ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230563

A Convolutional Neural Network Approach to General Game Playing

Yu Wang[1], Heng Zhang[2;*] and Guifei Jiang[3,4,5]

[1]College of Intelligence and Computing, Tianjin University, Tianjin, China
[2]Zhejiang Lab, Hangzhou, China
[3]College of Software, Nankai University, Tianjin, China
[4]Haihe Laboratory of Information Technology Application Innovation (HL-IT), Tianjin, China
[5]Tianjin Key Laboratory of Operating System, Tianjin, China

Abstract. General Game Playing (GGP), a research field aimed at developing agents that master different games in a unified way, is regarded as a necessary step towards creating artificial general intelligence. With the success of deep reinforcement learning (DRL) in games like Go, chess, and shogi, it has been recently introduced to GGP and is regarded as a promising technique to achieve the goal of GGP. However, the current work uses fully connected neural networks and is thus unable to efficiently exploit the topological structure of game states. In this paper, we propose an approach to applying general-purposed convolutional neural networks to GGP and implement a DRL-based GGP player. Experiments indicate that the built player not only outperforms the previous algorithm and UCT benchmark in a variety of games but also requires less training time.

1 Introduction

The ability to play games is usually regarded as a significant indicator of the intelligence level of an agent. Over the past few decades, the application of artificial intelligence in traditional game playing has made significant advances [1, 2, 3]. However, it is difficult for computers to generalize abilities from one game to others. The current techniques seem to be far away from the final solution of realizing Artificial General Intelligence (AGI). General Game Playing (GGP) was launched by Genesereth *et al.* in 2005 to create intelligent agents that can perform well in a variety of games without human intervention [4]. With decades of development, GGP has been regarded as a necessary milestone for AGI [5].

As an improved version of AlphaGo, DeepMind developed another program, called AlphaZero, to master games of chess, shogi and Go [6]. It can achieve superhuman performance in all three games. However, AlphaZero can only play two-player zero-sum board games, and the architecture of its neural networks, as well as the encoding method of the input and output, is manually designed according to the specific game rules.

To address these issues, Goldwaser and Thielscher extended AlphaZero to GGP and named it Generalized AlphaZero (GAZ) [7]. It removes the limitation on game type and can automatically design the neural networks according to the number of game propositions [8]. It is the first deep reinforcement learning algorithm that outperforms the UCT benchmark in all games [7, 9, 10], except in multiplayer cooperative games like Babel. However, unlike AlphaZero, GAZ employs fully connected networks instead of convolutional neural networks (CNNs). This is due to two main reasons: One is the challenge of extracting the necessary ordering knowledge for CNNs from Game Description Language (GDL). Another reason is that the existing CNNs can only process input tensors with known, certain dimension, while GGP games are unknown, which makes them inapplicable to GAZ.

It is important for the agent's network structure to align with the input structure to enable efficient feature extraction. Fully connected networks simply ignore the input structure, causing inefficiencies. By using topological information from game states, new agent built by generalized CNNs is believed to speed up the feature extraction process and thus reduce training time. Based on the above considerations, this paper implements a new DRL-based player for GGP. Our main contributions are summarized as follows:

- CNNs have been generalized from 2D to arbitrary dimensions, allowing them to automatically perform convolutions of different dimensions based on the input.
- An extended version of AlphaZero for GGP has been implemented, using the generalized CNNs. Only base-type propositions representing game states in the propositional network are used as the input of the neural networks, which is less than half the number used in GAZ. More importantly, given the game rules, the agent's network structure can be adjusted automatically.
- A number of experiments have been carried out to demonstrate the effectiveness of our approach. Experimental results show that the proposed agent outperforms UCT benchmark and GAZ in multiple games and is more efficient in training. Especially in game Babel where GAZ failed, it successfully outperforms the UCT with a perfect score.

The structure of the rest paper is organized as follows: Section 2 presents the previous DRL-based research for GGP and the background of CNNs. Section 3 introduces our proposed approach, and Section 4 describes experiments and results. Finally, we conclude this paper with a discussion of future work.

* Corresponding Author. Email: h.zhang@zhejianglab.com.

2 Background

2.1 GGP

In GGP, the game rules are written in the form of standard Game Description Language (GDL) [11]. GDL is a logic-based language, where a game state is described by specifying which propositions are true in that state.

To speed up game reasoning, a GDL-defined game can be converted into a propositional network (PropNet) [8]. PropNet can be seen as a graph representation of GDL, where each node represents a proposition or a logic gate. All propositions can be divided into three types: input propositions, base propositions, and view propositions [12]. The truth values of base propositions represent the state of the game, and the values of input propositions are set when the agent chooses the action to play. The rest are all view propositions, including the description of the goal, terminal condition, legal actions, etc.

In particular, the proposed agent in this work uses the optimized PropNet implemented by [13] with a method of the *OptimizingPropNetFactory* class[1] provided in the GGP-Base framework.

2.2 DRL for GGP

Deep reinforcement learning (DRL) is a subfield of machine learning that combines reinforcement learning (RL) and deep learning. It has had many successful applications in games [3, 6, 14, 15, 16, 17, 18] and has been recently introduced into GGP [7, 9, 19, 20].

