Master's Thesis

Investigation of possible improvements to increase the efficiency of the AlphaZero algorithm.

Christian-Albrechts-Universität zu Kiel
Institut für Informatik

written by: **Colin Clausen**
supervising university lecturer: Prof. Dr.-Ing. Sven Tomforde

Kiel, 22.7.2020

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

.....
Colin Clausen

Contents

1 Introduction

Games have been used for a long time as a standin of the more complex real world in developing artifical intelligence. Beating humans at various games has often been viewed as a milestone.

(TODO a few sentences here about progress on various milestone games).

In March 2016 a program called AlphaGo for the first time in history has defeated a top human player in the board game Go [1]. Go had eluded attempts at super human level play for a very long time.

Louis Victor Allis attributes [4] this to the large game tree size of 10^{360} possible games, compared to 10^{120} in chess [12], but also to the way humans use their natural pattern recognition ability to quickly eliminate most of the often 200 or more possible moves and focus on few promising ones.

This combination of the usage of hard-to-program pattern recognition with an extremely large game tree has prevented computers from reaching top human strength through game tree search algorithms based on programmed heuristics. AlphaGo solved this issue by using the strong pattern recognition abilities of deep learning and combining them with a tree search, allowing the computer to learn patterns, similar to a human, but also search forward in the game tree to find the best move to play.

Further development of the AlphaGo algorithm yielded the AlphaZero algorithm, which significantly simplified AlphaGo, allowing learning to start with a random network and no requirements for human expert input of any kind.

In the following thesis I want to investigate further possible improvements to reduce the computational cost of using AlphaZero to learn to play games.

(TODO here one could state some key results, once they exist...)

This thesis is structured as follows:

First I will look at previous work, starting with the basis of AlphaZero, Monte Carlo Tree Search, moving onto the various versions of AlphaZero with previously suggested improvements. Then a list of novel improvements will be described. Finally an extensive set of experiments will be presented, first establishing a baseline performance, then showing the results on the novel improvements.

2 Previous work

In this section previous work relevant to AlphaZero will be presented.

2.1 Monte Carlo Tree Search

Monte Carlo Tree Search, in short MCTS, is the idea to search a large tree (e.g. a game tree) in a randomized fashion, gathering statistics on how good the expected return for moves at the root of the tree is.

An early suggestion of this idea was made by Bernd Brügmann in 1993 [7], who suggested a tree search in a random fashion inspired by simulated annealing. He used the algorithm to play computer go and found promising results on 9x9 boards.

AlphaZero uses a variant of MCTS called UCT, which was formulated in 2006 [10] by L.Kocsis et al. and used in computer go for the first time in the same year [8]. UCT stands for "UCB applied to trees", where UCB is the "Upper Confidence Bounds" algorithm of the *multi-armed bandit problem* [6]. Previous to AlphaZero multiple other strong computer go programs based on MCTS have been released, such as Pachi [3] or CrazyStone [2].

In the *multi-armed bandit problem* a player is faced with a machine that has a set of levers, some of which return a high reward, while others return a low reward. The player is tasked with getting a high return from a fixed number of lever pulls. This creates a dilemma, where the player has to decide to explore, i.e. try a new lever to maybe find a higher return, or exploit, i.e. pull the lever with the highest known reward. This dilemma is a key problem of reinforcement learning and L.Kocsis et al. [10] apply it to tree searches, effectively viewing the decision of which move to play in a node of the game tree as a multi-armed bandit problem.

UCT as described by L.Kocsis et al. is a rollout-based planning algorithm, which repeatedly samples possible episodes from the root of the tree. An episode is a possible sequence of moves that can be played form the root of the tree up to the end of the game. The result of the episode is backpropagated upwards in the tree. UCT is thus an incremental process, which improves the quality of the approximation of the move values at the root with every episode sampled and can be stopped at any time to return a result.

For every action a, state s, tree depth d and time t an implementation of UCT needs to track the estimated value of a in s at depth d and time t $Q_t(s, a, d)$, the number of visits of s up to d and t $N_{s,d}(t)$ and the number of times a was selected in s at depth d and time t $N_{s,a,d}(t)$.

A bias term, shown in equation 1, is defined where C_p is a constant.

$$C_{t,s} = 2C_p\sqrt{\frac{\ln t}{s}} \tag{1}$$

$$argmax_a(Q_t(s, a, d) + C_{N_{s,d}(t),N_{s,a,d}(t)}) \tag{2}$$

MCTS with UCT selects actions at every node of the tree according to equation 2, updating the visit counts and estimated values of nodes as episodes are completed.

Equation 2 can be understood as weighting exploitation, in the form of the Q term, against exploration, in the form of the C term. The specific form of $C_{t,s}$ is shown to be consistent and to have finite sample bounds on the estimation error by L.Kocsis et al.

2.2 AlphaZero

TODO: Should I describe AlphaGo here in more detail? I don't really care about all the complications it did compared to AlphaZero, but it might still be interesting?

The AlphaZero algorithm [14] is the application of AlphaGoZero [15] to games other than Go. AlphaGoZero is a significant simplification of AlphaGo [13]. AlphaGo is the first program to reach superhuman performance in the Go by combining MCTS with deep learning.

AlphaGo used a complicated system involving initialization with example games, random roleouts during tree searches and used multiple networks, one to predict the value of a position, another to predict promising moves. AlphaGoZero drastically simplified the system by only using a single network and not doing any roleouts anymore, instead the network evaluation for the given positions is directly used.

