Learning Connect Four

Andrea Malleo AM101@NYU.EDU

Abstract

In this paper we scale down the framework of AlphaZero, investigating whether the simpler game of Connect Four can be solved with a commensurate reduction in model size and compute time. In the end it is clear that the expectations for training time reduction were set too high, and more dedicated optimization of certain parameters would be necessary to better replicate the results.

1. Introduction

General game playing is concerned with unspecialized AI systems capable of learning to play many different types of games well. Presented here is a reimplementation of the AlphaZero [5]method of mastering games through reinforcement learning. This method achieved world class performance for the games of chess, Go, and shogi. The sheer magnitude of the resources and time taken to achieve this feat is striking, but perhaps it just matches the complexity of the problems beings solved. Here, we adapt the framework to Connect Four. The key questions are how instrinsic to success is this scale, and how general purpose does this method turn out to be.

2. Background

There are just two game agnostic components to the AlphaZero system: the deep neural network and the Monte Carlo tree search algorithm. The network $(p, v) = f_\theta(s)$ takes the board state over T time steps as input. The outputs are p, a probability distribution vector over the actions to take, and v, the expected outcome of the game from the current state.

Connect Four is particularly simple in comparison to the games on which AlphaZero is benchmarked. In Connect Four, there are only as many legal moves as columns, and the maximum of number of turns is limited to the amount of squares on the board. One measure of the relative complexities of Chess, Go, and Connect Four is with respect to the state space. Van den Herik [8] define state space complexity as the number of legal positions reachable from the initial position of the game and tabulate this metric for various games. The metrics relevant to this project are reproduced in the table below.

Game	State Space Complexity
Connect Four	10^{14}
Chess	10^{46}
Go	10^{172}

Unlike Chess and Go, Connect Four has been solved via brute force search [1] and even via knowledge based methods [2]. The goal here is to demonstrate adaptability of this variant of Deep Q learning to this problem. The tendency for Deep Q learning to be unstable is well known and attributed to correlations between the training samples and the fact that the target values are themselves functions of the parameters of the network

being optimized. These concerns are addressed here as they usually are. Data samples are buffered over a batch of games, essentially freezing our Q network (f_θ) while it directs our policy during game play, and are randomly sampled from this batch to produce a less correlated training data set.

3. Main Text

3.1 Methods

The data used to train this network is generated via self-play aided by the Monte Carlo tree search. Every action in every game is decided by constructing a tree of the action space rooted at the current state. Repeated over a set of n trials, the game is advanced by traversing the tree along the path of the highest value states as determined thus far. If a leaf is reached before the game has ended, a scheme to complete the game must be chosen. In classic Monte Carlo tree search, moves are chosen at random until the game has a resolution. The method used in the AlphaZero paper calls for evaluation of the neural network at this leaf which will return an expected outcome of the game right away. The implications of the choice of playout or evaluation will be discussed in later sections.

Once a score has been obtained, all of the states along the path have their total values updated. [1] During this update step, the visit count for each of the states along the path is incremented. From total action value and visit count, we can report mean values of these nodes/states, which will effect the paths taken at the start of successive trials.

After n simulations have completed, a single action from the root is finally selected proportional to its visit count, which directly corresponds to its estimated mean value.

Each step produces one training data point. The history of the game over the last T timesteps is the input image, and the target policy π for this state corresponds to the visit probabilities of the root node's children. Once the real game finishes, all of these steps also have an game outcome target, z.

The neural network used in this paper is very shallow in comparison to that used in AlphaZero, as the state and action space is so reduced. There are just three convolutional layers before the net splits off into its policy and value heads. Each head has an additional convolutional layer followed by a fully connected linear layer. The Adagrad optimizer was chosen with an initial learning rate on the order of 10^{-3}. The loss function used for parameter optimization is the sum over mean squared error and cross entropy losses:

$$L = (z - v)^2 - \pi^T \log p \tag{1}$$

With such a small network, the real bottleneck in training time was the self-play for generating the data. This implementation parallelized gameplay across five python processes, which approached the maximum capacity for memory on the author's computer.

The Supplementary [6] materials in the original paper were referenced heavily and further insight was gained from Junxiaosong [7].

1. Note that at each level of the tree, the current player switches. Thus, the score of +1 for a win will be propagated up to every other layer in the tree, while the score of -1 for a loss will be added to the outcomes for the layers of the tree in between, corresponding to the other losing player. A tie yields a 0 outcome.

3.2 Experiments

The models were evaluated against two Connect Four players. The first, referred to as the MCTS player, makes actions using the same MC simulation method detailed above. The second player, referred to as the random player, takes actions by sampling from a uniform probability over all of the columns on the board. In the first experiment, three versions of the framework were compared, one in which the Q network is used as intended for outcome evaluation during simulations, a second where the random playout method is used instead, and a third combining the two approaches, switching to network use halfway through training time. The results of this stage are contained in Figure 1 and Table 1. All experimental settings are reported in Table 4 in the Appendix.

