AI, Brute Force, or Human:

Who is the best at Connect Four?

Project No: M/CS0801T

Abstract

In our project, we apply reinforcement learning to Connect Four and get a sense of what it takes for a computer to learn and master a game. We create and test three different machine players: random player, brute force, and AI (Q player). Our main conclusions are: Two Q players can jointly train themselves from scratch and show comparable performance compared to a Q player trained against a random player. A Q player trained from scratch can outperform a human. We also found that a Q Player can tie with a brute force player up to a depth of 11. Our training code and our GUI interface using Tkinter (which was used for data collection) can be found on Github.

Background

In this project, we focus on the game of Connect Four on a 4 by 4 grid. Connect Four is a simple two-player game, where you try to get four in a row, column, or diagonal. The twist with Connect Four that makes it differ from a larger version of Tic Tac Toe, is the fact that gravity rules apply, meaning that you choose the column you want to play in, and the first vacant spot in that column would be played.

There are a variety of things a machine player can do when playing a game. It can just do the bare minimum of 'following the rules', (playing in a vacant spot), which we call a **Random player**. This machine player doesn't have much of a strategy, it just chooses a random spot out of the vacant spots.

Our machine player can be a little smarter. When a winning move can be played, it can play it. If the opponent has a winning move, it can block it. Additionally, it can look ahead at future moves and choose the optimal one. This machine player uses the **min-max algorithm** (which we refer to in this proposal as **brute force**).

The min-max algorithm is designed to find the optimal move by maximizing your gain and minimizing your opponent's gain. The way this works can be visualized in a tree diagram. This is demonstrated in the picture below of a move in Tic Tac Toe.

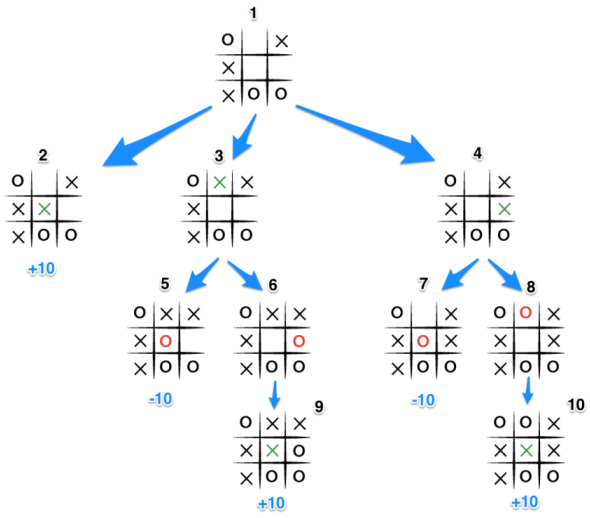


Image from Source [4]

In the picture above, it is the turn of player 1 (the machine player plays X). The machine player can play one of 3 moves, numbered 2, 3, and 4 in the diagram. In the game state numbered 2, we can see that the machine player wins, as there are 3 X's in the diagonal. This state is given a score of +10. This branch of the tree diagram ends here.

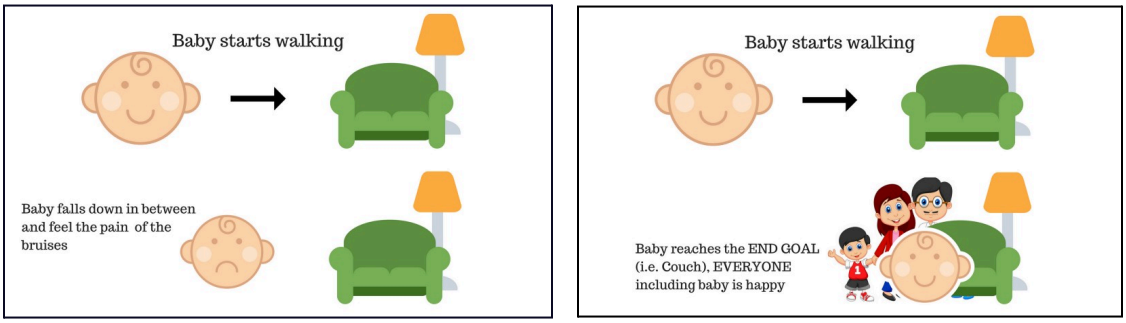
We can further branch off in the states numbered 3 and 4, and look at the opponent's move after the machine player played the corresponding move. We see that after the machine player has played state 3 or 4, the opponent can win (3 O's in the diagonal). This is given a score of -10.

