A comparison of machine learning methods using a two player board game

Drazen Draskovic,
University of Belgrade,
School of Electrical Engineering,
Department of Computer Engineering and Informatics,
drazen.draskovic@etf.bg.ac.rs

Milos Brzakovic,
University of Belgrade,
School of Electrical Engineering,
Department of Computer Engineering and Informatics

Bosko Nikolic,
University of Belgrade,
School of Electrical Engineering,
Department of Computer Engineering and Informatics

***Abstract* – The board games are usually performed by game theory algorithms: minimax and minimax with alpha-beta pruning. Tic-tac-toe (X - O) is the best-known two-player board game. The game tic-tac-toe, based on machine learning algorithms, has been shown in this research. The neural network has been developed and trained to play the game utilizing three implemented agents: an agent based on deep Q-learning, an agent based on policy gradient method and a random agent. The agent can play the game perfectly in the 10-minute training interval, on an average graphics processing unit.**

Keywords – machine learning, reinforcement learning, neural network, Q-learning, policy gradient method, deep learning

I. INTRODUCTION

The latest research in the field of machine learning is increasingly using neural networks [1][2]. The reinforcement learning is a type of learning that is used in situations when it is necessary to solve a problem by undertaking a series of actions, the joint effects of which provide a solution.

Nowadays, an increasing number of studies test the artificial intelligence capabilities in various games [3]. One of the most common experiments is that by using reinforcement learning, we encourage a deep neural network to explore the environment of a particular game, thereby revealing the tactics that leads to victory [4]. Furthermore, the evaluation of the network is carried out by competing against the people who are considered the masters of this game. At the moment of research within this study, artificial intelligence managed to win the championship in the game "Go" [5][6].

The aim of this paper is to compare the most important methods of reinforcement learning and their application in solving the famous X-O (Tic-Tac-Toe) game. It will be shown that a deep neural network can learn in a few minutes that actions from the current state lead to the maximum possible result. In order to come up with algorithms that can learn high-complexity games, it is necessary to start from simple principles and try out various techniques in simpler environments.

This section will explain the logical abstraction of the agent and the environment. Also, by introducing Markov's decision-making process, and the difference between the function of the policy and the value it will describe the process of implementing the experiment.

A. Agent

The agent is an abstraction of a part of the system that interacts with the environment and, based on the current state, selects an action that will potentially lead to the greatest reward. An example of an agent is a human player that makes the moves and by doing so he transitions from one state to another. Memory is often added to this part of the system, thereby allowing state, rewards, and actions to be remembered. This kind of implementation allows for better distribution of the reward, as well as batching process.

The role of the reward is to implicitly define the agent's goal. The reward should tell the agent what to do, not how to do something. In the case of chess, the prize would be +1 for the winner when he goes to the last state where he wins, -1 in case he goes to the state he loses or 0 in case of unresolved outcome.

B. Environment

The environment is a logical part of a system that has its current state and, under the influence of the action, switches to the next one and eventually returns a reward. One of the most important parts of the system configuration is to determine the logic for giving a reward. A classic example of the environment is the chessboard, where action is actually a valid move in this game, and the game state can be described by the positions of the white and black figures on the board.

It is important to see the difference between the two types of environment: deterministic, in which the state is clearly defined in which environment it passes in a particular action, and stochastic in which the system from one state under action can pass with certain probabilities in several different states. Further in the paper, the environment is defined as deterministic.

C. Markov decision process

Markov decision process (MDP) provides a theoretical framework for modeling the decision-making process in situations where the future is partly random and partly under the control of an agent [7]. The key claim of MDP is the following: if we look at the present, the future is independent from the past. In other words, the new state and reward depend only on the previous state and the actions taken, and not the entire process history. In the case of the X-O game, this means that if you are looking for the best move by observing the board, it is irrelevant which moves preceded the current state. In practice, and bearing in mind various reasons, the problem is often not presented as a MDP. For example, in the problem of autonomous driving, the same action in the same conditions can have different outcomes, because it depends on additional parameters such as traffic jam [8].

