Applying Machine Learning to Tic-Tac-Toe

Carl Rodriguez, Chris Barnes, and Thomas Tuttle

What is Machine Learning?

- Machine Learning is an algorithmic strategy where computers are given the ability to learn about problems from input data.
- It allows the application of computational power to complex problems in order to gain improving solutions.
- It focuses on the development of computer programs which can teach themselves solutions using data.
- These programs look at data in order to detect patterns and adjust accordingly as more data is discovered and input.
- Machine Learning is a rapidly growing field in the modern world. Its applications include:
  - Classification, such as in image processing, medical diagnoses, and fraud detection;
  - Online Recommendations based on a user's interests
  - And much more!

What is Rote Learning?

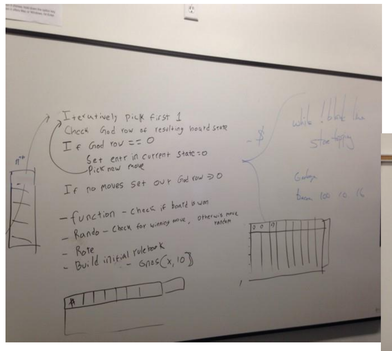
- Rote Learning is a basic form of Machine Learning.
  - In humans, Rote Learning is best defined as learning by memorization and repetition.
  - When applied to computers, Rote Learning encompasses the creation of a database by a computer that is checked and expanded as new data is input.
  - A notable example of Rote Learning in action is Arthur Samuel's Checkers Program. Created in the 1950s-1960s, it was a checkers-playing program that would remember every position it had ever been in, the results of those games, and the win condition of checkers to determine the best move in a given game.

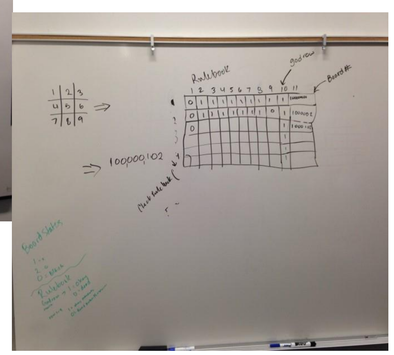
Project Goals

- **Primary Goal:** Use a rote machine learning algorithm to teach an A.I. not to lose at Tic-Tac-Toe
- **Medium:** Tic-Tac-Toe
- **Method:**
  - Create 2 A.I. programs: One to make random moves and one to train to not lose
  - Create a program with the ability to train an A.I. over a chosen number of games

Implementation

- We created a "rulebook" for the A.I. being trained that was updated over time. For an input board state, the rulebook would contain an array of all the possible moves by the A.I. At first, the rulebook would only prevent the A.I. from putting an O where there was already an X; however, if a certain move caused the game to be lost in the next term, the rulebook would update to prevent the A.I. from making that move on that specific board state.
- The A.I. was trained against an A.I. called Rando, which would always choose a move that would immediately win the game if possible and otherwise would make a random (legal) move.
- The trained A.I. would always make the first move allowed by its rulebook.
- The two A.I.s would play a specified number of games, with each unique board state being recorded into the rulebook as it appeared, as well as if the move made on that board state resulted in a loss (which would cause the A.I. to not make that move in the future when in such a board state).
- The program would run through the specified number of games, then give the number of wins, losses, and ties for each set of 100 games.
- Some Examples of our Pseudocode:

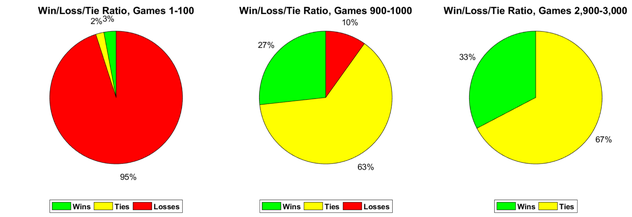


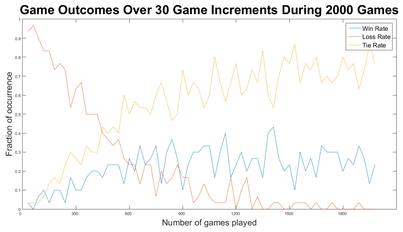


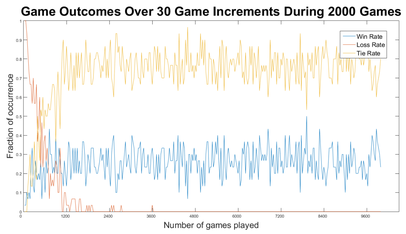
Results

- The program was run several times, with simulations containing 1000-10,000 games.
- While the random actions of enemy Rando A.I. would cause a decent variety of results, particularly in the first 1000 games, a consistent pattern emerged:
  - During the first 100 or so games, the trained program had a very high loss rate, approx. 80-90% each run.
  - However, the trained program's loss rate would drop visibly over each subsequent 100-game increment, down to approx. 10-20% by games 900-1000.
  - By around the 2500 game mark, the trained A.I. would stop losing to the Rando A.I. entirely.
  - After the A.I. stopped losing, its win rates and tie rates would usually stagnate.

Results







- Initial performance was poor, the rote algorithm would lose almost all of its games
- Over the first 500 games, losses decreased dramatically
- After around 1,000 games, the algorithm rarely losses
- By 3,000 games the algorithm practically never losses
- No losses occurred after 4,000 games

Conclusions

- The project was a success: After a few thousand games (just a few seconds of runtime with our code), the A.I. had been trained not to lose against the Rando A.I.
- There were several ways we could have expanded this project if given more time, such as:
  - Implementing a way for a user to play against the newly trained A.I./fill in for Rando
  - Training Rando to win rather than simply making random moves
  - Decreasing runtime by using trinary numbers and bit-shifting in the rulebook
  - Switching the order of the A.I.s
  - Having the trained A.I. make random rather than iterative rulebook choices
- Ultimately, this project taught us a lot about machine learning algorithms and how such logic could be applied to other games and real-world applications.