

**Machine Learning**

Homework 2

Supervisor:

Dr. Hashemi

Ali Mahmoodi

Atefe Rajabi

Winter 2024

**Index:**

[Problem definition](#Problem)

[Features](#Features)

[Training Process](#Train)

[Implementation](#Implementation)

[Result](#Result)

**Problem definition:**

The task set in Homework II involves the development of an artificial intelligence (AI) agent capable of playing the game of Tic-Tac-Toe on a 3×3 board. This game, characterized by its simplicity and finite number of possible states, presents an ideal scenario for applying function approximation techniques to learn optimal strategies. The challenge is to implement an algorithm that can learn to play Tic-Tac-Toe through interaction with the game environment, ultimately being capable of making decisions that increase its chances of winning.

**Features:**

The features extracted from the Tic-Tac-Toe board for training the model are designed to capture various strategic aspects of the game that could influence the decision-making process of the AI agent. Each feature is derived from the state of the board and is aimed at providing insights into potential moves.

1. f1 (Two in a Row for the Agent): This feature counts the number of rows, columns, and diagonals where the agent has exactly two of its symbols and the third cell is empty. This scenario signifies a potential win in the next move if not blocked by the opponent.

2. f2 (Two in a Row for the Opponent): Similar to `f1`, but for the opponent. It counts the number of rows, columns, and diagonals where the opponent has two of its symbols and the third cell is empty. This feature is critical for the agent to recognize situations where it needs to block the opponent to prevent losing.

3. f3 (Center Control): This feature checks if the center cell is occupied by the agent's symbol. Controlling the center in Tic-Tac-Toe is a strategic advantage, as it opens up more opportunities for creating three in a row.

4. f4 (Corners Occupied by the Agent): Counts the number of corners occupied by the agent's symbol. Occupying corners gives the agent additional chances to create three in a row since each corner is part of two lines (a row and a column) and two diagonals.

5. f5 (Single in a Row with Two Empty): This feature calculates the number of rows, columns, and diagonals with exactly one of the agent's symbols and two empty cells. It indicates potential lines the agent could work on to create three in a row, especially in the early stages of the game or to set up double threats.

6. f6 (Immediate Win Available): This is a binary feature indicating whether the agent has an immediate winning move available (i.e., a row, column, or diagonal where placing the agent's symbol would result in three in a row). This feature directly signals the agent to take the winning move if present.

7. bias

These features are chosen to give the AI a comprehensive understanding of the current state of the board from a strategic standpoint. They help the AI prioritize moves that lead to immediate wins, block the opponent's wins, take strategic positions (like the center and corners), and develop potential lines for victory. The linear regression model uses these features to evaluate and predict the outcome of potential moves, guiding the agent to make decisions that increase its chances of winning.

**Training Process:**

This model learns exclusively through gameplay, without relying on a pre-existing dataset.

Dataset Used for Training

- Gameplay-Based Learning: The model does not use a conventional static dataset. Instead, it generates its training data through repeated gameplay. In each game, the model collects features based on the current state of the board and the decisions (actions) made by the agents. This method allows the model to learn from a diverse set of game scenarios.

- Feature Collection: During each game, features are extracted based on the board's state whenever an action is predicted by either of the two agents (`ag1` and `ag2`). These features, along with the outcome of the game, are used to create the training data.

Performance Improvement Over Time

- Adaptive Learning: As the model plays more games, it encounters various board configurations and outcomes. This diversity enables the model to refine its parameters based on a wide range of experiences, improving its decision-making capabilities over time.

- Feedback Loop: After each game, the model updates its parameters based on the outcome. Wins and losses provide feedback, allowing the model to adjust its strategy. Winning moves are reinforced with positive labels, while losing moves are penalized with negative labels, creating a clear signal for the model to learn from.

- Randomization: The training data is randomized before each training session (`np.random.shuffle(ind)`). This step ensures that the model is not biased by the order in which data is presented, helping it to generalize better from the training data.

Train

- The training loop runs for a predefined number of games (`10000`). For each game, it dynamically generates training data based on the actions taken by the agents and the game's outcome.

- If a game ends in a draw (`stat == 1`), all moves made during the game are considered neutral and labeled with `0`. If there's a winner, moves by the winning agent are labeled positively (`100`), and the opponent's moves are labeled negatively (`-100`).

- The collected features and labels are then shuffled to randomize the training data, which is believed to improve the learning process by preventing the model from learning patterns based on the sequence of data.

- Finally, the model's `fit` method is called with the shuffled data and labels, allowing the model to adjust its parameters using gradient descent.