D. Policy and value

Value is a real number, which provides an assessment of the state or how good it is to pull some action from specified state.

978-1-5386-9301-8/19/$31.00 ©2019 IEEE

Policy indicates which action should be taken in a certain state. In theory, the optimal policy is the method of choosing the action in which an agent receives the maximum reward. It is considered that the ideal goal of reinforcement learning is the optimal policy. In practice, it is not possible to achieve an optimal policy, while striving to discover a good policy, due to specific constraints on many issues.

Figure 1 provides an example of iteration through values and policy. The board has 4x4 fields. The player aims at reaching one of the two corners, which are on the secondary diagonal of the matrix. In each field it is possible to perform 4 actions moving the player deterministically: up, down, left and right, respectively.

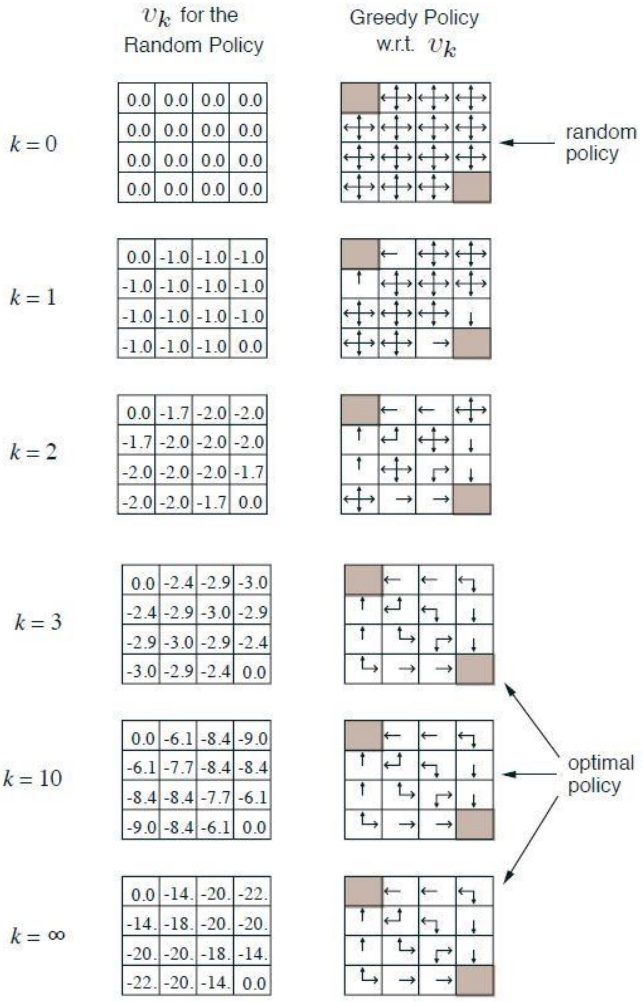

Fig. 1. Example of policies and values

In Fig. 1, on the left, there are matrices in whose fields there are values that indicate quality to be in that field. In the same figure, on the right, the policy is shown. In the first iteration (k=0), it can be noticed that policy requires one of the four possible actions to be selected in random fields in a random way. This principle of selecting the next move is called a random policy. In the next set of iterations, field values change and the agent finds it more convenient to be in the fields that are closer to the goal. Therefore, if politics behaves greedy, it will choose actions leading to the highest value field. The result of learning is the optimal policy that leads to the greatest gain.

There are two variants of learning: on policy and off policy [9]. When learning on the policy, the agent updates the values of the current state based on the values of the following states, observing the action of the current policy. An example of this learning is the SARSA on-policy algorithm, as well as all Policy Gradient methods [2]. With off policy, the agent updates the values of the current state based on the values of the following states, observing the action that follows the greedy policy, and then take action against the current policy. An example of this learning is the Q-learning algorithm.

