Connect X Competition

Joe Johnson CIS 472/572 Dr. Humphrey Shi

Foreword

When creating my original plan, I had little-to-no understanding of machine learning (at least from a practical standpoint). This led me to having a goal, but no idea of the means to achieve it.

By the time of the intermediary check-in, I had gained a little knowledge of machine learning, and this led me to find a theoretical approach to help me possibly achieve this goal, but no real idea on the actual code required to turn theory into reality.

As the final deadline approached, I realized that my goal would not be reached (at least by machine learning methods), and as such I decided to set my sights on a new goal.

1. Introduction

This document contains my steps, thought process, changes and discoveries from my entries into the Kaggle Connect X competition [1]. The general goal of the competition is to create an agent to compete against other agents in a game of Connect X. The agents then are paired up against other agents of a similar skill rating. After these pairs match, points will be deducted or added from an agent's overall rank based on the results of the competition. This gives the agent a rank within a leaderboard, similar to a chess rank or competitive videogame rank.

1.1. Rules of Connect X

Connect X is based off the Hasbro game Connect Four [2]. The game is played between two players, alternating turns placing tokens on a grid. The grid is comprised of six rows and seven columns. On a player's turn, they must place a token in one of the seven columns, the lowest available cell in that column will be the player's move. The objective of the game is to get four tokens in-a-row: whether it be horizontal, diagonal, or vertical (see Figure 1). The game continues until one player wins, or until no moves are left resulting in a draw.

Connect X is named as such because of the potential for future changes to the game rules. Kaggle stated that "[t] he default number is four-in-a-row, but we'll have other options to come soon."

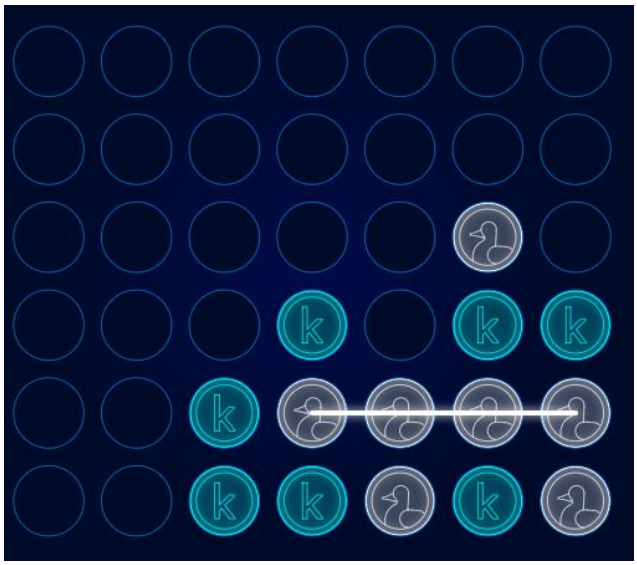

Figure 1: Example game board with 'white' winning

1.2. Objectives of my Research

The original goal I had was to reach the top 50% of the leaderboard using my machine learning agents. As my testing went on, I realized that this was a much steeper hurdle than I previously anticipated. I was able to achieve this goal (and then some), but not with a machine learning trained agent. I was only able to do this with a programmed MiniMax implementation.

After noticing this, I did more research into the entries that were holding the top spots in the leaderboard. I found that most of the implementation were not actually machine learning, but instead were forms of NegaMax, MiniMax or other "artificial intelligence" implementations. The leaderboard toppers that were actual machine learning were trained for several days (see Figure 2). I tried to have some versions that were trained for similar amounts of time, but despite Google Colab saying I had plenty of memory and processing overhead, I found that I would have connection

issues with Google Colab when passing the 40,000th game training session (roughly 11 hours in). As such, my explored states were lacking leading to less than satisfactory results in the ranking system.

At this point, I realized I would never reach the top 50% from one of my machine learning implementations (at least not with my current skill set). At this point I decided to focus more on a result comparison between my own MiniMax implementation and my Q-Learning Implementation.

