EasyChair Preprint

№ 7095

Creation and Implementation of a Set of Game Strategies Based on Training Neural Networks with Reinforcement Learning

Dmitry Kozlov and Olga Polovikova

November 28, 2021

Creation and implementation of a set of game strategies based on training neural networks with reinforcement learning

D S Kozlov[1] **and O N Polovikova**[2]

[1] Postgraduate at the Department of Informatics, Altai State University, Barnaul, Prospekt Lenina 61, Russian Federation
[2] Associate Professor at the Department of Informatics, Altai State University, Barnaul, Prospekt Lenina 61, Russian Federation

E-mail: dima190892@mail.ru, ponolgap@gmail.com

Abstract
The study explores the problems of reinforcement learning and finding non-obvious play strategies using reinforcement learning. Two approaches to agent training (blind and pattern-based) are considered and implemented. The advantage of the self-learning approach with reinforcement using patterns as applied to a specific game (tic-tac-toe five in a row) is shown. Recorded and analyzed the use of unusual strategies by an agent using a pattern-based approach.

1. Introduction

Reinforcement learning methods are one of the most promising directions in the field of artificial intelligence today. Real-time decision support systems are a typical example of such a method of computer self-learning. One of the main ideas of such systems is that initially, learning agents have no idea about the rules operating in the environment in which the agent exists.

At the first stage, it is necessary to create the very environment in which the agents will interact. At the second stage, the creator of the system needs to set the objective function for the agent, which must be maximized (or minimized). At the third stage, it is necessary to create events for which we will reward the artificial intelligence and for which we will punish it and launch the first eras of learning. During the learning process, the best performing agents will be selected as the basis for the next generation of learning.

At third stage, the most important events take place from the point of view of the analysis of subsequent results. It is impossible to predict in advance the outcome of reinforcement learning due to the randomness of decisions made in the early stages of learning. The following outcome is likely: at an early stage, the strategy that is weakest in the long run brings the highest payoff at the early stage of the game, artificial intelligence sees that this strategy has the highest payoff and rejects other strategies, and the further evolution of learning takes place on the basis of obviously the most weakest strategy in the long term. Possible ways to solve this problem are to change the rules of the game (both quantitatively for individual parameters and adding new parameters, creating new key events in the game, re-evaluating the importance of the objective function, using combined algorithms (like alpha-beta pruning)).

Reinforcement learning has made tangible progress over the past decade, from the first timid attempts to defeating humans in games like *go*, even though classical programming ruled out such an outcome. Today, there is already an urgent need for specialists who can create reinforcement learning models and analyze the results obtained during the training.

It is difficult to overestimate how important it is to analyze the agent training process itself - it is ineffective to simply create an environment, set rules and an objective function, and launch the training process in the hope that artificial intelligence will determine the optimal strategy by itself. It is necessary to continuously adjust the learning process to eliminate obviously ineffective strategies using the methods described above.

The purpose of this study is to implement two approaches to constructing neural networks for generating strategies that are not obvious to a person using the example of one game (a game of tic-tac-toe five in a row). The essence of this work is to create neural networks that will generate non-obvious game strategies.

Based on the implementation of two approaches in training neural network agents, one can compare their results, accumulate game strategies for subsequent analysis. During the game, you can request a hint from any of the two agents to indicate the strongest solution.

2. Artificial intelligence in board games

Board games have always played a key role in the evolution of artificial intelligence. A clear set of well-known rules allows you to simulate an environment for human interaction or for creating a multi-agent learning environment using object-oriented programming methods in a short time. [3]

In 1950, Shannon's publication "Programming a Computer to Play Chess" was published; in 1951, Turing published pseudocode for a program capable of playing chess. Half a century later, chess programs based on the classical approach to creating game artificial intelligence are confidently defeating the world chess champions (Garry Kasparov lost his second match against Deep Blue in 1997, Vladimir Kramnik lost his match against Deep Fritz in 2006). After the Kramnik match, games between people and programs lost their relevance due to the total advantage over human abilities [6][8]