After generating the entire tree, the machine player will pick the move that gives it a guaranteed win, which in this case is state 2. In both states 3 and 4, the opponent can win in the next turn, assuming it plays the optimal move.

The number of future moves it looks as is known as the **depth** of the minmax player. In our project specifically, we looked at depths 1, 6, and 11 of brute force.

This is a pretty good algorithm; it only has one major drawback. It takes a long time, as the number of moves to analyze grows exponentially. We decided to implement machine learning, specifically reinforcement learning, to increase the overall efficiency of our machine player.

Reinforcement Learning is a type of Machine learning where an agent learns to behave in an environment by performing specific actions and observing the rewards reaped from those actions. This is very similar to how humans learn; take how a baby learns to walk.



Images from Source [7]

By performing different actions to respond to the environment, the baby begins to understand how to walk. This is analogous to how a machine player will learn the strategy of Connect Four, by training on a training set and testing out the strategy.

A simple form of reinforcement learning is **tabular Q learning**, where the machine player learns a table that tells us the value (reward) of performing an action when the board is in a particular state. Once it learns this table for the Q function, the player (also referred to as AI agent) just needs to pick the move that gives the highest reward for a given state. The formal equation to pick a move is this:

$$\hat{a} = argmax_a[Q(s,a)]_,$$

Where Q is the Q function. The Q function takes the state (s) and action (a) and provides the expected reward when the action is taken by the agent when it observes state s.

Our AI agent learns the game by playing multiple rounds. During training, the agent keeps track of all the states and all the moves that it made. At the end of the game, it informs the agent whether it won or lost the game. Based on this information, the agent updates its Q table for the states and actions that it performed. Intuitively, if the AI agent wins the game, then it must have made good moves and those should have a higher reward. The actual math for this is given by the Bellman equation:

For final move:
$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha R$$
For previous moves:
$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\gamma max_a(Q(s',a'))$$

In this equation, we can see multiple variables, such as alpha, gamma, s, etc. Below we define what each of these actually mean:

Alpha - The learning rate (When it is close to zero: learning across many games, not just one game. Small importance is given to each individual game)

Gamma - the discount factor (When it is close to one, previous moves are given high importance)

