HEX playing AI

Contents

[Introduction 1](#_Toc122272985)

[HEX Game 1](#_Toc122272986)

[Related Work 1](#_Toc122272987)

[AlphaZero 1](#_Toc122272988)

[SLURM 2](#_Toc122272989)

[Analyzing the Code 2](#_Toc122272990)

[Bibliography 2](#_Toc122272991)

# Introduction

The goal of this project is to implement and train an AI to play the game HEX [1] using the AlphaZero reinforcement learning algorithm (RL) [2], [3] [4] created by DeepMind [5]. The institutes SLURM [6] cluster was used to offload training data generation to up to 20 machines in parallel. Since there were no restrictions on using existing code, we decided to go for an existing implementation of the AlphaZero algorithm and focused on optimizing this code to run on the cluster.

## HEX Game

Explain the game….

## Related Work

There are several projects on GitHub and other platforms where the AlphaZero algorithm has been implemented for various games. The most forked repository is alpha-zero-general [7] which implements the core algorithm of AlphaZero according to the original paper and contains example applications for games like Othello [8], TicTacToe [9], and others. One of the forks of this repository is called alpha-zero-hex [10] which implements the HEX game using the AlphaZero implementation of alpha-zero-general. This implementation doesn’t use any parallelization techniques and therefore is slow to train.

## AlphaZero

AlphaZero is a reinforcement learning algorithm developed by DeepMind that has gained lots of attention because its application in games like GO or Chess has led to superhuman level of play and convincingly defeated world-champion programs in each case [3]. It is astoundingly simple and yet very powerful as it combines techniques as the Monte-Carlo-Tree-Search (MCTS) [11] with a deep-learning approach [12] to overcome shortcomings in both approaches.

The MCTS is a heuristic search algorithm that simulates several game steps given an initial state and returns the best action based on the information it gathered during the simulations. In principle the search algorithm uses Upper-Confidence-Bound based exploration which focuses on expanding nodes that have been marked as leading to a win in previous simulations or randomly choose new nodes to expand.

The integration of deep learning into MCTS happens at the sampling level. Where the classical MCTS chooses randomly to expand on new nodes AlphaZero uses the current state of the incrementally trained neural network to predict probabilities for the next actions to choose.

While a pure MCTS algorithm is limited by the available memory to store simulation data and a pure neural network has only limited knowledge about a games development the combination of both make it decrease their shortcomings. In each training iteration a new instance of MCTS is created starting without any prior knowledge and new nodes are expanded using a trained network which can make more informed decisions than a random number generator. After several simulations these results are used to further train the network which in the next iteration should improve the predictive behaviour of the network and hence the quality of the MCTS search.

## SLURM

SLURM is an open source, fault-tolerant cluster management and job scheduling system. The FHTW has such a cluster active which students can use to run computationally expensive tasks distributed on several machines. It consists of 19 machines which each have a NVIDIA RTX 2080 TI installed of which two were down at the time of training.

The cluster can be controlled with view simple commands which are listed below, and their usage will be explained later.

* sinfo: shows the state of the cluster nodes.
* squeue: shows the current job queue.
* sacct: lists completed and running jobs.
* sview: graphical monitoring tool for jobs and nodes.
* salloc: allocates some resources on the cluster without starting a task.
* srun: runs a single job interactively.
* sbatch: runs a job in the background. Also runs job arrays.
* scancel: cancels a running job.

# Analysing the Code

As mentioned above this project uses an existing implementation of the alpha-zero-hex GitHub repository and focuses on adapting this code to be able to run it effectively on the SLURM cluster.

# Bibliography

[1] ‘*Hex* (board game)’, *Wikipedia*. Dec. 13, 2022. Accessed: Dec. 18, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hex\_(board\_game)&oldid=1127241285

[2] D. Silver *et al.*, ‘Mastering the game of Go without human knowledge’, *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.

[3] D. Silver *et al.*, ‘Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm’. arXiv, Dec. 05, 2017. Accessed: Dec. 18, 2022. [Online]. Available: http://arxiv.org/abs/1712.01815

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

[5] ‘DeepMind’. https://www.deepmind.com/about (accessed Dec. 18, 2022).

[6] ‘Slurm Workload Manager - Documentation’. https://slurm.schedmd.com/documentation.html (accessed Dec. 18, 2022).

[7] ‘suragnair/alpha-zero-general: A clean implementation based on AlphaZero for any game in any framework + tutorial + Othello/Gobang/TicTacToe/Connect4 and more’. https://github.com/suragnair/alpha-zero-general (accessed Dec. 18, 2022).

[8] ‘Reversi’, *Wikipedia*. Dec. 16, 2022. Accessed: Dec. 18, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reversi&oldid=1127715743#Othello

[9] ‘Tic-tac-toe’, *Wikipedia*. Dec. 14, 2022. Accessed: Dec. 18, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Tic-tac-toe&oldid=1127337707

[10] A. Agha, ‘alpha-zero-hex’. May 27, 2019. Accessed: Dec. 18, 2022. [Online]. Available: https://github.com/aebrahimian/alpha-zero-hex

[11] C. B. Browne *et al.*, ‘A Survey of Monte Carlo Tree Search Methods’, *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012, doi: 10.1109/TCIAIG.2012.2186810.

[12] ‘DeepLearning’. https://www.deeplearningbook.org/contents/intro.html (accessed Dec. 18, 2022).