Reinforcement learning methods in board and MOBA games

Hongyi Chai[1,2]

[1]University of California, Davis. 1 Shields Ave, Davis, CA 95616

[2]hychai@ucdavis.edu

Abstract. This article provided an introduction of applying reinforcement learning to games, including board games and video games like Backgammon, Go, and Dota2. The reason for choosing reinforcement learning to solve game problems was analyzed. The article also reviewed the reinforcement learning technique and introduced two optimizing learning methods, Temporal Difference learning and Q learning. Then, three important cases of using reinforcement learning to reach high level game skill, TD-gammon, AlphaGo, and OpenAI Five, were introduced. In the end, the future possibility of applying reinforcement learning in broader way was analyzed.

Keywords: board games, video games, reinforcement learning, Backgammon, Go, Dota2.

1. Introduction

Since the rules of games, especially board games, are clear-cut, and thus it is easy to imitate in a computer environment and easy to evaluate performance, games are often used as platforms for testing computer programs [1] [2]. After the introduction of modern machine learning, it was quickly applied to solve games. Supervised learning was tried by at an early stage, but this method also contained the flaw of human evaluation [3]. Therefore, the programs could only reach a high human level but cannot win against human players.

A better method called reinforcement learning was invented later in the 1980s [4]. This method was based on the animal psychology of trial and learning, combined with the value function and dynamic programming. The computer program could learn from self play by adjusting parameters within the function. This method fitted perfectly in the context of the games. Therefore, after the advent of reinforcement learning, there have been many breakthroughs.

From the 1990s to today, reinforcement learning was kept proving by scientists that it was the best way to train game AI. TD-gammon in the 1990s utilized the TD(λ) method, which solved the problem of implementing dynamic programming in machine learning [5]. Google Deepmind brought this method to the peak of board game AI after introducing AlphaGo. Different versions of AlphaGo all won over different top human Go players, including the reigning human player Ke Jie in 2017. Shortly, in 2019, the game AI OpenAI Five won against reigning human team OG in the game Dota2, proving that reinforcement learning even worked well in complex video games [6].

This paper will cover a basic review of reinforcement learning in section 2. And then three important reinforcement learning cases in games, TD-Gammon, AlphaGo, and OpenAI Five, will be described in section 3.

2. Reinforcement learning

The phrase "reinforcement learning" first appeared in the 1960s, but the development of this technique was slow for some years until it revived in the 1980s [4]. During these years, scientists tried to refine reinforcement learning in two main directions [4]. The first direction was based on the idea of trial and error. The AI tried to accomplish one goal many times, and the scientists hoped the AI was able to learn from failure in some ways. The second direction focused on calculating an optimal solution by using value functions and dynamic programming. There was also a minor third research direction focused on temporal difference algorithms. And finally, these three algorithms combined together to form today's modern reinforcement learning.

In reinforcement learning, as shown in figure 1, a program that can interact with the environment is called an agent, and the environment contains the problem that the agent needs to solve [7]. The agent is able to obtain the current state of the environment, and it can also take actions to alter the state of the environment. After each action taken, the agent will receive an immediate reward. And the agent will receive a series of immediate rewards after taking a series of actions until the game ends. The goal of the agent is to maximum accumulated reward [7].

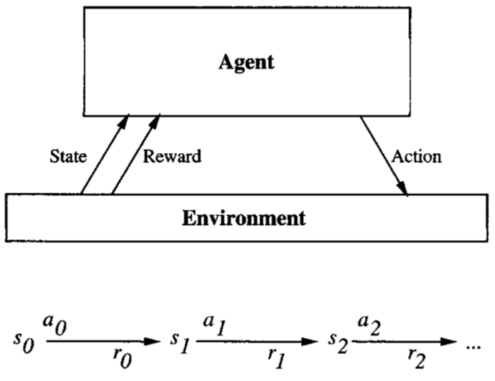
Agent
State
Reward
Action
Environment
$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$

Figure 1. The agent interacts with the environment. The goal of agents is learning a policy that takes a series of actions that maximizes its accumulated reward [7].