In 2017, DeepMind published a preprint of the AlphaZero program, which was based on reinforcement neural network learning algorithms. In the course of twenty-four hours of self-study, AlphaZero was able to achieve a level of chess that surpassed the strength of the best chess engines. The neural network approach in combination with Monte Carlo tree search has shown superiority over the traditional search in depth in practice. At the end of 2020, Vladimir Kramnik, together with DeepMind, published a report “Assessing Game Balance with AlphaZero: An Investigation of Alternative Rules in Chess [7].

Humanity has invented many board and computer games that are actively developing today using artificial intelligence. Each game requires a different approach to creating artificial intelligence.

As of today, the most studied are non-cooperative games with complete information, symmetrical, sequential. It is quite easy to apply to them the methods of both the traditional approach to artificial intelligence and methods based on reinforcement learning.

For games with incomplete information, creating artificial intelligence using the classical approach is difficult, and in such games, people beat machines more often. However, neural networks are changing the state of affairs in this area of knowledge: first, using neural networks, Limit Texas Hold'em was solved one on one, and in 2019 Pluribus neural network beat professional players in No Limit Texas Hold'em for six players. The model has learned some common strategies (such as donk betting, continuation bet, semi-bluff with draw)[4].

Almost every week, new ways of practical application of neural networks in various fields of human activity appear. The very structure of the world economy is changing, those companies that are not able to adequately respond to the widespread introduction of artificial intelligence methods into business will have to leave the market under the pressure of more highly efficient innovative companies. And at the origins of the development of general methods for creating artificial intelligence, developments in the field of AI were research in the field of discrete games with complete information. An important aspect in the practical application of reinforcement learning agents is the ability of these agents to adapt to the rapidly changing rules of the game. Modern investors are of little interest in an exchange bot that is not capable of an independent reaction to subtle changes in trading.[1][2]

3. Game development for AI

For the correct creation of the evaluation function, it is necessary to set the rules of the game. For this study, the option of playing on an 8 by 8 field was chosen (the size of the game tree with a 15 by 15 field will be 10^{105}, for an 8 by 8 field, the game tree will be reduced to 10^{27}). The sides sequentially make moves, trying to create a line of their elements exactly 5 figures in a row in any direction. If all

the squares are occupied and none of the opponents has a line of five of their elements in a row, then a draw is awarded.

Several types of agents will use reinforcement learning, the best of which will be determined later through a round robin tournament. The first agent will be trained almost entirely blindly. To speed up the learning process, the agent will receive a small penalty for each move made. The faster the batch ends, the higher the value of the objective function will be. The agent will receive 1000 points for winning the game, for creating a situation with the placement of four pieces in a row after his move, the agent will receive 800 points. For each of his moves, the agent will lose 10 points, for each move of the opponent, the agent will lose 20 points. The initial value of the objective function is zero. For a defeat, the second agent will receive a penalty of 1000 points.

The second type of agent has received many more patterns of behavior, and will also receive a bonus for every safe move in the game, which does not lead to a forced defeat in cases where the agent has a forced win.

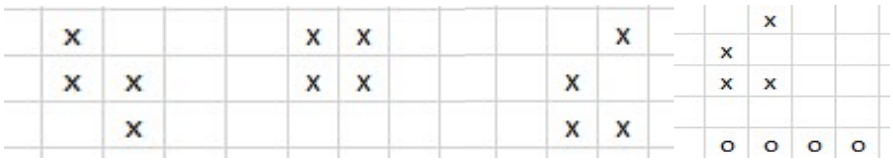

Figure 1. Simple patterns

Figure 1 shows examples of strong attacking patterns, the first two situations with correct answers to zeroes, you can try to defend with the move of zeroes, the third situation with the move of crosses leads to a forced winning of crosses in three moves. However, in the fourth situation, it is shown that the presence of a forced win in a position does not mean anything when the opponent has a forced win in one move.

