OLIVAW: Mastering Othello with neither Humans nor a Penny

Antonio Norelli, Alessandro Panconesi
Department of Computer Science
Sapienza University of Rome, Italy
norelli@di.uniroma1.it, ale@di.uniroma1.it

Abstract

We introduce OLIVAW, an AI Othello player adopting the design principles of the famous AlphaGo series. The main motivation behind OLIVAW was to attain exceptional competence in a non-trivial board game, but at a tiny fraction of the cost of its illustrious predecessors. In this paper we show how OLIVAW successfully met this challenge.

1 Introduction

Only a year after AlphaGo's landmark victory against Go master Lee Sedol another sensational development took place. An improved version of AlphaGo called AlphaGo Zero asserted itself as the strongest Go player in the history of the game (Silver et al. 2017). The remarkable feature of AlphaGo Zero was that, unlike its predecessor and unlike all previous game software, it learned to master the game entirely by itself, without any human knowledge. The paradigm successfully used by the AlphaGo family was an interesting blend of deep and reinforcement learning. As subsequent work quickly showed, this paradigm seems to be general and flexible enough to adapt to a wide array of games (Silver et al. 2018), (Schrittwieser et al. 2019).

These extraordinary successes came at a price however, and quite literally so. The amount of computational and financial resources that were required was so huge as to be out of reach for most, if not all, academic institutions. Not coincidentally these well-endowed projects and their follow-ups took place within giant multinational corporations of the IT sector (Tian et al. 2019; Lee et al. 2019). These companies deployed GPU's by the thousands and hundreds of TPU's. A recent study looked at the number of petaflops per day that were required to train AlphaGo Zero and other recent well-known results in AI (Amodei and Hernandez 2018). The paper shows an exponential growth with a 3.4-month doubling period. This is clearly unsustainable for most academic labs and departments and even the greatest majority of companies. Another aspect of the same problem is the amount of training needed. AlphaGo Zero required 4.9 million games played during self-play. While to attain the level of grand master for games like Starcraft II and Dota 2 the training required 200 years and more than 10,000 years of gameplay, respectively (Vinyals et al. 2019) (Pachocki et al. 2018).

Thus one of the major problems to emerge in the wake of these breakthroughs is whether comparable results can be attained at a much lower cost and with just commodity hardware. In this paper we take a small step in this direction, by showing that AlphaGo Zero's successful paradigm can be replicated for the game of Othello (also called Reversi). While being much simpler than either Chess or Go, this game is still rather sophisticated and has a considerable strategic depth. The game boasts a long history and a rich tradition. Each year an exciting world championship takes place in which accomplished players from all over the world vie for the world title.

Our Othello engine is called OLIVAW, a homage to the famous robot character invented by Isaac Asimov. We tested the strength of OLIVAW in three different ways. In one instance, we pitted OLIVAW against Edax, one of the strongest Othello engines. Perhaps the most interesting aspect of this set of matches was that OLIVAW managed to beat several times an opponent that explores tens of millions of positions in the game tree in the course of a single game. In contrast, OLIVAW's search of the game tree was limited to a couple of thousand positions.

We also tested OLIVAW against (presumably) human players of varying strength on the web platform OthelloQuest. But the most exciting challenge was a series of matches against top notch human players: a national champion and a former world champion.

The final outcome shows that in relatively short training time OLIVAW reached the level of the best human players in the world. Crucially, this has been achieved by using very limited resources at very low overall cost: commodity hardware and free (and thus very limited) cloud services.

2 Related work

The success of AlphaGo naturally stimulated several follow ups. One of the main questions was to determine the level of generality of the approach. A series of papers showed this level to be great indeed. One after the other a list of difficult games fell pray of the RL-with-oracle-advice approach. Silver et al. (2018) extended it to Chess and Shogi.

Recently Schrittwieser et al. (2019) added ATARI games to the list. Our work continues this line of research by adding

8×8 Othello to the list, paying special attention to the cost issue.

It is not clear a priori whether the approach scales down in terms of resources. Although cheaper in several ways, the agents in (Silver et al. 2018), (Schrittwieser et al. 2019), (Tian et al. 2019), and (Lee et al. 2019) still use thousands of GPU's or hundreds of TPU's to master board games. The recent KataGo (Wu 2019) reaches the level of ELF using 1/50 of the computation and implements several techniques to accelerate the learning. However these includes a set of targets crafted by humans which are very game-specific [1] thereby reducing the generality of the approach and reintroducing human knowledge in a crucial way.

Succesful low-cost reproductions of AlphaGo Zero came out in recent years, but only for very simple games like Connect-4 (Young, Prasad, and Abrams 2018) or 6×6 Othello (Chang et al. 2018), for which perfect strategies are known to exist.

Other works are dedicated to an analysis of the hyperparameters involved in AlphaGo Zero, with the objective of obtaining a faster and cheaper training. Wang et al. (2019) and Young, Prasad, and Abrams (2018) make several experiments in this direction, while Wu et al. (2020) investigates the possibility of tuning hyperparameters within a single run, using a population based approach on Go. We followed some insights provided by these works as reported in section 5.2.

Concerning the state-of-the-art of 8×8 Othello engines, algorithms became superhuman before the deep learning era, a fact heralded by the defeat of the world champion Takeshi Murakami at the hands of Logistello (Buro 1995). Today the strongest and most popular programs used by top Othello players for training and game analysis are Saio (Romano 2009) and the open source Zebra (Andersson 1997) and Edax (Delorme 1998).

