Generalized agent for solving higher board states of tic tac toe using Reinforcement Learning

Bhavuk Kalra
Department of Computer Engineering
Thapar Institute of Engineering and Technology
Patiala, India
bkalra_be18@thapar.edu

Abstract—Tic Tac Toe is amongst the most well-known games. It has already been shown that it is a biased game, giving more chances to win for the first player leaving only a draw or a loss as possibilities for the opponent, assuming both the players play optimally. Thus on average majority of the games played result in a draw. The majority of the latest research on how to solve a tic tac toe board state employs strategies as Genetic Algorithms, Neural Networks, Co-Evolution, and Evolutionary Programming. But these approaches deal with a trivial board state of 3X3 and very little research has been done for a generalized algorithm to solve 4X4,5X5,6X6 and many higher states. Even though an algorithm exists which is Min-Max but it takes a lot of time in coming up with an ideal move due to its recursive nature of implementation. A Sample has been created on this link [14] Min Max Implementation to prove this fact. This is the main problem that this study is aimed at solving i.e providing a generalized algorithm(Approximate method, Learning-Based) for higher board states of tic tac toe to make precise moves in a short period. Also, the code changes needed to accommodate higher board states will be nominal. The idea is to pose the tic tac toe game as a well-posed learning problem. The study and its results are promising, giving a high win to draw ratio with each epoch of training. This study could also be encouraging for other researchers to apply the same algorithm to other similar board games like Minesweeper, Chess, and GO for finding efficient strategies and comparing the results.

Index Terms—Reinforcement Learning, Game Theory, Artificial Intelligence Systems

I. PRE-EXISTING ALGORITHMIC SOLUTIONS

It is to be noted that tic tac toe is a biased game towards the first player and gives more winning chances to the player who starts the game leaving the opponent only a loss or draw as possibilities. However, it is already known and specified in [1] that if both players play rationally, the game would end in a draw. A player employs a strategy for playing the game in such a way that he either wins or draws but never loses. Efforts have been made on creating a fast and effective tic tac toe playing algorithm and similar other board games which employs a state-space search, ex. A modification of the Tic Tac presented in [2]. Below are listed some of the major contributions to the creation of various techniques for the game of tic tac toe. [3]

A major contribution made recently in this field is of Google Deepmind’s alphaGO which had a huge impact in this field by introducing a generalized deep reinforcement learning to the world of strategy board games by creating agents that could play many different Atari and similar board games like tic tac toe(chess, shogi and GO) and even managed to outplay the best chess engine at that time(StockFish 8) as stated in [4] and [5]. But the actual algorithm behind the agent has not been made fully public yet, which supposedly only required training of only 4 hours on google’s 64 GPU and 19 CPU using indirect feedback. Presumably, only the rules of the game were given as input the rest(strategies, tactics), the algorithm learned itself in the form of Self-Play. Similar was the adoption using qlearning utilized in this citation [6]. A similar study was covered in [7], where an attempt was made to improve the convergence rate. On similar grounds to Reinforcement Learning an interesting approach was taken to impose the human-like aspect to it and named it theory-based reinforcement learning. [8] [9]. AlphaGO like approach but implemented via one-hot encoding based vectors [10] [11]

To find the best pass for a 3X3 tic tac toe game researchers in [12] study employed hamming distance classifier based neural network, with time complexity for the algorithm being \( O(n^3) \) where \( n \) is the number of cells(3X3, \( n = 9 \)). Which would take a lot of time if the tic tac toe game were to be scaled up to 4X4,5X5 etc. A similar Methodology is employed by researchers in the paper using evolutionary programming in [13].

A more recent study explored a more interactive approach on how the game of tic tac toe should be played using drones. In that research the authors explored the algorithms such as QL algorithm, SV and SARSA. [14]

One of the more common solutions to the tic tac toe problem is using the Min-Max search algorithm. [15]

Borovska in his paper [16] has discussed the reliability of the Min-Max algorithm. This paper also discusses an Optimization approach(\( \alpha – \beta \) pruning) and the advantages and disadvantages of the Min-Max algorithm, which works on a basis of trying to minimize the loss and maximize the gain at each step of the way down the search tree in a recursive fashion, by attributing certain characteristics for the game. An optimization to it is covered in [17], [18] and [19].