Initial study attempting to apply Q-learning (a RL algorithm for learning the value of an action in a particular state) to GGP revealed that it can eventually converge, but very slowly [20]. When combined with Monte Carlo Tree Search (MCTS) [21], a fundamental and widely used algorithm in GGP, it still failed to beat UCT benchmark. In 2020, GAZ was introduced as a DRL-based agent for GGP that learns through self-play [7]. It is the first successful DRL-based agent to surpass the UCT benchmark in most games. We will describe GAZ in details later. The same year, GGPZero [9], which features a structure similar to GAZ but a different game reasoner, was released. This work analyzed various factors affecting agent performance, but its performance was inferior to GAZ.

The neural networks in the above three works only employed fully connected layers. Most recently, based on another GGP system, Ludii [22], and DRL framework, polygames [23], CNNs were employed in GGP [10]. It achieves very good performance on 15 board games, but is still limited to 2D, two-player, asynchronous games.

2.3 GAZ

GAZ [7] is the first deep reinforcement learning algorithm in GGP to outperform the UCT benchmark in most games except Babel [9, 19]. The main process of the algorithm is almost the same as in AlphaZero, as shown in Figure 1.

The neural networks are embedded in MCTS to guide simulations. The MCTS algorithm generates training data through self-play, which is added to the replay buffer. The constantly updated data in the replay buffer is then used to train the neural networks. Repeat the process until a high-quality policy is produced. The more accurate the network predictions are, the higher the quality of the data generated by MCTS, and conversely, the higher the data quality, the more accurate the trained networks will be.

[1] https://github.com/ggp-org/ggp-base/tree/master/src/main/java/org/ggp/base/util/propnet/factory

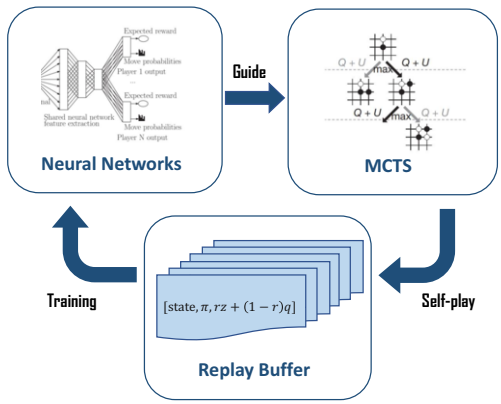

Figure 1: The main process of GAZ self-play training.

Neural networks are used to guide MCTS simulations. A simulation of MCTS can be divided into 4 steps: selection, expansion, evaluation, and backtracking. The notion of Monte Carlo tree is standard, see [24]. Given a state s and an action a, each edge (s, a) in the tree stores the data of network prior action probability $\boldsymbol{p}(s, a)$, visit counts $N(s, a)$, and action value $Q(s, a)$. In the selection phase, starting from the root node, continuously select the action a that maximizes the upper confidence bound until the leaf node is reached:

$$a = \text{argmax}_a \left(Q(s,a) + c_{puct} \frac{\boldsymbol{p}(s,a) \times \sqrt{N(s)}}{(1 + N(s,a))} \right) \tag{1}$$

where c_{puct} is a constant. After that, the leaf position s' will be passed through the neural network to obtain the prior action probability $\boldsymbol{p}(s')$ and the prior value $v(s')$. In the backtracking phase, the value $Q(s, a)$ of each action from the leaf node upward is determined by the mean of all the iterations that went through the node. That is, $Q(s,a) = \frac{1}{N} \sum_{s'|s,a \to s'} v(s')$.

The action probability $\boldsymbol{\pi}$ actually taken in self-play is determined by the visit counts of all actions at the root state, according to the results of 300 MCTS simulations. That is, $\boldsymbol{\pi}_a \propto N(s, a)$. At the end of the game, or when the game exceeds the maximum length, each role will be given a score z. Then the triples $(state, \boldsymbol{\pi}, rz+(1-r)q)$ will be added to the replay buffer, where r is a parameter with final score z, which is found to perform best when set to 0.5, and q is the Q value of each state after 300 MCTS simulations. The parameter r is introduced to take the results of the MCTS simulation into account, which is different from the original AlphaZero.

Regarding the structure of the neural network in GAZ, the network input is comprised of the values of all game propositions. All the inputs are passed through a series of fully connected layers, each one half the size of the previous one, and the final layer is of size 50. Then, starting with that layer as a common layer, each player's head doubles the size of the layer until it reaches the number of legal actions.

The following is how the networks were trained. For each given root state s, the neural network calculates its action probability estimate $\boldsymbol{p}$ and value estimate v, with current network parameter θ:

$$f_\theta(s) = (\boldsymbol{p}, v) \tag{2}$$

The purpose of network training is to make the output of the neural network $\boldsymbol{p}$, v approximate to $\boldsymbol{\pi}$ and $rz+(1-r)q$. So the loss function is defined as follows:

$$loss = ((rz + (1-r)q) - v)^2 - \boldsymbol{\pi}^T log\boldsymbol{p} + c \mid\mid \theta \mid\mid^2 \tag{3}$$

where c is a parameter controlling the level of L2 weight regularization (to prevent overfitting). When the game is over, the reward from GDL is between 0 and 100, and GAZ rescales it by dividing it by 100. This work also takes this setting.

As shown in Table 1, GAZ outperforms the UCT benchmark agent significantly in the two-player, zero-sum, turn-based games Connect-4 and Breakthrough, and is slightly better in the 3-player, mixed cooperative / zero-sum game Pacman 3p, but fails in the 3-player pure cooperative simultaneous game Babel.