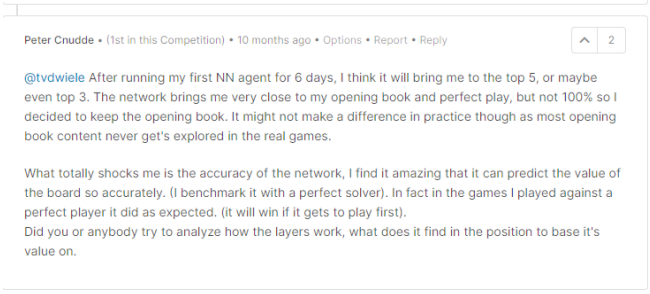

Figure 2: Comment from 1st place in competition 10 months prior to my entry

1.3. Q-Learning

Q-Learning is a model-free reinforcement learning. This is what led me to choosing this implementation as I wanted to try something different than all the model-based learning methods we learned in class. I stumbled across a Kaggle Notebook that outlined how to start with this method [3]. Which led me to look more into what exactly Q-Learning is. I found that Q-Learning is an Epsilon-Greedy Algorithm, meaning that it will explore new states Epsilon amount of the time (usually represented as number from 0 to 1) and the rest of the time it will choose what it has deemed to have the highest potential final score [4]. 'Q' in this formula has a few accepted definitions, one of the most accepted being 'Q' standing for quality. The 'Q' essentially represents the path it found to have the highest potential reward.

1.4. NegaMax

NegaMax is one of the implementations that is built into Kaggle and the Connect X environment for testing against. NegaMax is an agent that performs its moves based on what it calculates to give its opponent the lowest chance of winning (as opposed to doing a move that it believes to give itself the best odds of winning).

Some of the highest placing agents were implementations of NegaMax, or at least had elements of it mixed in (Figure 3). As previously stated, most of the top performing agents that I could get any information on were implementations of some form of either NegaMax or MiniMax.

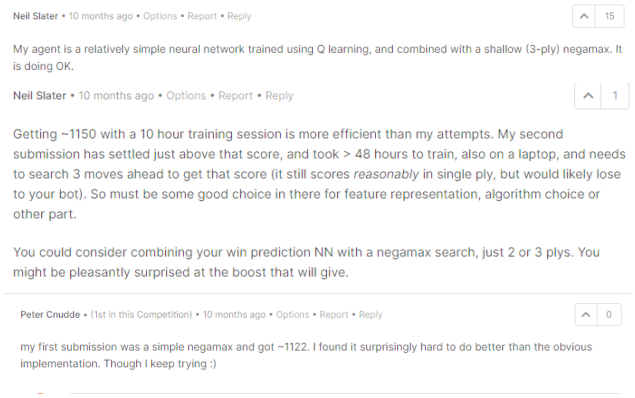

Figure 3: Top 10 (and former top 10) implementing NegaMax

1.5. MiniMax

MiniMax is an "artificial intelligence" formula that tries maximizing the potential outcome while assuming that the opponent will be making moves that will minimize the potential outcome of the one implementing the MiniMax formula. In the leaderboard, I saw two of the highest-ranking agents with Kaggle Notebooks to be implementing a form of MiniMax (Figure 4). This led to me doing my own implementation of a varying depth MiniMax (meaning I allow the changing of one variable in order to change the number of moves the agent looks ahead) to test against and to set as a comparison benchmark for my machine learning trained agents.