The researchers in [20] paper discussed a decision tree-based approach for implementing no loss state in tic tac toe. A Theoretical error is also highlighted in this paper that,
when both players play correctly the Min-Max does have an ideal No-Loss strategy. However, when the opponent plays non-optimally Min-Max has been shown to play non-optimal moves, dragging the game out for one extra move resulting to succeed at the next state rather than the planned state.

The Min-Max algorithm explores the game tree in detail first so with increasing complexity for higher board sizes the game tree in Min-Max becomes very large and it takes a long time to come up with a solution, especially at the starting when the whole board is empty. The time complexity of the Min-Max algorithm is \( O(b^m) \) and the space complexity is \( O(bm) \) where \( m \) is the maximum depth of the tree (9 for tic tac toe) and \( b \) is the number of legal moves at each node. In the next section, we will discuss how we can try to reduce this time for board sizes of 3x3, 4x4, and so on for the tic tac toe game specifically. Though the algorithm can be implemented for other board games as well and scaled up for higher board sizes with minimalist code changes. This reduction in time for different board sizes can be achieved by defining tic tac toe as a well-posed problem to improve decision taking time for an agent.

II. SELECTION OF TARGET FUNCTION REPRESENTATION

After the ideal target function, \( V \) is defined, the approximation function \( \hat{V} \) representation used by our agent will be defined in this section. The following board features are considered for a specific intermediate board state. A linear combination of these features will account for the definition of the function \( \hat{V} \):

- \( x_1 \): Number of Instances of Player-1’s Symbol(X) within an open row and column. Examples in Fig. 1 and Fig 2.
- \( x_2 \): Number of Instances of Player-2’s Symbol(O) in a row within an open row and column. Examples in Fig. 3 and Fig 4.
- \( x_3 \): No. of instances of 2 consecutive Player-1’s Symbol(X) in a row. Examples in Fig. 5.
- \( x_4 \): No. of instances of 2 consecutive Player-2’s Symbol(O) in a row. Examples in Fig. 6.
- \( x_5 \): No. of instances of 3 Player-1’s Symbol(X) in a row with an open box. Examples in Fig. 7.
- \( x_6 \): No. of instances of 3 Player-2’s Symbol(O) in a row with an open box. Examples in Fig. 8.
With all the features that will be extracted from a board state defined, the overall feature vector $\mathbf{X}$ will be ($x_0 = 1$ is for biasing) as explained in [21]:

$$
\mathbf{X} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}
$$

and corresponding weight vector will be

$$
\mathbf{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix}
$$

As a result, The learning program will represent $\hat{V}(b) = \mathbf{X}^T \mathbf{W}$ as a linear function of the form

$$
\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6 \quad (1)
$$

In machine learning terms $w_0$ through $w_6$ are called weights. They decide the relative significance of the feature for what they are defined for and a combined linear combination of them gives the utility of a particular board state. $w_1$ to $w_6$ are normal weights given to predefined features of the board above. While $w_0$ is a constant that adds to the importance of the board. [22].

### III. The Final Design

After defining all the components of the defined architecture, starting from defining the tic tac toe as a well-posed problem and solving it using reinforcement learning as a base and stochastic-gradient descent to fine-tune the estimated weight vector parameters for each training example generated during various iterations of several games played against itself. It’s time to put it all together.

Reinforcement-Learning is like an action-reward system. The agent is rewarded for taking a step in the right direction (in our case, Winning Tic Tac Toe) and penalized for wrong decisions. In our case, the RL Model uses a linear target function but various other possibilities like quadratic, cubic polynomials for a target function also exist.

Many learning systems in use today use these four key components: Performance System, Critic, Generalizer, and Experiment Generator, and the final design is based on these four components. More information and definitions about these components are given in the following sections.

The role of each component in the described architecture and its Implementation in an object-oriented manner is described in Fig. 9 and Fig. 10 respectively.

