Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

Mastering the game of Hive with Supervised Learning and Monte Carlo Tree Search

Dang Dang DANG05@ADS.UNI-PASSAU.DE
Fakultät für Informatik und Mathematik
University of Passau
94032 Passau, Germany

Editor: Dang Dang

Abstract

The game of Hive is a two-player abstract strategy game that has gained popularity in recent years due to its simple rules and challenging gameplay. However, despite its growing popularity, the development of an AI model capable of playing Hive at a high level has been limited. In contrast, the AlphaGo Zero and AlphaZero program recently achieved superhuman performance in the game of Go, Chess and Shogi, by tabula rasa reinforcement learning from games of self-play. In this paper, we apply a similar genetic version of these approaches to able to successfully create a strong-performance neural network with low-cost training resources. Starting from Supervised learning models trained on expert datasets, then combine with reinforcement learning using Monte Carlo Tree Search, our methods achieved a human-level performance in the game of Hive.

1. Introduction

Reinforcement learning is one of the most popular machine learning methods used in Artificial Intelligence (AI). It enables agents to make a sequence of decisions by learning from interactions with environment and being rewarded for successes. In details, agents are provided with all information regarding environment and choices. After making decision, they are informed feedbacks or rewards about how well the actions were taken. Therefore, reinforcement learning is considered to be closed to human learning process and agents are targeted to be as intelligent and decisive as a human being.

In gaming sector, computers trained by deep learning neural networks from reinforcement learning have demonstrated superiority over humans. One of the renowned examples is AlphaGo[4], the first program to achieve superhuman performance in Go using the combination of Supervised Learning and Reinforcement Learning. Or AlphaGoZero[5], the next generation of AlphaGo, with only trained with Reinforcement Learning using Monte Carlo Tree Search, be able to beat AlphaGo in only 24 hours of training[5]

However, there are only few games where human world champions have been beaten. Other games which entail complex rules are considered challenging for AI to mimic and beat human players. Besides, it also requires years of domain knowledge engineering and millions of dollars investing in computing resources to become successful. The state of AI in Hive board game leaves spacious room for improvement.

Hive is an addictive board game of two players developed by John Yianni in 2001[8]. Each player has 11 pieces that can move freely and to win over the opponent, one must surround the opponent's, Queen Bee. Hive has gained popularity not only from those who enjoy playing it but also from AI developers who have been attempting to develop a neural network capable of playing the game at a human level. However, the results still have some limitations due to expensiveness of getting expert data sets and an enormous amount of state space. As both players have the freedom to decide which moves to take, the state space grows exponentially with the number of moves, which is substantially large to handle in a reasonable time.

Similar to the development of AI player in Go and chess game where there are millions of data points and searches to dealt with and also millions of dollar investing in computer and engineers resources. Therefore, the paper will provide a better approach to the development of an AI for Hive that combines supervised learning and Monte Carlo Tree Search (MCTS). The goal of this research is to demonstrate how these two techniques can complementarily work together to create an AI that can play Hive at a high level and is superior to existing models. After several times revising and training the model, our model achieved 90% winning rate in self-play games.

2. Datasets

In order to train the AI model for Hive with limited training hardware [9], we required a large dataset of human gameplay for Supervised learning of policy networks. This dataset was collected through a combination of recorded games of human vs human, and human vs bot on "boardspace.net" game host platform. The human play was used to generate a set of high-quality moves and positions for the AI model to learn from, while the bot play provided a more diverse set of states, and moves for the AI model to learn about the environment, game rules, and movement of the pieces.

The expert play was performed by experienced players of Hive who were familiar with advanced strategies and tactics. These players generated a large number of high-quality moves and positions that were used to train the AI model. The recorded human games were obtained from online sources, such as the Hive community forums, and were selected based on the quality of play and the diversity of strategies and tactics used.

