Reinforcement Learning 2020
Assignment 4: Self-Play

Amin Moradi
me@mamin.io

Martijn Swenne
martijnswenne@hotmail.com

Bartosz Piaskowski
b.piaskowski@umail.leidenuniv.nl

May 22, 2020

1 Introduction

The pursuit for understanding artificial intelligence and being able to successfully design a machine that achieves human-level cognitive ability has never been so close. A milestone in this chase has been reached with the introduction of Deep Mind's AlphaGo program, which changed the course of history. It was then, in march 2016, when the machine was able to beat the world's best Go player at that time, Lee Sedol. This was a breakthrough event from the scientific perspective as games are known as one of the means of putting artificial intelligence to the test and confront it with a human in a fair and square battle. The next version of AlphaGo, namely AlphaGo Zero, released in the fall of 2017, took a completely different approach at learning how to play the game of Go. It was an example of *tabula rasa*, a program that was able to learn by playing against itself, which is called self-play. The performance of this software was astounding, greatly surpassing human-players and beating it's previous version in just three days.

2 Self Play

In case of AlphaZero, the core of it's self-playing reinforcement learning algorithm consists of two methods: **Monte Carlo Tree Search(MCTS)** and **Deep Neural Network(DNN)**. The former is used for game simulations and move assessment. It also contains an *exploration/exploitation* adjustment hyperparameter, which is a crucial part of every learning algorithm. The latter is responsible for estimating and improving the *policy statement* and calculating *action/move values*. Even though both these core components of the algorithm are independent from each other, they cooperate and exchange information which are necessary for training models and simulating games. The models trained in the process are compared and played against one another multiple times. If the new model outperforms

its successor and meets a certain threshold, it becomes the new base network, then used for MCTS simulations, policy improvement and value calculation. The name *"self-play"* perfectly reflects the general idea behind this policy iteration algorithm.

3 AlphaZero on Hex

In order to get familiar with the concept of self-play, we decided to explore an already existing implementation of it, which is based on the AlphaGo Zero paper. As there are multiple GitHub repositories, whose contributors attempted to replicate the functionality and core idea of a self-play based reinforcement learning program, we had the option to choose one for the scientific purpose of working on this assignment. The implementation of our choice is the one created by three students from Stanford University, Shantanu Thakoor et al.[1], who described it in their paper "Learning to Play Othello Without Human Knowledge". The authors focused on the game of Othello in their work, but the self-play algorithm they made available as open-source, after sufficient adaptation, can also be used for other games, such as in our case, the game of Hex.

3.1 Initial experiment

One of the instructions included in the assignment suggested getting familiar with the program by playing a couple of games of Othello against the pre-trained model and also taking a closer look at the *Coach.py* file, which is the core of the training process of the whole algorithm.

After this we had to implement our own version of Hex that worked using the structure of the github program. Most of the functions we needed for the structure was easily retrieved from the `hex_skeleton.py` and the Othello game.

There was one thing we really struggled with, and that was the fact that during training, it would show the available moves correct, but during human play, the available moves we switched in axis. If we fixed this in human play, the training would have the problem. We eventually fixed this by removing the `getCanonical` function-calls at several spots in our code.

3.2 AlphaZero Hex Training

After adopting AlphaZero General source code to our implementation of Hex, we used our GPU server to train our network. We selected the default hyper-parameters and trained

our network for 100 iteration that took us more than 16 hours of training. Figure1 shows the result of the first agent that we trained. Furthermore we trained two agent with same configuration. We will discuss that in the next section.