- The model parameters are stored in a ‘pkl’ file for further usages.

This training process allows the model to learn effective strategies for Tic-Tac-Toe by continuously adjusting its parameters based on the outcomes of thousands of games, improving its performance over time through direct interaction with the game environment.

**Implementation:**

This project is implementation of a Tic-Tac-Toe game that uses a combination of artificial intelligence techniques for the game's agents. The project is organized into multiple modules, each serving distinct functionalities that contribute to the game's AI logic, environment setup, and user interface. Below is a detailed explanation of each module within the script:

1. Agent Module

This module defines the `Agent` class responsible for the AI aspects of the Tic-Tac-Toe game. The class contains methods to encode the game board, extract features for AI decision-making, evaluate the board's state, and predict actions (moves). Key functionalities include:

- Initialization: Sets up the agent with a specific symbol (`x` or `o`), identifies the opponent, and assigns a machine learning model for decision-making.

- Board Encoding: Converts the game board into a numerical format where the agent's symbols, the opponent's symbols, and empty spaces are represented by `1`, `-1`, and `0`, respectively.

- Feature Extraction: Generates features from the board's state to be used in the decision-making process. Features include aspects like the number of rows, columns, or diagonals with two symbols of the same kind, the center occupancy, among others. (The features have been discussed before.)

- Board Evaluation: Uses the model to predict the outcome based on the extracted features.

- Action Prediction: Evaluates all possible moves to find the best action based on the model's prediction.

2. Environment Module

This module contains the `Environment` class, which manages the Tic-Tac-Toe game board and rules. It includes functionalities to get the current state of the board, make moves, check for valid moves, and determine if the game has finished. Key elements include:

- Initialization: Prepares an empty game board and sets the initial turn.

- State Management: Methods to get the current board state, apply actions (moves), and check their validity.

- Game Termination: Checks for game-ending conditions, such as a win for either player or a draw.

3. Linear Regression Module

The `LinearRegression` class in this module is a simple implementation of the linear regression algorithm used as the decision-making model for the AI agents. It includes:

- Initialization: Sets up the model with a specified learning rate and number of iterations.

- Gradient Descent: The core algorithm for adjusting the model's parameters based on the data it's trained with.

- Model Training and Prediction: Methods to fit the model to training data (extracted game features and outcomes) and predict outcomes based on new game states.

4. Game Interface Module

This part of the project utilizes the Pygame library to create a graphical user interface (GUI) for the Tic-Tac-Toe game. It includes the `TicTacToe` class with methods to:

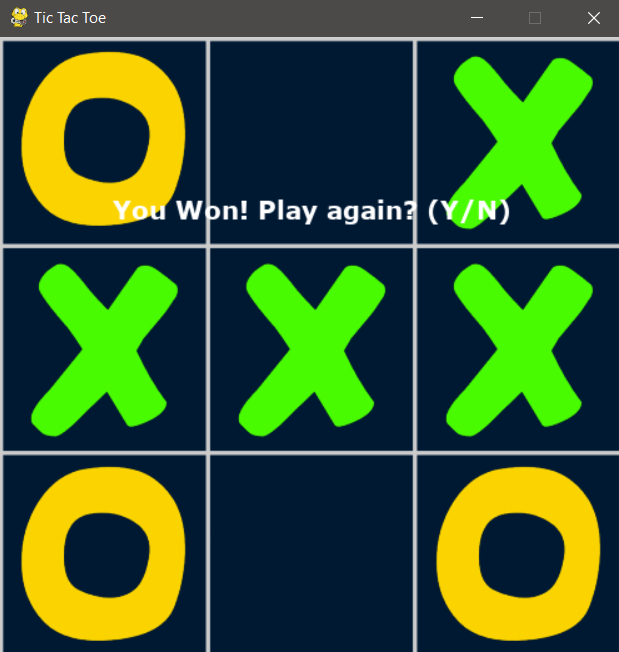
- Initialization: Sets up the game window, loads resources (images for the game board, `x`, and `o` symbols), and initializes the game environment and agents.

- Game Loop: Contains the main loop that handles events (e.g., mouse clicks for player moves), updates the game state, and redraws the game board.

- UI Updates: Methods to display messages, handle player decisions to continue playing after a game ends, and render the game board and symbols.

**Result:**

A Win state A Game Over state A Draw state



Parameters for the model:

1. [-1.42485182e-03]
2. [-3.13010588e-04]
3. [-2.89879592e+00]
4. [ 3.46089393e-05]
5. [6.96265787e-04]
6. [ 2.23320759e-04]
7. [ 1.13705660e+00]