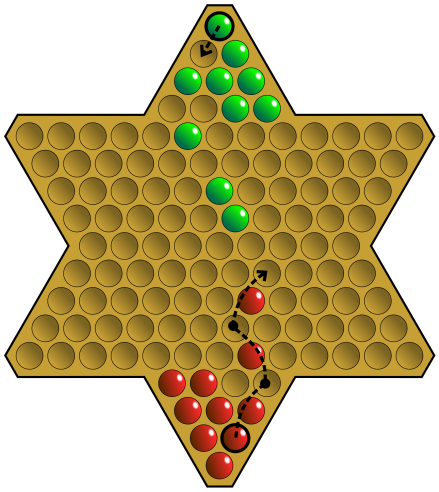

Figure 1: GUI of Chinese checkers game with two players (Wikipedia).

uses the weak labels generated by MCTS and learns the game strategy to complete the game in finite time in a self-supervised learning manner. We applied the algorithm to a three-player version of the game and simulated and visualized the game in Pygame.

2 BACKGROUND

2.1 Chinese Checker and Ludii Game Portal

Chinese checkers is a board strategic game. It has German roots and can be played by two to six players, either individually or with a partner (Morehead 2001). The object of the game is to be the first to use rolling or jumping to get all of your checkers to the corner of the game board opposite your starting corner. With ten pieces each, each player uses strategic positioning and moves to "jump" across the board. If two pieces are near each other, the piece next to them may jump and keep jumping. The remaining players carry on the game in the same manner. In a game with three players, each player will begin in one of three triangles equally spaced apart. Play begins in two pairs of opposing triangles if there are four players, and a two-player game should also be played from opposing triangles. We create and test our Chinese checker algorithms in the Ludii portal (Piette et al. 2020). Ludii portal is a generic game system created to play, rate, and create various games, such as board games, card games, dice games, and mathematics games, enabling the entire variety of traditional strategic games from around the world to be depicted in a single playable database. Games in the Ludii gaming system are structured groups of ludemes (units of game-related information). As shown in Figure 2, it depicts the Chinese checker game interface in Ludii, and the Chinese Checkers has a board with a total of 116 spaces and a maximum piece capacity of 60.

2.2 Game Learning Strategy

This learning strategy aimed to find a function that takes a board state and generates a probability distribution of moves that represents the likelihood that a certain move is the best. In other words, the primary problem can be stated as follows: Given a function f with parameters θ, how can we find parameters $\theta^{'}$? Specifically, we can create a probability distribution for each move to calculate a new probability distribution. Artificial neural networks (ANNs) could be used to find the best θ' due to their ability to be a highly general function. This means a trained neural network could classify data from the same class as the learning data it has never seen before. However, to train a neural network, it needs supervised signals from the "teacher" signals and let the neural network learn if it outputs the correct answer. One way to find the "teacher" signals is to try every possible move several times, then generate moves based on f until the game ends.