Our expert dataset came from "boardspace.net" [1] archive, which is a public and free board game host platform. The Hive dataset consists of a total of over 250,000 matches starting from early 2006 up to the present. However, in most of the dataset, over 150,000 matches are expansion versions of Hive [8] (with 3 more extra pieces set: Pillbug, Ladybug, and Mosquito) and the rest matches are normal Hive versions with 11 pieces set[8]. In the paper, we focus only on developing the policy network for the normal Hive version, as a limitation of our computing resources [9], we want to keep the game of Hive simple.

The normal Hive version on "boardspace.net" [1], comes with a board size of 26 by 26 (width and height) and is a total of around 46,700 matches. However, we can not fit the policy network with the input of board size 26 by 26, into our training machines with 11 Gigabytes of Vram [9]. As the result, for our Hive game, we need to limit the board size down to 12 by 12. According to (Figure 1), the number of matches that play inside the board size of 12 by 12, is over 41,200 matches.

With the board size limited to 12 by 12, we have around 8,000 matches of true expert matches, games that were played between humans vs humans. The rest of the matches were played between humans vs Min-Max Alpha-Beta [10] bots (Figure 1). The number of matches when bots played as black position is double compared when bots played as white position. As the result, our policy networks performed worse when played as black position.

For our Reinforcement learning policy networks with Monte Carlo Tree Search, we need to set the maximum number of turns of each episode, as this would speed up our running time during the training process. From the "boardspace" [1] dataset, the maximum match length is around 35 turns, and the percentile is under 60 turns (Figure 1). As the result, we selected 35 turns would be the maximum game length.

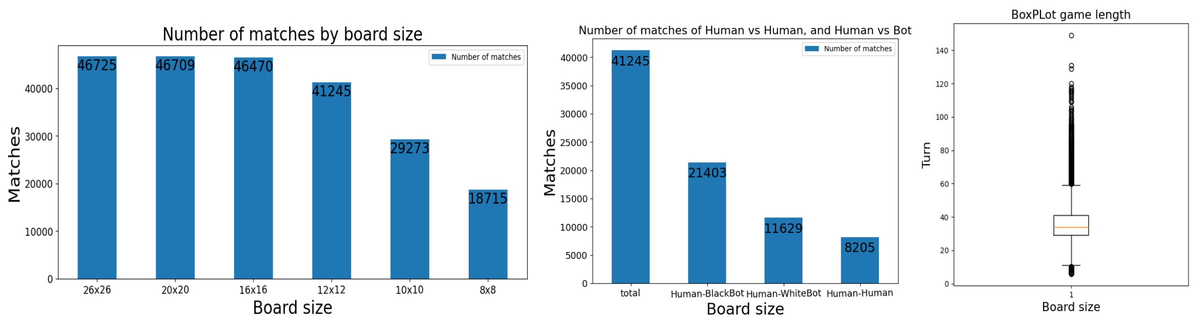

Figure 1: Self-play evaluation the winning percentage of 1000 matches playing again the policy network ρ_θ^1 (SL1). **a** Performace of all SL policy networks. **b** Performace of all RL policy networks with and without MCTS when selecting actions.

3. Methods

The game of Hive shares several similarities with chess, multi pieces kinds, dynamic movements, and trapped pieces. Recently, the AlphaGo Zero[2], AlphaZero[5] algorithm achieved superhuman performance in the game of Go and Chess, by representing Go knowledge using deep convolutional neural networks [2],[3], or trained solely by reinforcement learning from games of self-play [4],[5]. In this paper, we will apply a similar method in AlphaGo and AlphaZero, combining the supervised learning network trained from expert data and reinforcement learning algorithm with Monte-Carlo tree search (MCTS).

Supervised learning with expert data

For speeding up the run time of reinforcement learning with MCTS, in first stage of the training pipeline, we build a network on predicting expert moves in the game of Hive using Supervised Learning. The SL policy network $\rho_\theta(p, v|s)$ architecture is the combination of convolutional and recurrent layers with parameters θ. With the final softmax layer outputs a probability distribution over all legal moves a and tanh layer output for the value v.

$$p, v = \rho_\theta(s). \tag{1}$$

This neural network takes the board position s as an input and outputs a vector of move probabilities p with components $p_a = Pr(a|s)$ for each action a, and a scalar value v estimating the expected outcome z from position s, $v \approx \mathbb{E}[z|s]$.

