Replicating Alpha Zero for Connect Four

Adeet Parikh
Computer Science
University of Texas at Austin
Email: aaparikh@utexas.edu

Andrew Wu
Computer Science
University of Texas at Austin
Email: andrewmwu@utexas.edu

***Abstract*—Connect Four is a classic two-player game that, despite being solved, presents as an interesting benchmark game for deep learning agents. In this paper, we replicate techniques in Alpha Zero for Connect Four and introduce a faster data collection strategy that generates training data more efficiently, allowing for the training of an agent we dub ”Alpha Zero at Home.” We compare the performance of our novel agent against standard Alpha Zero, Alpha Beta Pruning, and an optimal solver, demonstrating a significant improvement in strength with our method, while using fewer computational resources.**

I. Introduction

Connect Four is a classic two-player, turn-based, deterministic, zero-sum, adversarial game that involves placing a single piece in a column each turn in a 7x6 board until either player gets four pieces in a row horizontally, vertically, or diagonally, leading to that respective player's win, or until the board is full, leading to a tie. In recent years, Alpha Zero, a self-play reinforcement learning and Monte Carlo Tree Search algorithm, has shown remarkable performance in games like Go, Chess, Shogi, and more [6]. In this paper, we present a novel neural network training approach that trains an Alpha Zero agent with considerably less compute. We achieve this by augmenting data collection with a greedy strategy to generate training data more efficiently, then we compare against Alpha Zero, Alpha Beta Pruning, and Optimal agents.

II. Related Work

Connect Four has historically been used for benchmarking AI algorithms [7] [8], although because it is a solved game [2], it does not receive much attention anymore. We find this to be beneficial because we can play against an optimal agent, and the game's relatively small but nontrivial branching factor compared to Go or Chess allows for achievable implementation and results from neural network based approaches without extreme levels of compute (41 TPU years [6]). Recently, AlphaGo and AlphaZero [6] have pioneered the use of deep reinforcement learning in combination with Monte Carlo Tree Search for adversarial, deterministic 2-player board games and has been applied to Chess, Shogi, Go, etc. These methods have dominated the state of the art for the time being, but a problem with these methods is the prohibitive levels of compute needed to train such algorithms. While there has been some recent work such as MuZero that makes AlphaZero deal with unknown rules and is more sample efficient, more work is needed to reduce the computational requirements of these algorithms [4]. There also exists MCTS approaches that take advantage of the structure of the game of Connect Four, but we focus on a more general aspect of the MCTS tree. We also utilize other classic neural network training techniques such as augmenting the data with board reflection and side reflection which have been employed in other board games and beyond [5].

III. Our Implementation

Our goal was to implement a neural network based agent to play Connect Four, based primarily on the methods in Alpha Zero [6]. The main obstacle was our lack of computing resources. We created both an agent that learns very similarly to Alpha Zero, as well as an agent that uses the same Monte Carlo Tree Search variant, but with a far less compute intensive training loop. This will be referred to as Alpha Zero at Home (sorry).

A. Implementing Alpha Zero for Connect Four

Our Alpha Zero agent is loosely based on the description in the KataGo paper [9]. The neural network is a multi-head network consisting of a 4x4 convolution, natural for Connect Four, and some linear layers. It takes the board state and current player as inputs, and outputs the win probability of the current player and a probability distribution over actions for the player to take in the given position. We were unsure how to choose a model size, because for a small game like Connect Four, too many parameters would be unnecessary. We generally saw performance increase with the number of parameters, though naturally, training and evaluation time increased as well. We kept our model relatively small, because with our limited compute, we needed to reduce our simulation time as much as possible.

The idea behind Alpha Zero's MCTS approach is to use the value (win probability) and policy prior generated by the network to guide its search. Unlike traditional MCTS, the Alpha Zero version does not perform rollouts to terminal states. While this is not very expensive in Connect Four, with a maximum of 42 moves, this makes a lot of sense for games like Go and Chess. Instead, after the expansion step, the algorithm immediately backpropogates the value of the newly expanded node to all nodes in the path. The search chooses the best child at each node using ”Polynomial Upper Confidence Trees”. For a given parent node n, we choose the child c which maximizes this formula described in [9].

$$PUCT(c) = V(c) + c_{PUCT} P(c) \frac{\sqrt{N(n)}}{1+N(c)}$$

$V(c)$ is the average predicted utility of all nodes in c's subtree, c_{PUCT} is a constant set to 1.1, $P(c)$ is the policy prior from the neural network, and $N(c)$ is the number of visits previously made to the given node.