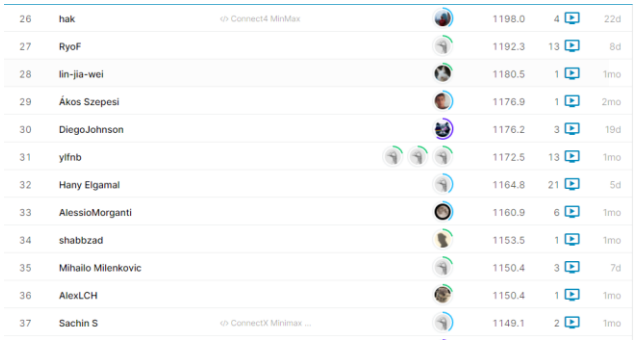

Figure 4: 26th and 37th place with notebooks on MiniMax

1.6. Parameters tested

My original implementation, based off another user's Kaggle Notebook I found very helpful, was done entirely with q-learning. After getting it to successfully run, I tried 5 variations of each learning rate, initial epsilon (chance of exploring), epsilon decay rate, minimum epsilon, gamma (discount rate) and alpha decay rate/step in conjunction (for a total of 1 control and 24 test scenarios) on a test of 5,000 training games each test averaging around 2 hours to train. The best results were submitted to the Kaggle Connect X competition and after several days settled at a

meagre score of 247.3 (at the time of writing this). I then took those parameters and applied them to a longer run of 20,000 trial games, taking around 7.5 hours to train with the epsilon decay rate scaled proportionally, this settled around 315.2 (at the time of writing this). This is another underwhelming score; it was at this point I investigated the average training time as mentioned in Section 1.2. I then tried several longer trial runs, once again with epsilon decay scaled proportional to number of games trained. All game counts of 50,000 or higher (inclusive) that I tried caused Google Colab to have connection issues (as mentioned in Section 1.2), so the largest game count I was able to train was 40,000, which took close to 15 hours to complete. This one still only reached 364.9 (again, at the time of writing this).

After this I implemented a MiniMax function based off a notebook I found [5]. I added the pieces into my ConnectX_Notebook (the included ipynb file) for testing comparison. After implementing these I then created an agent that calculated five moves ahead using these methods, this one showed extremely promising results, hitting an overall high of 1241 and sitting at 1213.5 at the time of writing this. This got me up to 20th overall on the leader board; thus, my goal was achieved, but not under the methods I had originally set for myself. I then implemented a seven-move depth MiniMax, but it took too long in calculation with my minimal optimization to pass certification. I was able to implement a six-move depth MiniMax, however, and this looked like it would reach my highest overall, but it seems to do comparable to the five-move depth MiniMax and is currently sitting at 1207 for its rank. All the current standings of my bots can be seen in Figure 5 and me on the leader board can be seen in Figure 6 [6].

Figure 5: Current Standings of my submissions

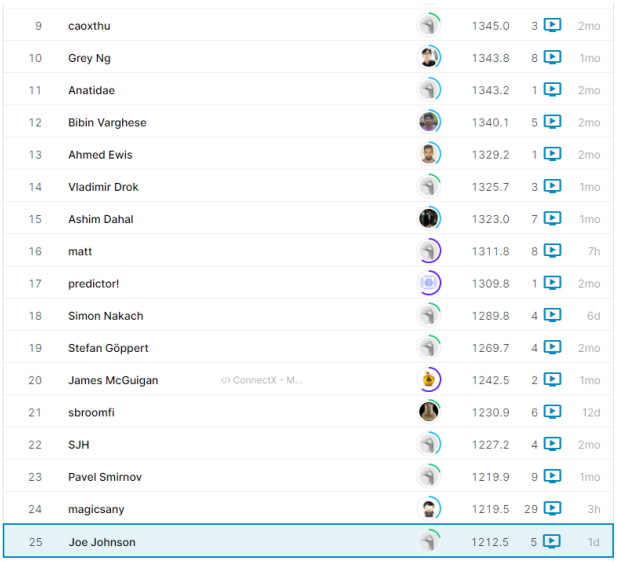

Figure 6: Current leader board standings

1.7. Results

All the Q-Learning trained submissions were tested against Connect X's included NegaMax agent, an agent that plays random moves and a 1-step look-ahead MiniMax agent. The test consisted of 100 games with the results of winning percentages reported as a decimal. The results for NegaMax were not included in the following Figures, but it ended with NegaMax winning 100% of the games against the trained agents.