2.4 CNNs

2D CNN has been commonly employed in feature detection and creation [25, 26, 27, 28]. 3D CNN was proposed to conduct human action recognition [29]. The main difference between 2D and 3D CNN is the spatial dimension of the filter sliding. 2D filters slide on the height and width of the input layer, while 3D filters on dim depth, height and width. Each time the filter slides a position, a convolution operation is performed to obtain a value. As the filter slides over the entire 3D space, the output structure is also 3D.

Note that convolution and Convolutional Neural Networks are not interchangeable concepts. While multi-dimensional convolution has been extensively employed in signal processing, image processing, and computational mathematics [30, 31], its utilization in neural networks has been limited to no more than 3D applications.

In 2020, a novel approach known as 4D CNNs (V4D) [32] was proposed to extract timeline information from long videos by designing a specialized 3D convolution kernel. Since it did not use a complete cubic block, the number of parameters for the convolution kernel was greatly reduced. However, the approach to V4D is still limited to 4D case, and the implementation process is actually done manually using 3D kernels.

3 Method

Since the fully connected layers have no topological information, GAZ is unable to exploit the topological information of propositions representing states. However, if the digits representing the position information in the description of the proposition can be extracted, the proposition can be reasonably arranged to pass through the CNNs. At the same time, using the extracted topology information, the agent can automatically adjust the parameters of the network according to different games, making the network structure more flexible and reasonable.

3.1 Multi-Dimensional Convolutional Layer

Existing convolutional networks are limited to processing tensors of certain dimensions, typically no more than 3. However, GGP lacks advance knowledge about the types of games it will encounter. Thus, to enable the use of CNNs in GGP, we first need to implement general and unified CNNs, which could be performed on tensors of any unknown dimension. The implementation process is given as follows:

Decide the shape of the input tensor. Performing an n-D convolution requires a given $n+2$ dimensional input tensor. Here we denote it as $[B, I_n, I_{n-1}, ..., I_1, C]$, where $I_k, k = 1, ..., n$, represents the length of the k-dimension of the input. The two extra dimensions B and C respectively represent the batch size and the number of input channels.

Algorithm 1 nD CNN operated by matrix calculation

Require: input tensor $\boldsymbol{x}$, number of filters F, kernel size $Ker = 3$, stride $S = 1$

1: Let $[B, I_n, I_{n-1}, ..., I_1, C] = \boldsymbol{x}.shape$

2: Padding 0 at the start and end of n dimensions of the input tensor for Pad number:

$$Pad = \left\lfloor \frac{Ker}{2} \right\rfloor \tag{4}$$

3: Create filters variable w of shape:

$$w\_shape = [Ker] \times n + [C] + [F] \tag{5}$$

4: Create bias variable b in the shape of $[1, F]$

5: Compute output shape $[B, O_n, O_{n-1}, ..., O_1, F]$, where

$$O_i = \left\lfloor \frac{(I_i + 2 \times Pad - Ker)}{S} + 1 \right\rfloor \tag{6}$$

6: Flatten the w tensor into a 2D matrix M_{filter} of shape $[Ker^n \times C, F]$

7: Extract patches from the padded input tensor $input'$, then form a 2D matrix M_{input} of shape:

$$[B \times O_n \times O_{n-1} \times ... \times O_1, Ker^n \times C] \tag{7}$$

8: Compute the output matrix:

$$M_{output} = ReLU(M_{input'} \times M_{filter} + b) \tag{8}$$

the shape of M_{output} is $[B \times O_n \times O_{n-1} \times ... \times O_1, F]$

9: Reshape the M_{output} into the tensor $output$ of shape $[B, O_n, O_{n-1}, ..., O_1, F]$

10: **return** $output$

The number of filters is represented by the parameter F. It is also the number of output channels. The kernel size of each filter is represented by the parameter Ker. If not given, the default is 3. To make the shape of the kernel automatically modified with dimension n, the length of the convolution kernel is the same in each dimension. For example, the kernel size in 2D convolution is $[3, 3]$, and in 3D it is $[3, 3, 3]$.

The stride is represented by the parameter S. If not given, the default is 1. The mode 'SAME' is used to pad the input tensor. This avoids the issue where the number of convolutional layers is affected by the size of the game board, leading to an unreasonable network structure.

The convolution calculation is actually taking the dot product with the kernel and the input patch. This can be accelerated by converting to matrix multiplication. Firstly, we need to expand each patch region of the convolutional part of the input image into a row vector. These row vectors together form a two-dimensional matrix in the sequence in which the convolution kernel slides. Then reshape the convolution kernel into a 2D matrix, and multiply the input matrix and the convolution kernel matrix to get the output matrix. Finally, reshape the output matrix into the desired size tensor. The detailed process is shown in Algorithm 1.

It is worth mentioning that, although this algorithm provide a unified convolutional neural network layer applicable for any dimension, it is still limited in practical use, due to the high-dimensional convolution kernel that leads to a sharp increase in the number of parameters to be trained. For higher-dimensional cases, we only ap-

Table 1: Performance of GAZ

Game	Game type	Performance against/compared to UCT		Evaluation
Connect-4	2-player, zero-sum, turn-based	70% winning rate		outperform significantly
Breakthrough	2-player, zero-sum, turn-based	100% winning rate		outperform significantly
Pacman 3p	3-player, mixed co-op, simultaneous	4.4 / 25 points	3.2 / 25 points	slightly better
Babel	3-player, cooperative, simultaneous	12.5 / 25 points	19 / 25 points	worse

ply this algorithm to TicTacToe 3d. For a $3 \times 3 \times 3$ chessboard, this algorithm is also feasible and effective.

