Learning to Play “Connect Four” with AlphaZero

Rok Cej
rc8309@student.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

AlphaZero is one of the most significant milestones in the field of deep reinforcement learning within the last decade. In this project, we describe our implementation of AlphaZero and its adaptation to the board game “Connect Four”. Our models managed to show improvement in a matter of hours, even surpassing the level of play of a human beginner.

KEYWORDS

AlphaZero, deep reinforcement learning, CNN, Connect Four

1 INTRODUCTION

In 2018, DeepMind introduced the AlphaZero algorithm [1, 2]. Given no domain-specific knowledge, except for the game rules, AlphaZero achieved a superhuman level of play in various board games, including chess, within 24 hours. It uses a deep reinforcement learning technique and learns by playing against itself.

The goal of this project is to implement the AlphaZero algorithm and use it to learn the game “Connect Four”. It is played on a 6x7 board and has a very small action space with a maximum of 7 possible moves per turn. However, despite its simplicity, it has an extremely large state space with a lower bound estimate of $1.6 \cdot 10^{13}$, which makes brute-forcing solutions impossible. There exists a perfect strategy, where the first player can win in at most 41 moves, but it requires a lot of insight and knowledge of the game. In this report, we describe our approach towards learning “Connect Four” without any domain-specific knowledge.

2 METHODS

The way AlphaZero trains can be simplified into a diagram shown in figure 1. On the outermost layer, it constantly cycles between self-play and model training. Self-play consists of simulating several games, then labelling each game state based on the outcome. During each turn, hundreds of Monte Carlo tree search (MCTS) iterations are performed to find the best move. The model, a convolutional neural network (CNN), is then trained based on the self-play results. The following subsections describe these steps in more detail.

2.1 Self-play reinforcement learning

The “brains” of AlphaZero is a convolutional neural network. The network takes a board state s as input and produces two outputs: a policy vector p (probabilities for each move) and a value v (game outcome prediction).

During the self-play step, AlphaZero plays 64 games against itself. On each move, a Monte Carlo tree search (MCTS) is executed using p and v as heuristics. This yields a much stronger policy π, which is used to play the next move. After the game ends, the outcome z is obtained. For each position i from the game, a training example (s_i, π_i, z) is stored.

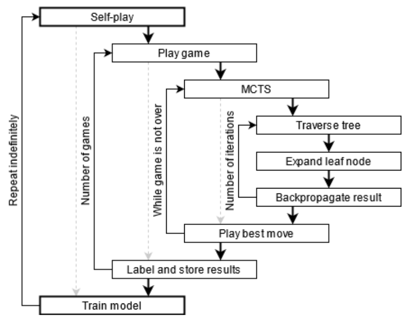

Figure 1: High-level overview of AlphaZero

After finishing all self-play games, the data is used to train the neural network. We used mini-batches of size 64. The model aims to bring it’s predicted policy p closer to the improved policy π, and its predicted value v closer to the game outcome z. In order to achieve this, the following loss function is used:

$$l = (z - v)^2 - \pi^T \log p + c\,||\theta|| \tag{1}$$

To prevent network weights from spiralling out of control, we added an L2 regularization parameter $c = 10^{-4}$. For training we used stochastic gradient descent (SGD) along with a learning rate decay schedule.

2.2 Monte Carlo tree search (MCTS)

The main idea behind MCTS is to use the predicted policy p and value v produced by the model as heuristics for executing a search that generates an improved policy π (in other words, the best move to play). MCTS works by building a tree of game states, which are connected by player actions. At the start of every search, the root node representing the current game state is created. Then, the tree is constructed through multiple iterations (100 in our case).

During each MCTS iteration, the tree is traversed until a leaf node s_L is found. The leaf node is then expanded by inputting its corresponding state into the neural network and creating a child for each legal move a. Based on the obtained policy p and value v, each child is assigned:

- a prior probability $P(s_L, a) = p(a)$,
- a visit count $N(s_L, a) = 0$,
- a mean-action score $Q(s_L, a) = 0$, and
- a total-action value $W(s_L, a) = 0$.

After the leaf node has been expanded, all previously visited nodes are updated in a backwards pass for each time step $t < L$:

- $N(s_t)$ += 1,
- $W(s_t)$ += v, and
- $Q(s_t) = \frac{W(s_t)}{N(s_t)}$.

