Polygames: Improved Zero Learning

Tristan Cazenave, Yen-Chi Chen, Guan-Wei Chen, Shi-Yu Chen, Xian-Dong Chiu, Julien Dehos, Maria Elsa, Qucheng Gong, Hengyuan Hu, Vasil Khalidov, et al.

ICGA Journal 0 (0) 1
IOS Press

Polygames: Improved Zero Learning

Tristan Cazenave [a], Yen-Chi Chen [b], Guan-Wei Chen [c], Shi-Yu Chen [c], Xian-Dong Chiu [c], Julien Dehos [d], Maria Elsa [c], Qucheng Gong [e], Hengyuan Hu [e], Vasil Khalidov [e], Cheng-Ling Li [c], Hsin-I Lin [c], Yu-Jin Lin [c], Xavier Martinet [e], Vegard Mella [e], Jeremy Rapin [e], Baptiste Roziere [e], Gabriel Synnaeve [e], Fabien Teytaud [d], Olivier Teytaud [e], * Shi-Cheng Ye [c], Yi-Jun Ye [c], Shi-Jim Yen [c] and Sergey Zagoruyko [e]

[a] *LAMSADE, University Paris-Dauphine, PSL, France*
E-mail: tristan.cazenave@lamsade.dauphine.fr
[b] *National Taiwan Normal University, Taiwan*
E-mail: zxkyjimmy@gmail.com
[c] *AILAB, Dong Hwa University, Taiwan*
E-mail: sjyen@mail.ndhu.edu.tw
[d] *University Littoral Cote d'Opale, France*
E-mail: dehos@univ-littoral.fr
[e] *Facebook AI Research, France and United States*
E-mail: oteytaud@fb.com

Abstract. Since DeepMind's AlphaZero, Zero learning quickly became the state-of-the-art method for many board games. It can be improved using a fully convolutional structure (no fully connected layer). Using such an architecture plus global pooling, we can create bots independent of the board size. The training can be made more robust by keeping track of the best checkpoints during the training and by training against them. Using these features, we release Polygames, our framework for Zero learning, with its library of games and its checkpoints. We won against strong humans at the game of Hex in 19x19 , including the human player with the best ELO rank on LittleGolem; we incidentally also won against another Zero implementation, which was weaker than humans: in a discussion on LittleGolem, Hex19 was said to be intractable for zero learning. We also won in Havannah with size 8: win against the strongest player, namely Eobllor, with excellent opening moves. We also won several first places at the TAAI 2019 competitions and had positive results against strong bots in various games.

Keywords: Zero learning, Board games

1. INTRODUCTION

In spite of AlphaGo (Silver et al., 2016), some games still resist to computers and for many games the computational requirement is still huge. We present Polygames, our open-source zero learning framework available at https://github.com/facebookincubator/polygames. It is based on Zero learning, combining Monte Carlo Tree Search and Deep Learning. It features a new architecture for accelerating the training and for making it size-invariant (Section 2.1). It allows neuroplasticity i.e. adding neutral layers, adding channels, increasing kernel size (Section 2.2) and warm start. Polygames also features a new tournament mode in order to make training more robust (Section 2.3). The framework provides a single-file API, that is generic enough for implementing many games, and comes with a library of games and many checkpoints. Basically, for adding a game, one must implement the transition function *next* (gameplay and reward), the mapping $s \mapsto tensor(s)$ representing a state as a 3D-tensor,

and a mapping for identifying an action to a 3-dimensional location in the 3D output tensor. One can for example duplicate the code in connectfour.h and modify it. Polygames made the first ever win against top level humans at the game of Hex 19x19 (Section 3.1) and Havannah 8x8 (Section 3.3).

1.1. Zero learning

AlphaGo and AlphaZero (Silver et al., 2016, 2017) proposed a combination between Monte Carlo Tree Search (Coulom, 2007) and Deep Learning. The version in (Silver et al., 2017) is a simplified and elegant version, learnt end-to-end.

1.1.1. Monte Carlo Tree Search

Monte Carlo consists in approximating values (i.e. expected rewards) by averaging random simulations. MCTS (Monte Carlo Tree Search) consists in biasing these random simulations: using the statistics from previous simulations, we increase the frequency of moves which look good. The most well known variant of MCTS is Upper Confidence Trees (UCT) (Kocsis and Szepesvári, 2006), which uses, for biasing the simulations, a formula inspired from the bandit literature: a move is chosen if it maximises a score as follows:

$$score_{uct}(s,a) = avg\ reward(next(s,a)) + k\sqrt{\frac{\log(num\ sims(s))}{num\ sims(next(s,a))}} \quad (1)$$

where s stands for a state, a for an action, $num\ sims(s)$ for the number of simulations in s, and *next* represents the dynamics of the game. An MCTS rollout consists in doing a simulation from the current board s_0 and playing using the above formula until a final state is reached. We keep from Monte Carlo a random component in MCTS rollouts: in states without any statistics, the selection is simply random move selection. When M MCTS rollouts have been performed starting in the current state s_0, we choose an action a maximizing e.g. $num\ sims(next(s_0,a))$.