3.2 Overall Framework

The framework of the proposed agent is roughly the same as GAZ. The main differences are in the input and the structure of the network, as shown in Figure 2. This agent first automatically extracts the base-type propositions from all nodes of the proposition network. They are then classified into two classes: CNN class and FC class, using the sequential information extracted from GDL. Propsitions in the CNN class go through a series of convolutional layers, and are then flattened into a one-dimensional vector. The propositions of the FC class are jointly input to the downstream fully connected layers.

It should be noted that the proposed agent is a general game player, as it can automatically set relevant parameters in the network without human intervention, according to the rules of different games, such as the number of filters in the convolution layer, the number of hidden layers in the MLP layer, etc. For readability, we illustrate the agent with the game TicTacToe.

3.3 Input of the Network

In contrast to GAZ, which uses all types of propositions in PropNet, the proposed agent only utilizes base-type propositions, resulting in a significant reduction in network input. The consideration behind this is that an optimal strategy is only related to the current state, and the existence of unimportant propositions could make the network more complex and interfere with learning. Moreover, a comparison of the number of base-type and all-type propositions in different games is shown in Figure 3.

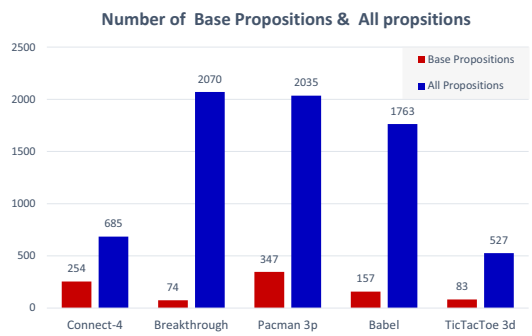

Figure 3: The number of inputs to our agent compared with the GAZ. Only the base propositions are used in our agent.

3.4 Exploiting Topology Information in GDL

For classification and ordering, the proposed agent firstly uses regular expressions to obtain the index information in GDL for all propositions. We illustrate this process with the game TicTacToe.

Table 2 lists each basic proposition id and its GDL description with a total of 29 items. The GDL item true (cell 1 1 b) says that in the current state, the cell with index (1, 1) is marked 'b'. Item true (control oplayer) means the current state is the turn of player 'o'. We next show how to automatically classify and rank these items.

First classify the items. Using the regex:

$$(? <= /s)/d + (? = /s) \tag{9}$$

to select all numbers in the items and then delete them, if the rest are exactly the same, group the items together. If a base proposition includes two or more digits, it is classified as the CNN class. Otherwise, a base proposition with no digit or only one digit is classified as the FC class. It should be noted that the regular expression used here is identical for all games and doesn't require any game-specific knowledge, which complies with the GGP principle.

The number of categories in the CNN class corresponds to the number of channels of input tensor. It specifies the number of planes needed to describe the current board state. For example, in TicTacToe, the 29 base propositions are automatically divided into 5 categories:

1. true (cell b)
2. true (cell o)
3. true (cell x)
4. true (control oplayer)
5. true (control xplayer)

A total of 27 base propositions of the first three categories are used as the CNN class, and the base propositions of the last two are used as the FC network part. The three categories input to the CNN network part represent the feature planes with states b,o,x respectively.

Then, sort the base propositions in the CNN class. The index of each base proposition in the plane corresponds to the digits in GDL, as extracted via regular expressions. Specifically, the two digits act as two-dimensional indexes, representing height and width. The three digits act as three-dimensional indices, representing depth, height, and width. Considering that some game indexes start from 0, adjust the index corresponding to each proposition accordingly. In this way, sort all base proposition ids according to the obtained index.

Finally, input the values of all the sorted propositions into the network. The base proposition in the FC class can be input into the FC network layer in any fixed order.

3.5 Network Structure

This section explains how the proposed agent automatically design convolution kernels and specific parameters in the network based on the topology information obtained.

For the convolutional layer, we fix the side length of the convolution kernel to 3 (the kernel size in 2D is 3×3, in 3D is $3 \times 3 \times 3$, ...). Three convolutional layers are used. The number of convolution kernels in the latter layer is twice of that in the previous layer. The number of filters in first layer F_1 is determined by the size of the board and the number of input channels of the CNN, which is equal to the power of 2 closest to $\max_i(Dim_{(i)}) \times channels$, for integer