---

Fig. 8. Possibilities of all O in a row

---

Fig. 9. Role of each component in the defined architecture

---

Fig. 10. Class Diagram for the defined Architecture
Algorithm 1 Proposed algorithm for Training of Agent

**Input:** numTrainingSamples(n)

**Output:** targetWeightVector(W), numAgentWins(nW), numAgentLoss(nL), numAgentDraws(nD)

*Initialisation:* 
\[ W = \{0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5\} \]

\[ \text{trainGamesCount} = 0, \]
\[ nW = nL = nD = 0 \]

1. while trainGamesCount != n do
2. boardStateInitial = expGen.generateNewProblem()
3. Obtain the solution trace of the game played by the agent by choosing best(legalMoves(b)), comparing utility value \( \hat{V}(b) \) for each legal move using the current \( W \).
4. for board in range(len(gameHistory)) - 1 do
5. featureV = extractFeatures(board) \( \| x_0 \ldots x_6 \)
6. calculate the approximate utility value \( \hat{V}(b) \) of intermediate board states using featureV and current \( W \) using equation (1)
7. end for
8. calculate the utility value \( V(b) \) of the final board state(Loss, Win, Draw)
9. \[ \text{trainingExamples} = \{ \text{featureV}_1, V_{\text{train}}(b_1), \text{featureV}_2, V_{\text{train}}(b_2), \ldots \} \]
10. \[ \text{finalScore} = \text{trainingExamples}[1] \]
11. Increment gameStatus counts nW, nL and nD based on the Final game result i.e FinalScore
12. Update \( W \) for all trainingExamples using LMS rule
13. for trainExample in trainingExamples do
14. \( V_{\text{train}}(b) \leftarrow \text{trainingExample}[1] \)
15. \( \text{featureV} \leftarrow \text{trainingExample}[0] \)
16. \( \hat{V}(b) \leftarrow \hat{W}.\text{featureV} \)
17. Substitute \( V_{\text{train}}(b) \) and \( \hat{V}(b) \) in eq (3), return the new \( W \)
18. end for
19. trainGamesCount \( \leftarrow \) trainGamesCount + 1
20. end while
21. return \( W, nW, nL, nD \)

### IV. Results

![Graph showing performance of agent with number of games played](image)

**Fig. 11.** Variation in performance of agent with number of games played(Coloured)

### TABLE I

| Games Played | Wins | Loss | Draws | Win/Draw |
|--------------|------|------|-------|----------|
| 10000        | 761  | 82   | 157   | 4.85     |
| 100000       | 7524 | 127  | 1049  | 4.56     |
| 1000000      | 78231| 382  | 113700| 5.82     |
| 10000000     | 139053| 16288| 24659 | 6.45     |
| 100000000    | 240136| 24202| 35662 | 6.73     |
| 400000000    | 326749| 30979| 42272 | 7.73     |
| 500000000    | 410749| 38336| 59015 | 8.07     |
| 600000000    | 494595| 45643| 59757 | 8.28     |
| 700000000    | 5808906| 52567| 66527 | 8.73     |
| 800000000    | 667257| 59554| 73189 | 9.12     |
| 900000000    | 753805| 66316| 79879 | 9.44     |
| 1000000000   | 840259| 73203| 86558 | 9.71     |
| 1100000000   | 926767| 79939| 93294 | 9.93     |
| 1200000000   | 1013183| 86815| 108802| 10.13    |
| 1300000000   | 1099240| 93803| 106957| 10.28    |
| 1400000000   | 1185745| 105555| 113700| 10.43    |
| 1500000000   | 1272729| 107135| 120406| 10.57    |

### TABLE II

| Games Played | w0  | w1  | w2  | w3  | w4  | w5  | w6  |
|--------------|-----|-----|-----|-----|-----|-----|-----|
| 10000        | 78.2| 9.8 | 106.8| 93.4| 97.1| 35.4| 113.7|
| 100000       | 70.6| 9.9 | 123.3| 31.0| 99.2| 43.0| 208.4|
| 1000000      | 72.8| 0.2 | 23.2 | 54.4| 158.4| 45.9| 208.0|
| 200000       | 44.3| 9.4 | 1.0  | 38.0| 85.9| 56.8| 175.5|
| 300000       | 81.4| 15.8| 11.1 | 12.6| 82.1| 63.0| 192.9|
| 400000       | 62.2| 157.9| 19.0| 42.8| 34.0| 41.0| 197.1|
| 500000       | 93.6| 17.6| 35.3| 29.5| 135.3| 29.8| 227.4|
| 600000       | 97.9| 168.2| 51.2| 29.0| 7.2 | 51.5| 218.5|
| 700000       | 67.3| 168.2| 41.9| 22.0| 47.5| 40.8| 188.3|
| 800000       | 77.7| 168.2| 2.2 | 22.9| 20.1| 62.5| 198.9|
| 900000       | 84.2| 168.2| 28.9| 29.4| 10.3| 41.0| 197.4|
| 1000000      | 67.0| 168.2| 11.4| 16.8| 12.9| 34.1| 197.5|
| 1100000      | 91.7| 168.2| 7.5 | 21.6| 20.8| 10.0| 202.5|
| 1200000      | 95.4| 168.2| 61.6| 76.9| 25.1| 33.0| 160.1|
| 1300000      | 58.4| 168.2| 76.4| 21.6| 19.4| 36.6| 183.3|
| 1400000      | 60.5| 168.2| 36.4| 30.6| 6.0 | 47.3| 209.5|
| 1500000      | 93.4| 168.2| 72.3| 12.4| 12.7| 18.9| 198.6|