The training loop starts by using the Alpha Zero MCTS implementation to play competitive games against itself. Since the agent is normally deterministic, we followed the suggestion in KataGo[9] to add Dirichlet noise to the policy prior. The causes the search to further explore nodes that it would have disregarded otherwise. Each state-turn pair is assigned either 1, 0.5, or 0, depending on the result of the game from the current player's perspective. These values are used to train the value head of the neural network. With each action, we also save the distribution of node visits from the parent, which is used to train the policy head of the network.

To increase the quality and diversity of the training data, we employed the Playout Cap Randomization technique from KataGo[9]. This works by simulating most moves with a low number of simulations (we chose 25), and with probability 0.25, choosing an action with a much larger number of simulations (500). Only the moves with high quality data are used as training examples. This allows us to train on higher quality data across more games, but comes at the cost of fewer total examples.

While we are confident that our implementation of Alpha Zero is correct, with our limited compute, we have not been able to train a high performing agent yet. It is able to play at a superhuman level, but still struggles against Alpha Beta pruning with high depth. During an overnight training run on a 2019 Intel i5 Macbook Pro, it was able to simulate approximately 5,000 games. We believe this is simply not enough data to learn meaningful information about the game.

B. Alpha Zero at Home

To allow faster data generation, we wanted an approach that does not require using MCTS at this stage of the training pipeline. To collect data, we used an agent which greedily picks the next state with the highest value according to the neural network. It also chooses a random action with probability 0.2 to make the agent explore a variety of states. This allows it to collect data from far more games in a smaller amount of time. This approach also allowed us to do data augmentation that isn't possible with the Alpha Zero approach. We could reflect the board and switch all the pieces, while still knowing the value, which wouldn't work when doing MCTS.

The true value of a state was approximated using Temporal Difference (TD(λ)) Learning. Unlike the Alpha Zero approach, this method discounts the state values by $\gamma = 0.9$, so states early in the game will have lower value. We also use $\lambda = 0.9$ to give a larger proportion of credit for the reward to distant states [1]. TD learning stabilizes learning by using the network's value of the next state to update the value of the current state. This way, the model can have more consistency between consecutive states instead of being pulled in opposite

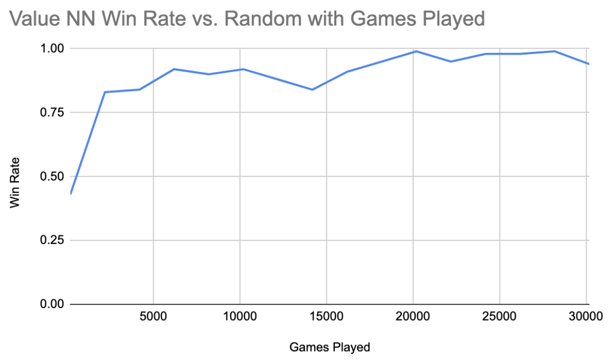


Fig. 1. Win rate of greedy value agent against a random player while training

directions from games with different outcomes. The value is 1 for a state where player 1 wins, -1 when player 2 wins, and 0 for a draw. It is trained using MSE Loss. Surprisingly, this performed better than BCE loss when testing.

The value network training loop was able to go through 30,000 games worth of training examples, using every state and its augmentations, in just a few hours. We struggled to find a good way of reliably evaluating the network, but as an approximation, we used the win rate of the greedy value agent against a random player (See Figure 1).

The model quickly reached a positive win rate, and was able to beat the random agent almost every time. Still, this was not good enough to consistently defeat a human player. We decided to follow the idea behind Alpha Zero MCTS, using the value network in conjunction with a policy network.

We created a policy network with a very similar structure to the value network, simply changing the output to a softmax over 7 outputs for each of the possible actions. In Alpha Zero, this policy network is normally trained using the MCTS visit distribution, but this is what makes its training so time intensive. We instead trained it to mimic the greedy value player, as this was an easy way to ensure that they worked together without using MCTS. To collect data, we had two random agents play out a game, then picked a random state from the game and had the greedy value agent self play from that point. Evaluation for this model was much easier, as all we needed to do was play out games and see whether the policy network output matched the action of the greedy player. The policy network consistently improved its accuracy on the validation states during training, so we let it train for 50,000 games (See Figure 2). Note that these were all partial games, played starting from random states, so it can train on a more diverse set of games than the value network, but a similar number of states.