As in chess before AlphaZero (Kasparov 2018), in order to reach the level of world-class players, Othello engines rely on a highly optimized minimax search (Neumann 1928) and employ handcrafted evaluation functions based on knowledge of the game accumulated by human beings[2] (Andersson 1997). Another key feature used by these more traditional engines are huge catalogues of opening sequences, distilled and stored in the course of decades by human players and, more recently, by software as well. Finally, typically these engines play the perfect game by sheer computational brute force by expanding the entire game tree as soon as 20-30 moves before the end. The conclusion is that, unlike Olivaw, traditional Othello engines make crucial use of knowledge of the game accumulated by humans and of brute force tree exploration.

A recent paper by Liskowski, Jaśkowski, and Krawiec (2018) presents an engine obtained by training convolutional neural network (CNN) with a database of expert moves. The engine was are able to defeat Edax 2-ply (tree search limited to two moves ahead), a level of Edax which is however much weaker than top Othello human players.

[1] For instance the use of ladder, a tactic that is specific of Go.

[2] For instance, patterns on the edge or the corner of the board, which are known to be of great importance in Othello.

3 Othello

Othello is a popular board game. Its very simple rules are explained in Figure 1. A typical game lasts for some 60 moves, with an average branching factor of 10. Like Go and Chess it is a perfect information game. Although simpler than these two, it has considerable strategic depth. Unlike English draughts, there is no known perfect strategy that can be played by computer (Mullins 2007).

Othello is played across the globe. There are professional players competing in official tournaments organized by world and national federations. The Othello world championship takes place every year.

The best softwares beat humans systematically but they rely on brute force for most of the game. For the initial stages of the game they access a huge databases of openings, which are known to be very important in Othello. An opening lasts between 10 and 20 moves. Some 20-30 moves before the end they play the perfect game by exploring the game tree in its entirety. In the middlegame these programs explore hundreds of millions of positions of the game tree per move, a clear indication of brute force at play. In contrast, as we shall see, OLIVAW explores only a few hundreds positions and does not use any database of openings.

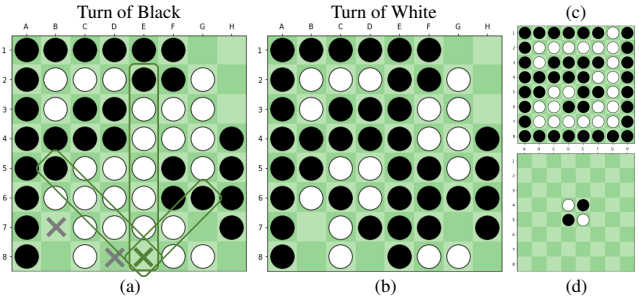


Figure 1: **Rules of Othello.** Othello is a turn-based game where the black and white player try to overcome the opponent in the final domination of an 8x8 board. **a.** Players move alternately by placing a new disk in an empty square in order to imprison one or more opponent's disks between the played disk and those of its own colour already on the board. It is possible to capture disks horizontally, vertically, and diagonally. Disks can be captured in one or more directions in a single move, capture always occurs in a straight line. Only moves capturing at least one disk are allowed, in the absence of moves the player skips the turn. It is not possible to pass the turn if there is at least one valid move. **b.** The imprisoned disks change colour and become owned by the player who moved. **c.** When none of the players can move, for instance when the board is full, the player with more disks on the board wins. Here black wins 40-24. **d.** A game of Othello begins with 4 disks placed in the center of the board in the shape of a X. Black moves first.

4 OLIVAW: the algorithm

The design of OLIVAW, our Othello engine, follows closely that of AlphaGo Zero (Silver et al. 2017). The main differ-

ence consists in a somewhat different and cheaper training process. The network architecture, while mimicking that of AlphaGo Zero, was scaled down.

Before discussing OLIVAW in detail, it is useful to describe its basic design.

4.1 The basic design

Like the AlphaGo series, OLIVAW uses reinforcement learning to build an oracle, in the form of a deep network f_θ (θ denotes the weights of the neural network). Given in input an Othello game state s, f_θ outputs a pair: $f_\theta(s) = (\boldsymbol{p}, v)$. The vector is a probability distribution over the moves that are possible from s. Intuitively, the higher the probability the better the move. The value v is the oracle's assessment of how good state s is, ranging from $+1$ (sure victory) to -1 (certain defeat).

The oracle is used to guide an exploration of the "near future" by a Monte Carlo Tree Search (MCTS) (Browne et al. 2012). In order to pick the next move, the game tree rooted at s is explored. Roughly speaking, in this exploration the moves that f_θ considers good are explored first (so that the branching factor is bounded) and the amount of new leaf explored per action is fixed (so that the depth of the exploration is also limited). The goal of this exploration phase is to produce a better estimate $(\boldsymbol{\pi}, q)$ of state s. When this is done, the best move according to $\boldsymbol{\pi}$ is played to reach a new state s', and the process is repeated.

What is noteworthy about this process is that while by itself f_θ is a rather weak player, using it in combination with MCTS gives rise to a very strong one, i.e. the estimates $(\boldsymbol{\pi}, q)$ are more reliable than $(\boldsymbol{p}, v)$. Let us call $A(f)$ the MCTS playing agent using f as oracle.

The crux of the approach is to generate a sequence of oracles $f_0, f_1, f_2, \ldots, f_t$ each better than the predecessors. This is done by generating a sequence of training sets $S_1, S_2, \ldots, S_t$ each better than the previous one. Training set S_i is used to train f_i. The process is initialized with a deep network f_0 with random weights.