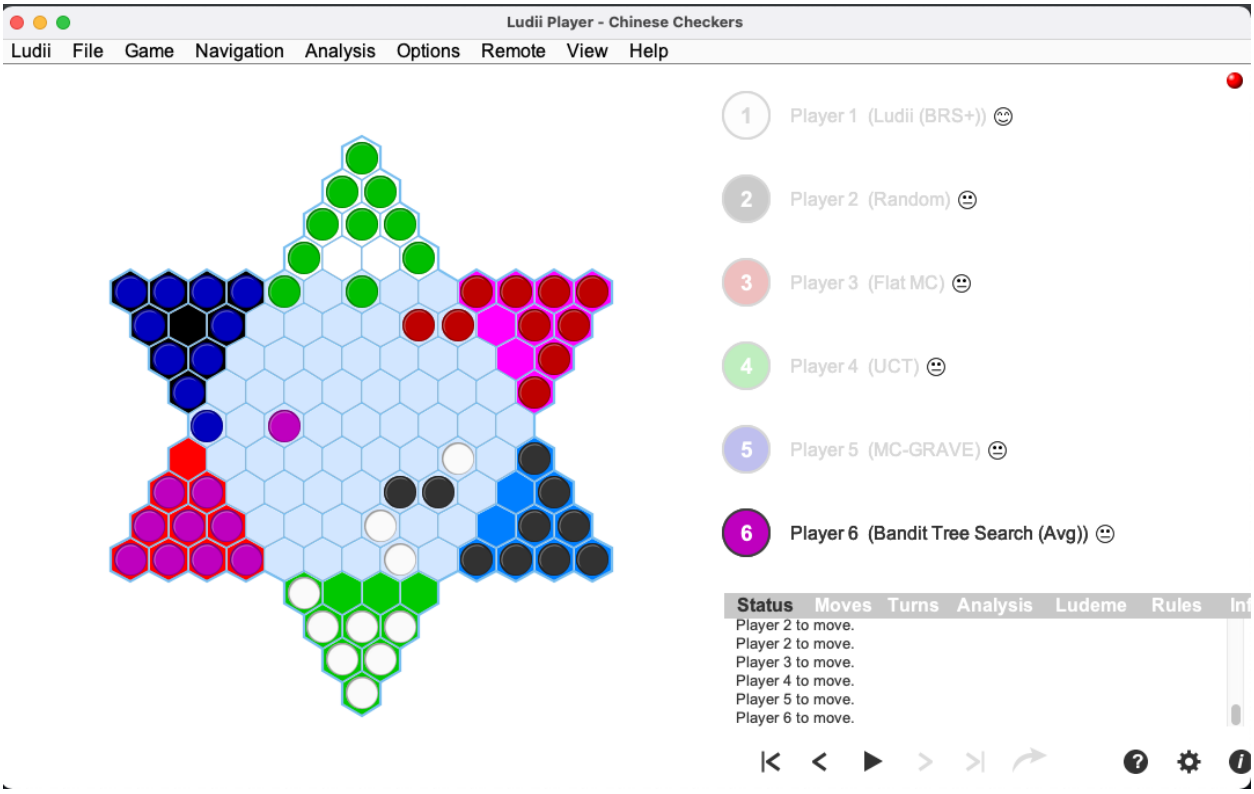

Figure 2: Ludii game portal.

In highly complex exponential systems with enormous actions and state space like in the game of Chinese checker, it is often impractical to analytically determine the most optimized move to be made. To mitigate this issue, given the law of large numbers from statistics, the more random trials performed, the more accurate the approximated quantity will become. Monte Carlo methods defer to simulations of potential outcomes and collect multiple samples from approximating the desired quantity. Each simulation in Monte Carlo methods utilizes elements of "randomness" and provides distributions of results to identify the most optimal deterministic solution to a problem (Bechhofer et al. 1991). Thus, through many simulations, the algorithm trends to increasing precision because the resulting distribution is getting more clearly identifying the solution. However, the computational complexity of evaluating all possible simulations is often limited. Given the central limit theorem, the distribution of the samples will form a normal distribution, the mean of which can be taken as the approximated quantity and the variance used to provide a confidence interval for the quantity (Cheng and Kleijnen 1999). To enhance the quality of simulations provided by the Monte Carlo method, the application of the Markov Chain Monte Carlo (MCMC) sampling model is formed by a possible sequence of events in which the probability of the event is determined by the state achieved in the previous event. This form of simulation manifests itself in Monte Carlo modeling as an equilibrium distribution, and the simulations of a variable are done to create unique distributions. Each simulation utilizes the information of the accuracy of the previous simulation (empirical mean) and modifies the parameters of the simulation to tend towards the equilibrium distribution. If a problem has a probabilistic interpretation, the probability distribution of a variable can be parameterized, and an MCMC sampler can be utilized. Random states of the samples derived from MCMC can be utilized to approximate a stationary distribution. The ultimate application of the Monte Carlo method in simulation is for optimization in finding the best element or solution based on many sets of simulated elements/scenarios. The Monte Carlo Tree Search (MCTS) with Markov Chain Monte Carlo (MCMC) sampling could be used to generate many random possible simulations of possible moves and determine the ideal move for the player. Therefore,

we use the MCTS approach to collect the weakly labeled training data to train the neural network model for better generalization of the game strategies, the workflow is as shown in 3.