1.1.2. Neural policies

Let us assume that we have a function *tensor* which maps a state to a tensor $tensor(state)$ (to be used in Eq. 2 below). A neural network (NN) can then be applied. We consider a NN with two outputs:

$\pi_{NN}(state)$ is a tensor,

$V_{NN}(state)$ is a real number.

The NN typically uses convolutional layers, and then two heads for those two outputs; in the original Alpha-Zero, it contains fully connected layers in both heads. If we assume that each possible action has an index in the output tensor, then $\pi_{NN}(s)$ can be converted into probabilities of actions by (i) multiplying by some temperature T and applying the exp function to each entry in it (ii) setting to zero illegal actions in *state* s (iii) dividing by the sum:

$$logit(a) = \pi_{NN}(s)$$
$$Proba_{NN}(s,a) = \frac{\exp(Tlogit(a))}{\sum_{a'}\exp(Tlogit(a'))}$$

Let us note $Proba_{NN}(s,a)$ the probability of action a in state s for the neural network NN.

1.1.3. Zero model of play: how the MCTS is modified by adding the NN

The zero-model is then as follows. First, the UCT formula (Eq. 1) is adapted as PUCT (Silver et al., 2017), as follows:

$$score_{puct}(s,a) = \bar{Q}(s,a) + k \times P_{NN}(tensor(s))(a) \times \frac{\sqrt{n(s)}}{n(next(s,a))} \quad (2)$$

In Eq. 2, $n(s)$ stands for *num sims*(s), $\bar{Q}(s,a) = avg\ reward(next(s,a))$, $tensor(s)$ is a tensor representation of the state, $P_{NN}(tensor(s))$, output of the action head of the neural network, is also a tensor shaped as the output space - e.g. in Go or Hex the output tensor shape is the board size[1]. Each action a is identified to a location in the output tensor of the action head, so that $Proba_{NN}(tensor(s))(a)$ is a real number, namely the logit of the probability of choosing action a according to the neural network in state s. Please note that the $\log$ has been removed. The parameter k has been multiplied by the probabilities provided by NN.

Second, when a simulation reached a state in which no statistics are available (because no simulation has been performed here), instead of returning the reward of a random rollout until the end of the game, we return the reward estimated by V_{NN}. The NN has therefore the double impact of (i) biasing the tree search (ii) replacing the random part of the rollouts by neural estimates of value.

1.1.4. Zero training: how the NN is modified by the MCTS

Let us generate games with a zero-model as above, using e.g. $M = 600$ simulations per move. Each time a game is over, we have a family of 3-tuples (s,p,r), one per visited state in this game:

- r is the final reward of the game.
- p is the tensor of frequencies in the MCTS simulations at s. More precisely, with a an action identified to a location in the output tensor as above, $p(a) = n_{mcts,s,a}/M$ where $n_{mcts,s,a}$ is the number of the simulations which have chosen action a, among the M simulations starting at s.

These 3-tuples are then used for training the network so that π_{NN} imitates p and V_{NN} approximates the reward. We also use a weight decay as a regularization, so that the overall loss is

$$Loss(NN) = \sum_{(s,r,p)} CrossEntropy(p, Proba_{NN}(s,.)) + ||r - V_{NN}(s)||^2 + c_{L2}||\theta||^2$$

1.1.5. Overall architecture

There is a server (typically for us 8 GPUs) and many clients (tested up to 500 GPUs and 5 000 cores in our experiments):

- The server receives 3-tuples (s,p,r) from the clients. It stores them in a replay buffer (with default size 1 000 000 in our implementation), in a cyclic manner. It trains the NN as in Section 1.1.4, also cycling over the replay buffer.
- The clients send data (3-tuples) to the server.

[1] The output of the neural network is actually corrupted by a Dirichlet noise: rather than $o = Proba_{NN}(tensor(s))$, we use $.75o + .25DirichletNoise(1/numMoves)$, where *numMoves* is the number of legal moves in s. This is close to what was already done in (Silver et al., 2017).

The number of clients should be tuned so that the cycles performed by the trainer are just a bit faster than the speed at which data are provided; Section 2.4 provides a robust solution for ensuring this, in particular for low computational power.

1.2. Other open-source frameworks

Many teams have replicated and sometimes improved the Alpha Zero approach for different games.

Elf/OpenGo (Tian et al., 2019) is an open-source implementation of AlphaGo Zero for the game of Go. After two weeks of training on 2 000 GPUs it reached superhuman level and beat professional Go players. Compared to Elf/OpenGo, Polygames features a wide library of games, boardsize-invariance, and growing architectures.

Leela Zero (Pascutto, 2017) is an open-source program that uses a community of contributors who donate GPU time to replicate the Alpha Zero approach. It has been applied with success to Go and Chess. Compared to Polygames, its library of games is smaller, it is not boardsize-invariant and does not support growing architectures.