To be specific, the environment responds to the agent's action based on the state-action transition function and reward function. Both functions take current state and action as parameters, and they would produce "next state" and "immediate reward" respectively. However, the agent doesn't know what function the environment is using. Therefore, the agent needs to find out these two functions during training. And basing on these two functions, the agent can find the maximum accumulated reward.

The main challenge the agents face is the credit assignment problem [8]. The agent needs to assign credits to previous actions when a reward is received. Since the policy for the agent is a series of actions, the immediate reward gained after taking an action might be the result of previous actions. Therefore, because previous actions are also crucial, they also deserve credits. However, it is hard to assign credits to previous actions because some of them are more important than the others. As a result, this becomes a main challenge in reinforcement learning.

One method in dealing with this challenge is the Temporal Difference (TD) method [8]. The TD method allows agents to update its evaluation from the temporarily successive experience during training [1]. In the conventional training process, the network is trained based on the difference between current evaluation and the true evaluation. But, in the TD method, the network will update its evaluation gradually based on the previous evaluation. The idea of optimizing is to optimize the weight in the neural network. And the final weight is just the initial weight plus the sum up of the weight difference at different timestep. In this way, the update of weight only depends on the previous estimation. This will make the algorithm perform better since the "true evaluations" used before are usually based on the human experience, and human experience is not absolutely correct. On the other hand, using the previous experience of the agent to update the function will not be influenced by human bias. This method is proved to converge to the true evaluation [1]. There are also other advantages. It also takes less memory and computation resources because it only needs to store temporary estimations and the updating algorithm is easy.

Another method dealing with this challenge is called Q learning [8]. This method defines a Q function $Q(s,a)$ that is able to directly give the accumulated reward of doing an action in such a state. Therefore, the quality of an action is summarized in a single function for future use. This function is updated based on the equation

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma\, maxQ(s',a') - Q(s,a)).$$

The Q function updates its value based on the maximum Q value in the next state-action pair. This method is very practical because calculating the true Q value iteratively and repeatedly costs lots of resources. The scientists prove that if the Q can converge, it must satisfy three conditions [7]: first, the system is a deterministic Markov Decision Process; second, the immediate reward has a bound; and third, during training, the agent has infinitely visited every possible state-action.

3. Reinforcement learning in games

Board games are considered as good platforms to perform testing for the concepts of machine learning. Board games like Go, Chess, and Backgammon are complex enough that cannot be learned with brutely enumeration methods, but they have a clear-cut rule and are easy to imitate in a computer program [1] [2]. These two features make them become the best platform for machine learning.

Reinforcement learning is intuitively suitable to solve games [8]. The game itself is the environment and the players are agents. Scores can be viewed as reward, and agents can only choose actions in a finite set. Reinforcement learning makes the computer agents reach master level, even superhuman level. The following three examples show the rise of reinforcement learning in board games, the peak this method reached, and the possible future implementation and potential of this method in the games area.

3.1. Backgammon

Backgammon is a classic two-player game which has a long history and it is known for its high probability and high randomness [2]. The player needs to roll two dice and move accordingly. When the two dice have the same point, the player can double the points on the dice and move accordingly. There is also a special rule that allows one player to “eat” another player’s checker to gain advantage, which further increases the complexity of the game.

Before the application of reinforcement learning, many attempts have been made to build a program that could reach the human master level of backgammon, but all of them are not success. One of the most famous ones was called Neurogammon, which won the backgammon competition in the first Computer Olympiad in 1989 [3]. This program mainly trained a neural network based on supervised learning and human data [3]. A group of backgammon experts provided data that described good moves and bad moves in a given situation. The network was told that good moves would score higher than the bad moves. And the desirable output of the programs were good moves [3]. However, there were tremendously huge numbers of situations in the backgammon games, and it was impossible for experts to give their evaluations for each of them. Moreover, the data given by the experts were not always right. Their understanding of the games might not be accurate and often contained many biases based on their experience [2].