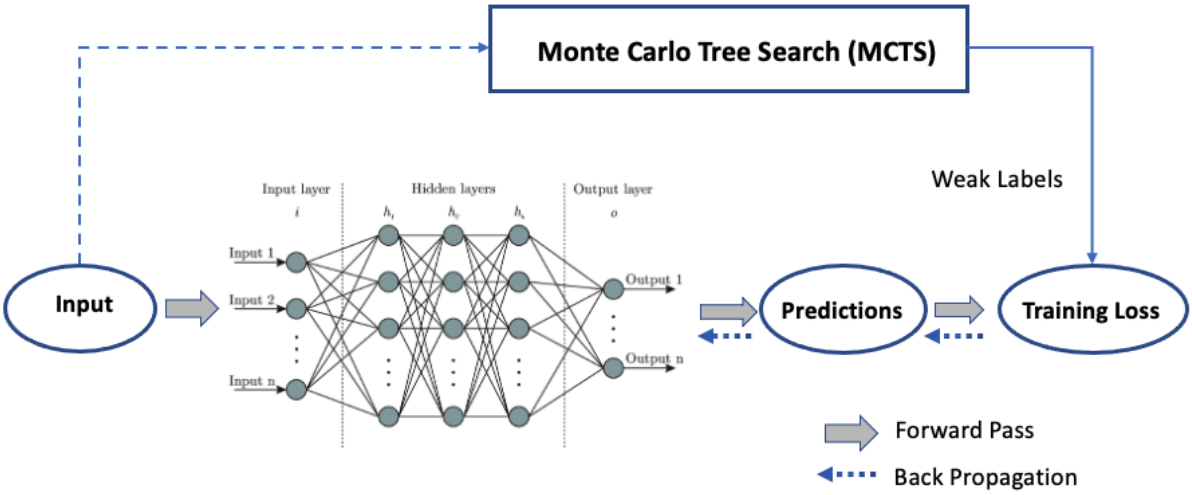

Figure 3: MCTS+CNN: Weak/self-supervised learning.

3 METHODOLOGY

3.1 MCTS with Upper Confidence Bounds Applied to Trees (UCT)

Monte Carlo Tree Search works in four phases. There's a tree traversal phase using Upper Confidence Bounds for rewards that are either 0 or 1 (UCB1) formula (Auer 2002). Then there's a node expansion phase where you add extra nodes to the tree. This is followed by a rollout phase where you do a random simulation of the game or the problem you're solving to find a value. As shown in Figure 4. Then there's a back propagation phase where you take the values found from the rollout, and it is put in appropriate places in the tree. Here is how it works in each phase:

Selection and Expansion: The selection starts at a node or state considered the root, then selects a child node to move to. The selection is based on the Upper Confidence Bounds (UCB1) formula. The selection steps are as follows:

- Step 1. Start with S_0 as the initial state is the current state.
- Step 2. If the current node is not a leaf node, explore the child nodes of the current node by calculating the UCB1 value of each of the states S_i and then choose the one that maximizes the UCB1 value as the current node to explore. The UCB1 value is calculated as :

$$UCB1(S_i) = Mean(V_i) + C \times Sqrt(Ln(N)/n_i,$$

  S_i: Current given state, V_i: average value of that state, N: parent visits, n_i: visits of this state, C is the constant used for fine-tuning.
- Step 3. Repeat step 2. until getting to a leaf node in the tree.
- Step 4. If the leaf node in step 3 has never been sampled before, simply do a rollout form that leaf node to find a terminal state with value; if it has been sampled before, then add a new child node

for all actions available at the current node into the tree, then move to the first of the child nodes and begin a rollout.

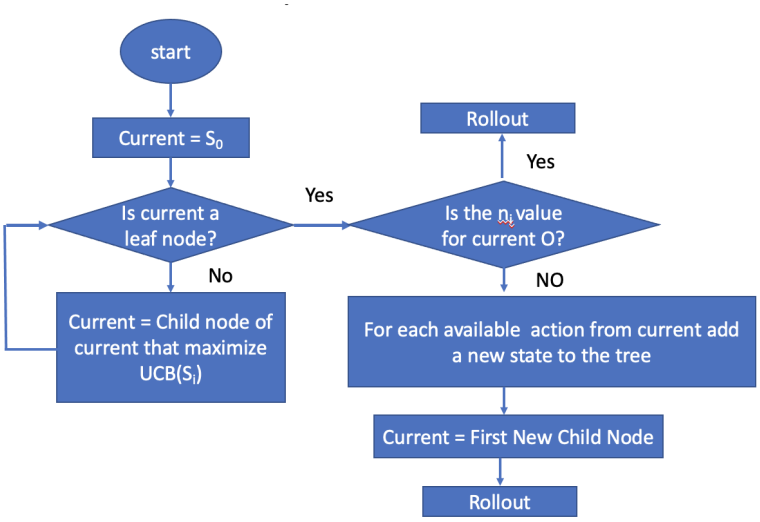

Figure 4: Tree traverse, expansion and rollout.