The two networks were used together to implement an MCTS agent that functions very similarly to Alpha Zero. Since the value is absolute (not based on the current player), it is multiplied by the current player's id (± 1) before using it in the PUCT formula.

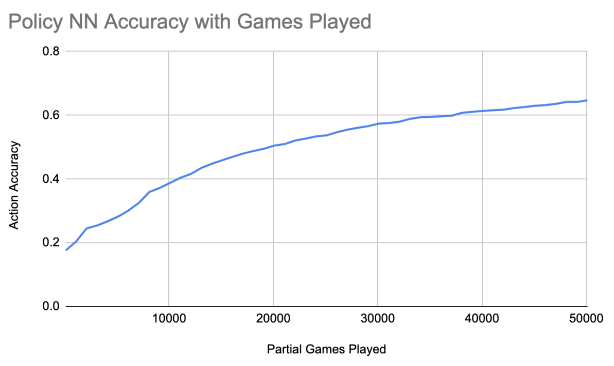


Fig. 2. Accuracy of policy network prediction of greedy value player actions while training

IV. Experiments

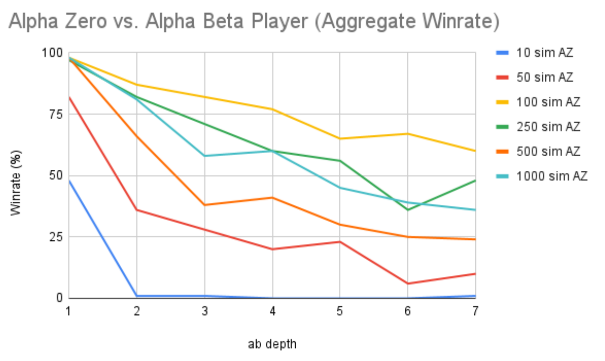


Fig. 3. Alpha Zero vs. Alpha Beta Player winrates at different depths and simulations

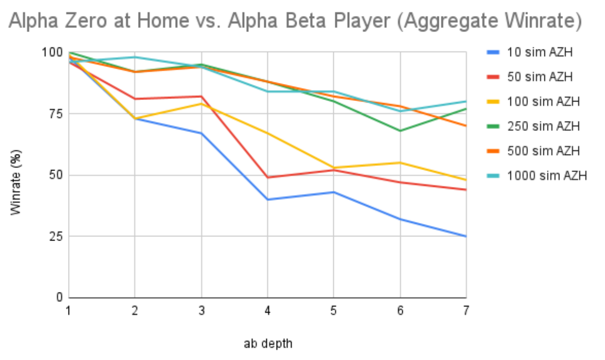
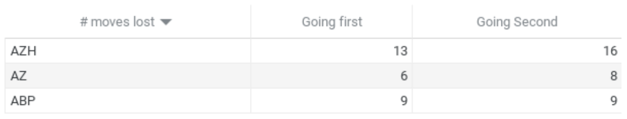


Fig. 4. Alpha Zero at Home vs. Alpha Beta Player winrates at different depths and simulations. Note that if we run Alpha Zero at Home versus Alpha Zero, Alpha Zero at Home would beat it every time with equal simulation count so we decided this statistic was not interesting.



# moves lost	Going first	Going Second
AZH	13	16
AZ	6	8
ABP	9	9

Fig. 5. AZ, AZH, ABP vs. Optimal player

For experimental evaluation, we ran 100 games for black and white first for Alpha Zero and Alpha Zero at Home with different simulations for the MCTS (10, 100, 250, 500, 1000) versus an Alpha-Beta Pruning agent with different depths (1 to 7) and graph the win rates in 4 and 3. We find that interestingly enough, for Alpha Zero the best simulation count was 100 simulations, while in Alpha Zero at Home, it follows the theory that higher simulations should result in a better agent. Overall, Alpha Zero at Home is a stronger agent by about 20 percent win rate across the board, and in general Alpha Zero at Home is relatively better at defending with Black than Alpha Zero is. Comparing all the agents against an optimal agent 5 [3], we find that the Alpha Beta Pruning agent and Alpha Zero agent perform similarly and perform 2-3 blunders per game resulting in losing in sub-10 individual moves while Alpha Zero at Home performs around 1-2 blunders per game resulting in losing in around 15 individual moves. Although this is a lot better, it is interesting to note that even though White should win every game, none of the agents can force this win and oftentimes will lose faster playing first than defending. More results including side statistics (white is going first, black is going second) are located in the Appendix.

