Reinforcement Learing on Board Games

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Vishal Kumar Chaudhary
(111501030)

under the guidance of

Dr. Chandra Shekar Laskhminarayanan

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

<u>CERTIFICATE</u>

*This is to certify that the work contained in this thesis entitled "**Reinforcement Learing on Board Games**" is a bonafide work of **Vishal Kumar Chaudhary (Roll No. 111501030**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my supervision and that it has not been submitted elsewhere for a degree.*

Dr. Chandra Shekar Laskhminarayanan

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

I would like to express my special thanks to Dr. Chandra Shekar Laskhminarayanan for his support and guidance throughout the course of the project. He gave me various opportunities to learn various things which couldn't be possible all by myself. He gave me ways to organize things and what is the scope of the project. I would also like to thank my family for their support and motivation.

Abstract

In this project, learning through self-play algorithm has been explored and has been applied on connect4, 2048, cricket. Our task is to improve the training process of board games. During self-play, data is being generated for the model to learn. So if we create good data and explore more rewarding paths then our agent can learn quickly. For experiment purposes, we have tried the self-play algorithm in 2048 with a branching factor of 4. Further, in this project we worked on two important things.

We have introduced KL-Upper Confidence Bound (KL-UCB) and Thompson sampling for sequential game connect4. In Spite of these bound came from the solution of multi-arm bandit problem they show significant improvement over the existing bound which was being used in the algorithm.

Learning through self-play algorithm has been implemented for sequential games but we have applied this algorithm on cricket in the simultaneous environment. Simultaneous games are more complex game than its sequential form because there is always randomness involved about the opponent.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Board games are one of the best way to test agents and reinforcement learning algorithms to test it viability.These are complex yet not too complex than that of agent in real environments. Example- chess is one of the complex game with state space of almost 10^{47} and decision tree of size 10^{123}. Similarly state space complexity of connect4 is 10^{12} and decision tree of 10^{21}. This motivates me to learn and contribute to algorithms in board games. The task of this project is to improve the learning rate of the algorithm. The traditional supervised machine learning method involves experts data and sometimes, it is not enough or the data is too noisy such that we cannot learn anything. In this project various games have been chosen depending upon its complexity to train agent within time constraints.

The project started with establishing fixing the baseline for some fixed game. We went directly to study the one of the most complex game chess. We thought to train agent with the help of dataset. But for training such network and work on big data has certain limitation. Like it requires huge computation (GPU and RAM requirements) and there should be enough examples for a certain stradegy to learn. This poses a problem that we don't know the quality of data though we had collected a good amount of data. So that stratdgy didn't worked.

We got to know about learning through self-play. We started trying to select a game that could suit our situation like low computation, trainable in days. We select 2048 game which has the branching factor of 4 which means at any given position we can have maximum of 4 valid moves. The motivation of chosing low branching factor game was to test the possibilty and scope of given algorithm. We tried and trained the 2048 and it was able to train in 2-3 hours with better results than the random moves. We tried with different type of neural network architecture. We found out that the irrespective of the network architecture, there was not significant change. So we move on applying the algorithm without changing the neural network architecture and we want to test the algorithm on some complex games with higher branching factor. We selected connect4 game with branching factor of 7. We tested and studied the algorithm of self-play and every piece of it on connect4. We established the baseline over this game and we will not change the neural network architecture in any experiments because of findings in the game 2048.

Connect4 is an example of sequential game in which player1 moves after the player2 has taken its action. So the task of agent is to take action which maximize its probability of winning. The advantage of training agent from self play is that it explores the game state space all by itself depending upon the reward it gets after taking those action.To make the probability space of more rewarding state higher, better bound should be choosen to select the action. We gave a shot to confidence interval of solutions of multi-arm bandit problem. KL-UCB [1] is found to be performing better than the Upper Confidence Bound (UCB). We applied this bound in the training process and got better results which excited us to seek for more better bound in the multi-arm bandit field. We found thompson sampling and experimented on it. We found out that it is performing as better as the KL-UCB. In the end our both results are being better than result of baseline which use UCB.