Tic Tac Toe is a biased game in favor of the first player, giving the player who starts the game more chances to win while leaving the opponent with either a loss or a tie. The game will end in a draw if both players play intelligently. A "Never-Loss" Generalised strategy allows the agent to win or at the very least keep a tie in any game.

The Learning rate \( \eta \) was set to 0.4 on a 3X3 Tic Tac Toe board using Algorithm 1. It was trained on Google Colaboratory for 6 Hours for testing.

Firstly the win ratio was tried after each iteration, which is defined by

\[
\text{Win Ratio} = \frac{\text{Number of games Won}}{\text{Number of Games Played}}
\]

but the results just kept providing ineffable results, so the next thing that was tried was the win/draw ratio, which highlighted the correct output of the agent after each iteration.

\[
\text{Win Draw Ratio} = \frac{\text{Number of games Won}}{\text{Number of Games Drawn}}
\]
The most amount of variation is observed in $w_5$ which is the Number of Instances of symbol “X” in a row and it is trying to maximize it to whatever extent possible while minimizing $w_6$ which results in opponent winning and giving it whatever minimizing number possible. These results are also plotted on graphs in Fig. 11 and Fig. 12. The most amount of variation is observed in $w_1$ in between 0.4 and 0.6(Base 1,000,000) which means the agent trying to explore different starting positions for starting the game. In theory, the first player playing to any corner gives the best winning chances. If the correct play is assumed from both the players then the game will always result in a draw.

Table II gives us insight on what features are important to agent it is namely $w_5$ which is the Number of Instances of symbol “X” in a row and $w_6$ which results in opponent winning and giving it whatever minimizing number possible. These results are also plotted on graphs in Fig. 11 and Fig. 12. The most amount of variation is observed in $w_1$ in between 0.4 and 0.6(Base 1,000,000) which means the agent trying to explore different starting positions for starting the game. In theory, the first player playing to any corner gives the best winning chances. If the correct play is assumed from both the players then the game will always result in a draw.

Table I gives us the observation that the agent keeps on improving through the win/draw ratio. Some of the games that were played by the agent resulted in Win and draw.

There are a few more conclusions that we can draw from this study:

- The values of weight vectors $w_5$ and $w_6$ in Table II suggest that there is a fair probability of having a no-loss tic tac toe strategy. However, a game like chess, which can also be solved using the state-space search Min-Max algorithm and in which the engines (agents) are constantly changing, cannot be compared to tic tac toe because it is a much more complex game.
- If the algorithm is extended to other games, changes to the number and types of feature vectors extracted, as well as the Target Function, must be investigated to better match the game. Since the linear target function used in this study may not be the best fit for other complex board games. For those board games Quadratic, Cubic, and other target functions may be needed.
- A significant amount of variable fluctuations can be observed for variable $w_2$. Making it an unreliable parameter whilst comparing it with other parameters chosen.
- A case very similar to $w_2$ can also be observed with $w_1$ but with a different pattern. A bell shape curve is observed for the initial stages, later on moving to a constant value. Making the parameter $w_2$ as unreliable.

In this research, only the training of the agent via Indirect Learning is explored. The future scopes of this study might include Indirect Learning for some portion of learning to complement with direct methods to offer a more comprehensive view of agent learning. But there are still a lot of optimizations that can be done to reduce the training time to reach optimal results via the proposed algorithm. This may inspire other researchers to conduct similar experiments for several other games as this generalized proposed algorithm can also be easily extended to be used to solve other perfect information board games like Checkers, Infinite Chess, etc.