The first submission, of 5,000 training games, explored close to 17,000 different possible states of the game (a paltry number when you consider there are 4.18×10^{11} possible permutations) [7]. The results are reported in Figures 7 – 10.

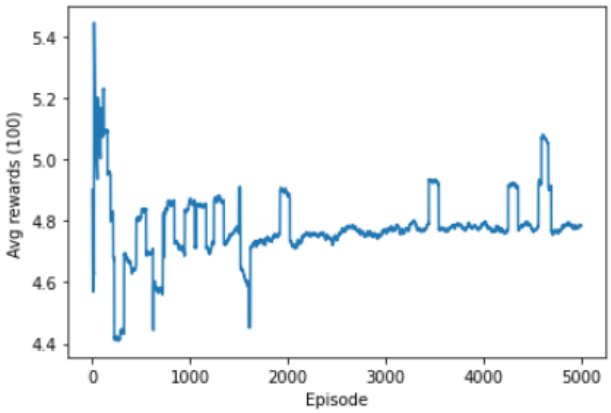

Figure 7: Average rewards over 5,000 training episodes

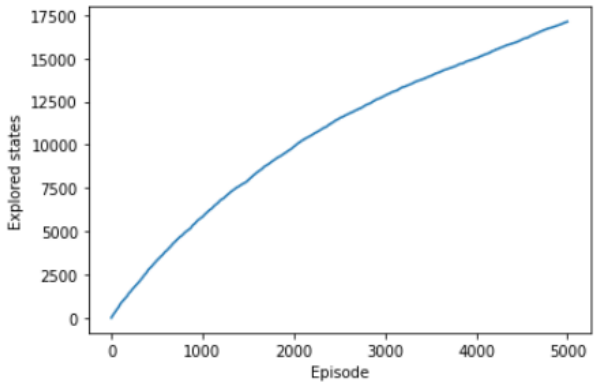

Figure 8: States explored over 5,000 training episodes

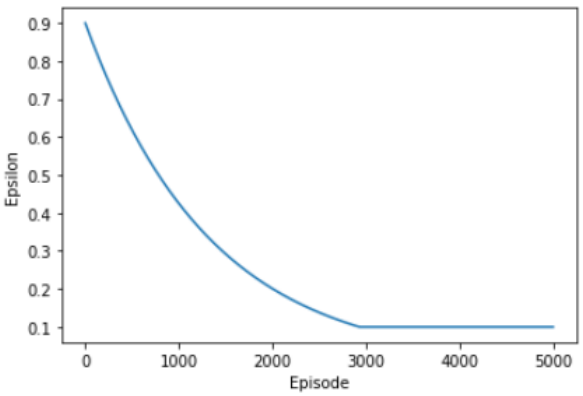

Figure 9: Epsilon value over 5,000 training episodes

```
my agent vs random
Agent 1 Win Percentage: 0.49
Agent 2 Win Percentage: 0.49
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

depth 1 vs negamax
Agent 1 Win Percentage: 0.53
Agent 2 Win Percentage: 0.45
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

my agent vs depth 1
Agent 1 Win Percentage: 0.0
Agent 2 Win Percentage: 1.0
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0
```

Figure 10: Winning percentages after 5,000 training episodes

The second submission of 20,000 training games reached almost 60,000 different possible states. The results are reported in Figures 11 - 14.