s - the current state, and s' is the next state

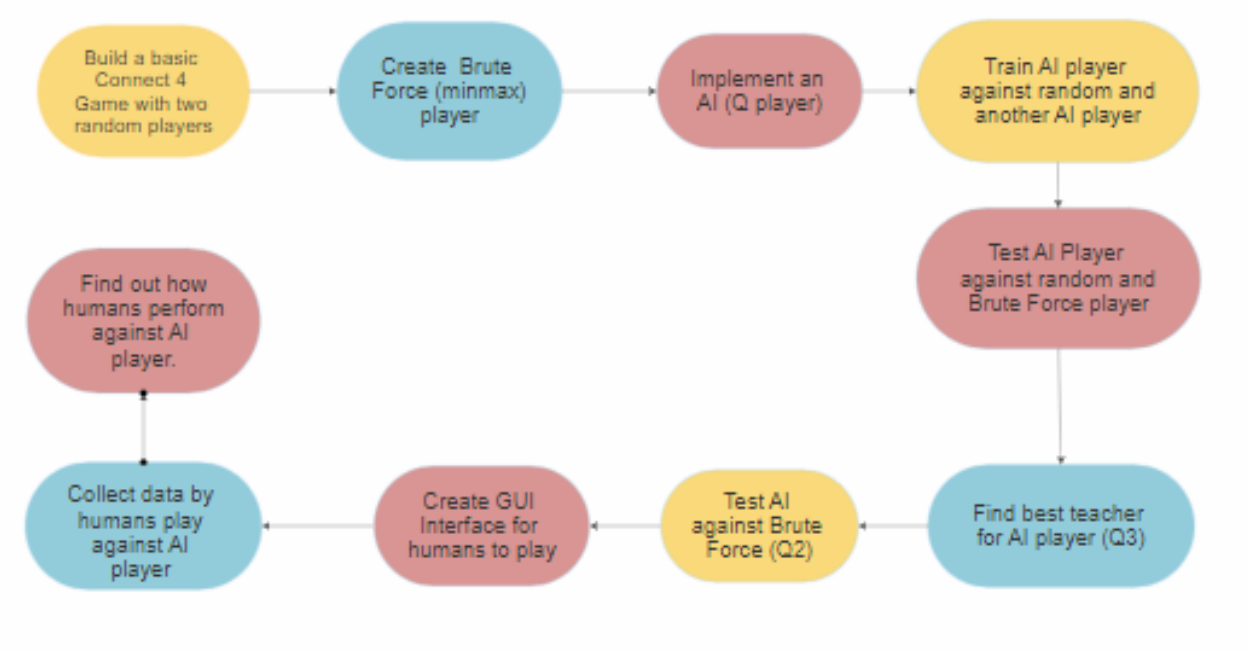
a - the action taken by the agent when it was in state s

Research Questions:

We aim to answer the following questions:

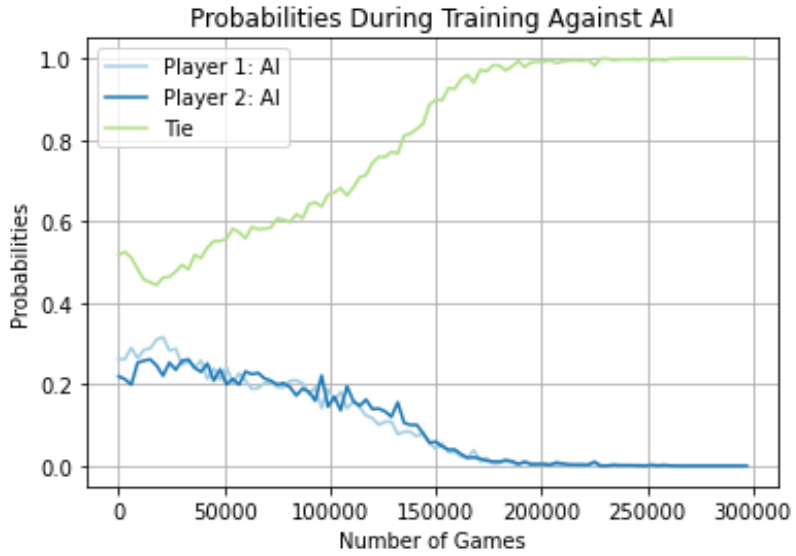
1. Is an AI Agent as good as a human in Connect 4?
2. Can an AI agent win against a brute force algorithm playing in Connect 4?
3. When training an AI agent, does it matter who the teacher is, whether it is random/brute force/another AI agent?