V. Conclusion

In this paper, we have presented a new greedy training strategy for Alpha Zero agents, which significantly reduces computation requirements for achieving decent-performing agents, which could potentially be used to bootstrap normal Alpha Zero training. Our experimental results show that our approach outperforms standard Alpha Zero in Connect Four across the board, but cannot beat an optimal solver and force a win going first despite making significantly fewer mistakes.

A. Future Work

With more time, we would like to obtain more training data for the Alpha Zero approach, and experiment with more training methods for our Alpha Zero at Home approach. With only 5,000 games of training, the Alpha Zero agent did not perform as well as it could have. We would likely need more compute to collect a significant amount of data, but it is something we would like to look into.

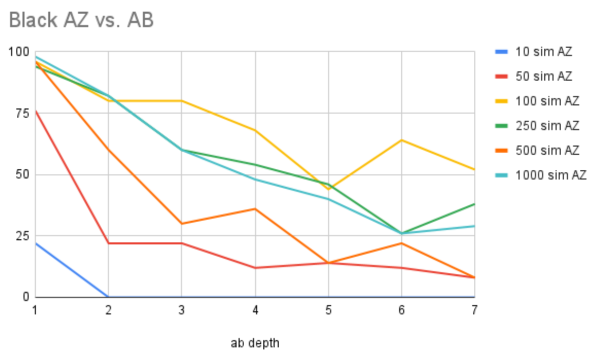
The Alpha Zero at Home approach worked well in our testing, but we would like to try some other variations. Currently, we have only tested the c_{PUCT} param value given in the KataGo paper, but we may need to vary it since the value is scaled differently. We also considered training the value network the same way, then training the policy network to mimic the MCTS visit distribution instead of the greedy value player.

VI. Appendix

ab depth	10 sim AZ	50 sim AZ	100 sim AZ	250 sim AZ	500 sim AZ	1000 sim AZ
1	48	82	98	97	98	98
2	1	36	87	82	66	81
3	1	28	82	71	38	58
4	0	20	77	60	41	60
5	0	23	65	56	30	45
6	0	6	67	36	25	39
7	1	10	60	48	24	36

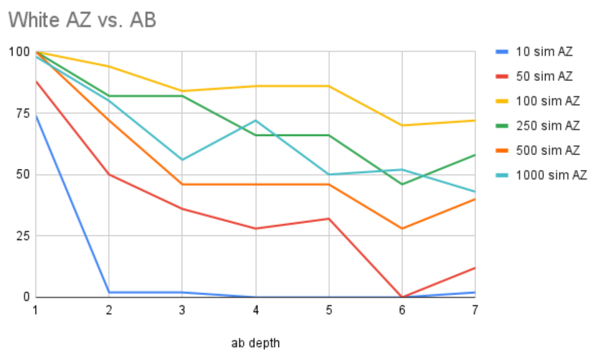
ab depth	10 sim AZH	50 sim AZH	100 sim AZH	250 sim AZH	500 sim AZH	1000 sim AZH
1	98	96	99	100	98	96
2	73	81	73	92	92	98
3	67	82	79	95	94	94
4	40	49	67	88	88	84
5	43	52	53	80	82	84
6	32	47	55	68	78	76
7	25	44	48	77	70	80

Fig. 6. Data for 3 and 4



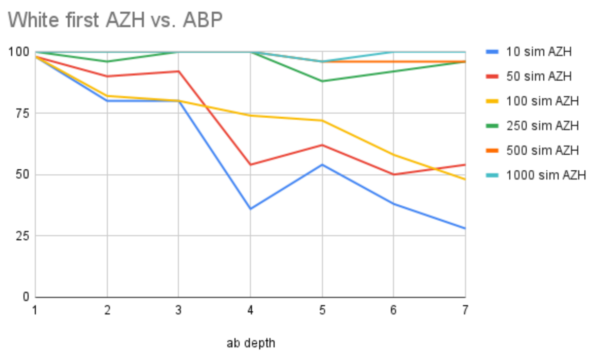
ab depth	10 sim AZ	50 sim AZ	100 sim AZ	250 sim AZ	500 sim AZ	1000 sim AZ
1	22	76	96	94	96	98
2	0	22	80	82	60	82
3	0	22	80	60	30	60
4	0	12	68	54	36	48
5	0	14	44	46	14	40
6	0	12	64	26	22	26
7	0	8	52	38	8	29