E. Temporal Difference and Monte Carlo process

Temporal Difference (TD) is a bootstrapping method where learning depends only on changes made in the current move [10]. It can be said that this method updates the approximation to aim for an approximation. The main problem of this type of learning is that the values depend on the initial state of the network, as shown in Figure 2. The advantage is that you do not have to wait until the end of the game to update and you do not need to know the environment model.

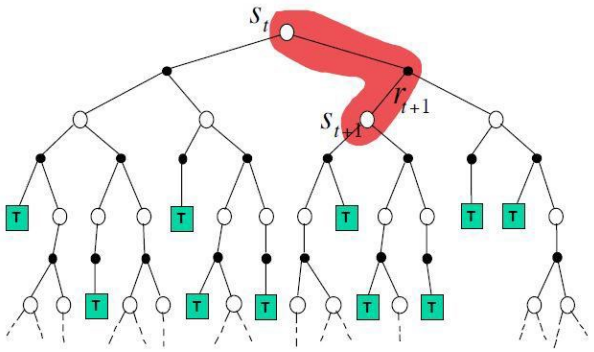

Fig. 2. Example of a Temporal Diference tree

Monte Carlo's learning approach is dependent on experience gained throughout the game [11]. In this way, the problem is solved on the basis of averaging a return reward during an episode. The disadvantage of this approach is high variance and, additionally, more parties are needed to reach the level of learning, as is the case in Temporal Difference [12]. Figure 3 shows the Monte Carlo Tree.

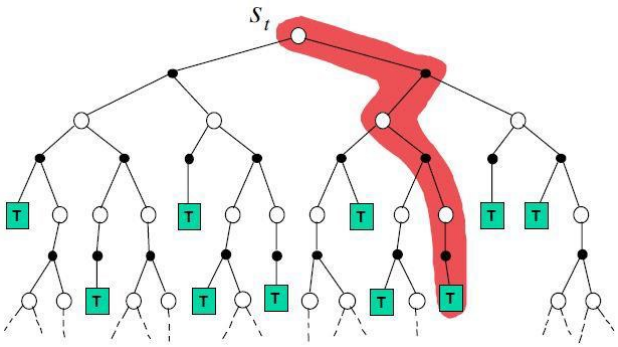

Fig. 3. Example of a Monte Carlo tree

II. THE ANALYZE OF PROBLEM AND USED ALGORITHMS

X-O is a game where two players play alternately, on a 3x3-size board. The goal of the game is to connect three characters X or O horizontally, upright or diagonally. There are 3^9 different states, but most are illegal and unattainable. The environment can generate 6046 different states and the

goal of the agent is to learn the best way to approximate these states. The game is simple and can be solved using the classic minimax algorithm, but this study demonstrates the application of more innovative solutions using machine learning.

A. Reinforcement learning

Reinforcement learning is a type of machine learning in which an agent as a neural network, tries to maximize profits while interacting with the environment [13]. The key assumption is that it is not known which of the actions taken was right in the given context, and which is not. Otherwise, it would be a problem of supervised learning. At each step at time t, the agent executes the action A_t, receive the observation O_t and reward R_t. If a sequence were made, the history of interaction would be obtained:

$$H_t = O_1, R_1, A_1, O_2, R_2, A_2, \ldots, A_{t-1}, O_{t-1}, R_t \quad (1)$$

Formally, the state is a function of history: $S_t = f(H_t)$. If Markov's decision-making process is concerned, we can apply the information that the current state depends only on the previous and not the entire history:

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, S_2, .., S_t] \quad (2)$$

On the basis of (2), the current reward can be defined as a mathematical expectation of the reward in step t + 1, observing the state s and pulling the action a:

$$R_s^a = E[R_{t+1} | S_t = s, A_t = a] \quad (3)$$

Finally, the total gain G_t can be calculated as the sum of all future rewards with a reduction factor: (4)

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = k = \sum_0^\infty \gamma^k R_{t+k+1}$$

There are two main approaches aimed at maximizing G_t:

- *Learning of value function* shows the advantage of being in the current state, following a certain policy.

  $$V_\pi(s) = E_\pi[G_t \mid S_t = S] \quad (5)$$

  The value of the state s is the mathematical expectation of the current gain for the state s, following the policy p. The representative of this approach is Q-learning.