Rollout or Simulation: The rollout or simulation is the phase in which random actions are taken, retrieve the landing state, and then take another random action to land in a new state. This process is iterated until a terminal state is reached. At that point, the value of the terminal stated is returned. The pseudocode of the procedure is explained below and depicted in Figure 5.

```
Procedure Rollout(S_i):
    Loop Forever
        If S_i is terminal state:
            Return the value(S_i)
        Else:
            A_i = Random (available.actions(S_i)
            S_i = Simulate (A_i, S_i)
```

Backpropagation: The backpropagation gets the value of the rollout and updates the nodes from the start of the rollout till the root node. The update consists of adding the rollout results to the current value of each node and increasing by one the count of visits at each of these nodes, as shown in Figure 6.

3.2 Game Model Formulation

In this study, a smaller six-piece variation is employed as an experiment to reduce processing complexity instead of playing with ten pieces. The board size before the transformation is 13 by 17 square with the original ten pieces; the board size turns 10 by 13 with six pieces after the transformation. A piece can traverse is restricted by the board size: 10 units in the horizontal direction (x_space) and 13 in the vertical direction (y_space), Any location a piece can move would be restricted to be inside the playable game board. To represent the board in a program, we use a 10×13 matrix, called the game matrix, in which the value 0 represents an empty slot, and value 1 represents the player's checkers. We use three game matrices

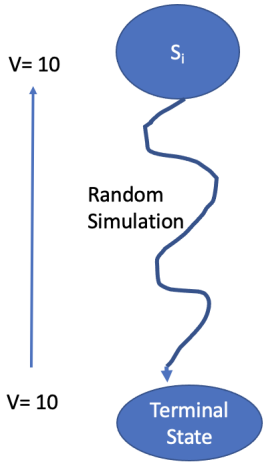

Figure 5: Rollout or Simulation.

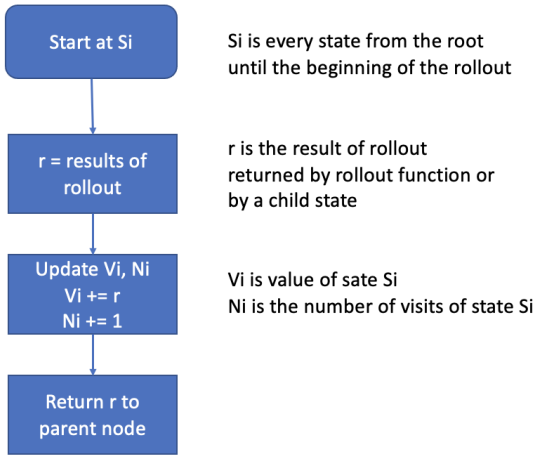

Figure 6: Backpropagation.

to represent three players. In applying the Monte Carlo method, we implement the following pseudocode to simulate a simulation in the process that determines the possible moves. When reaching the ending states and if there are only two moves between the player and the goal, a simple depth-first search will be used instead of counting the sum of the taxicab distance to get valid moves. The pseudocode listed below is for the simulation of finding new game states.

The pseudocode legit_moves and simulate_moves listed below show that assigning win rates to different game state outcomes when applying the Monte Carlo Tree Search. Firstly it generates a specific win ratio for each possible move a piece can make, and then the algorithm chooses a move that maximizes its chance of winning the game. The delta terms represent the change in position from each move. Each possible move the player could make is designated as a node object in the "tree search." For each node, a Markov Chain simulation is conducted, and the tree nodes are explored until they reach a final state. If it is a win for the algorithm, it transfers that information as a property of that node going back up the tree to the initial move to indicate that it has been successful. Through many simulations, the distribution of win rates tends to be the true mean for a certain move. The total explorations into a node will be recorded, and the win

rate will be determined accordingly. Additionally, the initial step in each simulation iteration to determine which node branch to explore will be determined by the highest win rate node. The random element enters as for any move that does not have a win rate assigned to it, a random value will be generated, and the highest value will be chosen to be explored.

```
Procedure Legit_Moves
    Array moves = function simulate_moves
        If  x_space < 10 and y_space <  13 Then
            For i:= 0 to moves/2 do
                If move < board Then
                    If EMPTY Then
                        set move to array
                    If OCCUPIED Then
                        Array new_moves = function simulate_moves
                            For j:=0 to new_moves/2
                                set move to array
    end
```

```
Function  Simulate_moves
    If  deltaY  +  1  mod  2 is 0 Then
         If  deltaX  mod  2  is 0 Then
            moveY  =  predictY  +  ((deltaY  +  1) % 2)
         If  deltaX  is  not  divisible  by  2  Then
            moveY  =  predictY  -  ((deltaY  +  1) % 2)
    If  move  <  board  Then
         If  EMPTY  Then
            set  move  to  array
            For  i := 0  to  4  do
                If  OCCUPIED  Then
                    Array  moves  =  function  simulate_moves
                    For  j := 0  to  moves / 2  do
                        set  move  to  array
    return  array
```