Crazy Zero by Rémi Coulom is a zero learning framework that has been applied to the game of Go as well as Chess, Shogi, Gomoku, Renju, Othello and Ataxx. With limited hardware it was able to reach superhuman level at Go using large batches in self-play and improvements of the targets to learn such as learning territory in Go. While the predicted final territory information (attribution of each location on the board) is not used by the Zero player, learning territory, as a side information in Go, increases considerably the speed of learning. Polygames does not have this clever use of side information. This is under progress. Polygames, on the other hand, features a large library of games, boardsize invariance, and growing architectures.

KataGo (Wu, 2019) is an open-source implementation of AlphaGo Zero that improves learning in many ways. It converges to superhuman level much faster than alternative approaches such as Elf/OpenGo or Leela Zero. It makes use of different optimizations such as using a low number of playouts for most of the moves in a game so as to have more data about the value in a shorter time. It also uses, as a side information for training the neural network, additional training targets so as to regularize the networks. Compared to KataGo, Polygames has a wide library of games, boardsize invariance, diversity of opponents for the training, and growing architectures.

Galvanise Zero (Emslie, 2019) is an open-source program that is linked to General Game Playing (GGP) (Pitrat, 1968). It uses rules of different games represented in the Game Description Language (GDL) (Love et al., 2006), which makes it a truly general zero learning program able to be applied as is to many different games. The current games supported by Galvanise Zero are Chess, Connect6, Hex11, Hex13, Hex19, Reversi8, Reversi10, Amazons, Breakthrough, International Draughts. Contrarily to Galvanise Zero, Polygames is based on compiled C++ game descriptions, and uses growing architectures and boardsize-invariant neural networks.

To the best of our knowledge, besides differences pointed out above, Polygames is the only framework featuring a population of models (the past checkpoints), ranked by an ELO scale and used for sampling sparring partners for the training. It is also the framework in which growing architectures are most systematized.

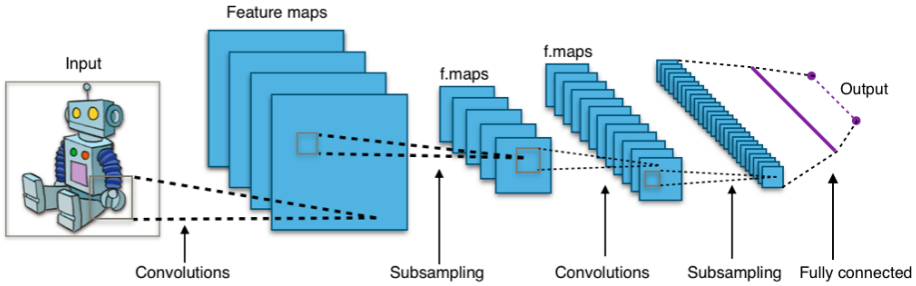


Fig. 1. Traditional deep convolutional network. The convolutional layers preserve the spatial coordinates of the shape - possibly not the depth, i.e. the number of feature maps. The number of parameters of those layers, therefore does not depend on the board size, and a same neural network can work in 9x9 Go and 19x19 Go (or Hex, etc). On the other hand, each neuron in the fully connected layers has one weight per location in its input tensor (plus the bias weight), so that a neural network with fully connected layers is not boardsize-independent. This can be solved in two different manners: (1) using only convolutional layers, which is possible when the output tensor has the same spatial shape as the input tensor - which is the case, for many games, in the case of the action head of the neural network in zero-learning (2) applying global pooling, before the fully connected layers, for reducing the spatial dimensions to (1x1). In Polygames, we apply (1) for the actor head of the neural network and (2) for the critic head, so that our neural networks are boardsize invariant (Section 2.1.2). The resulting architecture is presented in Fig. 2.

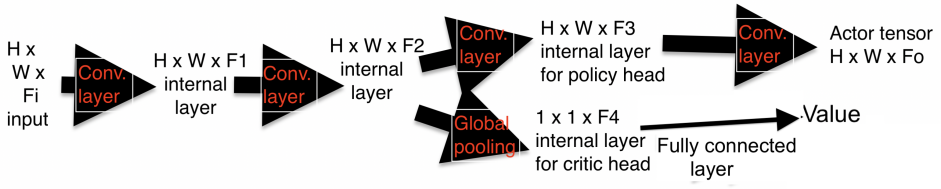


Fig. 2. Structure of a boardsize invariant neural network in Polygames. $H \times W$ is the board size (spatial coordinates). F_i is the number of input channels. F_1 and F_2 are the number of channels in the first and second convolutional layers (in real experiments there are way more layers). The input (more precisely, the tensor representation of the input state) is on the left. The output has two heads: the policy (also known as actor, at the top) and the critic, which approximates the value. The only fully connected layer is for predicting the value: importantly, it is based on the output of a global pooling layer, so that it does not depend on the boardsize. Using global pooling before the fully connected layer and convolutional layers everywhere else, we get a boardsize invariant neural network. F_1, F_2, F_3, F_4 are fixed parameters, independent of the board size.