The input s to the policy network is a representation of the board state. Each position s is described by a sparse vector of handcrafted features $\phi(s)$, including the position of individual pieces on the board, blocked pieces, pieces surrounding Queens, current turn number, and history move of the previous steps. (see detail Extended Data Table 1).

The parameters θ of the SL policy network ρ_θ are trained by target labels from expert datasets. For each step, the probability weight of action a taken by humans will be equal to 1: $p^*_a = 1$; and for action a taken by bots will be equal to 0.24: $p^*_a = 0.24$. At the end of the game, the terminal position s_t is scored according to the rules of the game to compute the game outcome z: 1 for a loss, 0 for a draw, and +1 for a win. For the rest state from s_0 to s_{t-1}, the expected outcome z will be equal to the next state multi with the discounted factor $\gamma = 0.99(2)$.

$$z_t = \gamma z_{t+1}. \tag{2}$$

We trained an SL policy network ρ_θ with 1 layer of convolutional with 256 channels and 19 residual blocks, from 1.5 million steps from "boardspace" game records with the limited map size setting of $12X12$. Starting from randomly initialized, the neural network parameters θ are updated so as to minimize the error between the predicted outcome v_t and the game outcome z and to maximize the similarity of the policy vector p_a to the expert probabilities p^*_a. Specifically, the parameters θ are adjusted by gradient descent on a loss function l that sums over mean-squared error and cross-entropy losses respectively.

$$l = (z - v)^2 - p^* \log p. \tag{3}$$

We have a total of 4 different SL policy network ρ_θ which was trained on different parameter, features and dataset. The network ρ^1_θ with full features, trained on the full expert data is our current best-performance SL policy network. The network ρ^0_θ is similar to ρ^1_θ without the history features set (Table 1). The network ρ^2_θ with full features is trained only on the human player dataset with the bot data removed.

The policy network ρ^3_θ with full features is trained on the rotation dataset. In the game of Hive, the location of the first step is always the center of the maps. For the second step, the location of the placed pieces would be 6 tiles around the center hexagonal. As the hexagonal is symmetrical, we rotate the state of the map to only exactly one tile adjacent to the center tile. This process will lower the computation space of the state input and act as a data augmentation method to generate more data. However, when testing the policy network ρ^3_θ do not gain performance compared with ρ^1_θ.

Actions and States representation

We limited the baord size to 12 x 12 to reduce state space so that model can be completed within a reasonable time and fit well with computer capacity. As each player has 11 pieces, there are 12 x 12 x 11 = 1584 actions in total. To capture all possible movements, it is

All Feature	
Feauture	Planes
P1 location	23
P2 location	23
History	8
Turn numbers	1
P1 and P2 locations	1
Total	56

In 23 Feature	
Feauture	Planes
11 pieces location	11
all player pieces	1
stack beetles	3
Pieces around Queen	1
trapped pieces	1
pieces move to your Queen	6
Total	23

Table 1: List features of State presentation

important to create different layers for each piece's position, even though they are the same kind of creature. We created 46 features to record what happens with 2 players during the game set, 23 for each which include 11 features representing 11 positions of 11 pieces using one-hot encoder method. To describe the current environment to adjust the policy and select optimal move decision, 1 feature was used to capture the positions of all pieces of one player appearing currently on the board. For example, when there are only 2 pieces on the board, the agent has the tendency not to move the Ant.