Procedure

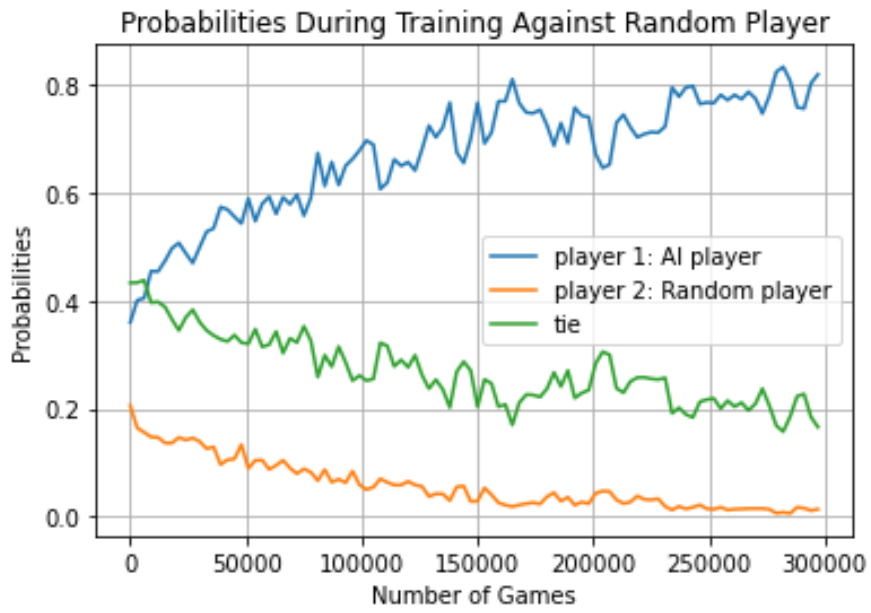


1. Begin by creating a simple Connect Four game in a 4 by 4 grid, with two random players.
2. Create a brute force or minmax player, that looks at future moves to determine the best move.
3. Implement the AI agent using Q learning and the bellman equation.
4. Run many experiments to determine how best to train the player.
5. To test the AI agent, have it play thousands of games against random and brute force players.
6. Based on the results from testing, answer questions 2 and 3.
7. To figure out how well it plays against a human, implement a simple GUI using Tkinter, and ask friends and family to play against the AI agent.
8. With the collected statistics answer question 1.