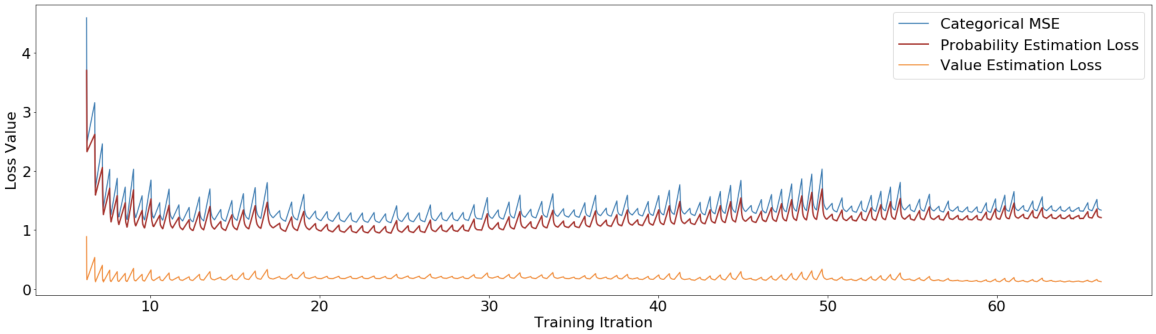

Figure 1: Loss values of the training process with default values of 25 MCTS simulation and 20 History size.

4 Tournaments

After training two A0G models, we had them and our agents from previous assignments play a tournament. We have 4 players, an Alpha-Beta agent with Iterative Deepening and Transition Tables, an MCTS agent and two A0G agents. Each round of the tournament, each player plays two games against each other player, one where they start, and one where the enemy starts. For our agents, this means 12 games are played each round, 6 where player one starts, and 6 where player two starts. Each game updates the TrueSkill rating of the corresponding two players. We have played 100 rounds, where each round the mean(=μ) of the ratings was stored. These ratings are shown in Figure 2.

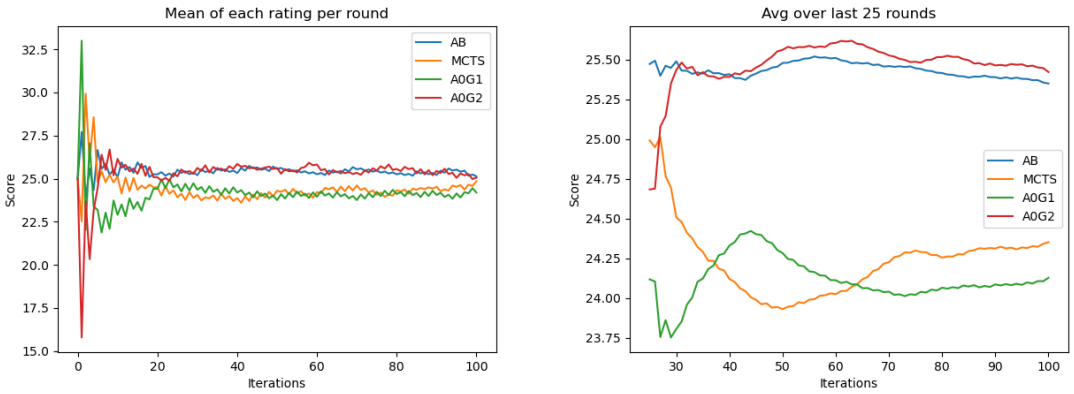

Figure 2: The results of the Tournaments over 100 rounds.

We knew from the previous assignments that our version of Alpha-Beta was very good,

and our version of MCTS was not that good. As seen in the results of the tournament, it shows that MCTS is one of the worse players, and Alpha-Beta is one of the better players. For A0G1 and A0G2 there is much to say. First of all, it is weird that both models show such a big difference in performance, compared to each other, but also to the other agents. This shows that the model does not train the same every time. We can see that A0G1 comes out the best, even though Alpha Beta comes pretty close, which does show that A0G still is the best performing algorithm out of all our agents.

5 Hyperparameters

One of the most challenging parts that we faced in this assignment was finding the best hyper-parameters. Training agents like AlphaZero are computationally expensive and in most cases we are dealing with an blackbox optimisation problem. Using methods like grid search for hyper-parameters or other traditional methods are very time consuming and in almost impossible. Companies like Google and Facebook usually take the advantage of their computational resources and find hyper-parameters sweet-spots. We can then perform local and exploitative search around that configuration.