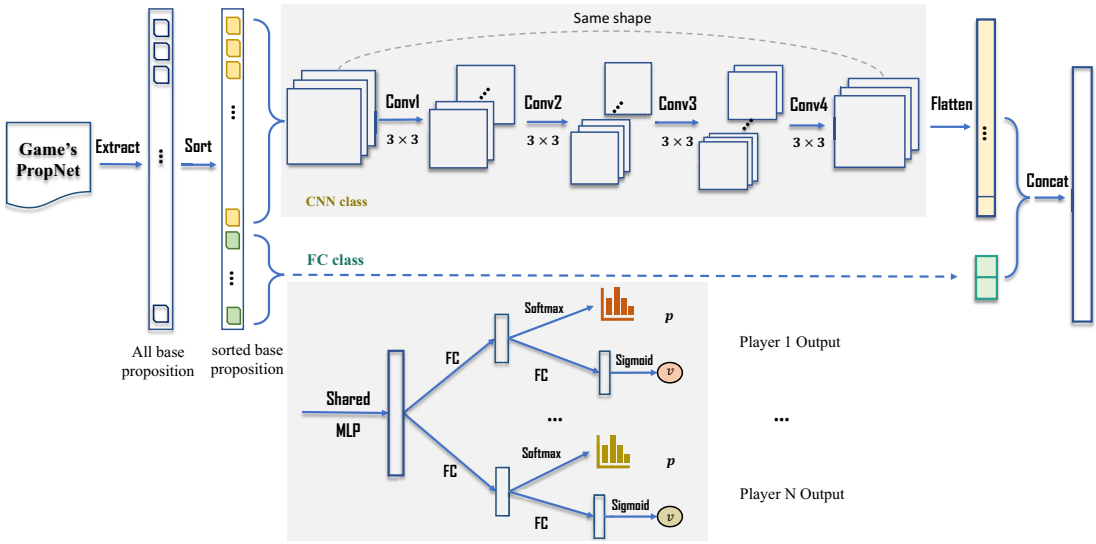

Figure 2: The overall framework of our agent.

Table 2: The ids and GDL of all base-type propositions in game TicTacToe

246 (true (cell 1 1 b))	172 (true (cell 1 1 o))	258 (true (cell 1 1 x))
158 (true (cell 1 2 b))	242 (true (cell 1 2 o))	180 (true (cell 1 2 x))
155 (true (cell 1 3 b))	35 (true (cell 1 3 o))	263 (true (cell 1 3 x))
234 (true (cell 2 1 b))	245 (true (cell 2 1 o))	145 (true (cell 2 1 x))
59 (true (cell 2 2 b))	210 (true (cell 2 2 o))	197 (true (cell 2 2 x))
300 (true (cell 2 3 b))	182 (true (cell 2 3 o))	225 (true (cell 2 3 x))
274 (true (cell 3 1 b))	232 (true (cell 3 1 o))	214 (true (cell 3 1 x))
13 (true (cell 3 2 b))	276 (true (cell 3 2 o))	127 (true (cell 3 2 x))
207 (true (cell 3 3 b))	209 (true (cell 3 3 o))	46 (true (cell 3 3 x))
108 (true (control oplayer))	265 (true (control xplayer))	

k. Formally:

$$F_1 = 2^k \approx \max_i(Dim_{(i)}) \times channels, \quad k \in Z_+ \tag{10}$$

where $channels$ means the number of planes representing the state features. $Dim_{(i)}$ represents the length of the i th dim. Tabel 3 shows the input shape and the number of filters of the three convolutional layers for testing games. It should be noted that the agent generates calculations for all data in the table automatically. To make the output shape the same as the input, the number of filters used for Conv4 in Figure 2 is set to $channels$.

Table 3: CNNs structures designed by the agent for testing games.

	Input shape	Filters num for 3 layers		
Connect-4	$7 \times 9 \times 4$	32	64	128
Breakthrough	$6 \times 6 \times 2$	16	32	64
Pacman 3p	$8 \times 8 \times 4$	32	64	128
Babel	$8 \times 17 \times 1$	16	32	64
ConnectFour 3p	$6 \times 8 \times 4$	32	64	128
TicTacToe 3d	$3 \times 3 \times 3 \times 3$	8	16	32

The MLP layer is designed as follows: it is first mapped to a fully connected layer of the same size, then the nodes of fully connected layers are halfed each time, until it falls below the size of the game board: $boardsize = \prod_i Dim_{(i)}$.

The current layer is stored as the common head state for all players. Then starting from the head state, we build a fully connected layer for each role to estimate the action probability p, and two fully connected layers to estimate the value v, where the nodes of the hidden layer is $boardsize$. All network layers use $ReLU$ activation, the final output action probability used is $Softmax$, and the output value used is $Sigmoid$.

4 Experiments

4.1 Evaluation Method

We evaluate the our agent's performance by playing against the UCT benchmark, whose variants have been state-of-the-art over many years [33, 34, 35, 36], and compare the result with GAZ. Moreover, we also conduct a direct comparison of our agent with GAZ. We do not compare with agents built on other systems, such as Ludii [10]. On the one hand, cross-platform competitions are not supported by the current system. On the other hand, the degree of code optimization varies between systems, so a direct comparison is not fair.

Four different types of games were considered in GAZ: Connect-4, Breakthrough, Pacman 3p and Babel. We evaluate our agent in all the four games. In addition, to show the generality of our agent, we conduct experiments in the game ConnectFour 3p and TicTacToe 3d.

Each player has 1 second simulation time per step. The win rate for each game is calculated from 50 games (two players each go first 25 rounds). We do not set random seeds in all the experiments. Each network was trained 3 times, starting with a randomly initialized network for each game. The evaluated data represents the average of three training sessions. The experiments were all run on an Intel Core i7 running at 3.2GHz and used a NVIDIA GeForce GTX 1060 6GB graphics card[2].