The generic step in this sequence of improvements is as follows. Let f_θ be the current oracle. During the so-called *self-play phase*, $A(f_\theta)$ plays a batch of games against itself. During each game a set of states S will be explored. For each $s \in S$ an updated (and hopefully better) assessment $(\boldsymbol{\pi}_s, q_s)$ will be computed. The set T of pairs $\{s, (\boldsymbol{\pi}_s, q_s)\}$ for $s \in S$ will be added to the training set. The intuition is that this way we can create a virtuous circle. As the assessments $(\boldsymbol{\pi}_s, q_s)$ become more and more accurate the training set becomes better and better. And, as the training set improves the assessment becomes more accurate.

We remark the main difference between OLIVAW and AlphaGo Zero resides in how this training set is constructed. Instead of $\{s, (\boldsymbol{\pi}_s, q_s)\}$, AlphaGo Zero only considers pairs $\{s, (\boldsymbol{\pi}_s, z_s)\}$ where s is actually played during the game, and z_s is the outcome at the end of the game. Thus, $z_s \in \{-1, 0, +1\}$. In contrast, besides this type of pairs, OLIVAW also adds to the training set pairs $\{s, (\boldsymbol{\pi}_s, q_s)\}$ for which s has been explored "a lot". In this way we collect a larger training set for the same number of simulated games. This is crucial since the cost of the *self-play* phase is the main contributor to the overall cost. This design choice is discussed in detail in section 4.2.

Once the new training set is obtained we switch to the *training phase*. The current neural network f_θ is further trained with the updated training set. At the end of this training we have a new configuration f'_θ. We want the move probabilities and value $(, v) = f'_\theta(s)$ closer to the improved search probabilities and value $(\boldsymbol{\pi}\,, \omega)$, where ω can be z or q. So we minimize the training loss:

$$L(\boldsymbol{\pi}, \omega, \boldsymbol{p}, v) = (\omega - v)^2 - \boldsymbol{\pi}^T \log \boldsymbol{p} + c||\theta||^2 \qquad (1)$$

As in AlphaGo Zero, we combine evenly the squared error on the value and the cross-entropy on the move probabilities, while the last term penalizes large weights in the neural network ($c = 10^{-4}$). This L2 regularization is used to prevent overfitting over the many training phases.

In a final *evaluation phase* we verify whether the new f'_θ is stronger than the old f_θ. To do so, we pit $A(f'_\theta)$ against $A(f_\theta)$. If the former wins significantly more often than the latter it becomes the new oracle. Otherwise we go through the training phase again to produce a new challenger.

The whole process is then repeated. And so on, so forth.

For a more detailed description of the reinforcement learning algorithm please refer to Silver et al. (2017).

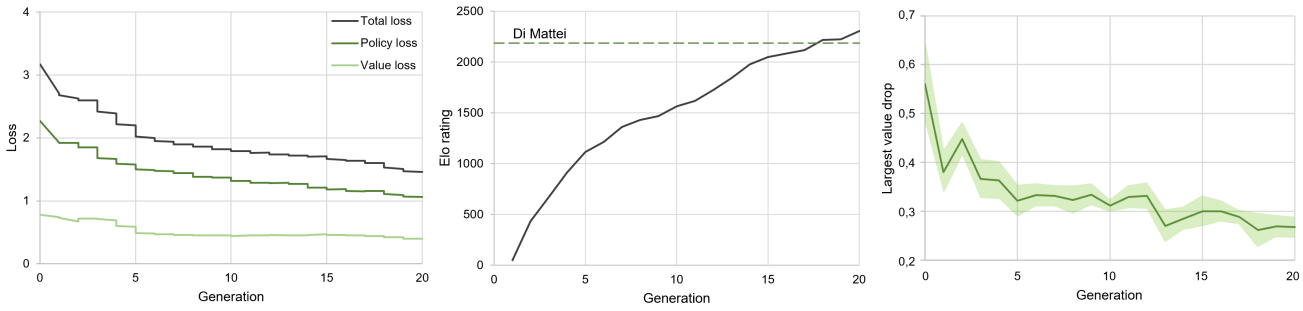
4.2 Low-cost faster training

With respect to AlphaGo Zero, OLIVAW introduces three main modifications in the training phase.

As remarked, while the training set of AlphaGo consists only of pairs of the kind $\{s, (\boldsymbol{\pi}_s, z_s)\}$, where s is a move actually played during self-play and z_s is the outcome of the end of the game, OLIVAW also considers pairs of the type $\{s, (\boldsymbol{\pi}_s, q_s)\}$, where s is a position in the game tree that has been explored above a certain threshold. The threshold value was set dynamically in order to have a training set of twice the size with respect to the AlphaGo approach. In other words, the number of pairs of type $\{s, (\boldsymbol{\pi}_s, q_s)\}$ added was roughly equal to that of the pairs of type $\{s, (\boldsymbol{\pi}_s, z_s)\}$. The pairs added were the ones with the largest number of visits. Our approach was broadly inspired by the convincing results of Young, Prasad, and Abrams (2018).

Adding noisy pairs might seem disadvantageous. In fact, besides the advantages discussed in the previous section, using only z as a signal has a big drawback. In a game with multiple errors, every evaluation of an early position based on the final outcome is almost random, while q offers a better assessment. On the other hand, q suffers from the limited horizon of the simulations; an early position with positive or negative consequences far ahead in the game may not be properly evaluated. So, a combination of the two signals might strike a better balance.