As there are 4 Beetles in each player's inventory, there is a likelihood that all of them will stay on top of the other 3 pieces. Therefore, 3 features were used to record the stacked layers of Beetle pieces. From the game rules, one of the pieces might be blocked by the Beetles, we created 1 feature to record the state when it occurs. Regarding Queen Bee' positions, 1 feature was generated for the adjacent tiles that contain the Queen and we also included 6 features representing a place where a piece can move into empty for Queen adjacent tiles. To ensure that the current environment is accurately reflected in the model, we used 1 feature to reflect the order of the current turn and 1 feature for the current positions of all pieces on the map. Finally, there are 8 features that were generated in the model to record history movements in the previous steps. In total, as there are 56 features created, the state presentation is the matrix of size 12 x 12 x 56.

Reinforcement learning of policy networks

For the second stage of the training pipeline, we applied a similar genetic AlphaZero algorithm [5], a policy gradient reinforcement learning (RL) [6],[7] to improve the SL policy network ρ_θ. The architecture of RL policy network ρ_ω is identical in structure to ρ_θ and its parameters ω are initialized from parameters θ of the policy network ρ_θ^1. Then the parameters ω are trained from games generated from the self-play processes when letting the ρ_ω play with itself using MCTS. In each position s, the MCTS search is executed, guided by the policy network ρ_ω. The MCTS search output a new search vector π representing a probability distribution over moves for each state s. These search probabilities usually select much stronger moves than the raw move probabilities p of the neural network $\rho_\omega(s)$[4]. Games are played by selecting moves for both players by MCTS, $a_t \approx \pi_t$. And in the end game state similar to SL process, we collect the game outcome . One key difference compared to both SL process and Alphazero methods[5], we set up the maximum length of each game and the outcome value = 0, when the games reached the maximum turn. This is for helping speed up the reinforcement learning run time. And the maximum length value is $max\_length = 55$ for the main game and $max\_length = 95$ for the MCTS processes. This difference is because we want the search probability π does not effect by the $max\_length = 55$.

Inside MTCS processes, Each edge (s, a) in the search tree stores a prior probability $P_{(s,a)}$, a visit count $N_{(s,a)}$, and an action value $Q_{(s,a)}$. Each simulation starts from the root state and iteratively selects moves that maximize upper confidence bound $(1 - c)Q_{(s,a)} + cU_{(s,a)}$ where $U_{(s,a)} = P_{(s,a)}/(1 + N_{(s,a)})$, and $c \in (0, 1)$ is control variable until a leaf node $(s^{'})$ is encountered [4, 12]. Another key difference between our approach and Alphazero, we add the reward function to the $Q_{(s,a)}$ (equation 4), helping to guild the MCTS to state which the policy network has opponent's Queen been surrounding more than its Queen pieces. Similar to the SL process, the policy vector p_a to the search probabilities π_a.

$$r_t = \frac{\sum opponent\_Queen\_surrounded - \sum your\_Queen\_surrounded}{4}. \tag{4}$$

For the list of the RL policy networks that we had trained from the based SL policy networks ρ^1_θ. The first iteration of training with a total of 16,000 games of MCTS, 50 simulations per move, is the policy network ρ^1_ω, which is our latest best performances network. The second iteration ρ^2_ω policy, with ρ^1_ω based networks, trained for another 16,000 games of MCTS, 50 simulations per move. The ρ^3_ω, ρ^4_ω and ρ^5_ω policy networks, are the continuous next iteration of training with, 3000 games of MTCS, 250 simulations per move.

4. Evaluation and Results

Evaluation of our Hive policy network comes with tremendous challenges. Even gained popularity in recent years, the game of Hive has a minimal number of players and research to develop artificial intelligence for Hive compared with well-known games like chess. As the result, we do not have public ELO-ranked bots for our models to play with.