In 1992, a program called TD-Gammon reached master level compared to human players. It was based on reinforcement learning and the evaluation method is TD(λ) method. It was one of the most famous implementations of the TD(λ) method after this method was mathematically proved. TD-Gammon was the first computer program that reached master level in the game of backgammon.

In TD(λ), the weight difference $\triangle w_t$ that needs to be updated is,

$$\triangle w_t = \alpha\,(P_{t+1} - P_t)\sum_{k=1}^{t} \lambda^{t-k}\, \nabla_w P_k$$

where P_t is the prediction of the final weight at timestep t. $\nabla_w P_k$ is the gradient, and λ is a discounted factor that allows gradual updating, which makes the updating process smoother.

TD-Gammon trained from raw with random strategies and self-play. The input was merely a current situation on the board. As shown in the table 1, with several version updates, TD-Gammon 2.1 has been trained for 1,500,000 games, and it only lost 1 point in 40 games when playing with top human players. This was not a victory toward humans, but it still demonstrated high-level skill and outperformed other contemporary backgammon programs [5].

Table 1. Results of testing TD-Gammon in play against World-Class human opponents [5].

Program	Training games	Opponents	Results
TD-Gammon 1.0	300,000	Robertie, Davis, Magriel	-13 pts/51 games (-0.25 ppg)
TD-Gammon 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts/38 games (-0.18 ppg)
TD-Gammon 2.1	1,500,000	Robertie	-1 pt/40 games (-0.02 ppg)

3.2. AlphaGo

AlphaGo was one of the most famous successful AI cases. It was designed by DeepMind under Google. After AlphaGo defeated world top Go player Lee Sedol and Ke Jie in 2016 and 2017 respectively, a lot more AI researchers began to focus on the algorithms and logic of Alpha Go [9]. After the publication of AlphaGo Zero, the method of reinforcement learning reached its peak in board games.

There were many versions of AlphaGo [10]. The first published version of AlphaGo, AlphaGo Fan, won the European championship Fan Hui in 2015. The version defeated Lee was called AlphaGo Lee and the version defeated Ke was called AlphaGo Master. After defeating the reigning world champion, DeepMind stopped playing with human players. The final version of AlphaGo was AlphaGo Zero, which won AlphaGo Master over 89 games in 100 games.

Both AlphaGo Fan and AlphaGo Lee utilized similar architecture and algorithms. Two separate deep neural networks as its policy network and value network. These two networks were both trained by human data and experience [10]. And a method called Monte-Carlo Tree Search (MCTS) was applied for lookahead searching. This framework was easy to understand and to implement, but had a great performance in game AI [11]. Starting from some states, the framework selected an action that balanced exploitation and exploration. And when it reached a state that never appeared in the tree, it started to simulate. The result of the simulation would be backpropagated to the whole tree, as shown in Figure 2 [11].

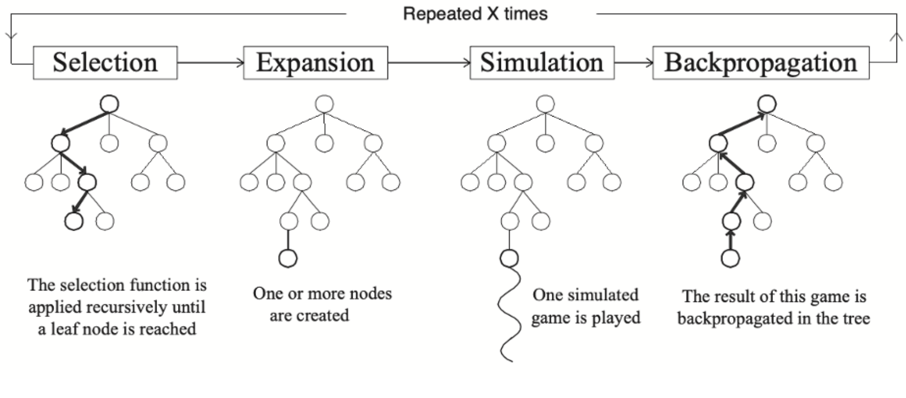
Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 2. Outline of a Monte-Carlo Tree Search [11].