2. INNOVATIONS

Some innovations in Polygames consist in using ideas from computer vision inside Zero learning (Section 2.1): global pooling, fully convolutional neural networks. We also use a diverse set of opponents for filling the replay buffer (i.e. the current model plays against previous models sampled according to their ELO rank as described in Section 2.3), and growing architectures (Section 2.2). These elements are detailed in the present section.

2.1. Structured zero learning

2.1.1. Fully convolutional models: taking into account the natural mapping between inputs and outputs

Many zero-learning methods are based on traditional convolutions, followed by fully connected layers. However, policy learning in board games is often closer to image segmentation (i.e. several output values per input pixel) than to classical image classification (fixed output size, independent of the

number of pixels) as actions are naturally mapped on boards (a vector of logits for each location on the board). More precisely, for many games:

- The input has various channels, and two dimensions matching the board size (spatial dimensions; see Fig. 1).
- Similarly, the output of the network has various channels, corresponding to various possible moves, and two dimensions matching the board size as well.

Therefore, we can apply fully convolutional models - not a single fully connected layer is necessary in the policy part. This contributes to scale invariance (Section 2.1.2) but also takes into account the partial translation invariance of policy learning. Polygames features fully convolutional networks (Shelhamer et al., 2017), residual fully convolutional networks (He et al., 2015), including with global pooling (Wu, 2019) as detailed below for scale invariance, and U-networks (Ronneberger et al., 2015).

2.1.2. Scale invariant models

The widely cited (Marcus, 2018) pointed out that zero-learning frameworks are not boardsize invariant. In the present section, we point out that Polygames learns in a boardsize invariant manner. As usually in zero learning, our neural network has two heads: one for the policy and one for the value. The one for the policy is fully convolutional (Section 2.1.1), and therefore it works independently of the input size, i.e. independently of the board size. The value part, however, does not have this property if it is fully connected. We therefore use global pooling as in (Wu, 2019). Global pooling replaces each channel c, of shape possibly $boardsize \times boardsize \times 1$, by several channels, such as the maximum and the average of c. This maximum and this average are computed over the $boardsize \times boardsize$ entries. We therefore get a boardsize-independent representation, of shape $1 \times 1 \times number\ of\ channels$. Our Hex19 model was trained in 13×13 and was immediately strong in 19×19 – though we needed a bit of fine tuning for the success story presented in Section 3.1.

2.2. Neuroplasticity

Several modifications are almost neutral when initialized close to zero:

- addition of residual blocks (i.e. switching from 12 blocks of 3 convolutional layers to 13 or 14 blocks of 3 convolutional layers);
- addition of new channels;
- extension of the kernel size (from 3×3 to 5×5 or 5×5 to 7×7, etc).

Polygames provides a script "convert" that makes such a growth of the neural network easy. Training can be resumed after such extensions of the neural architectures; we can train, then grow (while preserving the quality of the model as it remains almost equal to the previous model as new weights are close to 0), then resume the training with more degrees of freedom.

2.3. Tournament mode

In order to fight catastrophic forgetting (McCloskey and Cohen, 1989) or the red queen effect (oscillations of performance (Johansson, 2011)), we add a tournament mode, as follows. Each completed game is used for evaluating the ELO rank of players. Each client, when it must start a new simulated game, randomly draws a checkpoint in the archive. With *dev* the most recent checkpoint and

ELO_{dev} its ELO rank, the probability that a model with ELO rank ELO is drawn is proportional to $\exp(-\frac{ELO_{dev}-ELO}{400})$. Then the client simulates games between the MCTS using PUCT (Eq. 2) using "dev" and a MCTS using PUCT with that model. We experimentally notice that sometimes old models come back in the race.

2.4. Other features

2.4.1. Checkpoints

We provide checkpoints[2] for many games: Einstein Würfelt Nicht, Breakthrough, Havannah8, Havannah10, MiniShogi, Othello8, Othello10, Hex, and others.

2.4.2. Against overfitting

Heuristically, we consider that an example, in the replay-buffer, should never be seen more than 8 times. When the clients are not fast enough for filling the replay buffer, for example because of preemption of clients or slow game, we artificially add delays in the learning when an example is seen more than 8 times.

2.4.3. Easy addition of games

Adding a new game can be made by writing a new class that inherits from State and overrides a few methods (see the implementation of Connect Four[3] as an example).

2.4.4. Stochastic games and some partially observable games

Polygames can handle stochastic games (see e.g. our bot "randototoro" on LittleGolem, playing the game of Einstein Würfelt Nicht), which is not that common in existing frameworks but not conceptually hard: we just need random nodes, instead of only minimization and maximization nodes. The adaptation of MCTS to partial observation is more tricky: we cannot simulate the underlying dynamics from a given state, because players have only a partial observation. Actually, (Buffet et al., 2012) has presented a class of partially observable games that can be converted to an equivalent stochastic game, and therefore can be handled by MCTS (and, equivalently, by Zero learning as shown by Polygames): all games in which the visible information is the same for all players. This class includes a few two-player games: for example Chinese Dark Chess: there is hidden information, but the observable part is the same for both players. It also includes all one-player games, in particular Minesweeper and Mastermind. In those games, the undecidability results such as (Auger and Teytaud, 2012) do not apply.