When traversing the tree, action a_t at timestep t is selected by maximizing the upper confidence bound (UCB) score:

$$a_t = \mathtt{argmax}_a(Q(s_t, a) + U(s_t, a)) \tag{2}$$

$$U(s, a) = c_{\text{puct}} * P(s, a) * \frac{\sqrt{N(s)}}{1 + N(s, a)} \tag{3}$$

Here, c_{puct} serves as an exploration parameter. By increasing it, we also encourage exploration when choosing an action. In our case, we found that values between $2 < c_{\text{puct}} < 4$ work best.

Another innovation of AlphaZero was to combine MCTS with noise. To increase exploration at the start of every search, Dirichlet noise is added to the prior probabilities P of the root node. Since Dirichlet noise is normalized, we can add it as a weighted average:

$$(1 - x) * p + x * Dir(\alpha) \tag{4}$$

Here, x represents the exploration fraction and α represents the Dirichlet noise distribution parameter. As α get smaller, Dirichlet noise starts to resemble basis vectors ($[0, ..0, 1, 0, ..0]$). In our case, $x = 0.25$ and $\alpha = 1$ seemed to work well.

After completing all MCTS iterations, we can obtain the improved policy π by looking at the visit counts N of children of the root node. The probabilities are computed as follows:

$$\pi_a = \frac{N(s_0, a)}{N(s_0)} \tag{5}$$

During the first 10 moves of the game, the best move is sampled, using π as weights. The idea behind this is to increase diversity during the earlier stages of the game. Afterwards, the best move is selected as the move with the highest probability.

2.3 Encoding game information

In order to input a game state into our model, we first need to encode it. We represent each state as a stack of three 6x7 binary planes, resulting in a 3x6x7 binary image. The first plane encodes the first player's pieces, where 1 indicates the presence and 0 the absence of a piece. Similarly, the second plane encodes the second player's pieces. The third plane is a constant-valued plane used to indicate whose turn it is - zeros indicate the first player's turn, while ones indicate the second player's turn. Note that the structure of encoded game states generally shouldn't have a big impact on the results as long as all the information is provided.

2.4 Network architecture

AlphaZero uses a deep convolutional neural network. It consists of an input block, followed by 19 residual blocks, and an output block. For most convolutional layers we used 128 filters of size 3x3 (padding = 1).

The input block is structured as: [Conv > Batch norm > ReLU].

Each residual block is structured as: [Conv > Batch norm > ReLU > Conv > Batch norm > ReLU]. It also has a skip connection from the input to the second ReLU.

The output block consists of a policy head (best move prediction) and a value head (outcome prediction). They are structured as follows:

- Policy head: [Conv > Batch norm > ReLU > Fully connected (82, 7) > Softmax]
- Value head: [Conv (1 filter, size 1x1) > Batch norm > ReLU > Fully connected (42, 128) > Log Softmax]

3 RESULTS

Three different networks were trained for approximately 4 hours. The following parameters were used:

Model ID	x	c_{puct}	Num MCTS iters	Num epochs
#1	0.25	4	25	75
#2	0.25	4	100	30
#3	0.2	2	100	25

The models all played 100 games against each other and almost every match ended in a decisive 100-0 victory. During testing, all models used 100 MCTS iterations. Results are shown in the following table:

P1	P2	P1 wins	P2 wins	Draws
#1	#2	100	0	0
#1	#3	100	0	0
#2	#1	0	100	0
#2	#3	100	0	0
#3	#1	100	0	0
#3	#2	100	0	0
#1	#1	0	0	100
#2	#2	100	0	0
#3	#3	100	0	0

The first sections shows matches between different models, while the second section shows matches that models played against themselves. It appears that the model that had the first turn won most of the time, except in 2 matches.

Our models did not achieve superhuman performance. When playing against them, I managed to win a couple of times. However, they beat me most of the time.

4 CONCLUSIONS

AlphaZero is a fascinating algorithm, being able to learn games in an unsupervised manner without any domain-specific knowledge. Despite that, it is quite computationally expensive. Even for simple games such as "Connect Four", learning can take several hours, if not days. Our models showed significant improvement in a matter of hours, however, they did not achieve a superhuman level of play. One of the difficulties of working with neural networks is that they are virtually a black box, making it nearly impossible to understand what is happening inside. Finding the optimal parameters is often experimental and can involve a little bit of luck.

REFERENCES

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

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