- *Learning of policy function* directly decides which action to choose in a specified state.

  $$a = \pi(s) \quad (6)$$

  The action is given directly through the policy function above a specified state s. The representative of this approach is policy learning.

B. Q-learning

For the final Markov decision-making process, it has been shown that if Q-learning has an infinite time for exploring the environment, it can determine the values of all states and, therefore the optimal policy.

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a] \quad (7)$$

The Q-value (the function of state and action) is introduced, which numerically expresses how good it is to be in state s and play the action a. It is possible to express the value of the state through the function of state and action using (8).

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s, a) \quad (8)$$

$\pi(a|s)$ is a probability of withdrawal of the action a, provided that the agent is in the state s.

Using the Bellman equation, the current and future Q-value can be combined:

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \quad (9)$$

The value of the state s for action a is equal to the sum of the current reward and the reduced value of the future state S_{t+1} for the action A_{t+1}. The SARSAMAX algorithm, also known as Q-learning, is obtained by means of combining the Bellman equations for iteration, and the principle of the temporal difference in learning without policy. Formula (8) is used to update the value of the state and the action.

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma * max_a Q(S', a') - Q(S, A)) \quad (10)$$

There are two main problems of the proposed Q-learning algorithm: 1) in practice, it is not possible to provide infinite research time; 2) if more complex games are considered, the number of conditions is huge and it is not possible to place all variants in the memory. The solutions to the problem are the following: 1) it should be fulfil a good policy that is not optimal; 2) a huge number of states are approximated by a finite number of states, using a deep neural network. In this way Deep Q-learning is obtained.

C. Policy gradient learning

While Q-learning deals with learning the values of states and actions to derive good policy from them, the Policy Gradient (PG) method directly teaches the policy. If the parameters of the neural network are moved from θ, the gain function J can be represented according to the following formula:

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) \quad (11)$$

By combining (8) and (11), (12) is obtained:

$$J(\theta) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s, a) \quad (12)$$

$d^\pi(s)$ is a stationary distrubution sequences in history using policy π_θ.

Using the PG theorem, a new rule for calculation the gradient gain, can be reached:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s, a) \triangleq \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q^\pi(s, a) \nabla_\theta \pi_\theta(a|s) \quad (13)$$

There are several variants of PG, one of the basic REINFORCE method that uses the Monte-Carlo principle of

updating. Using (7) and (13), the formula for the network parameters updating can be extracted:

$$\nabla\theta J(\theta) = E\pi[Q^{\pi}(s,a)\nabla\theta ln\pi_{\theta}(a|s)] = E\pi[Gt\nabla\theta ln\pi_{\theta}(At|St)] \quad (14)$$

The REINFORCE algorithm can be demonstrated in several steps:

1) initialize θ randomly,
2) generate a sequence of batches according to policy π_{θ} : $S_1,A_1,R_1,S_2,A_2,R_{2,\ldots,}\ S_T$;
3) in iterations $t = 1,2, \ldots, T$:
   3.1) evaluate the estimate of Gt;
   3.2) update values according to the following formula:

$$\theta \leftarrow \theta + \alpha * \gamma t * Gt * \nabla\theta ln\pi_{\theta}(At|St) \quad (15)$$

D. Batching

The agent can have a memory in which it can remember the history of different parties. In case a Monte Carlo method is used, it is not necessary to update at the end of the game. It is possible to save a history in order to periodically take a number of instances (batch size) from the memory and simultaneously update the network.

It has been shown that the method of batching is exceptionally favorable when training is carried out on a graphics card (GPU). The update of the batch is executed very fast, as the dataflow graph is replicated by the memory of the GPU allowing parallel update of the network parameters.

III. The Research Results

The research demonstrated the progress of an agent using Q-learning algorithm and an agent using the policy gradient method. Network evaluation has been performed by summing up the number of batches lost by a smart agent against a random agent.