REFERENCES

[1] A. Singh, K. Deep, and A. Nagar, “A” never-loose” strategy to play the game of tic-tac-toe,” in 2014 International Conference on Soft
[2] M. T. Carroll and S. T. Dougherty, “Tic-tac-toe on a finite plane,” Mathematics Magazine, vol. 77, no. 4, pp. 260–274, 2004. [Online]. Available: https://doi.org/10.1080/0025570X.2004.11953263

[3] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018, vol. 1, pp. 10–13.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., “Human-level control through deep reinforcement learning,” nature, vol. 518, no. 7540, pp. 529–533, 2015.

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel et al., “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” arXiv preprint arXiv:1712.01815, 2017.

[6] D. H. Widyantoro and Y. G. Vembrina, “Learning to play tic-tac-toe,” in 2009 International Conference on Electrical Engineering and Informatics, vol. 1, 2009, pp. 276–280.

[7] F. Cruz, S. Maegg, C. Weber, and S. Wermter, “Improving reinforcement learning with interactive feedback and affordances,” in 4th International Conference on Development and Learning and on Epigenetic Robotics, vol. 1, 2014, pp. 165–170.

[8] P. A. Tsividis, J. Loula, J. Burga, N. Foss, A. Campero, T. Pouny, S. J. Gershman, and J. B. Tenenbaum, “Human-level reinforcement learning through theory-based modeling, exploration, and planning,” arXiv preprint arXiv:2107.12544, 2021.

[9] H. Wang, M. Preuss, and A. Plaat, “Adaptive warm-start mcts in alphazero-like deep reinforcement learning,” in Pacific Rim International Conference on Artificial Intelligence, vol. 1. Springer, 2021, pp. 60–71.

[10] B. Gu and Y. Sung, “Enhanced reinforcement learning method combining one-hot encoding-based vectors for cnn-based alternative high-level decisions,” Applied Sciences, vol. 11, no. 3, 2021, pp. 1291–1307.

[11] J. Scheiermann and W. Konen, “Alphazero-inspired general board game learning and playing,” arXiv preprint arXiv:2204.13307, 2022.

[12] N. F. Rajani, G. Dar, R. Biswas, and C. K. Ramesha, “Solution to the tic-tac-toe problem using hamming distance approach in a neural network,” in 2011 Second International Conference on Intelligent Systems, Modelling and Simulation, vol. 1, 2011, pp. 3–6.

[13] E. Karmanova, V. Serpiva, S. Perminov, A. Fedoseev, and D. Tssetserukou, “Swarmplay: Interactive tic-tac-toe board game with swarm of nano-uavs driven by reinforcement learning,” in 2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN), vol. 1. IEEE, 2021, pp. 1269–1274.

[14] S. Videgaín and P. G. Sánchez, “Performance study of minimax and reinforcement learning agents playing the turn-based game iwoki,” Applied Artificial Intelligence, vol. 35, no. 10, pp. 717–744, 2021. [Online]. Available: https://doi.org/10.1080/08839514.2021.1934265

[15] P. Borovska and M. Lazarova, “Efficiency of parallel minimax algorithm for game tree search,” in Proceedings of the 2007 international conference on Computer systems and technologies, vol. 1, 2007, pp. 1–6.

[16] V. Keswani, O. Mangoubi, S. Sachdeva, and N. K. Vishnoi, “A convergent and dimension-independent min-max optimization algorithm,” in International Conference on Machine Learning, vol. 1. PMLR, 2022, pp. 10939–10973.

[17] Y. Cai and W. Zheng, “Accelerated single-call methods for constrained min-max optimization,” arXiv preprint arXiv:2210.03096, 2022. [Online]. Available: https://arxiv.org/abs/2210.03096

[18] J. Abernethy, K. A. Lai, and A. Wibisono, “Last-iterate convergence rates for min-max optimization: Convergence of hamiltonian gradient descent and consensus optimization,” in International Conference on Algorithmic Learning Theory, vol. 1. PMLR, 2021, pp. 3–47.

[19] S. Sriram, R. Vijayarangan, S. Raghuraman, and X. Yuan, “Implementing a no-loss state in the game of tic-tac-toe using a customized decision tree algorithm,” in 2009 International Conference on Information and Automation, vol. 1, 2009, pp. 1211–1216.

[20] I. Goodfellow, Y. Bengio, and A. Courville, “Machine learning basics,” Deep learning, vol. 1, pp. 98–164, 2016.

[21] T. M. Mitchell, Machine Learning, 1st ed. USA: McGraw-Hill, Inc., 1997, pp. 1–11.