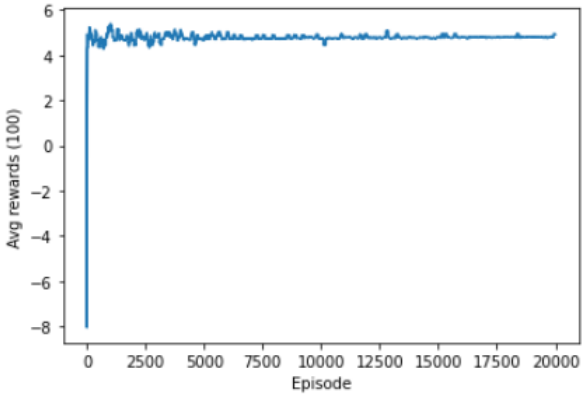

Figure 11: Average rewards over 20,000 training episodes

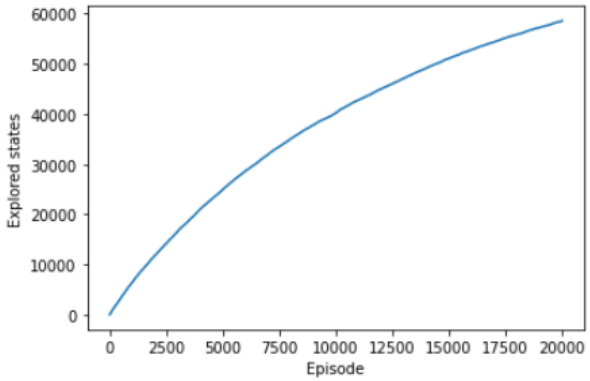

Figure 12: States explored over 20,000 training episodes

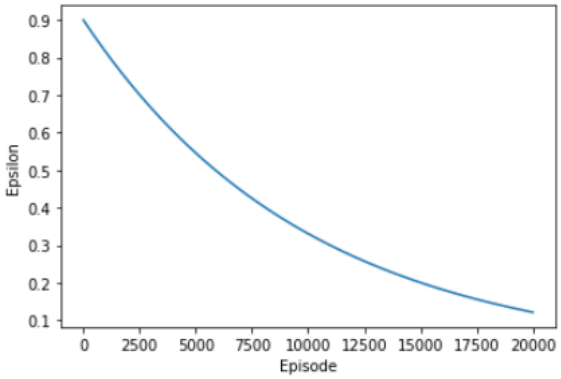

Figure 13: Epsilon value over 20,000 training episodes

```
my agent vs random
Agent 1 Win Percentage: 0.55
Agent 2 Win Percentage: 0.45
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

depth 1 vs negamax
Agent 1 Win Percentage: 0.53
Agent 2 Win Percentage: 0.43
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

my agent vs depth 1
Agent 1 Win Percentage: 0.0
Agent 2 Win Percentage: 1.0
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0
```

Figure 14: Winning percentages after 20,000 training episodes

The third submission of 40,000 training games achieved close to 90,000 different states explored. The results can be seen in Figures 15 – 18.