A. Q-learning

Fig. 4 shows the learning process of Deep Q-learning agent. In order to better show the fall line, the noise is removed, so that the y-axis set the average results of 20 episodes.

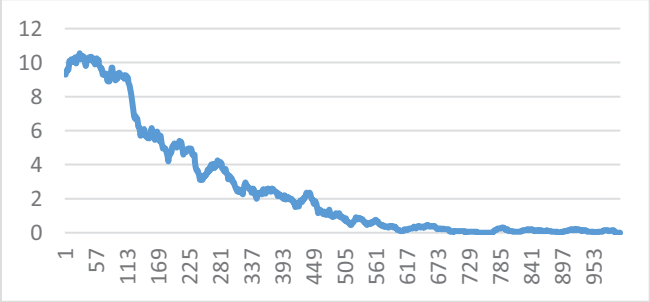

Fig. 4. The number of points that DQAgent loses after the episode

The graph is initially idle, because it takes some amount of time in order to fill the batch memory. After 60 episodes, the network starts learning very quickly, and after 750 episodes, the AI agent is playing almost without losing a game. The entire training process, accompanied by saving the network state, as well as the logging process, takes 10 minutes. The value of the gamma parameter is 0.5 and provides a high reward for the last move, as well as the sufficient impact of the last move of the same player. If the gamma is too high, it would be a rewarding move in the rugged past, which is not responsible for winning the players in this game. The best value of learning rate is approximately about 10^{-5}, because it uses batching in which a lot of updates are executed in parallel. The learning rate parameter is set to value 0,00002. The learning decay coefficient plays a role in reducing the learning factor over time. If the learning factor reduction is high, the algorithm will not be able to slow down the graphic quickly enough and the learning process will simply stop. If the learning coefficient were reduced, then the graph would not calm down, but would also have a high variation, even when it has shown the tendency toward zero. The learning decay is set to value 10^{-10}. The batch size and batch freq parameters depend on the power of the GPU. For the GPU in which training was conducted, it was sufficient to have 1024 updates (batch size) for every 64 episodes (batch freq). Parameters epsilon = 1.00 and epsilon decay = 0.0005 show that the policy in this game that was monitored by the agent during the update could be completely random. Epsilon did not have to be reduced, because the number of game states was small.

B. Policy gradient learning

Figure 5 shows the process of a PG agent learning. Similar to Q-Agent, noise is removed. Between 60 and 140 episodes an agent quickly finds a good policy, then the learning speed drops, so a total of 500 episodes is needed to learn a good enough policy. The entire training process, together with the network memory and the logging process, takes 12 minutes. The problem of high variance can be noticed between episodes 250 and 450. The value of the gamma parameter is set to the same value of 0.5. Also, the learning rate parameters, batch size and batch freq remained at the same values as in Q-learning. The Epsilon parameter does not exist because it is on-policy learning type. The learning decay is set to value 8^{-10}. The role of this coefficient is to reduce the learning factor over time. It is extremely important to adjust this parameter well with the gradient policy method so that the agent would complete the training successfully. It had been found in this study, that if the value of this parameter was 2^{-10}, the agent would lose one of the 20 games played against the random players on average.

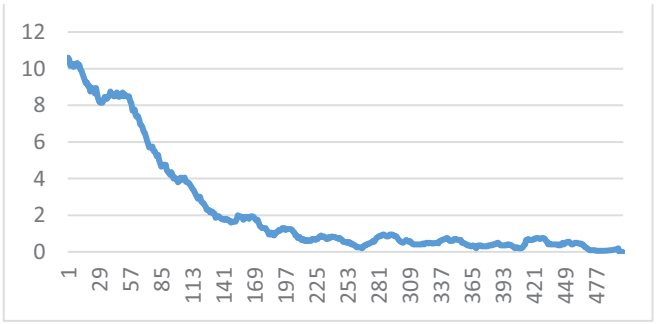

Fig. 5. The number of points that PGAgent loses after the episode

IV. Conclusion

In this research, an agent that plays the Tic-Tac-Toe game, and the one that is able to play the game perfectly in a 10-minute training interval on an average GPU, has been presented. The agent is realized with three different method implementations. The proposed solutions allow training process using a neural network.