The difference between AlphaGoZero and AlphaZero is mainly that AlphaGoZero involved comparing the currently trained network against the previously known best player by letting them play a set of evaluation games against each other. Only the best player was used to generate new games. AlphaZero skips this and just always uses the current network to produce new games, surprisingly this appears to not give any disadvantage, learning remains stable.

The main advantage of the "Zero" versions is that they do not require any human knowledge about the game apart from the rules, the networks are trained from scratch by self-play alone, so unlike AlphaGo, AlphaGoZero does not require supervised initialization of the network with a dataset of top level human play. This allows the algorithm to find the best way to play without human bias which seems to slightly increase final playing strength. Additionally it allows to use the algorithm for research of games for which no human experts exist, such as No-Castling Chess [11].

2.2.1 The AlphaZero algorithm

The AlphaZero algorithm [14] uses MCTS with the UCT formulation, as described in section 2.1, and modifies it to incorporate a deep neural network which serves as a heuristic to bias the tree search and provide evaluations of unfinished games. The

network is then trained to predict the final result of the MCTS search directly, as well as the final result of games played by MCTS against itself. This allows to form a closed cycle of self improvement, which starts with a random network and has been shown to successfully learn to play various games, such as chess, shogi or go on the highest level, if given substantial computation resources.

Work written concurrently to AlphaGoZero describes a similar algorithm tested with the game hex [5]. They describe the algorithm as thinking slow, via MCTS, and fast, via the network alone, in reference to results from human behavioral science [9]. AlphaZero in that sense learns in a similar fashion as humans: At first a time intensive "thought process" is used to reason about a new problem, but with pratice good decisions can be made much faster.

The network in AlphaZero is provided with an encoding of a game situation and has two outputs: a policy describing move likelihoods and the expected value of the situation for the players, i.e. it predicts what move should be played and who is likely to win in the given situation.

MCTS with UCT is modified to include the network outputs, which biases the search towards moves that the network believes to be promising. To this end for every node of the search tree some statistics are stored, shown in table 1.

$N(s,a)$	The number of visists of action a in state s.
$W(s,a)$	The total action value.
$Q(s,a)$	The average action value, equal to $\frac{N(s,a)}{W(s,a)}$.
$P(s,a)$	The network policy output to play action a in state s.

Table 1: Statistics tracked per node in the AlphaZero MCTS

As described before in chapter 2.1, UCT plays through episodes in an iterative fashion, starting at the root of the tree and playing moves until it finds a terminal game state.

Unlike plain UCT, AlphaZero does not play out entire episodes till terminal game states, but rather calls the network whenever a move is mode that reaches a node in the search tree which has not been reached before.

The analysis of the network is then backpropagated upwards the search tree, updating the node statistics in the process. The tree search then moves down the tree again, using the updated statistics, until it reaches again an unknown node to be evaluated by the network.

The tree grows until some fixed number of nodes is created. The output is the distribution of visits to the actions of the root node, as a high visit count implies the tree search considers a move to be worthwhile and has analyzed it thoroughly.

To use the network outputs in UCT, the formulations are changed. The action a in a given node is selected accoring to equation 3, where $U(s,a)$ is a term that is inspired

by UCT but is biased by the network policy, as seen in equation 4. C_{puct} is a constant to be chosen, the same as C_p in UCT.

$$argmax_a(Q(s,a) + U(s,a)) \tag{3}$$

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

Using this tree search games are played out, the resulting game states are stored with the MCTS policy and the final game result. The network is then trained to predict the MCTS policy for the states and the final result of the game.

The self play learning hinges on the assumption that the policy generated by MCTS with the network is always better than the one by the network alone. In pratice this appears to hold, as learning is quite stable even very complex games such as go. Intuitively this makes sense, as the MCTS effectively averages many network predictions into one, forming a sort of ensemble of network evaluations on possible future positions.

A drawback of this approach is the high computational cost, as in practice for good results the search tree to play a single move has to be grown to at least hundreds of nodes, requiring the same number of forward passes through the network to play a single move. For more complex games it takes millions of games to be played until the highest level of play is reached with a network having millions of parameters. This motivates my work in researching possible improvements to the algorithm that allow learning to progress faster or with less example games played.

2.3 Extensions to AlphaZero

Many improvements to the AlphaZero algorithm have been proposed, typically aiming at reducing the extreme computational cost of learning to play a new game.

3 Evaluated novel improvements

3.1 Network modifications

3.2 Playing games as trees

3.3 Automatic auxilary features

4 Experiments

4.1 Baselines

4.1.1 AlphaZero implementation

4.1.2 Extended AlphaZero

4.2 Results on novel improvements

4.2.1 Network modifications

4.2.2 Playing games as trees

4.2.3 Automatic auxilary features

4.2.4 Evolutionary hyperparameters

References

[1] http://www.straitstimes.com/asia/east-asia/googles-alphago-gets-divine-go-ranking. Accessed: 2020-05-17.

[2] Crazystone go engine. https://senseis.xmp.net/?CrazyStone.

[3] Pachi go engine. https://github.com/pasky/pachi.

[4] Louis Victor Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen, 1994.

[5] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pages 5360–5370, 2017.

[6] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[7] Bernd Brügmann. Monte carlo go. 1993.

[8] Sylvain Gelly, Yizao Wang, Olivier Teytaud, Modification Uct Patterns, and Projet Tao. Modification of uct with patterns in monte-carlo go. 2006.

[9] Daniel Kahneman. Thinking, fast and slow. new york. 2011.

[10] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[11] Vladimir Kramnik. Kramnik and alphazero: How to rethink chess. https://www.chess.com/article/view/no-castling-chess-kramnik-alphazero. Accessed: 2019-11-29.

[12] Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[13] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[14] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.