With the given achievement in sequential games, we also wanted to try on simultaneous games. Simultaneous games are the game in which both player make their move at the same time. So there is randomness involved and the agent have to choose action considering the possible move made by adversary. There is only a very little work on this area. We have taken cricket game in simultaneous environment.So we have bowler team and they chooses which bowler to bowl for the over. We have also batting team which choses shot which can be go run for $0, 1, 2, 4, 6$ irrespective of the knowledge of the bowler. This shot from the batting team is fixed for the whole over for the simplicity of game. The motive of bowling network is to minimize the number of runs made instead number of matches won whereas the motive of batting network to score runs. Since we want to investigate the learning through self-play in simultaneous game which is much harder for network to learn than the sequential games, we made this cricket environment to be simple and easy to investigate.

Chapter 2

Review

2.1 Reinforcement learning by self-play

The traditional algorithm involves training agent with help of expert data, guidance or hard coded heuristic in that domain. They also don't discover paths which was not present in data. Reinforcement learning through self-play [2] is the state of art algorithm for board game agent. In this algorithm, the agent learns new things by themselves without any human intervention. This algorithm has three parts, self-play, training, selection of new best agent. The agent tries to improve on the previous version of itself. This type of algorithm was not possible before because it requires a lot of computating power. It is very different to other supervised algorithms which requires a good source of data and the model is dependent upon the noise in data. But in the above algorithm it does not require data from outside, rather it create data by selecting highly probable and rewarding moves.

2.1.1 Self-play

The agent learns from the examples created by its previous version. There is one neural network f_θ which predicts actions probability and probability of winning if it played with itself (meaning same policy) from that state.

$$(p, v) = f_\theta(s)$$

. So (p, v) is the prediction of the network. With the help of this network combined with tree search, it produces better strategy for agent in next iteration. The tree search involved is Monte Carlo Tree Search (MCTS) [2].

After doing serveral MCTS iterations, one can estimation action probability with the help of number of visits taken to other state after taking action a. $\pi_a \propto N(s,a)^\tau$ where $N(s,a)$ is the number of time a action has been taken from state s and τ is temperature parameter. z is win or lose at the end of game. So (π, z) is the examples created after the self play and π is supposed to be better policy than the p because of the tree search simulations. The exisiting method uses upper confidence bound for action selection. Our method differs here using different bound for action selection. The bound used are being KL-UCB [1] and Thompson sampling [3]

2.1.2 Training

As previously mentioned, there is one neural network with two head one predicting policy and other probability of wining. We train this neural network with (π, z) data.

$loss = (z - v)^2 + \pi^T log(p)$

2.1.3 Selection of Model

Selection of neural network is done with the help of tournaments between the new trained version of neural network and the previous neural network which was before training with the new data. Depending upon the thresold we decide the new network as our best network or not.

