Implementing minimax algorithm and reinforcement learning on HEX game

Hung-Jun, Qiu Nian-Rong, Wu Yu-Chun, Chen
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Abstract

Hex is a two player abstract strategy board game in which players attempt to connect opposite sides of a rhombus-shaped hex board. We implement Minimax algorithm and reinforcement learning on hex board game.

1. Introduction

Computer Games is a popular branch of Artificial Intelligence. Motivated by many success on using AI to beat chess players expecially for world champion, we came up of an idea of playing a strategy chess game with AI. Our goal is to test several approaches on the Hex board game in order to demonstrate how reinforcement learning can improve board game strategy. The rule of Hex game is two plyers take turns to place in an empty hexagonal cell and the first player to form a connected path of their counters linking the opposing sides of the board marked by their colour wins. Because the time complexity of the Hex game is PSPACE complete, the issue is exceedingly difficult. When the board size exceeds ten, the particular winning strategy cannot be proven.

In our project, we set our baseline as Minimax Search Algorithm and BFS and our main approach is Reinforcement Learning with Monte Carlo Search Algorithm. Furthermore, we tried to figure out how different times of training model can have impacts on the performance. You can refer our project to our Github repository.

2. Related Work

Google DeepMind released the paper [2] about training an AI model that play *Go* with reinforcement learning. To be more specific, they trained their network through supervised learning with human's games before applying reinforcement learning. After the supervised learning, they trained their network through the combination of Monte Carlo tree search with two convolutional neural networks: one of them is policy network, and the other one is value network.

Inspired by Google DeepMind, we decided to adopt a similar may to train an AI model to play *Hex*: combining Monte Carlo tree search [1] with two convolutional neural networks. [3] Different from how Google DeepMind trained their model, we did not train our model through supervised learning. We just went straight into the self-training step. However, we train our networks in the same way that Google DeepMind does.

3. Methodology

I. Baseline

(a) Minimax Search
We can realize that if we can give different state an evaluation, then we can use this algorithm to traverse the board and multiple depth. During my turn, record the maximum value of the state, then on the opponent's turn, choose the minimum value.

(b) heuristic function and path finding
We want to know how to evaluate different state. Therefore we can see that the score as one expected steps- rival's expected steps. We convert a state into a graph and connect the nodes to the board's boundaries. When the state is given, the BFS algorithm can be used to discover the shortest path to the opposite side of the board.

II. Main Approach

(a) Monte Carlo Search
Given the state(S) and we have to find the best action.

For all the valid actions in state s, Nsa(s,a) stores the times that each action has been visited. As a result, the probability of choosing the action a is based on $Nsa(s,a)/\sum(Nsa(s))$ for the state S. How can an action been visited? In each iteration, choosing the largest value of score(a) for all the valid actions in the given state s. Noted that

$$score(a) = Q(s,a) + \frac{n * Ps(s,a) * \sqrt{(Ns(s))}}{(1 + Nsa(s,a))}$$

Ps(s,a) is a two-dimensional dictionary that contains the probability distribution of all actions from a given set of states. For each valid action in a given state, the policy network will evaluate a value v. As a result, for each action,

$$Ps(s,a) = \frac{v}{\sum Ps(s,a)}$$

Qsa(s,a) is getting the value v from the next state which is the result of state s doing the action a. For the next state, repeat the calculation on the top of this paragraph (e.g. choose the largest value of score(a)). If the state is ended, it will return 1 for win and -1 for lose. If Qsa(s,a) is not visited, then Qsa(s,a) equals to v. However, doing the same action a in the same state s doesn't mean that we will get the same result. If Qsa(s,a) was visited before, return the average of all the v.

In the begining, the value of score(a) depends more on Ps(s,a) because Q(s,a) equals to 0 in the begining. As Nsa(s,a) gets higher and higher, the influence of division part gets smaller, so Q(s,a) will be more important.