The second variation concerns a dynamic adjustment of the MCTS simulations. During the *self-play phase* OLIVAW selects each move after 100, 200 or 400 MCTS simulations from the current game state, using a higher number of sim-



(a) (b) (c)

Figure 2: **Training process. a.** Training loss across generations. The stepwise trend is due to the shifting training window. **b.** Performance of the i-th generation MCTS agent. The ELO ratings were computed using the evaluation games and the first match vs the national champion Alessandro Di Mattei. **c.** Absolute largest value drop in a game across generations. We show the averages using standard deviations as confidence interval.

ulations in the higher generations[3], as opposed to AlphaGo Zero that runs a fixed 1600 simulations in every generation. The rationale is to move quickly from early generations, where a dataset generated by low-depth MCTS still provides enough signal for an improvement, as noted by Wang et al. (2019).

Finally, OLIVAW uses a dynamic training window. The training set is a sample of 16,384,000 positions (minibatch size of 1024) from the games generated by the last generations. We gradually increase the generations included in the training window from the last 2 to the last 5, the idea is to exclude quickly the games played by the first very weak generations. AlphaGo Zero uses as training set a sample of 2,048,000 positions from the last 500,000 games, so drawing always from the games generated by the last 20 generations. This tweak proved effective in the Connect4 implementation of AlphaZero by Young, Prasad, and Abrams (2018).

5 Resources and training process

OLIVAW was entirely developed, trained and tested on Colaboratory, a free Google cloud computing service for machine learning education and research (Carneiro et al. 2018).

OLIVAW code is completely written in python, from the core MCTS and Neural Network classes implemented in Numpy (Virtanen et al. 2020) and Keras (Chollet et al. 2015), to the simple GUI based on Matplotlib (Virtanen et al. 2020). The *self-play*, *training* and *evaluation phases* take place on three self-contained distinct notebooks sharing the same memory. All the code will be made publicly available upon acceptance.

Concerning local resources, we took no more advantage than a laptop equipped with a browser and an internet connection.

5.1 Hardware used

The hardware specifications of a Colaboratory virtual machine at the time of the training were:

- CPU: 1 single core hyper threaded Xeon Processor, 2.3Ghz, 2 threads.
- RAM: ≈ 12.6 GB.
- Harware accelerators (if used):
  - GPU: 1 Nvidia Tesla K80, 2496 CUDA cores, 12GB GDDR5 VRAM.
  - TPU v2-8: Google Tensor processing unit equipped with 8 TPU cores.

The generation of games during the *self-play phase* is the most computationally expensive process in the learning algorithm. In our hardware configuration, a single game takes 6 to 30 seconds depending on the number of MCTS simulations, due to the high number of GPU calls.

Nevertheless, game generation can be naturally performed in parallel, by running several instances of the self-play notebook and saving the generated datasets on a shared memory. This result has been achieved by stimulating a small crowdcomputing project thanks to the ease of sharing of Colaboratory. 19 people contributed to the generation of games through their Google accounts, even from smartphone.

The *training phase* can not be executed in parallel but can be highly accelerated using the TPU runtime of Colaboratory, the acceleration factor is approximately 10 respect to a GPU K80. A single training phase requires ≈ 1.5 h.

The *evaluation phase* consists in 40 games between agents of different generations and can be run in less than an hour using the GPU runtime.

5.2 Training process

The version of OLIVAW discussed in this article is the result of a single training run lasting 30 days, 20 generations and $\approx 50,000$ games. We refer to the *i-th generation* as the *i-th succesful update of the weights of* f_θ.

An exhaustive evaluation of our specific choices in the design of the learning algorithm and a proper tuning of the hyperparameters for 8×8 Othello would have required several runs, not compatible with our main objective of mastering the game with limited resources. Nevertheless, as discussed

[3] OLIVAW switched to 200 simulations between the 4th and the 5th generation, and to 400 simulations between the 11th and 12th generation.

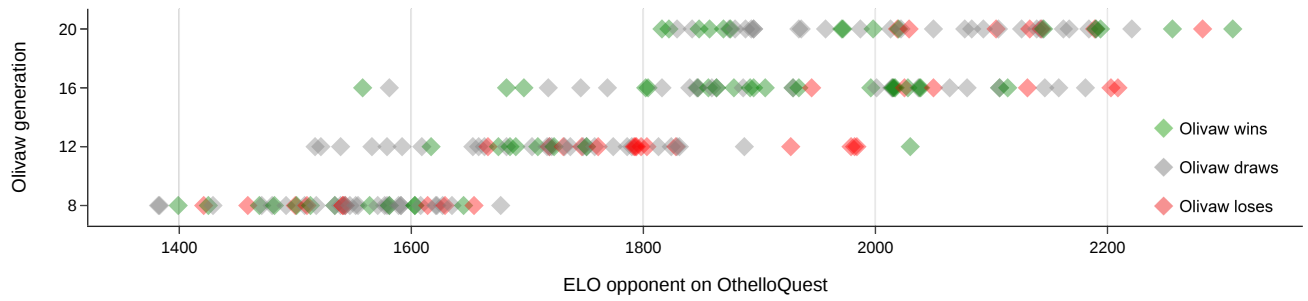


Figure 3: **Score of OLIVAW games on OthelloQuest at different generations.** OLIVAW played anonymously using 400 MCTS simulations. Here are reported the last 50 games played by each version of OLIVAW, excluding the positioning games. The performance ratings on these 50 games of generations 8, 12, 16 and 20 are respectively 1557, 1687, 2074 and 2100.

in Sec. 4.2, our choices find evidence in Wang et al. (2019), which tested 12 relevant hyperparameters on 6×6 Othello in about 36 runs, and in Young, Prasad, and Abrams (2018) which used the simpler game Connect4.

Training objective. The progression of the losses across the generations reflects the evolution of the training dataset, see Figure 2a.

