Reinforcement Learning: Playing Tic-Tac-Toe

Jocelyn Ho[1], Jeffrey Huang, Benjamin Chang, Allison Liu and Zoe Liu

[1]Georgia Institute of Technology

ABSTRACT

Machine learning constructs computer systems that develop through experience. Applications surround disciplines in daily life ranging from malware filtering to image recognition. Recent research has shifted towards maximizing efficiency in decision-making, creating algorithms that quickly and accurately process patterns to generate insight. This research focuses on reinforcement learning, a paradigm of machine learning that makes decisions through maximizing reward. Specifically, we use Q-learning – a model-free reinforcement learning algorithm – to assign scores for different decisions given the unique states of the problem. Widyantoro et al. (2009) have studied the effect of Q-learning on learning to play Tic-Tac-Toe. However, the study yielded a win/tie rate of less than 50 percent. We believe that does not represent an effective algorithm to exploit the benefits of Q-learning fully. In the same environment, this research aims to close the gaps in the effectiveness of Q-learning while minimizing human input. Data were processed by setting the epsilon value as 0.9 to ensure randomness, then consecutively decrease with a constant rate as possible states increase. The program played 300,000 games against its previous version, eventually securing a win/tie rate of approximately 90 percent. Future directions include improving the efficiency of Q-learning algorithms and applying the research in practical fields.

Introduction

Background

AlphaGo is the first computer to defeat a human professional player in Go, a board game that contains 10^{172} possible position combinations (Silver et al., 2016). AlphaGo continued to improve by training against artificial and human intelligence, eventually defeating first-tier players around the world (Deepmind, 2021). AlphaGo opens new avenues for artificial intelligence to advance in disciplines dominated by humans. With the example of AlphaGo, artificial intelligence can be applied to fields existing in fiction, such as autonomous robots, chatbots, and trading systems (Ritthaler, 2018).

Motivation

Given the trend of artificial intelligence like AlphaGo, we intend to solve real-world problems through a similar automated approach. Board games are the simplest means to train a computer given their adequate environment of having certain rules and possibilities for outcomes (Ritthaler, 2018). Under a perfect information system, artificial intelligence could calculate exact outcomes and derive a goal of interest by figuring out how to maximize utility. Considering the multitude of existing board games, Tic-Tac-Toe proved to be the simplest and most comprehensible game. Though possible positions (360,000) are relatively less than Go, Tic-Tac-Toe still provides the complexity and nuance to be solved with deep neural networks such as reinforcement learning (Ritthaler, 2018).

Research Context and Goals

There are at least two methods for a computer to reach an outcome from previous actions: decision tree and reinforcement learning (Sutton et al., 2018). Decision trees are graphs that contain branches to represent specific conditions, with outcomes located at their ends (Sutton et al., 2018). Each node in the decision tree constitutes a given action for which the condition is met. However, the decision tree requires sorting all the possible combinations, taking an enormous amount of memory. Hence, many people use an extended version – the min-Max algorithm – to estimate the quality of action by backtracking results. The opponent attempts the optimal move and a value is assigned to the outcome for this algorithm (Sutton et al., 2018). The depth of the tree depends on the number of moves; to optimize performance, the evaluation of function with respect to depth is used to estimate the final value of the match. Nevertheless, the algorithm uses massive state space and has no evaluation functions that are sufficient for an immense amount of outcomes. Underlying the aforementioned flaws of different algorithms, we intend to use reinforcement learning, an algorithm that automatically finds a balance between exploration of pathways and exploitation of knowledge – to train the computer to play Tic-Tac-Toe (Friedrich, 2018).

Through reinforcement learning, the computer will optimize rewards through interaction with the environment and update itself with a better prediction based on its experience. In Q-learning, each action comes with a reward based on the outcome, a value calculated based on the current state, and optimal action from the previous state (Watkins, 1992). These variables allow machines to calculate a new value and repeat the process until the game terminates. After many consecutive simulations, machines would experience different patterns, allowing them to accurately estimate the probability of winning the game (Watkins, 1992). We aim to demonstrate a high success rate through Q-learning and yield a success/tie rate above the existing literature today.

