# Final Project

Joshua Rinaldi, joshua.rinaldi@colorado.edu
Carlos Lawrence, carlos.lawrence@colorado.edu

## Abstract

One popular area of research in Machine Learning is teaching a computer how best to play a game. People have done this on games such as Super Mario World, Battleship and Brick Breaker. We set out to create a program that would play tic-tac-toe game that would learn as it played the game more on and be able to eventually beat a human player.

## 1. Introduction

When on Google, if one were to search for "Machine Learning Tic Tac Toe" a multitude of results would come up. There are numerous results that can be found which give tutorials on how to train a model to play tic tac toe. There are also some links that will take you to a tic tac toe game where you play against a machine. We wanted to do the same on our own, and then eventually train the model against itself, interested to see if both "players" would become equally proficient at the game, or if one player would become drastically better than the other. We started out by making our own implementation of tic tac toe and then later on added in a computer player using Reinforcement Learning to learn how to play the game and compete with both human and computer players.

## 2. Related Works

One person, Christopher J. MacLellan made a project that was almost exactly the same as what we set out to do, his process was different, but the end result was the same. There are also plenty of other examples readily available after an internet search where people have programmed a computer to play and learn how to play tic tac toe. In addition to this, there are multitudes of tic tac toe games where there is an option to play against a computer, and while we cannot say for sure if these games use Machine Learning, it can be a safe guess that there are some that do.

Machine Learning is not new to the video game world. As mentioned before, it is a very popular practice for a person to write a program to play a game as a project, there are also many video games in existence where the computer will learn how you play and how to counter your actions in the game. Implementation of Machine Learning into video games is something that can be fun for the developer, and make the game more challenging for the player.

## 3. Playing Tic Tac Toe

Our model uses the full game board state to determine the best next move. The state of the board is recorded as an array with a 0 for an unclaimed space, a -1 for a space claimed by the other player, and a 1 for a space claimed by the current player. Each player has a vector that represents the board in this manner, so Player 1's vector would be the inverse of player 2's, and it is this vector that is altered after each move.

As the game moves on, each player's history is also stored in a vector. The history is essentially a snapshot of the game board at the start of a player's turn, and the move that the player made at that turn. It is this vector of histories which is used by the computer to learn how to play the game. When the game has concluded, the history vector is used to update the bias vector for the given game state.

This knowledge base exists in a file and consists of every state the computer has seen to date, as well as a vector containing a weight for each square on the board, given the current state of the board. Before being trained, the file is empty. After each game that is run, this file is updated. Any previously unseen states are added to the file, and those states that are already in the have their weight vectors updated. Whenever the machine wins, the weights for the moves made at those states are rewarded. They are punished whenever the machine loses and a tie will reward both. Ties are rewarded to encourage the models to learn how to take the game to a tie. Statistically speaking, two expert players should be able to draw any game to a tie, so we wanted to encourage the model's ability to tie games as much as winning. In an attempt to improve our model, we focused on changing the update function for the bias vector, which would be used to actually allow assign new weights to different moves.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

### 3.1. Linear Function

This was the simplest of the functions that we tried. In the event of a win, we would add one point, in the event of a loss we would subtract one point. If there was a draw we would add one for both players. The issue with this function is that it allows for an unlimited upper and lower bound on the expected reward. The issue with this is that much larger numbers need to be stored, and this has the potential to get messy when the computer performs hundreds of thousands of training runs.

### 3.2. First Sigmoid Function

Following the linear function, we then normalized all of our results to fit within a range of (0, 1). This yielded numbers that were much easier to handle than our linear function. Another advantage of using this function is that it prevented over confidence in certain moves, something which can happen very easily operating under a linear function. However, we were hoping to have a range of (-1, 1) rather than a range of (0,1).

### 3.3. Hyperbolic Tangential Functional

The third function that we tried was a hyperbolic tangential function. The advantage that we got out of this function was that it yielded the range of (-1, 1) that we were looking for. However, this function greatly increased the complexity of our code and slowed down the process by a factor of 6. Because of this decreased performance, we opted to go back to a sigmoid function.

### 3.4. Second Sigmoid Function

The new function that we used was a modified version of the sigmoid function. It is:

$$\frac{1}{1 + e^{(-(a-0.5)*5)}}$$

Insert graphs

The reason that we opted to use this function as opposed to a traditional sigmoid function is that a traditional sigmoid function would cluster the majority of our results between 0.5 and 0.7, and as the model ran longer and longer, the grouping of these results would begin to get tighter and tighter around 0.6. Instead, with the function that we use it keeps our results even distributed in the range of (-1, 1) and allows for much cleaner numbers to work with. The reason that this works better for our model is that in our model we can look at the highest value in our weight vector and take into consideration all values around it. This will allow for the model to choose options other than those with the absolute highest weight and therefore avoid getting stuck on certain moves, preventing overfitting. It is far easier to differentiate these values under the new sigmoid function, with a wider range of values, than to do so on the original sigmoid function which had a much tighter spread.

http://blog.ostermiller.org/tic-tac-toe-strategy