3.3 Optimizing Results using Convolutional Neural Netowrk

Convolution Neural Network approximates the function *f*, and we use it to simulate the game strategy. To encode the input for the CNN network, we convert the game board array from the size of 10x13 into a 13x13 input frame with 1 where there is a piece for a given player, -1 for the filler places, and 0 for everywhere else. There are three players, so the input is 3x13x13. The output of the neural network requires that we choose a piece to move and a location to move it to. Additionally, to specify which piece moves to which location, for six pieces, there are six 13x13 arrays. The correct moves were generated using a basic depth-first search on the 13x13 grid. Additionally, to distinguish which piece in the input corresponds to which layer in the output, another six 13x13 arrays need to be added to the input that corresponds to the locations of each player piece. The input is a 9x13x13 matrix, and the output is a flattened vector of a 6x13x13 matrix.

As shown in Figure 7, the game state is mapped as an array with dimensions of 9x13x13 input into the neural network, followed by the input layer, a convolution layer with 64 filters with the size of 5x5 is added. Followed by the first convolutional layer, there is a batch normalization layer and rectified linear unit (ReLu) layer. Then the second, third, and fourth convolutional layers are added. In these convolutional layers, the filter sizes are all 3x3 but in different numbers of channels: 32, 16, and 16. There is no pooling layer in between the convolutional layers. After this, each of the six input layers went through a convolutional+batch+ReLU layer and then were flattened to go through a linear layer to get the output. The layers were all processed separately because the indices of the other pieces are irrelevant to determining the goodness of a move for a specified piece. Finally, the six output layers for each piece were concatenated, and the softmax function was applied to ensure they were all positive and summed to one.

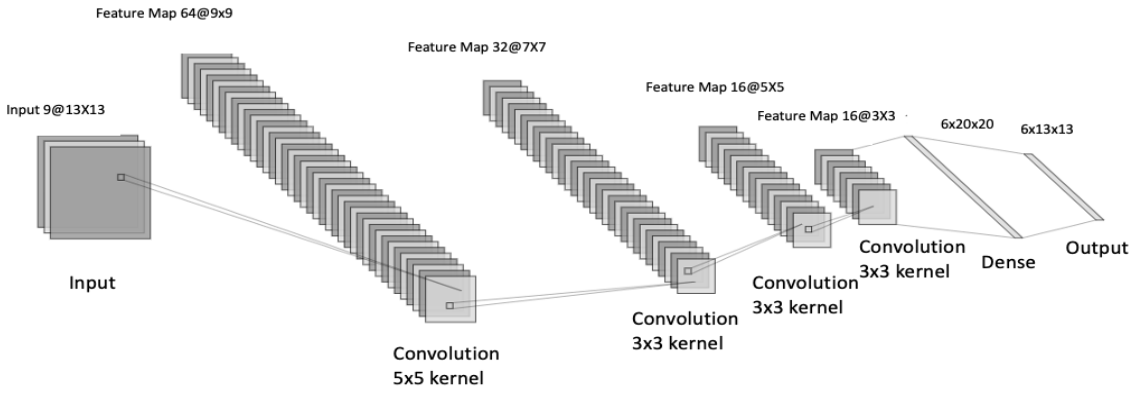

Figure 7: Convolution neural network structure.

4 EXPERIMENT RESULTS

Pygame is used to simulate the game. At first, We train the neural network for 3, 6, and 12 hours without using MCTS to search for the best moves; the best moves are as long as each piece of the player is moving towards the goal positions and the taxicab distance to the goal position of that piece is decreasing. We save the networks as baseline models CNN-3, CNN-6, and CNN-12. Then, we transfer the weights of the baseline model CNN-3 and train the neural network based on the pre-trained CNN-3 model without MCTS for 6 and 12 hours. We then saved these networks as pre-trained models CNN-6p and CNN-12p. At last, we train the CNNs using MCTS searching for the best moves on top of the pre-trained models of CNN-6p and CNN-12p for 6 and 12 hours. We saved these networks as CNN-6MCTS and CNN-12MCTS. To evaluate the results, we let a random network play against CNN-6 and CNN-12 for 300 rounds; let the same random network play against CNN-6p and CNN-12p for 300 rounds, and let the same random network play against the CNN-6MCTS and CNN-12MCTS for 300 rounds. The experiment results are shown in Table I. From the result, it shows that with the 3-hour pre-trained model, the CNN's performance is improved than training the CNNs model from scratch. Furtherly, using MCTS to fine-tune the CNNs, significantly improved the performance of the CNNs model, and it gives the best results. The playing time is significantly reduced within the reasonable time limit.