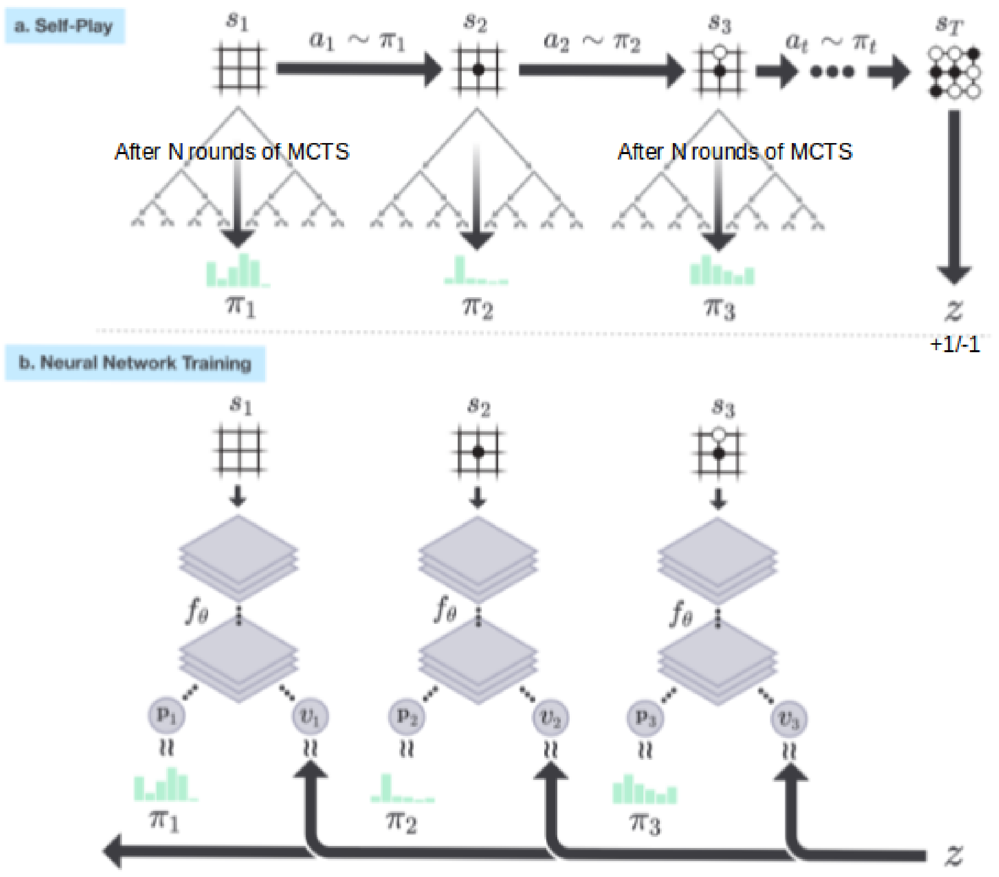

Fig. 2.1 a. self-play: each action $a \sim \pi$ is done after N simulation and single example is created with values depending upon lose or win and probability depending upon number of actions taken. b. Neural network is trained with the examples created during self-play

2.2 confidence bound

So far Upper confidence bound is being using during the Monte Carlo Tree Search (MCTS)[4] which minimize the regret.

$$A_t = argmax_i \left\{ Q_i(t-1) + c * P(s,a_i)\sqrt{\frac{2 * \log N}{N_i}} \right\}$$

A_t is action taken at time t, Q_i is average reward after taking i^{th} action, c is temperature parameter, $P(s,a_i)$ is the probability of taking i^{th} action in state s, N is total number of simulations and N_i is number of time i^{th} action has been taken.

2.3 Conclusion

In multi-arm bandit problem, this bound has been shown to have .

Chapter 3

Algorithm I

Game which have been used to test the validity of the algorithm is connect4. Connect4 is a two player game. So the agent have to chose one valid action depending on the board state in such way that he will win the game. This problem is similar to multi-arm bandit problem in which one has to chose arm to maximise the reward or which minimize the regret. During the self play and doing Monte Carlo Tree Search(MCTS) we are using KL-UCB to select action given the history of rewards. In multi-arm bandit problem, KL-UCB has less regret than the UCB. We want to see whether this also work and improves the learning time of agent.

For each state and action of game we store reward during self play and the create data for agent to learn upon those examples.

Step 1: For each state having k valid actions

Step 2: Choose every action once.

Step 3: Then choose A_t action at t - time

$$A_t = argmax_i \ \max \left\{ x \in [0,1] : d(\hat{x}_i(t-1), x) <= \frac{\log N}{N_i} \right\}$$

3.1 confidence bounds

3.1.1 KL-UCB

3.1.2 Thompson Sampling

3.2 Simultaneous Games

3.3 Conclusion

3.4 Conclusion

In this chapter, we proposed a distributed algorithm for construction of xyz. The complexity of this algorithm is $O(n \log n)$. Next chapter presents another distributed algorithm which has linear time complexity based on xyz.

Chapter 4

Algorithm II

The algorithm presented in previous chapter has $O(n)$ time complexity. We further propose another distributed algorithm in this chapter based on xyz which has linear time complexity.

4.1 Construction

Write ...

4.2 Improved Method

Write...

4.3 Conclusion

In this chapter, we proposed another distributed algorithm for XYZ. This algorithm has both time complexity of $O(n)$ where n is the total number of nodes. In next chapter, we conclude and discuss some of the future aspects.

Chapter 5

Conclusion and Future Work

write results of your thesis and future work.

References

[1] T. Lattimore and C. Szepesvri, “Bandit algorithms book,” p. 131. [Online]. Available: https://tor-lattimore.com/downloads/book/book.pdf

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of go without human knowledge.”

[3] D. J. Russo, B. V. Ro, A. Kazerouni, I. Osband, , and Z. Wen, “A tutorial on thompson sampling,” 2017.

[4] “add this.”