As expected the largest drops in the losses are between generations and not during the single training phase, i.e. a f_θ fresh out of training fits better the new games it was not trained on than the ones just leaving the training window. Another indication of continuous learning can be spotted in the almost constant trend assumed by the value loss from the 6-th generation on, showing that new games provide always fresh information to learn. Note how the largest discontinuity in the total loss is in correspondence of the increase of MCTS simulations from 100 to 200 between the 4th and 5th generation.

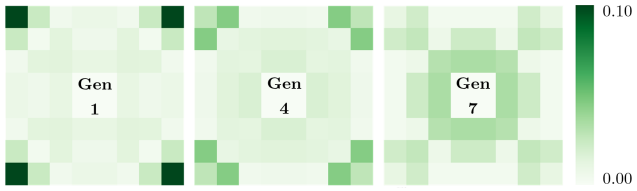


Figure 4: **Location of crucial moves by generation.** OLIVAW attributes high relevance to the conquest of the corners in the early generations, similarly to human beginners. This shifts to the accesses to the corners, and then towards the center of the board, as we would expect from a more experienced player.

Largest value drops by generation. During the self-play phase, the final positions in a game correspond to values close to 1 for the winner and -1 for the loser. Conversely, the evaluations of initial positions are close to 0 for both players. Recording the value trend in a game, we can spot the action corresponding to the largest value drop for the loser, interpreting its magnitude as the size of the most relevant error. We expect a decreasing trend of this drops across generations, both in mean and standard deviation. The rationale is that the defeat of an experienced player is less likely attributable to a single blunder. Such online measure provides a further evidence of improvement in the level of play of OLIVAW during the training run, see Figure 2c.

Othello players may also find insights about the style of learning of OLIVAW looking at the distribution on the board of these crucial actions across generations, see Figure 4.

5.3 General details to replicate training conditions

In the following we report all salient hyperparameters to reproduce this work. In each generation OLIVAW plays 2500 games against itself (25,000 in AlphaGo Zero). Each move is selected after 100, 200 or 400 MCTS as explained in Sec. 4.2.

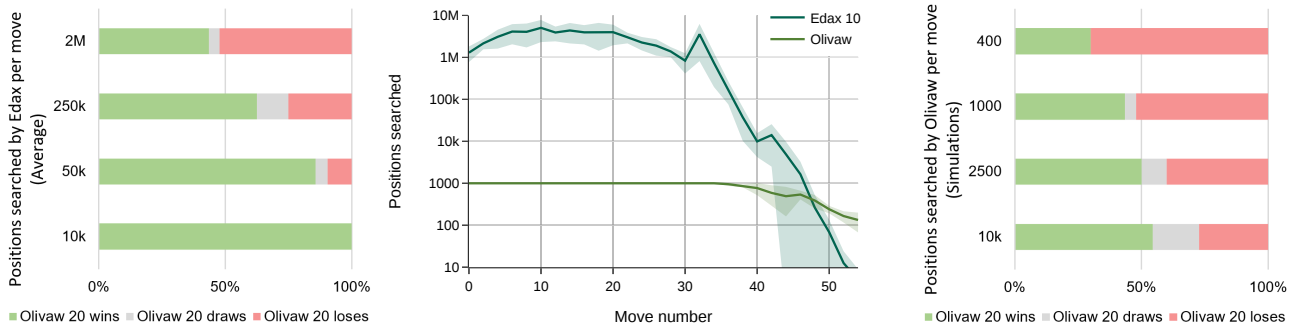
During the *self-play phase*, the first 20 moves of each game are selected extracting at random according to $\boldsymbol{\pi}$ to favor exploration, the remaining moves are selected taking argmax($\boldsymbol{\pi}$). As in AlphaGo Zero we use virtual losses (Segal 2010).

When selecting a move from state s with MCTS, in the root node we add Dirichlet noise to the prior probabilities computed by the neural network oracle:

$$P(s_0, a) = (1 - \epsilon)p_a + \epsilon x_a \quad \boldsymbol{x} \in \mathbb{R}^B \tag{2}$$

Where B is the number of legal moves from s_0, and $\boldsymbol{x}$ is a point sampled from the symmetric Dirichlet probability distribution X_α, with $\alpha = \min(1, 10/B)$, we used an $\epsilon = 0.25$ as in AlphaGo Zero. A symmetric Dirichlet noise with an $\alpha < 1$ tends to unbalance the move probability distribution $\pi(a|s_0)$ towards a specific action, favoring an exploration in depth of the subsequent variant. In games with high branching factor, this behaviour is desired to preserve the asymmetric nature of the MCTS search. So, the higher the branching factor of the game, the lower the α parameter of the symmetric Dirichlet noise used ($\alpha = 0.03$ in AlphaGo Zero).

When generating the dataset, not all games are played until the end to save computation. If during a game a player



(a) (b) (c)

Figure 5: OLIVAW **vs Edax. a.** Score of OLIVAW generation 20 searching 1000 positions per move (Borassi version) vs Edax at levels 10, 8, 6, and 4 (maximum search depth) on 20 games per game series. **b.** Positions searched per move across a game for Edax 10 and OLIVAW. The trend reported represents the mean and standard deviation for each move on 10 games. Notice the peak at 20 moves from the end where Edax is programmed to search deeper in preparation for the endgame. **c.** Score of OLIVAW generation 20 searching 400, 1000, 2500 and 10k positions per move vs Edax 10.

values a position under a resignation threshold v_{resign}, the game ends and the opponent is considered the winner. v_{resign} is chosen automatically playing a fraction of the games until the end, so that less than 5% of those games could have be won if the player had not resigned. In early generations OLIVAW plays 10% of the games until the end, increasing progressively this fraction to improve its strength in finals. This is because, differently from Go, Othello games are usually played until the very end, when no more moves are available. (AlphaGo Zero plays always only 10% of the games until the end).