5 CONCLUSION AND FUTURE WORK

In this paper, we implemented a Convolution Neural Network on top of the Monte Carlo Tree method to improve the convergence of the Chinese Checker game. Our simulation results show that the CNN-based model could learn from the weak label generated by MCTS, implement weak/self-supervised learning

Table 1: Experiment results.

	Wins		
Network	**Random**	**CNN-6***	**CNN-12***
CNN-*Baseline**	49	75	176
CNN-*pretrained**	63	89	148
CNN-*MCTS**	29	75	196

without human interventions and execute the game strategy in a finite time. We also applied the pre-trained CNN model in different scenarios to compare its performance to reduce the playing time. To advance this research in the future, we will consider implementing more players on top of our existing model to furtherly refine the simulation results.

ACKNOWLEDGMENTS

This work is supported by the Air Force Research Lab while the author was visiting as visiting research faculty in the Visiting Faculty Research Program and mentored by Dr. Benjamin Ritz. This work is also partially supported by funding from the Commonwealth Cyber Initiative (CCI).

REFERENCES

Auer, P. 2002. "Using Confidence Bounds for Exploitation-Exploration Trade-Offs". *Journal of Machine Learning Research* 3(Nov):397–422.

Bechhofer, R. E., A. J. Hayter, and A. C. Tamhane. 1991. "Designing Experiments for Selecting the Largest Normal Mean when the Variances are Known and Unequal: Optimal Sample Size Allocation". *Journal of Statistical Planning and Inference* 28:271–289.

Cheng, R. C. H., and J. P. C. Kleijnen. 1999. "Improved Design of Queueing Simulation Experiments with Highly Heteroscedastic Responses". *Operations Research* 47(3):762–777.

Dignum, F., J. Westra, W. A. van Doesburg, and M. Harbers. 2009. "Games and Agents: Designing Intelligent Gameplay". *International Journal of Computer Games Technology* 2009(837095):18.

He, S., W.-B. Hu, and H. Yin. 2016. "Playing Chinese Checkers with Reinforcement Learning". Technical report, Stanford University.

Liu, Z., M. Zhou, W. Cao, Q. Qu, H. W. F. Yeung, and V. Y. Y. Chung. 2019. "Towards Understanding Chinese Checkers with Heuristics, Monte Carlo Tree Search, and Deep Reinforcement Learning". https://arxiv.org/abs/1903.01747, Accessed October 15, 2022.

Morehead, A. H. 2001. *Hoyle's Rules of Games*. Berkley.

Piette, É., D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. 2020. "Ludii – The Ludemic General Game System". In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, edited by G. D. Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, Volume 325 of *Frontiers in Artificial Intelligence and Applications*, 411–418: IOS Press.

Wang, Y., Z. Yang, H. Qiu, and X. Liu. 2018. "Application and Improvement of UCT in Computer Checkers". In *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*. November 23rd-25th, Nanjing, China, 274-278.

Wikipedia. "Chinese checkers". https://en.wikipedia.org/wiki/Chinese_checkers. Accessed October 15, 2022.

Yisi, W., M. N. A. Khalid, and H. Iida. 2020. "Analyzing the Sophistication of Chinese Checkers". *Entertain. Comput.* 34:100363.

AUTHOR BIOGRAPHIES

YAN LU is a Research Assistant Professor of Virginia Modeling, Analysis, and Simulation Center at Old Dominion University. She holds a Ph.D. in Modeling and Simulation, and her research interests lie in Deep Learning, Machine Learning, and Trustworthy AI. Her email address is y2lu@odu.edu.

SACHIN SHETTY is a Sachin Shetty is an Executive Director for the Center of Secure and Intelligent Critical Systems in the Virginia Modeling, Analysis and Simulation Center at Old Dominion University. He holds a joint appointment as a Professor

with the Department of Computational, Modeling, and Simulation Engineering. His research interests lie at the intersection of computer networking, network security, and machine learning. His email address is sshetty@odu.edu.