For self-containedness, let us explain the reasoning in (Buffet et al., 2012). We do not know the state, so instead of a state, s is the history of observations: we must make a decision in a given s. This concept of history of observations is correctly defined because, by assumption, the history of observations is the same for all players. Given an history of observations s (including actions) and an action a, we can solve the transition by simulating $next(s,a)$ by rejection sampling as follows:

(1) Consider $r_1,\dots,r_n$ all sources of randomness up to "state" s.
(2) Randomly draw $r_1,\dots,r_n$ and simulate a game with actions as in s and random outcomes as in $r_1,\dots,r_n$.

[2] http://dl.fbaipublicfiles.com/polygames/checkpoints/list.txt
[3] https://github.com/facebookincubator/polygames/blob/master/games/connectfour.h

(3) If results are not consistent with the observation in s, go back to step 2.

This is fully developed in (Buffet et al., 2012) in the case of MCTS and implemented inside Polygames for Minesweeper and Mastermind. In the case of Minesweeper, we implemented the more sophisticated method from (Buffet et al., 2012), faster than the rejection sampling.

3. SUCCESS STORIES

Results here are obtained using (i) trainings stabilized by the tournament mode (Section 2.3), (ii) architectures growing in order to double the number or parameters each time the ELO rank stagnates (Section 2.2) (iii) boardsize-invariant architectures (Section 2.1), with top performance typically obtained after 9 days with 500 GPUs (usually playing at a very strong level after 3 days).

3.1. Beating humans at Hex19

According to (Bonnet et al., 2016), "Since its independent inventions in 1942 and 1948 by the poet and mathematician Piet Hein and the economist and mathematician John Nash, the game of Hex has acquired a special spot in the heart of abstract game aficionados. Its purity and depth has lead Jack van Rijswijck to conclude his PhD thesis with the following hyperbole (van Rijswijck, 2006): « Hex has a Platonic existence, independent of human thought. If ever we find an extraterrestrial civilization at all, they will know Hex, without any doubt.» " The rules are simple. Black and white fill an empty cell, in turn (Fig. 3). Black wins if it connects North and South, White wins if it connects West and East. The game is made more fair by a pie rule: at the second move, the second player can decide to swap colors. The game is hard because, as a connection game, its reward is based on a global criterion (no local criterion to sum). Fig. 3 shows the first win against a top level human.

3.2. TAAI competition

At TAAI 2019, Polygames was ranked first in Othello10 (3 programs), Breakthrough (4 programs) and Connect6 (2 programs)[4]. For more statistical significance, it was also successfully tested against

- WZebra and Ltbel (Othello8),
- the winner of TAAI 2018 at Connect6, namely Kavalan,

and won all games (4 games against Ltbel, 4 games against WZebra 15 games against Kavalan). We show in Fig. 5 one of the games won against Kavalan at Connect6.

3.3. Havannah

Havannah was invented specifically for being hard for computers (Fig. 6). It follows the game play of Hex, also with hexagonal cells but now on an hexagonal board, and winning conditions are more diverse:

- Connecting two of the six corners wins (15 possibilities);