For further analysis, a simple notation of games was developed for convenient storage of played games. It is also logical to use hash tables to optimize calculations, since then you will not have to re-evaluate the positions that can be obtained by rearranging moves.

The notation is based on the classical chess notation - the number of the move, who moved, the coordinates of the cell to which the move was made, is indicated, and the result of the game is also noted. The simplest example of notation: 1. Xc3 Os4 2. Xd3 Od4 3. Xe3 Oe4 4. Xf3 Of4 5. Xg3 # 1-0. The importance of keeping a record of games is also that at any time you can take a game played between agents and analyze it.

In the process of playing with itself, the network, using a Monte Carlo tree search, selects the maximum value of the objective function for a given search depth and alpha-beta pruning parameters.

The used architecture of the neural network is combined - backpropagation of an error together with reinforcement learning. Sometimes in the learning process, the agent degrades, when the strength of his game begins to decrease in the learning process. This happens sometimes due to incorrect alpha-beta clipping parameters when a branch is pre-estimated as unpromising for further calculation of options. The degradation of an agent should not be confused with overfitting; this problem is solved by the correct selection of parameters.

4. Evaluation function

The assessment function is based on the approach from chess, when the objective function is an integral indicator of the assessment functions of various game aspects, multiplied by the importance coefficient.

Table 1 – Parameters of evaluation function

Parameter	Designation	Coefficient of importance	Commentary
Contempt	C	0,1	This parameter shows the agent's attitude to a draw result. If it is

			negative, then the agent will consider a draw as negative for himself. If positive, then the agent will consider the draw outcome as positive for himself
Control over the center fields	S	0,15	This parameter shows how important the agent will consider the occupation of the central squares (the square, bounded by the squares c3, c5, f3, f5)
The minimum number of moves to win, if we imagine that the opponent does not make reciprocal moves	M	0,5	An important parameter for playing tic-tac-toe five in a row
The presence of strong attacking patterns (used only by an agent who bases his play on patterns)	P	0,25	A specific parameter used only by an agent who bases his play on patterns[5]

The formula for the evaluation function will be:

F (p) = 0.1C + 0.15S + 0.5M + 0.25P

for a pattern-based agent.

For an agent using blind learning, the parameter 0.25P will be discarded, and accordingly, the remaining 0.25 must be redistributed by three other parameters and will take the form: F (p) = 0.133C + 0.1995S + 0.665M.

This evaluation function is valid for the X-stitch command. For zeros, the following equality is true: F (p) = - F (q), which is symmetric with respect to zero and for negative values of F (p) shows the preponderance of zeros, for F (p) = 0, complete equality of the position, and for positive values, the preponderance of crosses.

The logic of the evolution of the learning process is as follows - a model, in the course of the game it plays with its previous version 50 games for crosses and 50 games for zeroes. With a winrate of 55% for noughts and 50% for noughts, the model is considered successful and replaces the current one. When the model changes, a random noise figure is added to each of the importance parameters of the scoring function.

The objective function for the second agent sometimes finds surprising solutions to maximize the value of the objective function - due to the depth of search and bonuses to the value of the objective function, it continues to lead the game, having a forced win in 1-2 moves. The agent chooses a solution that does not lose the forced win and continues the game.

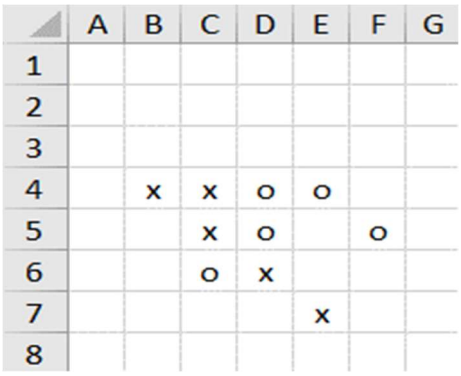

Figure 2. Demonstration of the game between the first and second agent

In this game situation, it is the move of crosses, and there are two immediately winning moves - Xa3 and Xf8, but searching in the Monte Carlo tree allows the agent to look a move ahead, to see that for any answer of zeros after Xc3, crosses have a move that ends the game instantly. The move Xc3 was chosen by the agent precisely because it allows to complete the pattern.