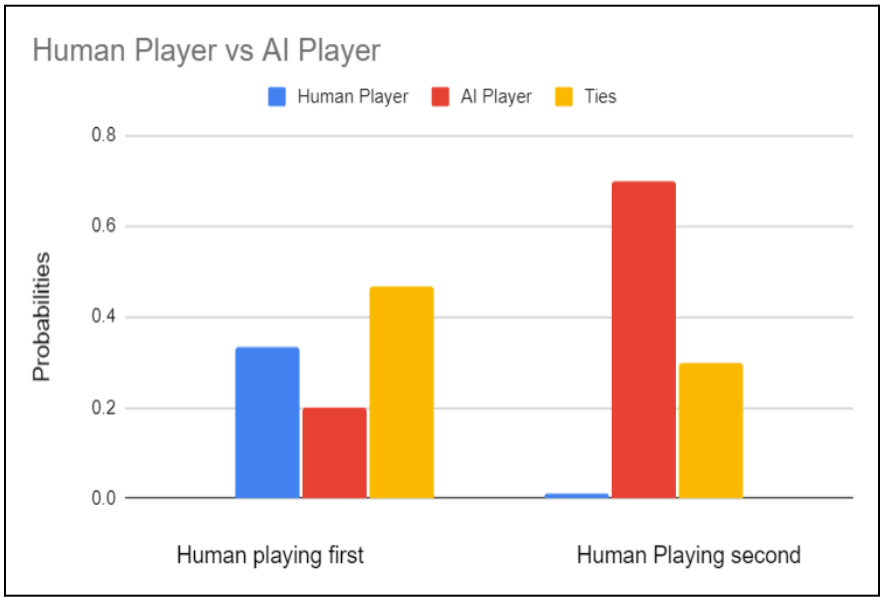
Results and Analysis



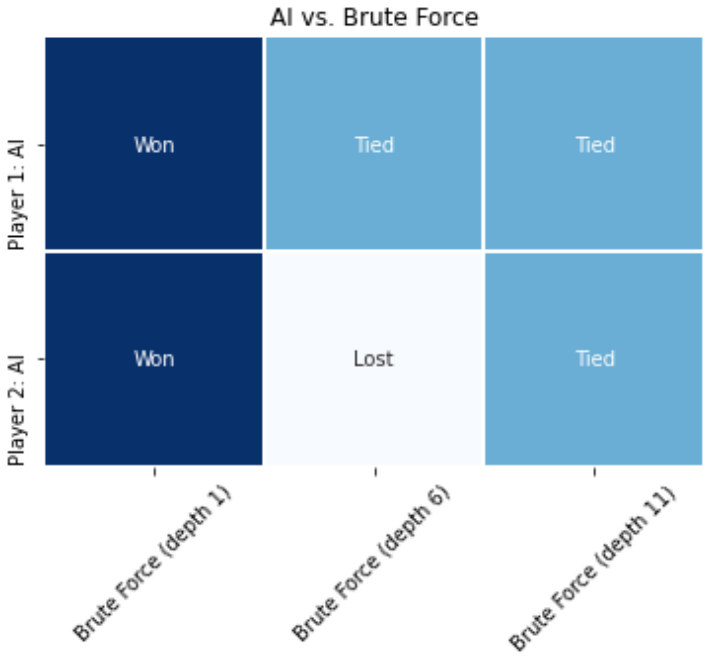
This plot shows what happens when 2 AI players train against each other over the course of 300,000 games. As shown, the probabilities of each player individually winning converges to zero and the probabilities of ties goes to 1. Despite the fact that both players started from scratch, they both became so good that they could not defeat each other. For this exact result, specific values were used in the Bellman equation: the learning rate was 0.01, the discount factor was 0.99, and the initial q value was 5.0.



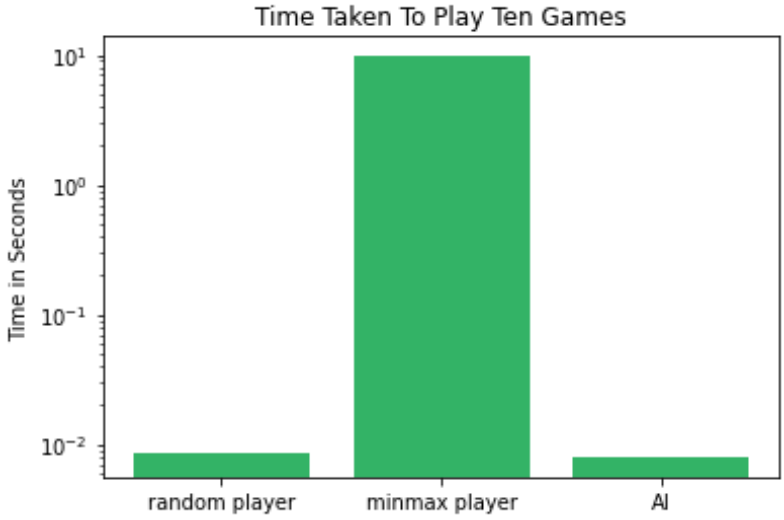
This plot shows what happens when an AI player trains against a random player over the course of 300,000 games. The plot shows a lot more variation, and this is partially because we are training against a player that plays randomly. However, we can see that the probability of the AI winning increases as the games proceed, which is consistent with what we expected. For this result, we once again used another set of specific values in the Bellman equation: the learning rate was 0.01, the discount factor was 0.99, and the initial q value was 3.0.



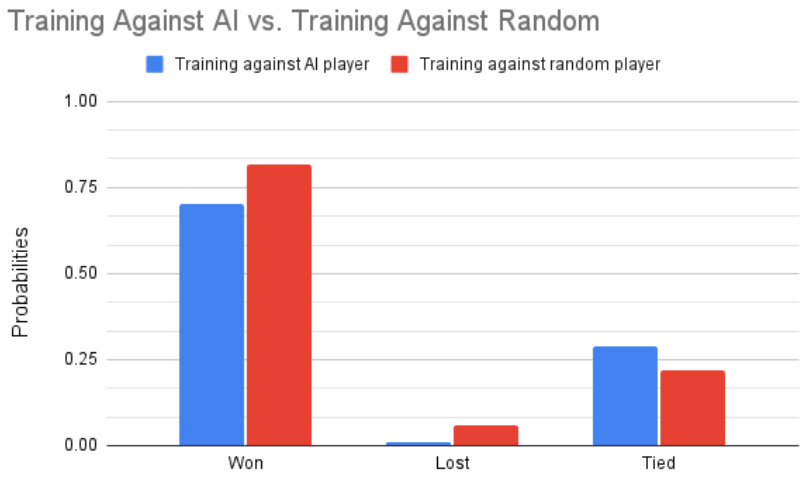
We were interested in seeing who wins when having a human and AI play against each other. We asked friends and family to play this game and obtained data for this plot. (Note: the AI we asked them to play against was trained against another AI). When the human played first, their winning probabilities of winning were slightly better than AI. However, when the human played second, their winning probabilities were much worse.