AlphaGo Zero outperformed its predecessors by applying two important modifications [10]. First, only one neural network was used in AlphaGo Zero. The neural network could produce both a vector of probabilities and a value. The vector gave the probabilities that the agent chose each action, and the value was a prediction of the win rate of the current player. Moreover, a simpler tree search replaced MCTS that suited this single neural network. The second modification, also the more important one, was that AlphaGo Zero abandoned the data from human-players and trained solely on self-play games [10].

The result of these modifications was astonishing. AlphaGo Zero outperformed AlphaGo Lee in only one and half days starting from the raw, even though AlphaGo Lee was trained for several months back then. After 3 days training, AlphaGo Zero beat AlphaGo Lee with a 100:0 win rate. It then quickly outperformed AlphaGo master as well. Purely self-play reinforcement learning was proved to be absolutely practicable [10]. The performance evaluated by Elo rating method is shown below in figure 3.

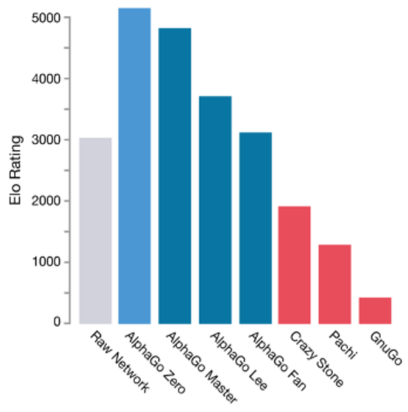

Figure 3. Final performance of AlphaGo Zero and other agents evaluated by Elo rating method [10].

AlphaGo started the new era of AI since it integrated the whole knowledge of AI at that time and showed the strength of this technology [9]. It also started an era of making decisions based on sufficient data and computations rather than logical reasoning.

3.3. *OpenAI Five*

OpenAI Five was a reinforcement learning AI system designed by OpenAI company used to play the video game Dota 2, one of the famous Multiplayer online battle arena (MOBA) games. Complex video games provided a great platform to imitate real-world problems. These problems usually contained lots of aspects and had a long-time horizon.

Dota 2 is one of those video games. To be specific, one match of Dota 2 usually lasts 45 minutes, and the agent needs to decide an action for about every 0.13 second. To contrast, Go only has about 150 actions per match. Moreover, the agent needs to observe about 16,000 values from the game per step and has an action space for over 8,000 actions, while Go only has a pool of about 250 actions per step. Beside the volume of the game, Dota 2 is also imperfectly observable. The agent is not able to see the whole map due to the rules of the game. Information like the position of the enemies and parts of the map are usually hidden. Therefore, the agent is required to have the ability to speculate a lot of information.

The policy optimizing method of OpenAI Five was the Proximal Policy Optimization (PPO). It was a kind of policy gradient method. PPO used multiple epochs of stochastic gradient ascent to perform minibatch update. It retained the advantages of the trust-region method but was much easier to code and required less samples to achieve goal function [12]. Policy gradient method was based on the idea of optimizing a “surrogate” objective function. And the astonishing part of PPO was that it used a novel surrogate function,

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)],$$

where θ is the vector of policy parameters and $\hat{E}_t[\ldots]$ is the empirical expectation over timesteps. The first parameter in the min $r_t(\theta)\hat{A}_t$ means the probability ratio of the new policy to the old policy, times the estimated value of advantage function. And the clip means the probability ratio must be in the interval of $1-\epsilon$ and $1+\epsilon$. ϵ is usually 0.1 or 0.2. This "surrogate" objective function was inspired by the one used in trust region methods. But this surrogate function performed better when optimizing it using stochastic gradient ascent. When the OpenAI Five team implemented this method, they used Generalized Advantage Estimation (GAE) to estimate the value of advantage function. GAE was able to provide a steady and fast environment during training. The agent contained a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN). This network performed very well on large scale acoustic modeling and efficiently exploited parameters of model [13].