Literature Review

Many studies related to the utilization of machine learning through simple board games arise since the nascent of computer-operated programs such as those in chess. In 1949, Claude Shannon first started to develop a computer-operated chess program (Shannon, 1950). Building on Shannon's work Alan Turing developed a computer-simulated checker player (Morris and Jones, 1984). In 1966, MacHack 6 by Greenblatt (1967) became the first computer-operated chess program that defeated a human player in a tournament. The program uses the approach of searching techniques where the state is the board configuration and operations are all potential steps. It uses a game tree with a depth of four levels, and choosing certain choices and levels will maximize a specific utility function.

In 1989, Watkins first introduced Q-learning, a model-free reinforcement learning algorithm (Watkins, 1989). Ever since the introduction of the algorithm, many studies have built themselves upon it, such as those of Even-Dar and Mansour (2001) and Hu and Wellman (2003). Several previous studies have covered the utilization of reinforcement learning on simple board games. Widyantoro et al. (2009) apply a Q-learning algorithm to play Tic-Tac-Toe. In the study, a new update rule is established by only updating the Q-value when transitioning from the final move back to the first move. Though its partial-board representation yields comparable results to that of a human player, its full-board representation only has a win/tie rate of less than 50%. Thus, our study will implement a new design for the reward setup and utilize optimistic initialization to encourage the agent's learning and improve the efficiency of this study.

Methods

In reinforcement learning, no prior information on the potential value of actions is given. The essential goal is to maximize the cumulative rewards of tasks through exploring the environment and exploiting learned materials. In Tic-Tac-Toe, the learning agent repetitively plays a standard game: a 3 x 3 board where three X's or O's are placed

consecutively diagonally, vertically, or horizontally to win. The learning agent must consider both the immediate and subsequent future rewards to achieve a combination of delayed rewards.

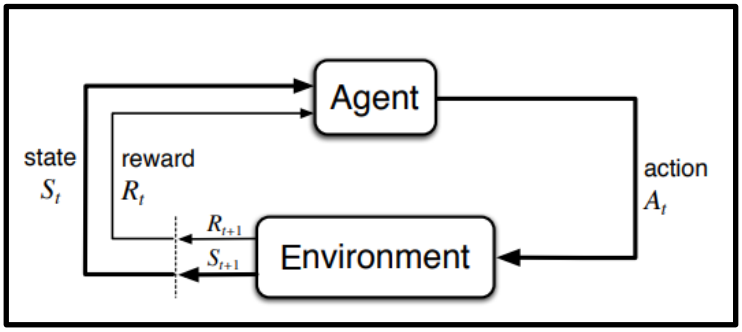

Figure 1. A figurative representation of the State-action-reward-state-action (SARSA) algorithm, which follows the Markov decision process policy. At time t, the agent will take action A_t, which responds to the environment and receives reward R_t and state S_t.

In a reinforcement learning environment, the learning agent's policy is a learned strategy dictating the agent's actions as a function of the environment and its current state. The reward signals are numerical rewards after an action. Agents alter their policy to maximize reward signals: strategy is established by prioritizing actions with high rewards in the future. With the eventual reward being +1 after a win and punishment being -1 after a loss, the agent's policy will alter to avoid the actions leading to the low reward.

A finite Markov Decision Process consists of the interactions between the agent and environment, with factors such as the action, state, and reward. This process can be represented by a stochastic series of possible actions that motivate the agent to seek various rewards, which leads to partially random and partially machine-controlled outcomes. Therefore, it operates through a value function of step t. In the current state s, the agent could choose action a. The system is moved to the next state s', which only depends on the current state and the action. A corresponding reward R is given after the action. The probability of state s' could be represented by the transition function $pa(s, s')$, a solution to the Bellman equation, which shows the relationship between the current state and successor states. In the state s, action a is decided by policy π. The probability function p then decides the corresponding reward r and the next state s'. By weighing possible steps on the probability of occurring and averaging them, the Bellman equation provides the immediate reward by action a in state s and the maximum reward in the next state.

$$\boldsymbol{Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot maxQ(s_{t+1}, a) - Q(s_t, a_t))}$$

Equation 1: Bellman Optimality Equation is called iteratively to update the Q value until it converges to its optimal. On each update, the Q value is modified by adding earned reward $\boldsymbol{r_t}$ and the estimated optimal future value $\boldsymbol{maxQ(s_{t+1}, a)}$. These values are affected by learning rate α, which determines the step size towards a minimum loss function for each iteration, and discount factor γ, which determines how much the agent should consider the calculated optimal future value into consideration for its next move.