Although hyper-parameter tuning is such an abstract topic in AlphaZero, an effort of a deep analysis and guidelines for optimal tuning has been introduced by Wang et al. [2] in their research paper. The authors described the role of all tunable settings in the algorithm and compared three different variants in respect to training losses, time efficiency and strength of the models/players. The information included in that publication allowed us to save some time on the computationally intensive experiments by starting off with what we believed, optimal initial parameter setting.

There are 5 parameters to explore for AlphaZero implementation that we used. As we were short in resources, we only tried two different combinations that we thought have the most effect on the training of the network. We used 2 different combinations of 'numMCTSSims' and the 'numItersForTrainExamplesHistory' for values 25 VS 50 and 20 VS 32. Figure3 shows the results.

In order to tackle the issue of efficient hyperparameter setting, more scientists are leaning towards the use of **meta-gradient** in reinforcement learning. The idea assumes using them as a way of approximating the value function by interacting and learning about the environment. Such approach has been tested on the Atari 2600 environment and the results proved to have a positive effect on the state-of-the-art algorithm, increasing its learn-

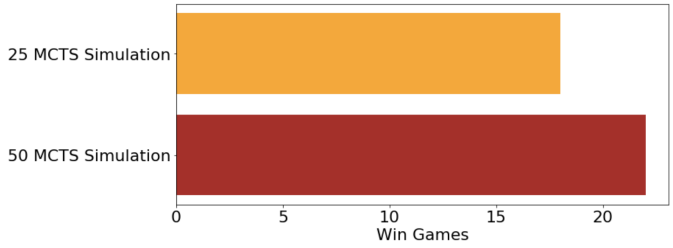

Figure 3: Comparison between two Hyper-parameter settings.

ing capabilities by a significant margin as concluded by Xu et al. [3]. Another instance of using meta-gradient in reinforcement learning has been described by Zahavy et al. [4]. The method they propose is called STAC (Self-Tuning Actor Critic) and is an extension to DeepMind's IMPALA (Importance Weighted Actor-Learner Architecture) algorithm, used for multi-task learning. The focus is put on the ability to tune multiple hyper-parameters simultaneously using meta-learning and the results showed that it's application is not only beneficial for the model's performance but also for the computational efficiency.

6 The Essence of Self-play

In this section we would like to take a deep dive into the idea and essence of self-play and why it plays a crucial role in the success of intelligent agents. As mentioned in the lecture, self-play was introduced first in TD-Gammon[5] game in 1992 which is not very recent compared to other RL and DL approaches. Self-play enables an agent to interact with its environment in the context of competition with itself. This means, in a balanced self-play environment there are two agents equally good that challenge each other for survival. We have seen similar behaviours in nature as well; The size of the human brain increased dramatically as it created societies and became a social being. As human interacted and got challenged by other humans with the same capabilities (as good as themselves) in the same environment, the evolution peaked. This is the same situation if RL self-play approach.

One of the observations that we had in DQN and AlphaZero was the difficulty for an agent to learn what exactly the task we are training it on is. In DQN we trained a model to play the game of Breakout where our results showed that in the first 750 episodes the agent was still struggling with basic movements. This phenomenon can be improved with self-play. One of the major advantages of self-play is that once we add compute into interaction with the environment, we turn random moves to trainable data and that is where Neural

Networks shine. Self-play will play a crucial role in pursuing general intelligence.