[2] https://github.com/littleWangyu/CNN-Approach-to-GGP

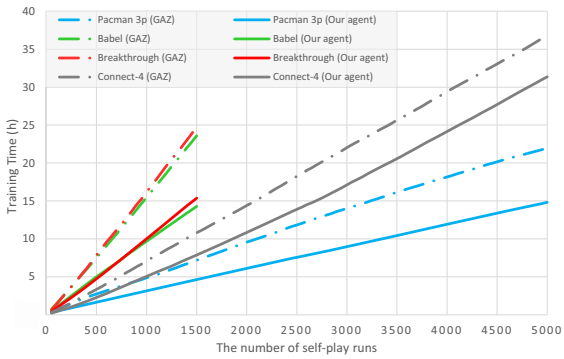

Figure 4: Comparison of training time between our agent and GAZ.

4.2 *Results and Analysis*

The training time of GAZ and our agent for different games is shown in Figure 4. The dotted line is the training time of GAZ, and the solid line of the same color represents ours. It can be seen that the training time of our agent is significantly reduced, since the fact that our neural network has a much smaller number of inputs.

It should be noted that, the training time requied in our agent is still more than the 10 minute startclock in GGP competitions, which is why our agent is currently unable to participate in the GGP competition. This could be accelerated by using multiple processes to collect data, or even parallelizing the MCTS algorithm [37]. In addition, it is actually natural and necessary to provide a general agent with enough time to learn a new game.

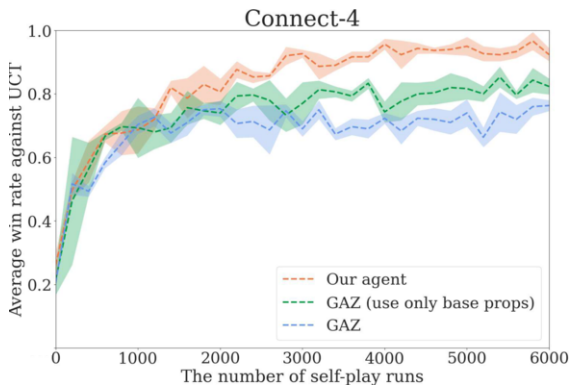

Figure 5: Win rate against UCT in Connect-4.

Furthermore, We conduct ablation experiments on game **Connect-4** to demonstrate the effect of using only base-type propositions as network input. As shown from Figure 5, GAZ that only uses the base-type proposition as the network input has increased the winning rate against UCT benchmark by nearly 10% compared with the previous one. And, with our CNNs, the winning rate can reach 95%.

We then evaluate the performance of our agent and GAZ against UCT on game **Breakthrough**. The experiment found that our agent and GAZ can beat UCT with a 100% winning rate after training. To further compare them, we relax the simulation time of the opponent's UCT to 4s per second, while our agent and GAZ are still 1s. The result is shown in Figure 6. As can be seen, when playing against the more powerful UCT, our agent can still achieve complete victory, while the GAZ winning rate is about 90%.

For DRL-based agents, under the same experimental conditions, it is more reasonable to compare two agents with the same training time than with the same amount of self-play. As a result, we also evaluate our agent against GAZ directly in the above two 2-player zero-sum games with a fixed training time limit.

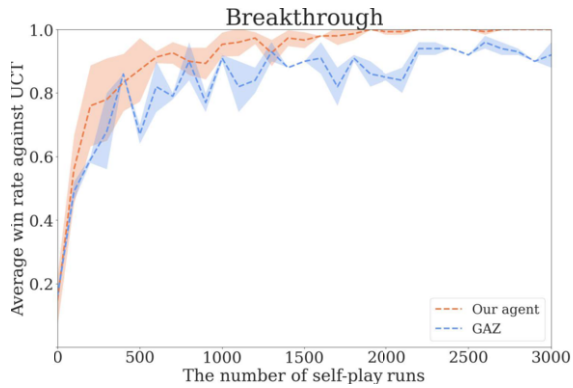

Figure 6: Win rate against UCT in Breakthrough.

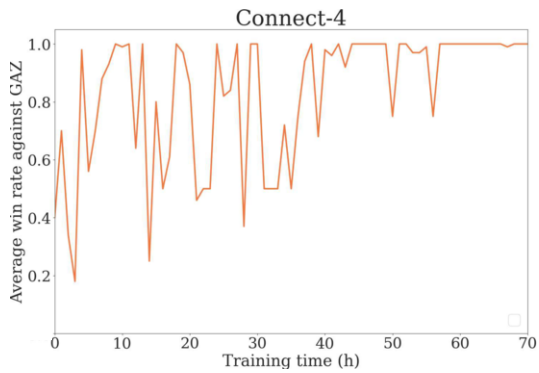

Figure 7: Win rate against GAZ under the same training time in Connect-4.

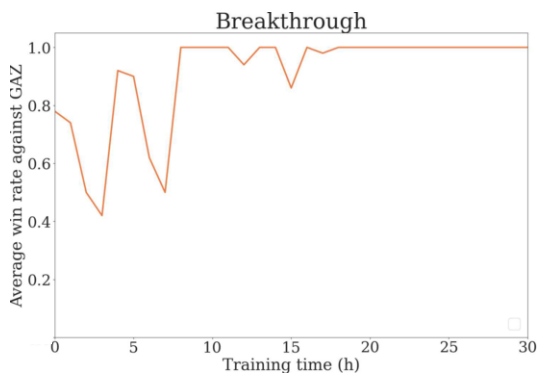

Figure 8: Win rate against GAZ under the same training time in Breakthrough.