[4] https://www.tcga.tw/taai2019/en/

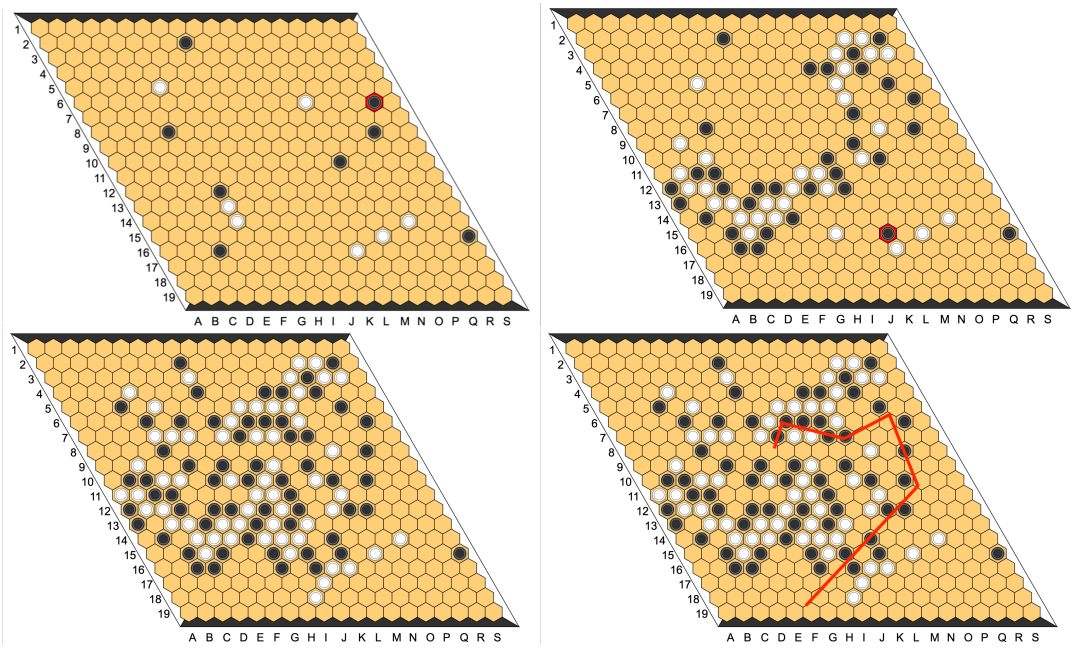


Fig. 3. Game of Hex 19x19 with Pie rule played by Polygames against Arek Kulczycki, Winner of the last LG tournament, ranked first for the ELO ranking (ELO 2248) on the LittleGolem server. First: opening. Second: at that stage, the human (White) seems to win - two solid groups are connected to East and West respectively, and look close to connect each other. However (third), the situation in the center is quite complicated and later it turns out that Black can win by one of two possible paths (last: White can not block both H7 and an attack on the left side). Source: LittleGolem. Our bot played multiple games since this win and never lost one, including games against Galvanise-Zero (Fig. 4). We use a single Quadro GP100, 5 minutes per move.

Game	Opponent	Rating	Tournament	Moves	Result
#2157792	thepatzer	1259	hex19.ch.24.2.2 Size 19	1	win
#2157791	Wickedestjr	1563	hex19.ch.24.2.2 Size 19	12	
#2157790	piotr	1558	hex19.ch.24.2.2 Size 19	39	win
#2157789	Aleksander Siatecki	1626	hex19.ch.24.2.2 Size 19	0	win
#2157784	ypercube	1773	hex19.ch.24.2.2 Size 19	3	win
#2157779	Nathan F Miller	1798	hex19.ch.24.2.2 Size 19	30	
#2157773	Bernhard Herwig	1808	hex19.ch.24.2.2 Size 19	13	
#2157766	Marius Halsor	1822	hex19.ch.24.2.2 Size 19	62	
#2155761	Ocross	1835	hex19.in.DEFAULT.172 Size 19	121	win
#2152151	piotr	1558	hex19.in.DEFAULT.171 Size 19	145	win
#2148744	Nagy Fathy ★	1701	hex19.in.DEFAULT.170 Size 19	81	win
#2143744	lguser	1303	hex19.in.DEFAULT.169 Size 19	12	win
#2139789	ferrarifeleches	1394	hex19.in.DEFAULT.168 Size 19	86	win
#2134882	German_Rodriguez_Perez	1387	hex19.in.DEFAULT.167 Size 19	1	win
#2133114	zastrzyk	1369	hex19.in.DEFAULT.166 Size 19	36	win
#2131362	Ganelon	1460	Hex 19x19 Size 19	59	win
#2131310	smktzw	1513	hex19.in.DEFAULT.165 Size 19	29	win
#2130503	gzero_bot	1841	Hex 19x19 Size 19	128	win
#2130344	gzero_bot	1841	Hex 19x19 Size 19	83	win
#2130098	gzero_bot	1841	Hex 19x19 Size 19	175	win
#2129789	Arek Kulczycki	2248	Hex 19x19 Size 19	119	win

Fig. 4. Results of our bot Mootwo since its game in Fig. 3.

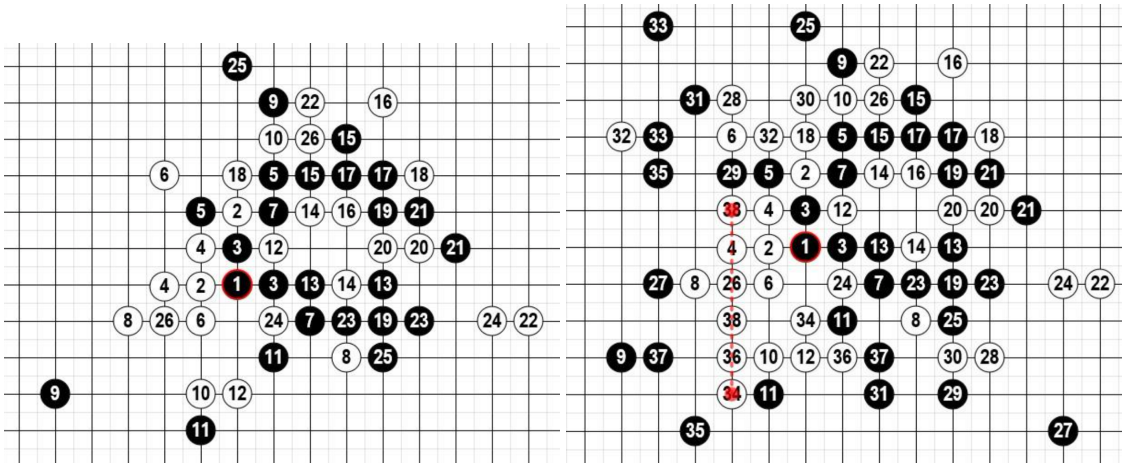


Fig. 5. Game won by Polygames against Kavalan: move 26 (left), which made the situation excellent for Polygames, and final position (right). Polygames won 15 games out of 15.