Fig. 8. Black first winrates for Alpha Zero vs. AB



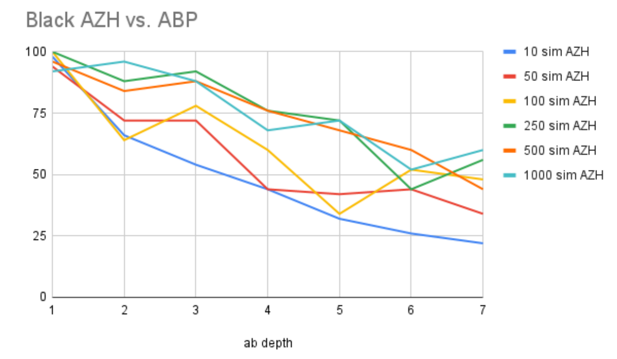
ab depth	10 sim AZ	50 sim AZ	100 sim AZ	250 sim AZ	500 sim AZ	1000 sim AZ
1	74	88	100	100	100	98
2	2	50	94	82	72	80
3	2	36	84	82	46	56
4	0	28	86	66	46	72
5	0	32	86	66	46	50
6	0	0	70	46	28	52
7	2	12	72	58	40	43

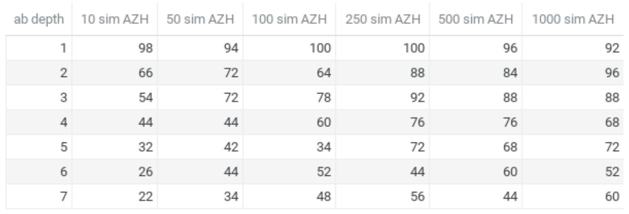
Fig. 7. White first winrates for Alpha Zero vs. AB



ab depth	10 sim AZH	50 sim AZH	100 sim AZH	250 sim AZH	500 sim AZH	1000 sim AZH
1	98	98	98	100	100	100
2	80	90	82	96	100	100
3	80	92	80	100	100	100
4	36	54	74	100	100	100
5	54	62	72	88	96	96
6	38	50	58	92	96	100
7	28	54	48	96	96	100

Fig. 9. White first winrates for Alpha Zero at Home vs. AB





ab depth	10 sim AZH	50 sim AZH	100 sim AZH	250 sim AZH	500 sim AZH	1000 sim AZH
1	98	94	100	100	96	92
2	66	72	64	88	84	96
3	54	72	78	92	88	88
4	44	44	60	76	76	68
5	32	42	34	72	68	72
6	26	44	52	44	60	52
7	22	34	48	56	44	60

Fig. 10. Black first winrates for Alpha Zero at Home vs. AB

REFERENCES

[1] Apr 2023. URL https://en.wikipedia.org/w/index.php?title=Temporal_difference_learning&oldid=1149325698. Page Version ID: 1149325698.

[2] Victor Allis. A knowledge-based approach of connect-four: The game is solved: White wins. Master's thesis, Vrije Universiteit, Amsterdam, Netherlands, 1988. URL http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf.

[3] Pascal Pons. Connect 4 solver, n.d. URL https://connect4.gamesolver.org/. Accessed: 2023-04-24.

[4] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

[5] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:60, 2019. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

[6] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. Dec 2017. URL http://arxiv.org/abs/1712.01815. arXiv:1712.01815 [cs].

[7] Lukas Tommy, Mardi Hardjianto, and Nazori Agani. The analysis of alpha beta pruning and mtd(f) algorithm to determine the best algorithm to be implemented at connect four prototype. In *IOP Conference Series: Materials Science and Engineering*, volume 190, page 012044. IOP Publishing, 2017. doi: 10.1088/1757-899X/190/1/012044. IAES International Conference on Electrical Engineering, Computer Science and Informatics, 23-25 November 2016, Semarang, Indonesia.

[8] Hui Wang, Michael Emmerich, and Aske Plaat. Monte carlo q-learning for general game playing, 2018.

[9] David J. Wu. Accelerating self-play learning in go. Nov 2020. URL http://arxiv.org/abs/1902.10565. arXiv:1902.10565 [cs, stat].