This paper highlights details that greatly accelerate the learning process, such as representation of the environment, propagation of the rewind, and the use of batching process. It has been described how hyperparameters affect the update process itself by comparing two methods of reinforcement learning. Along with the adjustment of similar parameters, it has been shown that the Q-learning method is faster, because the prediction is rarely used due to the use of epsilon-greedy policies with a high randomness parameter, but it needs several episodes to achieve a good policy. The PG method is slower because it uses predictions during updating parameters (on-policy), but requires fewer episodes to achieve satisfactory results.

Both the research and the realization of the proposed agent represent an excellent template that could be used in more complex games. If Q-learning and PG methods do not converge fast enough, they can make their connection and create an Actor-Critic method, possibly A3C, which has been proven to be a very good method for learning more complex games.

ACKNOWLEDGMENT

This research is supported by the Ministry of Education and Science of the Republic of Serbia, project number TR-32047.

REFERENCES

[1] L. Xiaomin, P. Beling, R. Cogill, "Multiagent Inverse Reinforcement Learning for Two-Person Zero-Sum Game," IEEE Transactions on Games, vol. 10, pp. 56-68, March 2018.

[2] M. Xu, H. Shi, Y. Wang, "Play games using Reinforcement Learning and Artificial Neural Networks with Experience Replay", 17th IEEE/ACIS International Conference on Computer and Information Science, Singapore, Singapore, June 2018.

[3] I. Kachalsky, I. Zakirzyanov, V. Ulyantsev, "Applying Reinforcement Learning and Supervised Learning Techniques to Play Hearthstone," 16th IEEE International Conference on Machine Learning and Apllications, Cancun, Mexico, December 2017.

[4] K. Shao, D. Zhao, N. Li, Y. Zhu, "Learning Battles in ViZDoom via Deep Reinforcement Learning," IEEE Conference on Computational Intelligence and Games, Maastricht, Netherlands, August 2018.

[5] D. Silver, R. Sutton, M. Muller, "Reinforcement learning of local shape in the game of Go," Proceeding of the 20th international joint conference on Artificial intelligence, pp. 1053-1058, Hyderabad, India, January 2007.

[6] D. Silver, et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, pp. 484-489, January 2016.

[7] E. Beaudry, F. Bisson, S. Chamberland, F. Kabanza, "Using Markov decision theory to provide a fair challenge in a roll-and-move board game," Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, Dublin, Ireland, August 2010.

[8] E. Walraven, M. Spaan, B. Bakker, "Traffic flow optimization: A reinforcement learning approach," Engineering Applications of Artificial Intelligence, Elsevier, vol. 52, pp. 203/212, June 2016.

[9] D. Silver et al., "Deterministic Policy Gradient Algorithms," Proceedings of the 31st International Conference on Machine Learning, vol.32, Beijing, China, June 2014.

[10] Y. Osaki, K. Shibahara, Y. Tajima, Y. Kotani, "An Othello evaluation function based on Temporal Difference Learning using probability of winning," IEEE Symposium On Computational Intelligence and Games 2008, Perth, WA, Australia, December 2008.

[11] A. Fabbri, F. Armetta, E. Duchene, S. Hassas, "Knowledge Complement for Monte Carlo Tree Search: An Application to Combinatorial Games," IEEE 26th International Conference on Tools with Artificial Intelligence, Limassol, Cyprus, November 2014.

[12] K. Kao, I. Wu, S. Yen, Y. Shan, "Incentive Learning in Monte Carlo Tree Search," IEEE Transactions on Computational Intelligence and AI in Games, vol. 5, no. 4, pp. 346-352, December 2013.

[13] A. Jeerige, D. Bein, A. Verma, "Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing," IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, USA, January 2019.