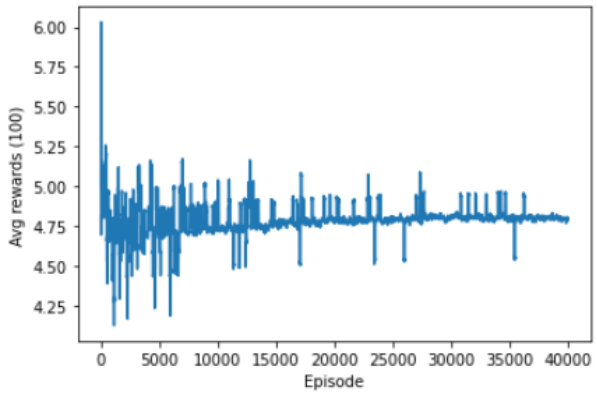

Figure 15: Average rewards over 40,000 training episodes

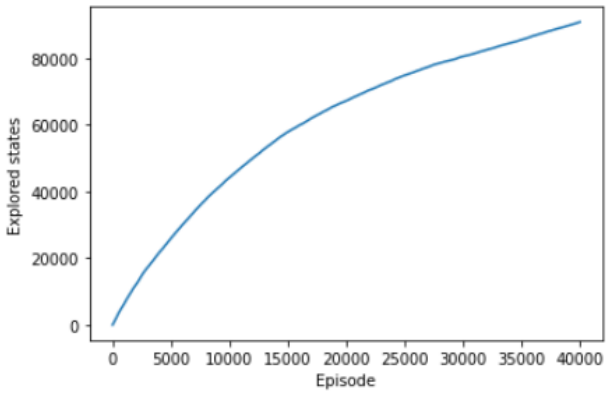

Figure 16: States explored over 40,000 training episodes

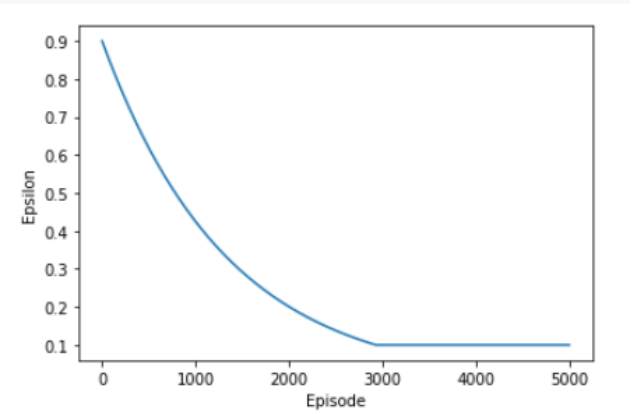

Figure 17: Epsilon value over 40,000 training episodes

```
my agent vs random
Agent 1 Win Percentage: 0.57
Agent 2 Win Percentage: 0.43
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

depth 1 vs negamax
Agent 1 Win Percentage: 0.45
Agent 2 Win Percentage: 0.54
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

my agent vs depth 1
Agent 1 Win Percentage: 0.0
Agent 2 Win Percentage: 1.0
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0
```

Figure 18: Winning percentages after 40,000 training episodes

The fourth and fifth submissions (the MiniMax submissions) can be seen facing MiniMax and each-other in Figure 19.

```
depth 5 vs negamax
Agent 1 Win Percentage: 0.97
Agent 2 Win Percentage: 0.0
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

depth 6 vs negamax
Agent 1 Win Percentage: 0.98
Agent 2 Win Percentage: 0.01
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0

depth 5 vs depth 6
Agent 1 Win Percentage: 0.5
Agent 2 Win Percentage: 0.5
Number of Invalid Plays by Agent 1: 0
Number of Invalid Plays by Agent 2: 0
```

Figure 19: MiniMax results of depth 5 and 6 vs each-other and NegaMax

1.8. Conclusion

I found that my methods of pure Q-Learning lacked the ability to come remotely close to an average result. I believe if I stuck to a pure form of Q-Learning I would need to train on a machine I could just let run for many days on end and/or get creative with my networking.

All the top contenders seemed to have some implementation combining one form of "artificial intelligence" (at least) combined with some form of training. Many also included opening move books pulled from an external source. I failed to implement any of these in my trials, so I can't say for sure how much faster my agents could have trained or how much better they could have performed, but I hypothesize they would perform significantly better.

As I am taking this class simultaneous to CIS 471 (Intro to Artificial Intelligence), I am proud of how well my MiniMax implementation performed. That being said, I see many areas I can improve upon within my machine learning knowledge and skills moving forward. I hope to carry all I have learned from this class and this assignment into any future machine learning projects I might have.

References

[1] https://www.kaggle.com/c/connectx/overview
[2] https://shop.hasbro.com/en-us/toys-games?q=((navigation.brand.restName%3Ahasbro-games))
[3] https://www.kaggle.com/phunghieu/connectx-with-q-learning
[4] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56
[5] https://www.kaggle.com/hak999/connect4-minmax-conventional-baseline
[6] https://www.kaggle.com/joehoho
[7] https://mathworld.wolfram.com/Connect-Four.html