Figure 7 and Figure 8 show the win rate of our agent directly against GAZ. After a certain training time, our agent can beat GAZ with a 100% win rate in both Connect-4 (after 60 hours) and Breakthrough (after 18 hours).

Pacman 3p[3] is a three-player, mixed cooperative / zero-sum, non-board game. One player controls Pacman and two other players, each controlling a ghost, collaborate against the Pacman. Experiments show that before being caught by two UCT-played ghosts, our

[3] http://games.ggp.org/base/games/pacman3p/pacman3p.kif

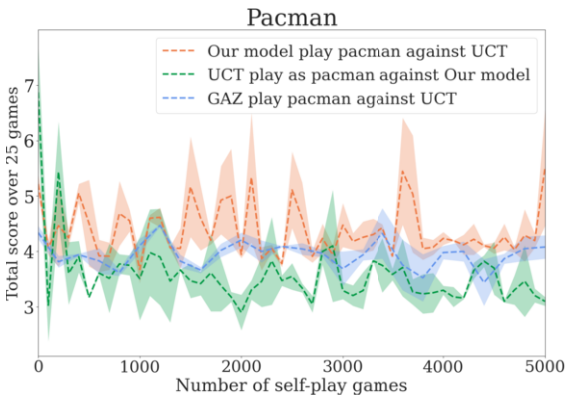

Figure 9: Total score over 25 games when playing as Pacman in Pacman 3p.

agent-played Pacman scored higher than GAZ. Figure 9 shows the Pacman's total score over 25 games. Since the ghosts caught Pacman on all runs and scored full points, the scores for the ghosts are not shown in the figure.

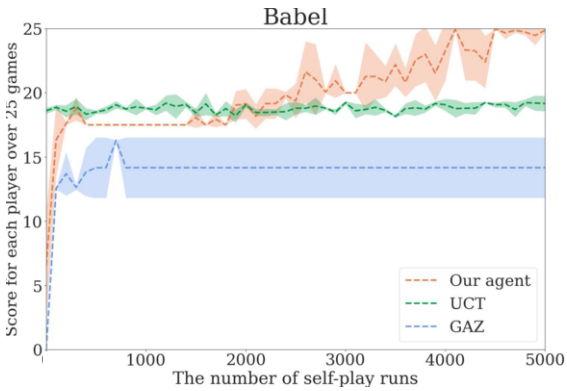

Figure 10: Score for each player over 25 games in Babel.

Babel[4] is a tower building game for 3 people. It is a pure cooperative, simultaneous, non-board game. Each player gets the same score (between [0, 1]) at the end of each game.

Figure 10 shows that our agent achieves a perfect score stably after 5000 rounds training, significantly outperforming UCT and GAZ. The game process, which is visualized in the appendix, shows that different players played by our agent have learnt to work together, so that they can get full marks, while those played by UCT and GAZ have not.

ConnectFour 3p[5] is a 3-player, assistant variant of the game Connect-4. The three roles are 'Red', 'Yellow' and 'Blue'. 'Red' is the 'Yellow' assistant, and 'Yellow' is the 'Blue' assistant. A line of four cells is worth 100 points to the role that fills them first, 50 points to the assistant, and 0 points to the failure.

We observe the player's performance through the score of 'Red', this is because 'Red' has no assistant and is the most disadvantaged side. Figure 11 shows the total score over 25 games when playing as 'Red'. According to experiments, our agent outperforms GAZ, while GAZ outperforms the UCT benchmark.

TicTacToe 3d[6] is a two-player simple game in 3D space. Both our agent and GAZ can quickly learn the strategy of being the first

[4] http://games.ggp.org/dresden/games/babel/rulesheet.kif
[5] http://games.ggp.org/base/games/3pConnectFour/3pConnectFour.kif
[6] http://games.ggp.org/base/games/tictactoe_3d_2player/tictactoe_3d_2player.kif

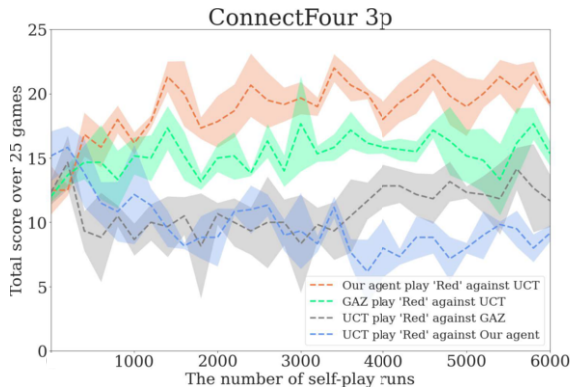

Figure 11: Total score over 25 games when playing as 'Red' in ConnectFour 3p.

to win. As a back-hand, they also have a strong counterattack ability when playing against the UCT benchmark. Moreover, when they play against each other directly, it is a draw.

5 Conclusion and Discussion

This paper has proposed a convolutional neural network approach to GGP, considering that fully connected networks in GAZ could not utilize the semantic and topology information present in game description language (GDL). We first generalize CNNs from two to arbitrary spatial dimensions, and then use them to implement an extended version of AlphaZero for GGP. Following that, using generic regular expressions, we extract the ordering and semantic information in GDL, allowing CNNs to be applied to GGP. The network structure of the proposed agent can be automatically adjusted according to game rules, making it more adaptable and suitable for a wide range of games. Furthermore, only base propositions representing the current game state in the propositional network are used as neural network inputs, greatly accelerating network training. Experimental results indicate that the proposed agent outperforms the UCT benchmark and GAZ in all games tested, with a more efficient training speed.