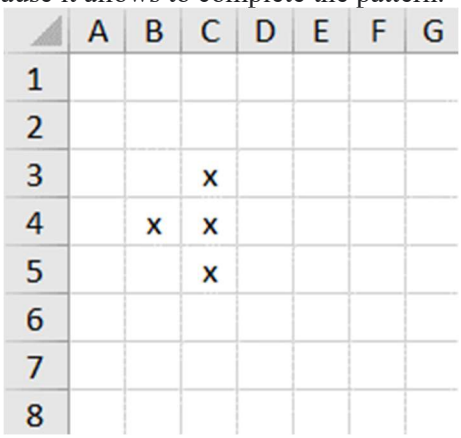

Figure 3. Drawing up a pattern in a won position

Not the most common behavior for self-learning systems. In the long term, such moves can help open up previously unexplored tactics when playing tic-tac-toe five in a row, thereby moving the level of the game forward.

5. Conclusions

It should be noted that without optimization, the computation time is unjustifiably delayed, and the power of modern computers does not allow calculating large game trees to the end in a short time. However, even if they could count them to the end - this would not be a reason to abandon optimization - a decrease in the potential number of users due to high system requirements did not benefit any of the software products.

In the course of training two agents, the advantage of a training approach using the game features of a particular game (behavior patterns) was revealed, however, two thousand training epochs were clearly not enough to compete with a person on equal terms. But in machine learning, such a number of epochs is a rather low indicator, and the loss to the network was not devastating.

There are several possible approaches to solving this problem: to increase the number of learning epochs up to 10,000, or to try to introduce additional parameters (the density of the arrangement of one's own cells, a more complete disclosure of the opponent's dangerous patterns). Also, during the training, cases of unusual agent behavior based on patterns were revealed. In some situations, he had a forced win in one move, but calculating his objective function in depth, when he found that his opponent could not win it earlier, he tried to build a new pattern, continuing the game, and receiving an additional bonus to the objective function for the completed move and for building a pattern. This feature, when applied

to other games (both multi-agent and with one agent), will be able to help reveal strategies that were previously hidden due to their non-obviousness.

The presented research highlighted the relevance of using neural networks to search for non-obvious game strategies. One of the possible further directions of research could be a study in changing the strategies of a neural network with changed rules of the game (the size of the playing field, rules from related games like Renju, Gomoku, and so on, the number of playing sides, two moves in a row).

References:

[1] Creswell A., Bharath A.A. Denoising Adversal Autoencoders // arXiv, 2017.

[2] Deep Generative Adversal Networks for Compressed Sensing Automates MRI/ M/Mardani et al. // arXiv, 2017.

[3] Emergent Tool Use from Multi-Agent Interaction // arXiv, 2019. https://openai.com/blog/emergent-tool-use/

[4] Goodfellow I.J., Pouget-Abadie J., Mirza M., Bing Xu, Warde-Farley D., Ozairy Sh., Courville A., Bengio Y.. Generative Adversarial Nets. http://arxiv.org/abs/1406.2661v1

[5] Goodfellow I. NIPS 2016 Tutorial: Generative Adversarial Networks. https://arxiv.org/abs/1701.00160v4

[6] Shannon C. Programming a Computer for Playing Chess // Philosophical Magazine, Ser.7, Vol. 41, No. 314. March 1950

[7] Tomašev N., Paquet U., Hassabis D., Kramnik V. - Assessing Game Balance with AlphaZero: Exploring Alternative Rule Sets in Chess. 2020

[8] Turing A. Computing Machinery and Intelligence // The Mind, V. 59 (1950), P. 433-460.