In the *training phase*, neural network parameters θ are optimised using stochastic gradient descent with momentum $\lambda = 0.9$ and a stepwise learning rate annealing. Starting from a learning rate of 0.003, OLIVAW switched to 0.001 in the 4th generation and to 0.0001 in the 11th generation.

Concerning the architecture, OLIVAW uses a Residual Network (He et al. 2016) as AlphaGo Zero. The game state input s is a $8 \times 8 \times 2$ binary tensor in OLIVAW, in contrast to the deeper $19 \times 19 \times 17$ binary tensor of AlphaGo Zero. We do not need layers representing past positions in Othello since the game state is fully observable from the board position. Even the information on the player's turn is not necessary since Othello is symmetrical respect to the player, we can assume that in each position is always white to move, flipping all the discs if it is the turn of black.

Therefore the input s is processed by a single convolutional block and then by a residual tower of 10 residual blocks (39 in the strongest version of AlphaGo Zero). The output of the residual tower is then processed by the value head and the policy head that output respectively the value v of the position and the move probabilities $\boldsymbol{p}$. The structure of each block coincides with the correspondent one of AlphaGo Zero, except for the output shape of the policy head, $8^2 + 1 = 65$ in OLIVAW instead of $19^2 + 1 = 362$.

6 Attaining world-class level in Othello

During the learning run we tested the level of play reached by OLIVAW through a series of anonymous online games against human players. We chose OthelloQuest[4], the most popular Othello platform frequented also by top human players. A selection of OLIVAW generations played 5-minute games using 400 simulations per move. Results are summarised in Figure 3.

We tested OLIVAW also against Edax (Delorme 1998), the best open-source Othello engine (Liskowski, Jaśkowski, and Krawiec 2018). Like other top traditional engines, Edax is based on a highly optimized alpha-beta tree search (negamax) (Knuth and Moore 1975) using tabular value functions[5]. For these games Edax used no opening book and its search depth was limited from 4 to 10. Edax is a deterministic engine and when it played against OLIVAW the same single game was repeated again and again. To circumvent this problem we switched to random XOT openings, a popular Othello variation where the first 8 moves are chosen at random from a list of 10,784 sequences ending in an almost even position, i.e. positions judged between -2 and +2 discs advantage for black by Edax 16. The results are summarised in Figure 5, detailing the amount of positions searched per move by OLIVAW and Edax in each game series.

Finally, we organized three live matches with the support of the italian Othello Federation, two with the 2019 italian champion Alessandro Di Mattei, ranked today among the top 150 Othello players in the world, and a formal match with the former World champion Michele Borassi, ranked in the top 50 and who finished in the top five in his last World Othello championship appearance in 2018[6].

[4] http://wars.fm/reversi

[5] Unfortunately, a detailed description of how these value functions are obtained is not publicly available.

[6] World Othello ratings updated to 2021/01/24, official page of

First match against the national champion Alessandro Di Mattei - Best of 5 - 2018/11/27

Di Mattei	OLIVAW 14	32-32	C4E3F6E6F5C5C3C6D3D2E2B3B4C2B6A4B5D6A3A5A6F3F4G4F7D1F1D7E1C1B1G6C7E7F8D8H6F2G1G5C8B8G7B7E8G2A8A7H1G3H2H3H4B2A2A1PAH5PAH8G8H7
OLIVAW 14	Di Mattei	32-32	D3C3C4C5B4D2D6C6E6D7B5A3C7C8B6A6E3E7A4F2F8F5F6F4A2E8F7G5G6H6C2B7F3A5A8A1E2B3D8B8B2G4G3B1C1H3F1E1D1G7H4H5A7G1H8G8H7G2H2PAH1
Di Mattei	OLIVAW 14	43-21	C4E3F6E6F5C5C3C6D3D2E2B3C1C2B4A3A5B5A6F4F3B6E7D1E1G5G4G6H5D6B1H3H4H6F7F8D8A4A2E8G8G3D7C8B8G7H8H7A7C7H2G2G1B7A8F1F2H1PAA1PAB2
OLIVAW 14	Di Mattei	36-28	F5F6E6D6E7G5C5C6C4F3D7D8C7C8F4B6G6G4H5H6H4H3G3B5E3B3B4C3A5A6D3C2D2D1A3H2C1B1F7A4A7B2E2E8G8G7F8H8H7G2F2B7A8B8H1F1G1E1A1A2
Di Mattei	OLIVAW 14	33-31	C4E3F6E6F5C5F4G6F7C3H6G4G3D7E7F3F2H3D3E2E1C6D6G5D2C7C8C2B1E8B8F1G1G8H5H4B7B5A5B4F8B6D8A8H2C1D1H7H8G7B2B3A3A4A7A6PAA2PAG2H1PAA1

Second match against the national champion Alessandro Di Mattei - Best of 5 - 2018/12/04