Multi-dimensional convolution has been widely used in the fields of signal processing, image processing, and computational mathematics [30, 31], but before our work, only 2D and 3D convolutions had been used in neural networks. To the best of our knowledge, this work is the first one which uses high-dimensional convolution in neural networks.

There is much work to be done in the future. Firstly, the proposed agent could be improved with further fine-tuning, e.g., adjusting the number of filters, network layers, and simulations in the MCTS during training. Secondly, the proposed agent is currently limited to processing GDLs that represent location information numerically. For games such as chess that use a letter and a number to represent a cell, the game rules need to be manually rewritten first. However, this could be addressed by identifying the succession relationship in GDL, for example, '(succ a b)', as described in [38]. Thirdly, while the agent is no longer limited to playing 2D, two-player, asynchronous board games, it is still constrained to games with a fixed plane size and a regular, grid-like structure. Further work can be done to remove this restriction and allow the agent to play games with more flexible structures. Lastly, exploring other scenarios where n-D CNNs could be applied could also be an interesting avenue for future research.

Acknowledgements

We would like to thank anonymous referees for their helpful comments and suggestions. This work was supported in part by Key Research Project of Zhejiang Lab (No. K2022PD1BB01) and Soft Research Project of Zhejiang Lab (No. K2023PDAK01).

References

[1] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[2] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook the world man-machine checkers champion. *Ai Magazine*, 17(1):21–21, 1996.

[3] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[4] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62–62, 2005.

[5] Rodrigo Canaan, Christoph Salge, Julian Togelius, and Andy Nealen. Leveling the playing field: Fairness in ai versus human game benchmarks. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19, New York, NY, USA, 2019. Association for Computing Machinery.

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

[7] Adrian Goldwaser and Michael Thielscher. Deep reinforcement learning for general game playing. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1701–1708, 2020.

[8] Evan Cox, Eric Schkufza, Ryan Madsen, and Michael Genesereth. Factoring general games using propositional automata. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, pages 13–20, 2009.

[9] Alvaro Gunawan, Ji Ruan, Michael Thielscher, and Ajit Narayanan. Exploring a learning architecture for general game playing. In *Australasian Joint Conference on Artificial Intelligence*, pages 294–306. Springer, 2020.

[10] Dennis JNJ Soemers, Vegard Mella, Cameron Browne, and Olivier Teytaud. Deep learning for general game playing with ludii and polygames. *arXiv preprint arXiv:2101.09562*, 2021.

[11] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. 2008.

[12] M. Genesereth and M. Thielscher. *General Game Playing*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2014.

[13] Chiara F Sironi and Mark HM Winands. Optimizing propositional networks. In *Computer Games*, pages 133–151. Springer, 2016.

[14] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[15] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Andrei A Rusu, Veness Joel, Marc G Bellemare, Graves Alex, Riedmiller Martin, Andreas K Fidjeland, and Ostrovski and Georg. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–33, 2015.

[16] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[17] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2140–2146. AAAI Press, 2017.

[18] Matej Moravč ík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, may 2017.

[19] Sili Liang, Guifei Jiang, and Yuzhi Zhang. Combining M-MCTS and deep reinforcement learning for general game playing. In Jie Chen, Jérôme Lang, Christopher Amato, and Dengji Zhao, editors, *Distributed Artificial Intelligence - Third International Conference, DAI 2021, Shanghai, China, December 17-18, 2021, Proceedings*, volume 13170 of *Lecture Notes in Computer Science*, pages 221–234. Springer, 2021.

[20] Hui Wang, Michael Emmerich, and Aske Plaat. Monte carlo q-learning for general game playing, 2018.

[21] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 2012.

[22] Eric Piette, Dennis JNJ Soemers, Matthew Stephenson, Chiara F Sironi, Mark HM Winands, and Cameron Browne. Ludii–the ludemic general game system. *arXiv preprint arXiv:1905.05013*, 2019.

[23] 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. Polygames: Improved zero learning. *ICGA Journal*, 42(4):244–256, 2020.

[24] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[29] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[30] Chang Eun Kim and Michael G. Strintzis. High-speed multidimensional convolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(3):269–273, 1980.

[31] M. V. Rakhuba and I. V. Oseledets. Fast multidimensional convolution in low-rank formats via cross approximation, 2014.

[32] Shiwen Zhang, Sheng Guo, Weilin Huang, Matthew R. Scott, and Limin Wang. V4d:4d convolutional neural networks for video-level representation learning, 2020.

[33] Shiven Sharma, Ziad Kobti, and Scott Goodwin. Knowledge generation for improving simulations in uct for general game playing. In *Australasian Joint Conference on Artificial Intelligence*, pages 49–55. Springer, 2008.

[34] Yngvi Bjornsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):4–15, 2009.

[35] Karol Walędzik and Jacek Mańdziuk. An automatically generated evaluation function in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):258–270, 2013.

[36] Maciej Świechowski and Jacek Mańdziuk. Self-adaptation of playing strategies in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):367–381, 2013.

[37] Richard B Segal. On the scalability of parallel uct. In *International Conference on Computers and Games*, pages 36–47. Springer, 2010.

[38] Gregory Kuhlmann and Peter Stone. Automatic heuristic construction for general game playing. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, page 1883–1884. AAAI Press, 2006.