The whole training process is shown below in figure 4. OpenAI Five used a large number of CPUs and GPUs during training and trained for 10 months. A "Rollout Worker" contained 51,200 CPUs that ran the game parallelly. The Rollout Worker ran the game in half speed because in this speed the Rollout Worker was able to run slightly more than 2 times games parallelly. Four fifths of games were played using the latest policy and 1/5 of games were played using the older policy. All game data was transferred to the Optimizer part, which contained 512 GPUs. The game data was stored in the local buffers of GPUs and GPUs would calculate the gradient using the game data selected randomly from the buffer. Then, the GPUs would give the new version of parameters to the Controller every 32 gradient steps. The Controller was able to stop and restart training. It communicated with the Forward Pass GPU group with the latest parameters. Finally, the Forward Pass GPU group ran forward passes to the Rollout Worker, which complete the whole process.

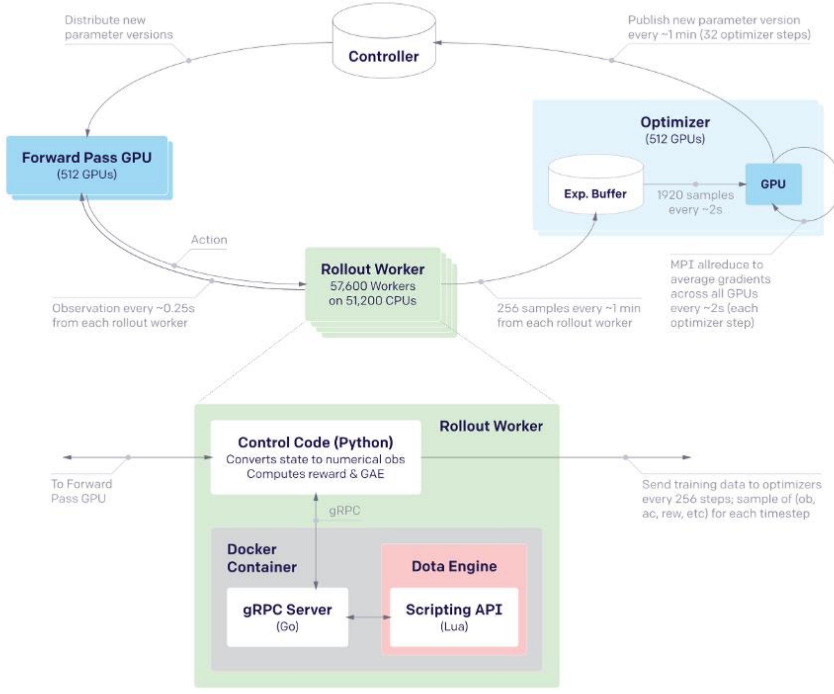
Distribute new parameter versions
Controller
Publish new parameter version every ~1 min (32 optimizer steps)
Optimizer (512 GPUs)
Forward Pass GPU (512 GPUs)
Exp. Buffer
1920 samples every ~2s
GPU
Action
Rollout Worker
57,600 Workers on 51,200 CPUs
Observation every ~0.25s from each rollout worker
256 samples every ~1 min from each rollout worker
MPI allreduce to average gradients across all GPUs every ~2s (each optimizer step)
Rollout Worker
Control Code (Python)
Converts state to numerical obs
Computes reward & GAE
To Forward Pass GPU
Send training data to optimizers every 256 steps; sample of (ob, ac, rew, etc) for each timestep
gRPC
Docker Container
Dota Engine
gRPC Server (Go)
Scripting API (Lua)

Figure 4. System overview of OpenAI Five [6].

OpenAI Five used the narrative information that human players could also get from the game as the agent's observation, rather than the pixel information. The team did that because first they wanted to focus on logical processing rather than visual processing, and second, it was improper to track and to analyze all the pixels for 45 minutes [6].