Di Mattei	OLIVAW 18	27-37	C4E3F6E6F5C5C3C6D3D2E2B3B4A3E7C2D6F1D1F4E1C1B6F3B2D7C8G5H4C7D8G6H6H5G4H3H2B7B1B5A8A1A5E8F7G7A4F8H7A6A2H8G8H1G3F2A7B8G1G2
OLIVAW 18	Di Mattei	40-24	E6F6F5D6E7G5C5C6E3C4D7F8B4D3C3A3B5B3B6C8A4A5A6A7F4C7G6H6F7G8H4H5H7C2D2F2F3D1E2G2E1C1B1G4F1B2A1A2A8G3E8D8B8B7H8G7H1H2H3G1
Di Mattei	OLIVAW 18	27-37	C4E3F6E6F5C5C3C6D3D2E2B3C1C2B4A3A5B5A6B6A4A7E7E1D6D7C8F3C7F8F7G5H4G6H5D1F1F2B1H6H7G4H3F4G3E8B7D8G8B2B8G7A2A1PAG1G2H2H1
OLIVAW 18	Di Mattei	45-19	C4E3F6E6F5C5C3B4D3C2D6F4E2F3D2C6G5G4F2G3H3H4B5H6A3B6D7A4A5A6B3C1E1A2D1F1G2E7E8F7H2C8F8D8C7G8B7G6H5H1G1B8G7A8A7B2A1B1PAH8H7

Match against the former World champion Michele Borassi - Best of 3 - 2019/01/19

OLIVAW 20	Borassi	35-29	E6F4C3C4D3D6F6E7F3C5F5G4B5G5C6E3D7B4G6F7E2C8B6C7A5H5E8F2B3F8D8E1H6A4A3C2C1A7H3H4A6A2D2D1G3G2H1H2B2B8A8B7A1B1G8G7H8H7F1G1
Borassi	OLIVAW 20	35-29	D3C5F6F5E6E3C3D2F4F3C2G4D1D6E2F2G3H4G1F7H3G6C4H2G5B4H5H6E7B3C7C6D7D8B5A5E8F8A6C8A4F1E1B1B6A2C1H1A3A7G7G8H8H7B7B2A1A8B8G2
Borassi	OLIVAW 20	47-17	D3C5F6F5E6E3C3D2F4F3C2G4D1D6E2F2G3G5E1F1G1D7H5G6H6H3C4B4H7H2F7B3B6F8G7C6A3A6A5A4B5B1C1H1A7A2A1B7E8D8A8C7B2G8G2E7H8B8H4C8

Table 1: OLIVAW **games versus top Othello players.**

6.1 Matches against a national champion

Two informal best-of-five series were organized against the italian champion Alessandro Di Mattei. They took place between 27 November and 4 December 2018 in Rome. The first series against Di Mattei saw the 14-th generation of OLIVAW very close to the national champion, who emerged victorious after two initial draws. A post-match analysis of the games showed OLIVAW losing from winning positions in the final moves. This feedback led to the decision of simulating all the subsequent *self-play games* until the very end, to strengthen OLIVAW's late game. The second series against generation 18 ended with a resounding 4-0 victory for OLIVAW. This boosted our confidence and we threw down the gauntlet against a former world champion. All matches are shown in Table 1.

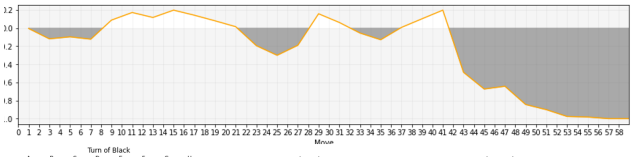
6.2 Match against a former World champion

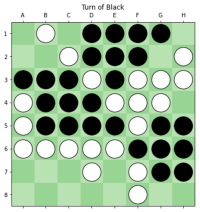
Former world champion Michele Borassi picked up the gauntlet and a formal series was organized. The match was sponsored by the Sapienza Computer Science department and was open to public. The formula was a best-of-three with 30 minutes for each player for the whole game, as in the world championship. After the good results against Di Mattei, we decided to keep the MCTS simulations at 1000 (few seconds to make a move) and to stop the training at generation 20, after ≈50,000 games simulated in *self-play*.

OLIVAW won the first game as black, losing the second and third one with white. The final score was thus 2-1 against OLIVAW. All matches are shown in Table 1.

Othello players may find interesting move 43 - C1 of Borassi in game 3. It is a highly counter-intuitive and the only option to avoid defeat: with a master stroke the world champion snatched victory from the jaws of defeat, saving the honor of the human race. Borassi spent more than one-third of its time on that single move, OLIVAW did not consider it enough, as evidenced by the large value drop recorded on move 43 (see Figure 6). This act of hubris proved fatal for OLIVAW.

the World Othello Federation https://www.worldothello.org





Borassi (**B**) vs Olivaw gen 20 (**W**) – **47 - 17**

1.D3 2.C5 3.F6 4.F5 5.E6 6.E3 7.C3 8.D2 9.F4
10.F3 11.C2 12.G4 13.D1 14.D6 15.E2 16.F2 17.G3 18.G5
19.E1 20.F1 21.G1 22.D7 23.H5 24.G6 25.H6 26.H3 27.C4
28.B4 29.H7 30.H2 31.F7 32.B3 33.B6 34.F8 35.G7 36.C6
37.A3 38.A6 39.A5 40.A4 41.B5 42.B1 43.C1 44.H1 45.A7
46.A2 47.A1 48.B7 49.E8 50.D8 51.A8 52.C7 53.B2 54.G8
55.E7 55.G2 56.E7 57.H8 58.B8 59.H4 60.C8

Figure 6: **Match between OLIVAW generation 20 and the former World champion Michele Borassi, final game.** The graph shows the confidence of winning of OLIVAW; -1 a sure defeat, +1 a sure victory.

7 Conclusion

After one month of training, using only free (but limited) cloud computing resources, OLIVAW has achieved a world-class level in the game of Othello, defeating the national champion Alessandro Di Mattei and losing only in the last game of a series to the former World champion Michele Borassi. The outcome of games played against Edax, one of the strongest Othello engines, corroborate this general conclusion.