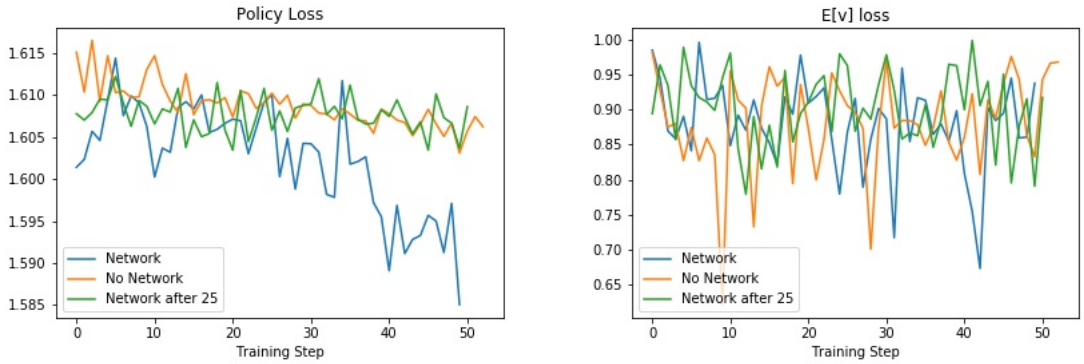


Figure 1: Loss over time comparison. What looks like superior convergence for the network design does not yield better validation metrics.

Table 1: Performance ratings are over an average of 500 rounds of play.

	Nework	No Network	Half Network
Win rate v. MCTS	0.846	0.856	0.838
Draw rate v. MCTS	0.046	0.042	0.034
Win rate v. Random	0.494	0.538	0.518
Draw rate v. Random	0.092	0.11	0.103

There are several takeaways from the data. First is that the convergence of the model set up as prescribed in AlphaZero looks to be faster than setups using random playout for state value estimations. However when it comes to performance against the MCTS and Random players, this approach does no better. In fact in certain instances [2] , the No Network model

2. During the presentation I reported that the MCTS network was increasing probability of a column as the number of plays in that column increased. This was observed at the time. For the settings of these experiements specifically, even the MCTS models output almost uniform probabilities. The version of the model produced in the final experiment does exhibit this reported strategy.

demonstrated a discernible strategy whereas the ps for the Network model were close to uniform at every step of the game.

An explanation for this trouble is possibly rooted in the loss function (1). The network is provided target distributions π which are the visit probabilities of the direct descendants of the root state after a full MC tree search. As noted previously, this double use of the Q network makes for a non-stationary target π, and gradient methods of this approach are not guaranteed to converge. It is worth noting here that in AlphaGoZero [4], the prior work of AlphaZero, the value estimation is derived by combining network evaluation and rollout outcomes.

However, a second related observation to make is these models lose more often to the Random player than to the MCTS player. This speaks to a fundamental flaw in the search algorithm.

Let us more closely examine how exactly actions are chosen. From [6], an action from state s at time t is selected as

$$a = \arg\max_a (Q(s_t, a) + U(s_t, a)) \tag{2}$$

Where Q is state value estimation from f_θ (3)

$$U = C(s)P(s,a)\frac{\sqrt{N(s)}}{1 + N(s,a)} \tag{4}$$

P(s,a) is the prior probability of taking action a in state s (5)

N(s) is parent visit count (6)

N(s,a) is node visit count (7)

The lead suspect is the value for C in the U term, influencing which action is chosen, and specifically controlling the rate of exploration. This is a reported parameter taken directly from the Alpha Zero paper and very likely needs to be tuned to Connect Four specifically. Searching for the optimal C(s) is a critical part of future work.

Finally, a look at the graph of expected game outcome loss shows little progress over time. One may wonder whether the relative simplicity of Connect Four hinders the strength of state as a signal for expected outcome. The board state before a win can be identical to the board state before a loss, and this may happen more often than it does not.

In the baseline experiments, $n = 400$ rounds of Monte Carlo trials are used at each action step. While this is half of what is used in AlphaZero, the question could be asked how many are necessary to capture the ideal action to take when there are only five choices on a 5x5 board. Performance metrics in Table 2 show that 25 seems sufficient.This support a rough scaling heuristic of n to the square of the number of possible moves, since on average [3] there are 35 possible moves in a state of chess, and $35^2 \sim 800$ is the cited number n used in AlphaZero. We confirm that there is a lower limit before performance degrades in the final column of Table 2.

So far, the number of timesteps of history provided in a training image has been $T = 1$. The next set of experiments investigated whether including more history would improve outcomes. $T = 5$ and $T = 10$ were chosen. The data in Table 3 suggests that all other paramaters held constant, no positive change in performance is observed. Therefore the decision to only use the state of the board at time t is retroactively justified.