In our research, the Bellman equation is used in the Q-learning model. In the model, the machine assigns the Q value, which is the expected reward with a higher value being more desirable, to every state-action pair. The Q values are updated iteratively based on the current state, future states, and potential actions. In Tic-Tac-Toe, the state is the board position while the action is the game move. At the end of each match, the result is associated with the move that caused the result. The machine will then work back the game history recursively and update the Q values. Next, the epsilon-greedy strategy is employed to ensure the agent's familiarity with all game moves instead of only

reinforcing Q values that are already high. The epsilon-greedy strategy is a method that balances exploration and exploitation by agent by choosing the two choices randomly: a random move with probability 1-ε from the Q-table or a random move with probability ε.

Results and Data Analysis

Through reinforcement learning, the Q-learning algorithm, coded with Python, allows an agent to find the solution that yields the greatest reward. When the agent is using the epsilon, the agent explores with a random move. During each move, a reward is collected and the Q-value is generated from a given state and action with the Q-learning equation. The data will then be stored in the Q-table, where each slot for the Q-table corresponds to a board state. The discount factor and learning rate affect the convergence of winning/tie rates. By following the policy that yields the most reward, the machine learns from the Q-table.

As the computer explores the possibilities, its epsilon decreases to encourage the agent to play the optimal action while ensuring a certain degree of exploration. Using a randomly generated decimal between 0 and 1, the computer determines whether it plays under the trained policy. The computer will play a random move if the generated number is within the epsilon value. However, if this generated number is greater than the current epsilon, the algorithm will be processed, utilizing the Q-learning approach.

For the first one thousand episodes, we set the epsilon to 0.9 to allow the computer to generate random moves, producing enough data sets to be stored in the Q-table; the table is updated with the Bellman equation. We then decrease the epsilon by ((1-epsilon)*10)/episodes to reduce the rate of exploration of the board, allowing the agent to have a greater possibility to play with optimal policy. By evaluating the value the Q-learning equation computed, the algorithm enables the computer to play strategically by following the policy that yields the greatest rewards.

Similarly, the opponent in the game was coded with the same Q-learning algorithm to yield greater results. However, different from the player, the opponent does not update its Q-table during the plays. The opponent only updates its Q-table to the agent's Q-table once every 1000 episodes to increase the efficiency of the agent's policy.

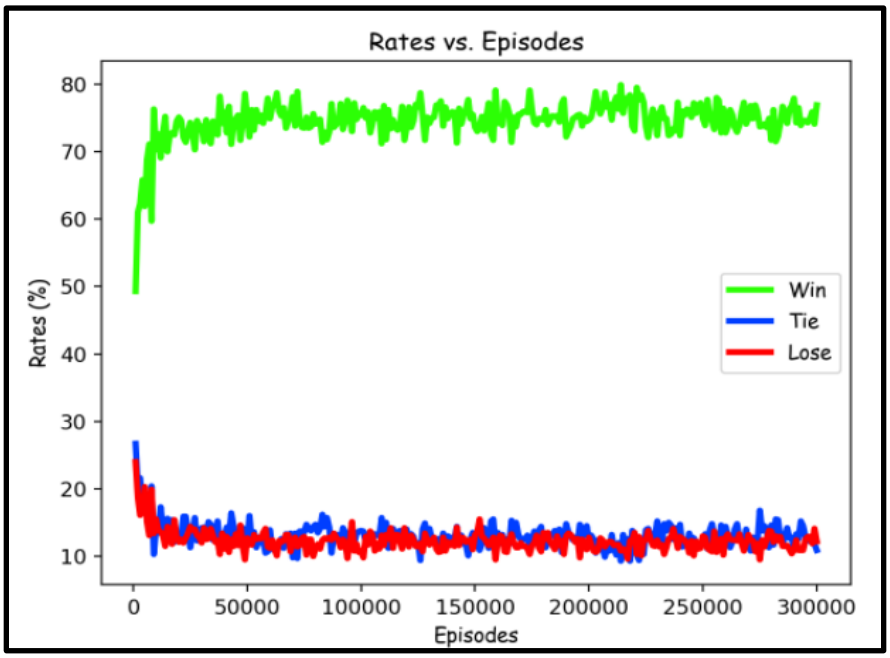

Figure 2. After playing 300,000 episodes, the agent will win on an average of 75%, break/tie for about 15%, and lose 10% of games played