Since the training process was unprecedentedly long, the environment changed periodically during the training process. These changes were caused by modifying the training process and the updating of the game itself. However, training the agent from raw after each change was infeasible because of the limited resources. Therefore, a novel method called “surgery” was introduced during the training process. During the surgery process, the system would be offline and would turn the old model into a new model, even though the number of parameters might not be the same. This method was sufficient to enable the agent to reach superhuman performance, but it is not perfect. The OpenAI team performed a “Rerun” which was trained from raw in the final version of games. It outperformed the OpenAI Five and got a win rate of 98% against OpenAI Five. As a result, the method of maintaining the performance of agents in a constantly changing environment in long-term training was worth studying.

Despite the astonishing performance of the OpenAI Five, it was based on some compromises. The first compromise is that the agent was only able to select in a pool of 17 heroes, while there were 117 heroes to be chosen in the game. The second one was that the agent could not control multi-units at the same time. There were some heroes and items that allowed one player to control multiple units at the same time, but the agent could not do this. The third one was that some strategies of agents were fixed. Strategies like item purchasing, ability learning, and courier controlling were set by human hand scripts. After 10 months training, the OpenAI Five defeated the reigning world champion in 2019, demonstrating the superhuman ability of the AI. The team also opened the AI program to the public. Surprisingly, 29 teams managed to defeat this AI in 42 matches. This meant the AI still had many imperfect aspects, especially when it encountered new situations that it never met in the training.

4. Conclusion

One thing to be noticed is that TD-Gammon would perform better if adding some human scripts. Compared to the AlphaGo Zero case, this shows that the implementation of TD method by the TD-Gammon still had room for improvement. The subsequent agents like AlphaGo Zero and OpenAI Five both show that training without human language, but with larger and more resourceful training would result in a better AI in the end. Despite the tiny error TD-Gammon made, TD method was still the most promising algorithm back then. Reinforcement learning began to show its power because of the TD method. It substantially enhanced the calculation efficiency and allowed the computer back then to calculate large reinforcement learning.

The peak of reinforcement learning was reached when DeepMind published AlphaGo. AlphaGo didn’t invent any new AI theory, but it integrated the most advanced AI methods back then and implemented them into a usable AI. Moreover, AlphaGo also broke the traditional causal relationship between problems and solutions. It showed that with sufficient data and calculations, problems can be solved without logical confirmation. This led information technology to a new era [9].

After board games, complex video games became the next target of reinforcement learning experts. OpenAI Five showed that the basic idea of reinforcement learning still works well on games with complex rules, but the training process needed a huge amount of time and resources. The new focus of reinforcement learning now becomes how to deal with tasks with a long-time horizon and in a changing environment. Although the OpenAI team provided the “surgery” method to deal with it, the method is far more than perfect and need revisions. Moreover, the tasks and environments in the future are going to be predictably more complex. The methods to solve problems in these situations without compromising are still vital and urgent.

Reference

[1] Sutton, Richard S. "Learning to predict by the methods of temporal differences." Machine learning 3.1 (1988): 9-44.
[2] Tesauro, Gerald. "Temporal difference learning and TD-Gammon." Communications of the ACM 38.3 (1995): 58-68.
[3] Tesauro, Gerald. "Neurogammon: A neural-network backgammon program." 1990 IJCNN international joint conference on neural networks. IEEE, 1990.

[4] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.
[5] Tesauro, Gerald. "TD-Gammon, a self-teaching backgammon program, achieves master-level play." Neural computation 6.2 (1994): 215-219.
[6] Berner, Christopher, et al. "Dota 2 with large scale deep reinforcement learning." arXiv preprint arXiv:1912.06680(2019).
[7] Tom M. Mitchell. “Machine Learning.” (1997).
[8] Justesen, Niels, et al. "Deep learning for video game playing." IEEE Transactions on Games 12.1 (2019): 1-20.
[9] Wang, Fei-Yue, et al. "Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond." IEEE/CAA Journal of Automatica Sinica 3.2 (2016): 113-120.
[10] Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.
[11] Chaslot, Guillaume, et al. "Monte-carlo tree search: A new framework for game ai." Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 4. No. 1. 2008.
[12] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
[13] Zia, Tehseen, and Usman Zahid. "Long short-term memory recurrent neural network architectures for Urdu acoustic modeling." International Journal of Speech Technology 22.1 (2019): 21-30