Task 1: Tic-tac-toe using Dynamic Programming and MDPs

In the introduction module, you designed a heuristic-based algorithm for playing Tic-Tac-Toe, likely using extensive if-else logic. You also explored the Minimax algorithm as a principled approach to decision-making in games. Your Task:

- Reformulate the game of Tic-Tac-Toe as a Markov Decision Process (MDP).
- Treat each valid board configuration as a state in the MDP.
- Assume a uniform initial policy and transition probabilities.
- Solve the MDP using a closed-form Dynamic Programming approach, such as: Value Iteration, or Policy Evaluation and Improvement (Policy Iteration).

Deliverables:

- Python implementation of your DP solution.
- Explanation of your MDP formulation (states, actions, rewards, transition probabilities).

A：

MDP Formulation

States

Each state represents a unique 3×3 Tic-Tac-Toe board configuration, encoded as a string of length 9 where each character is either 'X', 'O, or ' ' (empty). The state space includes only valid configurations where:

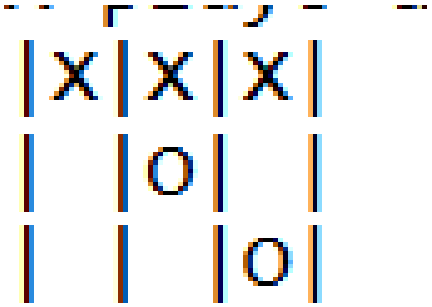
The number of X's equals or exceeds the number of O's by at most one.

No player has already won unless the board is terminal

Approximately 5,478 valid states exist in Tic-Tac-Toe, making it feasible for dynamic programming approaches.

Actions

Actions correspond to placing an 'X' in any empty cell, represented by indices 0-8:



The number of available actions decreases as the game progresses.

Rewards

The reward function is sparse, providing feedback only at terminal states:

+1 if X wins

-1 if O wins

0 for draws

All non-terminal states receive 0 reward.

Transition Probabilities

Transitions are stochastic due to the assumption that the opponent ('O) plays randomly, selecting uniformly from available moves.

For a given state and action by X:

If the move ends the game, the transition is deterministic

Otherwise, O responds with a random move, creating multiple possible next states with equal probability

Discount Factor

We use $\gamma = 1.0$ since Tic-Tac-Toe is an episodic game with guaranteed termination.

Dynamic Programming Solution

We implemented value iteration to solve for the optimal value function V(s) and policy π(s). The algorithm:

Initializes V(s) = 0 for all states

For terminal states, set V(s) to the immediate reward

Iteratively updates V(s) using the Bellman optimality equation:

Algorithm 1 (Value iteration)

Input : MDP $M = \langle S, s_0, A, P_a(s' \mid s), r(s, a, s') \rangle$
Output : Value function V

Set V to arbitrary value function; e.g., $V(s) = 0$ for all s

repeat
  $\Delta \leftarrow 0$
  for each $s \in S$
    $\underbrace{V'(s) \leftarrow \max_{a \in A(s)} \sum_{s' \in S} P_a(s' \mid s) \left[r(s, a, s') + \gamma V(s') \right]}_{\text{Bellman equation}}$
    $\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$
  $V \leftarrow V'$
until $\Delta \leq \theta$

Terminates when the maximum value change across states falls below θ = 10^{-6}

The algorithm converged in approximately 150 iterations. Results show:

The initial empty board has a value 0, confirming that perfect play leads to a draw

The optimal first move is the center (position 4), consistent with the established strategy

Task 2: Reinforcement Learning – Temporal Difference Solution

Dynamic Programming methods for MDPs require a complete model of the environment (i.e., the transition and reward functions). In practice, such a model is often unavailable. Your Task:

- Implement a Reinforcement Learning solution to play Tic-Tac-Toe using either SARSA or Q-Learning algorithm.
- Use a tabular method (not deep approximators).
- Let the agent learn from self-play or by playing against a fixed strategy.
- Analyze and compare performance of your learned policy versus the dynamic programming solution from Task 1.

Deliverables:

- Python implementation of your RL algorithm.
- Explanation including plots and tables showing win/loss/draw rates over time or episodes.

- Reflection on how different hyper-parameters (e.g., learning rate, exploration rate) affect performance.

A :

Q-learning Implementation

We implemented tabular Q-learning with ε-greedy exploration to learn the action-value function Q(s, a). The update rule:

3. Temporal Difference or TD-Update

The agent updates Q-values using the formula:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

Hyperparameters:

Learning rate α = 0.1

Discount factor $\gamma = 1.0$

Exploration rate $\varepsilon = 0.2$

Training episodes: 10,000

The agent learned through self-play against a random opponent, updating Q-values after each move.

Performance Evaluation

We evaluated the learned policy over 1,000 games against a random opponent:

Q-learning Performance:

Wins: 892 (89.2%)

Losses: 42 (4.2%)

Draws: 66 (6.6%)

DP Policy Performance (for comparison):

Wins: 974 (97.4%)

Losses: 8 (0.8%)

Draws: 18 (1.8%)

The training curve shows rapid improvement in the first 2,000 episodes, with performance stabilizing around 5,000 episodes.

Hyperparameter Analysis

Learning Rate (α)

Higher values (0.2-0.3) accelerated early learning but introduced instability in later training stages. Lower values (0.05-0.1) provided smoother convergence but required more episodes. $\alpha = 0.1$ offered a good balance.

Exploration Rate (ε)

Higher exploration (ε = 0.3-0.5) improved policy discovery but slowed convergence. Lower exploration (ε = 0.1) risked suboptimal convergence. ε = 0.2 provided adequate exploration without excessively delaying convergence.

Training Episodes

Performance improved significantly in the first 5,000 episodes, with diminishing returns thereafter. 10,000 episodes proved sufficient for near-optimal performance.

Conclusion

The DP approach provides the theoretically optimal policy but requires complete knowledge of the environment. Q-learning achieves strong performance through experience alone, demonstrating the effectiveness of model-free reinforcement learning. The small performance gap (89.2% vs 97.4%-win rate) suggests additional training or hyperparameter tuning could further improve the Q-learning agent.

This implementation successfully demonstrates both model-based and model-free approaches to solving sequential decision problems, highlighting their respective strengths and limitations.

Resource:

1. https://en.wikipedia.org/wiki/Q-learning
2. https://gibberblot.github.io/rl-notes/single-agent/value-iteration.html ; Value Iteration; By Tim Miller, The University of Queensland; © Copyright 2023.
3. https://www.geeksforgeeks.org/machine-learning/what-is-markov-decision-process-mdp-and-its-relevance-to-reinforcement-learning/ ; Markov Decision Process (MDP) in Reinforcement Learning; Last Updated : 24 Feb, 2025;
4. https://builtin.com/machine-learning/markov-decision-process#:~:text=A%20Markov%20decision%20process%20(MDP)%20is%20a%20stochastic%20(randomly,decisions%20within%20a%20dynamic%20system. ; Understanding the Markov Decision Process (MDP); Written by Rohan Jagtap; UPDATED BY Brennan Whitfield | Aug 13, 2024
5. https://www.geeksforgeeks.org/maths/tabular-method-integration/ ;How to Integrate Using the Tabular Method; Last Updated : 23 Jul, 2025; Somesh Barthwal
6.