- Connecting three of the six sides wins (20 possibilities);
- Realizing a loop (even if it does not contain empty cells) wins.

According to (Lorentz, 2015), "The state of Havannah programming is still in its early stages. Though the top programs do play at a reasonable level, about the level of somebody who has played the game for 6 months or a year, they still play with a very unnatural style, and often win their games by virtue of tactical shots missed by the human opponent. Our feeling is that Havannah programs cannot be expected to play at an elite level until they learn to play a more natural, human-like game."

Draws are theoretically possible but very rare. The game is also played with a pie rule. Some decent players have been defeated by computers, but never the best humans until Polygames. On Littlegolem, an early version won 3 games out of 4 against *Mirko Rahn* (Elo rank 2133), and 2 games out of 2 against *tony* (Elo rank 2167), who belong to the top four players on this website (see Figure 6 middle and right for examples of games). Then, a new version was trained and won two games out of two against *Eobllor* (Elo rank 2415): both games were quite short, with excellent opening moves by Polygames (Fig. 7). This bot, our strongest version, was using a single Quadro GP100, 40 minutes per move with a shared GPU (used simultaneously 50% for other applications), whereas the human could use a lot of times as by default on LittleGolem (there were several days per move, using vacation days and the 10 days main time + 24 hours per move - our bot was slowed down as requested by Eobllor just for reducing the overall speed but we did not use the additional thinking time).

4. CONCLUSIONS

We propose a state-of-the-art framework, called Polygames, that can play to various games. It is based on zero learning, with innovations detailed in Section 2, and had success stories detailed in Section 3. Polygames contains new architectures, allows architecture search thanks to neutral components addition and is stabilized by a tournament mode and an overfitting fighting method. It was widely tested in the TAAI competition and on LittleGolem (www.littlegolem.net). The source code is publicly available under an open-source license. Our plan includes ablation studies and experimenting some of the innovations - e.g. the one-player case, including partial observability as in Section 2.4.4.

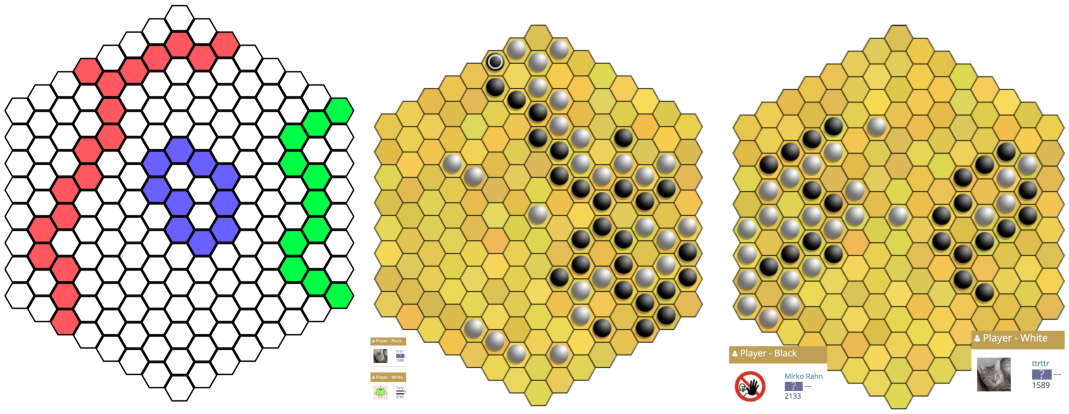
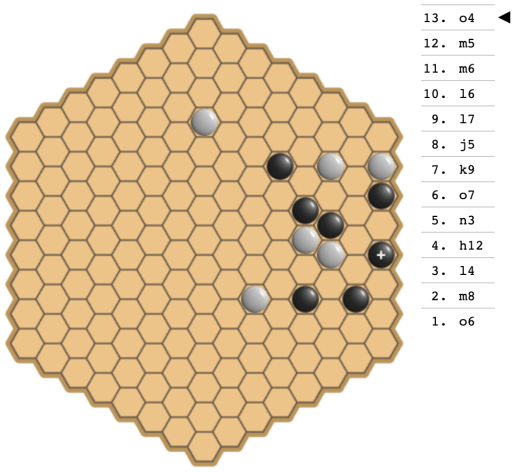
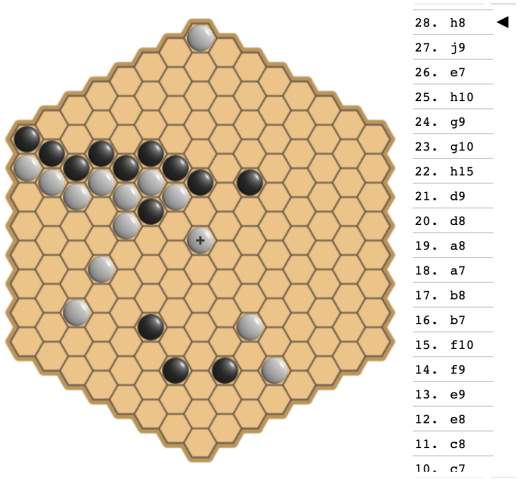


Fig. 6. Left: The game of Havannah and the three different ways for winning. Middle: a win by Polygames against Tony, Elo rank 2167. Right: a win against Mirko Rahn, Elo rank 2133, winner of many Havannah tournaments. Sources: wikipedia (left) and LittleGolem (middle and right).