OLIVAW learned to play Othello knowing only the rules of the game, starting *tabula rasa* as AlphaGo Zero, demonstrating once again the generality of this learning paradigm.

Still, the level of play reached by OLIVAW is not yet superhuman. This would be the natural next step for OLIVAW and is left for future work. Such an achievement should come using a limited amount of resources in the training phase and satisfying the constraint of a small number of MCTS simulations per move in the test matches, i.e. an amount of search per decision closer to the one of a human than to the one of a traditional engine, as our 1000.

Acknowledgments

This research was partially supported by a Google Focused Award and by BICI, the Bertinoro International Center for Informatics.

We thank Alessandro Di Mattei and Michele Borassi for agreeing to play against OLIVAW; Paolo Scognamiglio and Alessandro Di Mattei for introducing us to the Othello world and the helpful discussions on OLIVAW's style of play, Roberto Sperandio for the gripping match commentary, Benedetto Romano for the discussion on traditional engines, Leonardo Caviola for having introduced the public to Othello during the match-day, and the whole Italian Othello Federation.

Thanks to everyone who contributed to generate the training dataset through Google Colaboratory: Dario Abbondanza, Armando Angrisani, Federico Busi, Silva Damiani, Paola D'Amico, Maurizio Dell'Oso, Federico Fusco, Anna Lauria, Riccardo Massa, Andrea Merlina, Marco Mirabelli, Angela Norelli, Oscar Norelli, Alessandro Pace, Anna Parisi, Tancredi Massimo Pentimalli, Andrea Santilli, Alfredo Sciortino, and Tommaso Subioli.

References

Amodei, D.; and Hernandez, D. 2018. AI and Compute. URL https://blog.openai.com/aiand-compute.

Andersson, G. 1997. Zebra. URL https://web.archive.org/web/20200226073330/http://radagast.se/othello/zebra.html.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1): 1–43.

Buro, M. 1995. Logistello: A strong learning othello program. In *19th Annual Conference Gesellschaft für Klassifikation eV*, volume 2. Citeseer.

Carneiro, T.; Da Nóbrega, R. V. M.; Nepomuceno, T.; Bian, G.-B.; De Albuquerque, V. H. C.; and Reboucas Filho, P. P. 2018. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access* 6: 61677–61685.

Chang, N.-Y.; Chen, C.-H.; Lin, S.-S.; and Nair, S. 2018. The Big Win Strategy on Multi-Value Network: An Improvement over AlphaZero Approach for 6x6 Othello. In *Proceedings of the 2018 International Conference on Machine Learning and Machine Intelligence*, 78–81.

Chollet, F.; et al. 2015. Keras. URL https://keras.io.

Delorme, R. 1998. Edax. URL https://github.com/abulmo/edax-reversi.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* .

Kasparov, G. 2018. Chess, a Drosophila of reasoning. *Science (New York, N.Y.)* 362(6419): 1087.

Knuth, D. E.; and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artificial intelligence* 6(4): 293–326.

Lee, B.; Jackson, A.; Madams, T.; Troisi, S.; and Jones, D. 2019. Minigo: A Case Study in Reproducing Reinforcement Learning Research. In *RML@ICLR*.

Liskowski, P.; Jaśkowski, W.; and Krawiec, K. 2018. Learning to play othello with deep neural networks. *IEEE Transactions on Games* 10(4): 354–364.

Mullins, J. 2007. Checkers solved after years of number crunching. *New Scientist* .

Neumann, J. v. 1928. Zur theorie der gesellschaftsspiele (German). *Mathematische annalen* 100(1): 295–320.

Pachocki, J.; Brockman, G.; Raiman, J.; Zhang, S.; Pondé, H.; Tang, J.; Wolski, F.; Dennison, C.; Jozefowicz, R.; Debiak, P.; et al. 2018. OpenAI Five URL https://blog.openai.com/openai-five.

Romano, B. 2009. *SAIO: Un sistema esperto per il gioco dell'othello (Italian)*. Ph.D. thesis, University of Naples Federico II. URL http://www.romanobenedetto.it/tesi.pdf.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2019. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265* .

Segal, R. B. 2010. On the scalability of parallel UCT. In *International Conference on Computers and Games*, 36–47. Springer.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science (New York, N.Y.)* 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature* 550(7676): 354–359.

Tian, Y.; Ma, J.; Gong, Q.; Sengupta, S.; Chen, Z.; Pinkerton, J.; and Zitnick, C. L. 2019. Elf opengo: An analysis and open reimplementation of alphazero. *arXiv preprint arXiv:1902.04522* .

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575(7782): 350–354.

Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Jarrod Millman, K.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C.; Polat, İ.; Feng, Y.; Moore, E. W.; Vand erPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van

Mulbregt, P.; and contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17: 261–272.

Wang, H.; Emmerich, M.; Preuss, M.; and Plaat, A. 2019. Hyper-Parameter Sweep on AlphaZero General. *arXiv preprint arXiv:1903.08129* .

Wu, D. J. 2019. Accelerating Self-Play Learning in Go. *arXiv preprint arXiv:1902.10565* .

Wu, T.-R.; Wei, T.-H.; Wu, I.; et al. 2020. Accelerating and Improving AlphaZero Using Population Based Training. *arXiv preprint arXiv:2003.06212* .

Young, A.; Prasad, A.; and Abrams, I. 2018. Lessons From Implementing Alpha Zero. URL https://link.medium.com/ylGDD6F7V9.