Our best approach methods for evaluating the policy network are to let the policy networks play with their previous version and play with us. For self-play evaluation, both SL and RL policy networks play with the best SL policy network ρ^1_θ. For each match, the first 4 turns will be random to create different matches each time. For the rest of the game, the action was selected by maximum probability, $a = arg\,max_{Pr(a|s)}$. We would remove the matches with a similar end-game state, the game with the draw result, and the game reaching the maximum game length. Each policy network played with the policy network for 1000 matches as white position and 1000 matches as black position, then we compared the final winning percentages between all policy networks.

$$P_{wr} = \frac{\sum win\,matches}{1000} \tag{5}$$

The self-play evaluation would lead to biased results, as we only can not evaluate how good the policy networks comparing with the human level. As the result, we add another layer of human evaluation. For human evaluation, we played with the best RL policy network ρ_{ω}^{1} with using 500 MCTS simulations per move then we collected and analyzed the game records to understand the key play strategy and weakness of the policy network.

Self-play Evaluation

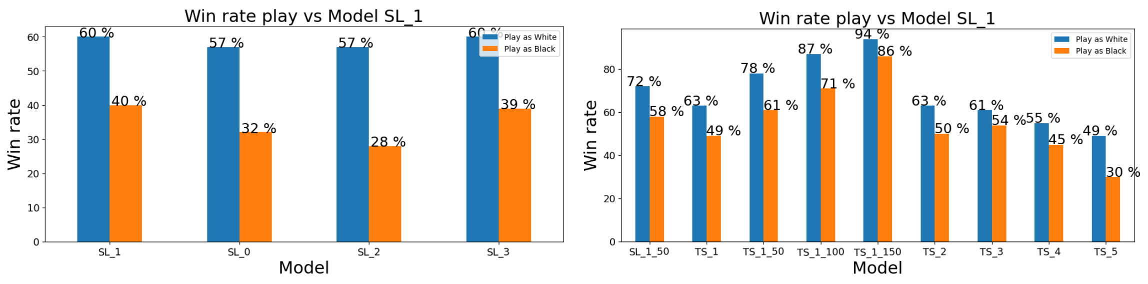

Figure 2: Self-play evaluation the winning percentage of 1000 matches playing again the policy network ρ_{θ}^{1} (SL1). **a** Performace of all SL policy networks. **b** Performace of all RL policy networks with and without MCTS when selecting actions.

When the policy network ρ_{θ}^{1} playing with itself, the white position winning percentage $P_{wr}^{W} = 60\%$ larger than the black position winning percentage $P_{wr}^{B} = 40\%$. In this case when both players were the same network $P_{wr}^{W} + P_{wr}^{B} = 100\%$. The policy network ρ_{θ}^{0} (SL0 - *without history features*) with $P_{wr}^{W} = 57\%$, $P_{wr}^{B} = 32\%$ and ρ_{θ}^{2} (SL2 - *without bot dataset*) with $P_{wr}^{W} = 57\%$, $P_{wr}^{B} = 28\%$, performance worse than ρ_{θ}^{1}.

All the policy networks, SL and RL, when playing as black position always have weaker performances comparing playing as white position. This happened because of **1**)The white player as the first player always has more advantages. **2**)In the expert dataset, the number of black bot actions is double the number of white bot actions.

Comparing the policy network ρ_{θ}^{1} and $\rho_{\theta\_50}^{1}$, we see the huge gained in performances with using the MCTS when selecting actions, P_{wr}^{W} increase 12% and P_{wr}^{B} increase 18% with only 50 simulations per move. And when we increased the number of simulations per move, the policy network also gain performance according to the P_{wr}^{W}, P_{wr}^{B} of the ρ_{ω}^{1} with 50, 100, 150 simulations per move.

The RL policy network ρ_{ω}^{1} (TS_1) which is the based network ρ_{θ}^{1} after trained with 16,000 matches of 50 MCTS simulations per move, play better than the based network ρ_{θ}^{1}. However, after continuing using the ρ_{ω}^{1} (TS_1) as based network for RL methods, all the next generation of RL networks started to decrease the performance, ρ_{ω}^{2} with 16,000 matches of 50 simulations, ρ_{ω}^{3}, ρ_{ω}^{4} and ρ_{ω}^{5} with 3,000 matches of 250 simulations. This would be a result of overfitting, as these policy networks trained on a small number of matches.