7 What future developments can we expect?

When it comes to future developments in the field of self-play based reinforcement learning algorithms, the room for upgrade is very wide. A recently published paper presents an extension to the original AlphaZero, called MuZero, introduced by Schrittwieser et al. [6]. The aim of the creators of this program was to take a more generalized approach that would make it more applicable for games with complex environments and high variety of dynamic present. The concept is based on reinforcing the learning algorithms so it has the ability to do planning, similarly to what humans do, which in this case was predicting the reward, the policy and the value function. Muzero is a model-based algorithm, which in contrast to the AlphaZero, uses a function that generates a hidden layer, serving as an abstract equivalent of the environment. This way the model can learn without any knowledge of the rules and heuristics. The results showed that the algorithm was able to outperform its successor and also achieve super-human level performance in the majority of explored 57 Atari games. Its ability to operate in a completely unknown environment without human interactions might herald the creation of a generalized models that would be able to deal with real world domains.

Another interesting take on the future of reinforcement learning algorithms are ones that incorporates the idea of using evolution strategies. A population-based optimization algorithm has been introduced by Wang et al. [7] in their research paper and it is called "Paired Open-Ended Trailblazer" (POET). There is significant relevance between this kind of algorithm and self-play concept included in AlphaZero. The similarity is that both programs learns how to understand and solve an environment using self-produced data, consecutively upgrading its inner loop algorithm and progressing. The difference lies in the used techniques, namely MCTS vs Evolutionary Algorithm as the policy improvement method. We believe that the aforementioned papers are part of the gradual process of paving the way towards the future of reinforcement learning.

8 Discussion

We have examined numerous approaches in solving games with RL and most of them originate from the same family of algorithms. In the previous section, we talked about the

importance and essence of self-play and how it makes agents create automated-learning environments. But training and researching in this field require a large amount of computation power. Of course, this is due to agent starting from a fully random state without any prior knowledge of its environment. We think it is possible to smoothen the learning process by some priors of the environment of the agents long term strategy. Although we observed improvement by making our agents blind to the environment in MuZero[6], we think with optimising exploration strategy, agents can improve in training as well as their results.

9 Conclusions

In this assignment, we got hand-on with Deep Mind's AlphaZero, one of the biggest leaps in ReInforcement learning agents. Training and testing AlphaZero thought us a lot about the importance of self-play and curriculum learning. Throughout our course, we started with the simples heuristic algorithms and got insights into how agents can explore an environments search space and select the best options. Further down the road, we learned about the effect of randomness in exploring and exploiting the game tree and how the intuitions behind Multi-Armed bandit problem can help us in Monte-Carlo Tree Search. Training zero-prior agents was always a challenge and we have experienced the effect of a neural network on such tasks. Deep Q-Networks opens a new horizon on how an agent can learn from high dimensional data like images. We have come a long way in the field of reinforcement learning to explore SOTA algorithms like DeepMind's MuZero. Without a doubt, RL will have a major influence in the recent future as autonomous driving is being regulated and the AI community is devoting more time to this topic.

10 How to run the code

We extended the AlphaZero-General implementation with our own code. The HexGame code can be found in the folder `HexGame`. If you run `python main_assignment.py`, you will be prompted with a textual interface. Here you can choose between playing versus the trained A0G1 player or running a tournament between the four agents described in Section 4. The first option is always two games, one where you start and one where A0G1 starts. For the second option you are able to specify the number of games each bot plays against each other bot, where each round the beginning player is swapped.

References

[1] S. Nair, "alpha-zero-general." `https://github.com/suragnair/alpha-zero-general`, May 2020.

[2] H. Wang, M. Emmerich, M. Preuss, and A. Plaat, "Hyper-parameter sweep on alphazero general," 03 2019.

[3] Z. Xu, H. van Hasselt, and D. Silver, "Meta-gradient reinforcement learning," *CoRR*, vol. abs/1805.09801, 2018.

[4] T. Zahavy, Z. Xu, V. Veeriah, M. Hessel, J. Oh, H. van Hasselt, D. Silver, and S. Singh, "Self-tuning deep reinforcement learning," *ArXiv*, vol. abs/2002.12928, 2020.

[5] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[6] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *ArXiv*, vol. abs/1911.08265, 2019.

[7] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions," *ArXiv*, vol. abs/1901.01753, 2019.