(b) Self-Play (training step1)
We let two players calculated by two old nerual networks play several games and store each state for our dataset. We can get the function of pi(s), which stores the probability of each valid actions in the state s from MCTS algorithm. The method of choosing the next action is to randomly choose a valid action based on the probability pi(s). After playing several games, we get the dataset.

(c) Train the new model (training step2)
For each time of training, we need to train two Convolutional Neural Network. One is policy network, and the other is value network. Policy network: input a state, output the probability of all valid actioins. Value network: input a state, output a value evluating how good the state is. We implement gradient descent, reduce the total loss and update the model. loss function of policy network is cross entropy, and loss function of value network is mean squared error. Total loss is sum of them.

(d) Testing new models
Let the old network and new network play several games. In the execution part, we select the best action by Monte Carlo Tree Search. We only choose the model when it's winning rate is more than 60%; otherwise, we reject and start a new iteration in our report.

4. Experiments

I. Baseline and AI agent comparison

(a) performance

In minimax with depth 1, the win rate of level 1 is identical to that of level 2. We can demonstrate that our algorithm is superior to the one found on the internet. As to our baseline method in depth 2 we expected it to have a greater win rate. However, the baseline method failed to fight against the one on the internet. We considered that such result stemmed from the fact that our algorithm prioritizes defense over attack, resulting in the case that even if the baseline method was close to win, it stayed in the principle of valuing defence. The essential winning moves were sometimes not taken by our baseline method.

It is perplexing that minimax with depth 2 performs poorly when fighting against the level 1 bot on the Internet but won against the level 2 bot.

For level 3, our baseline lost every single game against the Internet bot.

(b) thinking time
As the depths increase, times the baseline method it takes to calculate the next step get larger.

Table 1. baseline vs. web 11*11 on 6*6 hex board

	level 1	level 2	level 3
depth 1	72.5%	70%	0%
depth 2	50%	92.5%	0%

II. Main approach and baseline
Our initial goal was to train an 11*11 model that could outperform our baseline and the AI agent found on the internet. However, we had to make some compromise due to the overwhelming training time. Getting two neural networks to play approximately 150 games costs 3 hours, not to mention the training and testing that follows. By searching the internet, we discovered that we needed to train for about 2000 games to beat normal human players. As a result, we change our objective into training a model to play Hex in size 6*6.

(a) performance

In every comparison, we expected our performance to improve as the training time increased. In some columns, the win rate increases as the

training time increases. However, we are unsure whether our theory can be applied to the outcome because of the high win rate.

It is expected to perform better as the number of MCTS simulations increases. However, it appears that the three forms differ only slightly. The reason for this could be that there is no need to simulate too many times in such a small size.

(b) thinking time
As the number of MCTS simulations grows, our main approach requires more thought time. Similarly, as the depth increases, our baseline requires more time to think. Our models take longer to train as the number of MCTS simulations increases.

Table 2. Compare different training times main approach vs. baseline(depth) with MCTS 25 times on 6*6 hex board

	depth1	**depth2**
100	95%	95%
200	97%	97%
300	99%	94%
400	88%	94%
500	98%	100%

Table 3. Compare different training times main approach vs. baseline(depth) with MCTS 50 times on 6*6 hex board

	depth 1	**depth 2**
100	94%	100%
200	95%	96%
300	97%	97%
400	98%	98%
500	98%	96%

5. Contribution

- Yu-Chun, Chen: (1/3) Web Scratch, Finding path
- Nian-Rong, Wu: (1/3) MiniMax algorithm, Report
- Hung-Jun, Qiu: (1/3) Reinforcement Learning

References

[1] Broderick Arneson, Ryan B Hayward, and Philip Henderson. Monte carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258, 2010. 1

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 1

[3] Kei Takada, Hiroyuki Iizuka, and Masahito Yamamoto. Reinforcement learning for creating evaluation function using convolutional neural network in hex. In *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 196–201. IEEE, 2017. 1