Table 2: Effect of Number of trials per MC Simulation

	400	100	25	5
Win rate v. MCTS	0.838	0.834	0.86	0.646
Draw rate v. MCTS	0.034	0.038	0.034	0.07
Win rate v. Random	0.518	0.509	0.532	0.47
Draw rate v. Random	0.103	0.1046	0.106	0.1

Table 3: Effect of Number of Timesteps in State

	T=1	T=5	T=10
Win rate v. MCTS	0.834	0.775	0.762
Draw rate v. MCTS	0.038	0.056	0.068
Win rate v. Random	0.509	0.506	0.508
Draw rate v. Random	0.1046	0.114	0.102

With takeaways from each experiment determining the settings for the final model (see Table 4), the batch size and number of batches was increased substantially, and the network was trained on a game sized board. See the graph in Appendix A for the slow decline of policy loss and the stagnant value loss. The problems identified in Experiment 1 are still apparent here, and no appreciable difference in validation metrics is seen. However at this iteration, tangible action strategies can be observed in play. Please see the implementation site [3] for videos that attempt to demonstrate that this network learns to put higher probability on a column as the number of plays in the column increases. Unfortunately, it is apparent that the model cannot discern between the two players, nor is it clearly aiming to connect four of its kind in a row.

3.3 Conclusion

In this paper we identified two model parameters that do indeed seem to be functions of the game complexity and were significantly dialed down without harm to learning Connect Four. We reaffirm the need to specifically tune the exploration parameter C, and consider this a significant roadblock to better performance. We also suggest that instability of deep Q learning remains present in the proposed framework and an inescapable aspect of overcoming it is train time and compute power. Not enough of either has been applied yet to this adaptation to achieve good results.

References

[1] J. Allen. A note on the computer solution of connect-four. *D.N.L. Levy, D.F. Beal (Eds.), Heuristic Programming in Artificial Intelligence: The First Computer Olympiad*, pages 134–135, 1989.

3. https://github.com/Akmalleo3/ConnectFour_AlphaZero

[2] L.V. Allis. A knowledge-based approach of connect four. *M.Sc. Thesis, Vrije Universiteit Report No. IR-163, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam*, 1988.

[3] Victor Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, Maastricht, The Netherlands, 1994.

[4] Thomas Hubert David Silver. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[5] Thomas Hubert David Silver. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362:1140–1144, 2018.

[6] Thomas Hubert David Silver. Supplementary materials for a general reinforcement learning algorithm. *Science*, 362, 2018.

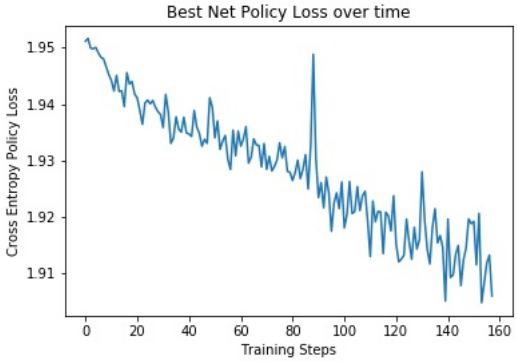
[7] junxiaosong. Alphazero_gomoku. https://github.com/junxiaosong/AlphaZero_Gomoku, 2019.

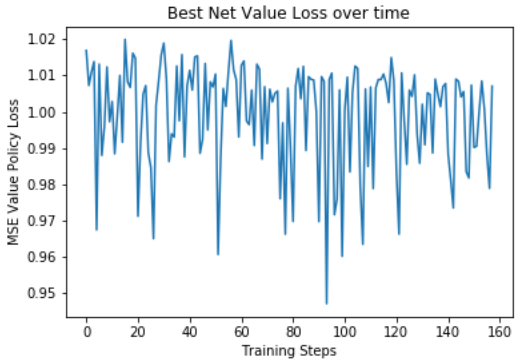
[8] H.Jaap van den Herik, Jos W.H.M. Uiterwijk, and Jack van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1):277–311, 2002.

Appendix A.

Table 4: Hyperparameters

	Games	Batch Size	T	M	Batches	Board Size	Network
Experiment 1							
1	40	200	1	400	50	5x5	Yes
2	40	200	1	400	50	5x5	No
3	40	200	1	400	50	5x5	Half
Experiment 2							
1	40	200	1	400	50	5x5	Half
2	40	200	1	100	50	5x5	Half
3	40	200	1	25	50	5x5	Half
4	40	200	1	5	50	5x5	Half
Experiment 3							
1	40	200	1	25	50	5x5	Half
2	40	200	5	25	50	5x5	Half
3	40	200	10	25	50	5x5	Half
Experiment 4/Best:							
	80	2048	1	50	150	6x7	Half


Best Net Policy Loss over time
Cross Entropy Policy Loss
1.95
1.94
1.93
1.92
1.91
0
20
40
60
80
100
120
140
160
Training Steps


Best Net Value Loss over time
MSE Value Policy Loss
1.02
1.01
1.00
0.99
0.98
0.97
0.96
0.95
0
20
40
60
80
100
120
140
160
Training Steps