Conclusion

Our research has shown that a learning agent, via reinforcement learning, can master playing simple games such as Tic-Tac-Toe with a high winning rate after receiving a sufficient amount of training. Throughout our experiment, our agent has developed its playing strategy by utilizing the Bellman equation and the Q-learning algorithm to recall its previous moves and discover the optimal Tic-Tac-Toe action. By the result of our experiment, the agent has around a 90% win/tie rate after 300,000 episodes of training. We believe that the winning rate can increase even more to 100% with a better tuned Q-function and more training episodes.

Suggestions

Future research will focus on improving the machine's learning strategies, as well as the related winning rate for the agent. We are looking forward to discovering a strategy to maximize learning without relying on long-term training, which is more efficient as it will take a much shorter time for machines to complete.

We wish to apply our research regarding the adjustments of the learning rate, decay exploration rates, as well as their effects on agents' abilities not only on Tic-Tac-Toe but also on similar games that can be advanced with reinforcement learning. Though the states, actions, and rewards vary by program, the fundamental quality of Q-learning algorithms remains model-free, and thus the research can be applied to a wide variety of programs, ranging from games to other fields where credible unsupervised learning is essential to quick operations and exemplary results. Finance, business, medicine, and industrial robotics are some examples of these fields.

Acknowledgements

I would like to express my deep and sincere gratitude to Benjamin Chang, freshman from John Hopkins University, and Jeffrey Huang, senior from Pacific American School, for their unwavering support in co-authoring and providing constructive feedback and reviews.

References

AlphaGo. DeepMind. (n.d.). Retrieved July 20, 2022, from https://deepmind.com/research/case-studies/alphago-the-story-so-far

Cleanwater Analytics. (2018, January 18). Using Q-learning and deep learning to solve tic-tac-toe (2017 Clearwater DevCon). YouTube. Retrieved July 20, 2022, from https://www.youtube.com/watch?v=4C133ilFm3Q

Even-Dar, E., & Mansour, Y. (2001). Neural Information Processing Systems. In Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic. Retrieved July 20, 2022, from https://dl.acm.org/doi/abs/10.5555/2980539.2980734.

Friedrich, C. (2018, July 20). Part 3-tabular Q learning, a tic tac toe player that gets better and better. Medium. Retrieved July 20, 2022, from https://medium.com/@carsten.friedrich/part-3-tabular-q-learning-a-tic-tac-toe-player-that-gets-better-and-better-fa4da4b0892a

Hu, J., & Michael. (2003). Nash Q-Learning for General-Sum Stochastic Games. Journal of Machine Learning Research. https://doi.org/10.1162/1532443041827880

Morris, F. L., & Jones, C. B. (1984). An early program proof by Alan Turing. IEEE Annals of the History of Computing, 6(2), 139–143. https://doi.org/10.1109/mahc.1984.10017

Shannon, C. E. (1950). Xxii. programming a computer for playing chess. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 41(314), 256–275. https://doi.org/10.1080/14786445008521796

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and Tree Search. Nature, 529(7587), 484–489. https://doi.org/10.1038/nature16961

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. The MIT Press.

TORRES.AI, J. (2021, September 24). The bellman equation. Medium. Retrieved July 20, 2022, from https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7

Watkins, C. (1989). Learning from Delayed Rewards (thesis). Cambridge University, Cambridge.

Watkins, C. J., & Dayan, P. (1992). Technical note. Reinforcement Learning, 55–68. https://doi.org/10.1007/978-1-4615-3618-5_4

Wunder, M., Littman, M., & Babes-Vroman, M. (2010). International Conference on Machine Learning. In Proceedings of the 27th International Conference on Machine Learning.