We next wanted to find out what happens when we have an AI and Brute Force play each other. We decided to analyze 3 different minmax depths of 1,6, and 11. As shown here, the AI agent wins against lower depths, but mostly ties against players of higher depths.



To figure out which player would be the best teacher, we first eliminated minmax player because it was not at all time-efficient. As shown here, compared to the random and AI players, the minmax player takes much more time to play 10 games. To train over 300,000 games would take a very long time, so we decided to just compare random and AI and see which is a better teacher.



To fully investigate this, we plotted winning, losing, and tying probabilities of two differently trained AI players. All of their probabilities are close, suggesting that both plates are very comparable.

Final Conclusions

1. Overall, the AI player plays better than the human player**.** It is interesting to see that humans can still play slightly better than AI when playing first.
2. We can see that the AI player generally wins against depth 1, but mostly ties against players of higher depths.
3. It was impractical to train against a brute force player because it took too much time to do so. However, training against random and AI produce very comparable trained Q players.

   We think this is because when training against a random player, AI gets more exposure to different board states than when training against another AI. However, training against another AI means the opponent is getting better, so the teacher improves as well. This means that they both should be comparable as they are positives to training both ways.

Bibliography

1. Cheung, Daniel. “An introduction to Q-Learning: reinforcement learning.” *freeCodeCamp*, 3 September 2018, https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/. Accessed 13 December 2022.
2. “Connect Four.” *Pencil and Paper Games*, http://www.papg.com/show?2XLU. Accessed 9 November 2022.
3. “Connect Four.” *Wikipedia*, https://en.wikipedia.org/wiki/Connect_Four. Accessed 9 January 2022.
4. Fox, Jason. “Tic Tac Toe: Understanding the Minimax Algorithm — Never Stop Building - Crafting Wood with Japanese Techniques.” *Never Stop Building - Crafting Wood with Japanese Techniques*, 13 December 2013, https://www.neverstopbuilding.com/blog/minimax. Accessed 15 December 2022.
5. Kaasinen, Jani, et al. “Understanding the Bellman Optimality Equation in Reinforcement Learning.” *Analytics Vidhya*, 13 February 2021, https://www.analyticsvidhya.com/blog/2021/02/understanding-the-bellman-optimality-equation-in-reinforcement-learning/. Accessed 17 December 2022.
6. Kadari, Prathima. “Introduction to Reinforcement Learning for Beginners.” *Analytics Vidhya*, 21 February 2021, https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners/. Accessed 4 December 2022.
7. Liverani, Chris. “A brief introduction to reinforcement learning.” *freeCodeCamp*, 27 August 2018, https://www.freecodecamp.org/news/a-brief-introduction-to-reinforcement-learning-7799af5840db/. Accessed 9 January 2022.
8. Sofela, Oluwatobi. “Minimax Algorithm Guide: How to Create an Unbeatable AI.” *freeCodeCamp*, 8 December 2020, https://www.freecodecamp.org/news/minimax-algorithm-guide-how-to-create-an-unbeatable-ai/. Accessed 13 January 2022.
9. “tkinter — Python interface to Tcl/Tk — Python 3.10.1 documentation.” *Python Docs*, https://docs.python.org/3/library/tkinter.html. Accessed 2 January 2022.

10. “Creating the (nearly) perfect connect-four bot with limited move time and file size.” *Towards Data Science*, https://towardsdatascience.com/creating-the-perfect-connect-four-ai-bot-c165115557b0. Accessed 7 March 2022.