Human Evaluation

After playing with our best performance policy network (). The RL policy network performed at a similar level of Intermediate human players. The policy networks know to perform basic pieces movement, how to win the game, how to protect your queen piece, block opponents' pieces,

Our network play strategies: **1.** Try to place pieces around Queen opponents as soon as possible. **2.** Place many pieces on the board as fast as possible. **3.** Instead try to unblock your's trapped pieces, the policy network would keep placing new pieces. **4.** Protect your's queen pieces from being stacked up by the opponent's Bettel pieces. Our network weakness: **1.** Easy to be blocked by the opponents. **2.** Weak solutions to unblock trapped pieces. **3.** Beatle pieces do not have many moving strategies. **4.** In most cases, the policy networks are lost when do not have many unblocked pieces left to move. From this analysis, we can see that our policy networks have the same level of performance as an Intermediate player.

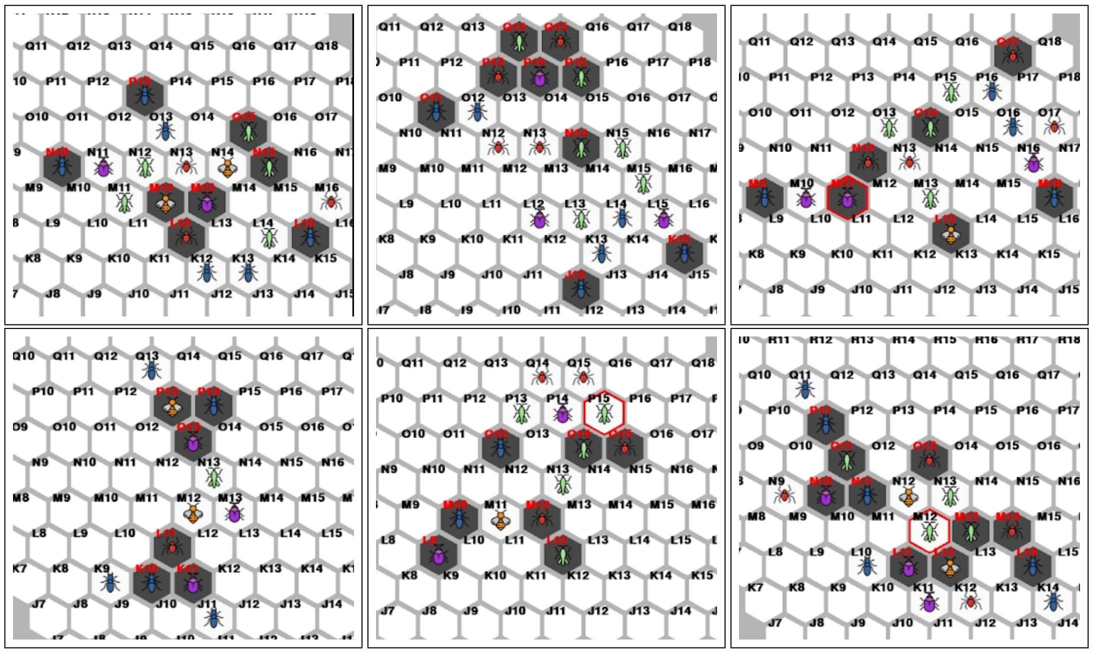

Figure 3: Sample game recorded between us and the ρ_{ω}^{1} with 500 simulations of MCTS played as white **a; b; c**. Played as black **d; e; f**.

5. Discussion

With the combination of search-based algorithms and deep learning, the AlphaGo Zero and AlphaZero methods by Deepmind have easily achieved superhuman performance in abstract strategy-based two players board games like Chess, Shogi, and Go. Hower to achieve this, the Deepmind team had access to unlimited computation resources, that why for reinforcement learning process, they can simulate 44 million matches of Chess, 24 games of Shogi, and 21 million games of Go [6][7]. As the result, for us to replica the same methods and applying to Hive, is a huge challenge.