Polygames plays Black. Last move (white cross) excellent move according to human analysis. http://www.trmph.com/diagram/1,havannah,8,o6m8l4h12n3o7k9j5l7l6m6m5o4,-1



Polygames blocked a bridge at the top, and has a double threat cycle (close to the center, left) + strong fork structure. http://www.trmph.com/havannah/game/lg-2153398

Fig. 7. The two games won against Eobllor (Elo rank 2415 before starting to play against Polygames), who dominates most competitions in Havannah. *Eobllor* won most games against the previous version of Polygames (Fig. 6), so we retrained it and got the present results. We note that the 3 fastest games lost by Eobllor, who rarely loses a game, are these 2 games and the game he lost against the previous version.

5. ACKNOWLEDGEMENTS

We are grateful to Wikipedia and LittleGolem as image sources and platforms for running experiments. We are grateful to human players who accepted to play with our bot, in particular Eobllor who beat successive preliminary versions of our bot. This work was supported in part by MOST under contracts 109-2634-F-259-001-through Pervasive Artificial Intelligence Research (PAIR) Labs, Taiwan. Tristan Cazenave is supported by the PRAIRIE institute.

REFERENCES

Auger, D. & Teytaud, O. (2012). The Frontier of Decidability in Partially Observable Recursive Games. *International Journal of Foundations of Computer Science*, *23*(7), 1439–1450. revised 2011, accepted 2011, in press. https://hal.inria.fr/hal-00710073.

Bonnet, É., Jamain, F. & Saffidine, A. (2016). On the complexity of connection games. *Theor. Comput. Sci.*, *644*, 2–28. doi:10.1016/j.tcs.2016.06.033.

Buffet, O., Lee, C.-S., Lin, W. & Teytaud, O. (2012). Optimistic Heuristics for MineSweeper. In *International Computer Symposium*. Hualien, Taiwan. https://hal.inria.fr/hal-00750577.

Coulom, R. (2007). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings of the 5th International Conference on Computers and Games*. CG'06 (pp. 72–83). Berlin, Heidelberg: Springer-Verlag.

Emslie, R. (2019). Galvanise Zero. https://github.com/richemslie/galvanise_zero.

He, K., Zhang, X., Ren, S. & Sun, J. (2015). Deep Residual Learning for Image Recognition. *CoRR*, *abs/1512.03385*. http://arxiv.org/abs/1512.03385.

Johansson, H. (2011). Self-learning Robots using Evolutionary and Genetic Algorithms.

Kocsis, L. & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006* (pp. 282–293). Springer.

Lorentz, R. (2015). Early Playout Termination in MCTS. In A. Plaat, J. van den Herik and W. Kosters (Eds.), *Advances in Computer Games* (pp. 12–19). Cham: Springer International Publishing.

Love, N., Hinrichs, T. & Genesereth, M. (2006). General game playing: Game description language specification.

Marcus, G. (2018). Innateness, AlphaZero, and Artificial Intelligence. *CoRR*, *abs/1801.05667*. http://arxiv.org/abs/1801.05667.

McCloskey, M. & Cohen, N.J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In G.H. Bower (Ed.), *Psychology of Learning and Motivation* (Vol. 24, pp. 109–165). Academic Press. doi:https://doi.org/10.1016/S0079-7421(08)60536-8. http://www.sciencedirect.com/science/article/pii/S0079742108605368.

Pascutto, G.-C. (2017). Leela Zero. https://github.com/leela-zero/leela-zero.

Pitrat, J. (1968). Realization of a general game-playing program. In *Information Processing, Proceedings of IFIP Congress 1968, Edinburgh, UK, 5-10 August 1968, Volume 2 - Hardware, Applications* (pp. 1570–1574).

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W.M. Wells and A.F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (pp. 234–241). Cham.

Shelhamer, E., Long, J. & Darrell, T. (2017). Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 640–651.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, *529*(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K. & Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR*, *abs/1712.01815*. http://arxiv.org/abs/1712.01815.

Tian, Y., Jerry Ma*, Qucheng Gong*, Shubho Sengupta*, Chen, Z., Pinkerton, J. & Zitnick, C.L. (2019). ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero. *CoRR*, *abs/1902.04522*. http://arxiv.org/abs/1902.04522.

van Rijswijck, J. (2006). Set colouring games. *Ph.D. thesis, University of Alberta*.

Wu, D.J. (2019). Accelerating Self-Play Learning in Go. *CoRR*, *abs/1902.10565*. http://arxiv.org/abs/1902.10565.