In spite of having 11 pieces set for each player compared to 20 pieces in Chess, the average branch factor in Hive is 46 compared to 35 in Chess. Combing that with a large board size and more restricted movement rules, the computation for training Hive AI is more expensive than Chess games. Therefore, our running time during reinforcing learning increases exponentially with the number of simulations: 22 hours for 16,000 games of 50 simulations, and 24 hours for 250 games of 250 simulations. For that reason, we make changes to the original AlphaZero method to speed up our RL training process. We got a huge performance jump from the based SL policy network ρ_θ^1 to the first RL policy network ρ_ω^1. However, to continue to gain more performance we need to increase the number of self-play matches and the number of simulations per move which would require us more computation resources.

Another approach would be using Proximal Policy Optimization Algorithms (PPO) [11]. We will create environments and let the PPO agent play again our best RL policy network ρ_ω^1 with 100 simulations of MCTS per moves. The PPO policy would continuously update and train on small batches of rollout matches. This approach would help to solve the overfitting problems.

6. Conclusion

In this paper, we presented a similar genetic algorithm from AlphaZero methods to the development of AI for the game of Hive that was able to achieve strong performance with limited computation resources. Starting from AlphaZero combined supervised learning and Monte Carlo Tree Search (MCTS), with the customization of our State and Action presentations to create a model that could play Hive at a human level. The key contribution of our research is the demonstration of how supervised learning can be used to train an AI model on a labeled dataset of human games. This approach allowed us to leverage the knowledge and experience of expert players to train the AI model, resulting in a more effective and efficient training process for reinforcement learning. Additionally, the use of MCTS allowed the AI to search for the best moves in real-time, making it a strong opponent in the game of Hive

One of the key challenges in the development of AI for games is the high computational demand, especially for high branching factors like Hive. Our study addressed this challenge by using maximum game length, adding rewards and discounted factors to value return from the policy network leading to achieving strong performance with limited computation resources. Our findings may be good contributions to researching the field of developing more general and computation-efficient deep neural networks.

However, there is still room for improvement in our models. For example, the training dataset could be expanded to include expansion versions of the Hive game to include a greater diversity of human gameplay and more advanced strategies and tactics. Adding the ranking system creates weight metrics to quantify the quality of human actions. This would mean, we need better computation resources as the size dataset is quintuple. Additionally, the AI model could be further optimized to improve its performance with PPO [11] algorithm.

In conclusion, while there is still room for improvement in our approach, this research represents an important step forward in the development of AI for Hive. The combination of supervised learning and MCTS provides a promising approach for the development of advanced AI models for games and has the potential to be applied to other complex decision-making tasks. Moreover, our results demonstrate that it is possible to achieve strong performance with limited computation resources, making this approach accessible to a wider range of researchers and practitioners.

References

[1] Online Hive games database, boardspace, 2023 URL: *http://www.boardspace.net/hive/hivegames/.*

[2] Chris J. Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in Go using deep convolutional neural networks. In *International Conference on Learning Representations* 2015.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. In *Nature, 529(7587):484–489* 2016.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. In *Nature, 550:354359* 2017.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. In *arXiv:1712.01815* 2017.

[6] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Mach. Learn. 8, 229–256* 1992.

[7] Sutton, R., McAllester, D., Singh, S. Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems, 1057–1063* 2000.

[8] Hive game official website. *https://www.gen42.com/games/hive.*

[9] Training computation machines: 16 cores 5950x CPU - 128 Gigabytes of Ram - Nvidia 1080ti, 11 Gigabytes of Vram.

[10] Kunihito Hoki and Tomoyuki Kaneko. Large-scale optimization for evaluation functions with minimax search. In *Journal of Artificial Intelligence Research (JAIR), 49:527–568* 2014.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. Proximal Policy Optimization. In *arXiv:1707.06347* 2017.

[12] Rosin, C. D. Multi-armed bandits with episode context. Ann. Math. Artif. Intell. In *61, 203–230* 2011.