Mastering the game of Go with deep neural networks and tree search

Silver et al (2016)

https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/36349047/report_florian_brunner.pdf

What is AlphaGo & AlphaZero?

AlphaGo defeated professional Go(囲碁) player in 2015.
However, AlphaGo relied on a large number of game records.

AlphaZero is the successor to AlphaGo.
It achieved performance that surpassed AlphaGo by playing against itself without using human game data.

Algorithm overview

The most powerful algorithm for Go is an exhaustive tree search.
However, the game size of Go is very huge, so it is impossible.

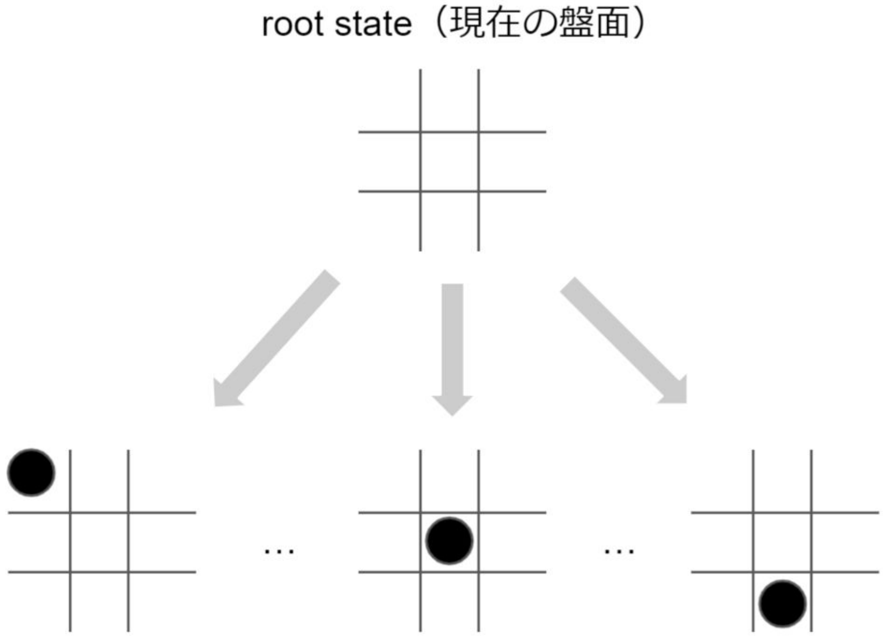
That’s why Monte Carlo tree search (MCTS) is used.
In this way, A board state is evaluated from the results of countless random plays starting from that board state.
The evaluation of a board state depends on only random plays, so it can handle problems that are difficult to design evaluation functions for.

AlphaZero uses MCTS, which provides significant performance improvements.

Pure Monte Carlo Tree Search - Part1

Go is too complex, so I’ll explain it using a simpler game called Tic-tac-toe(三目並べ).

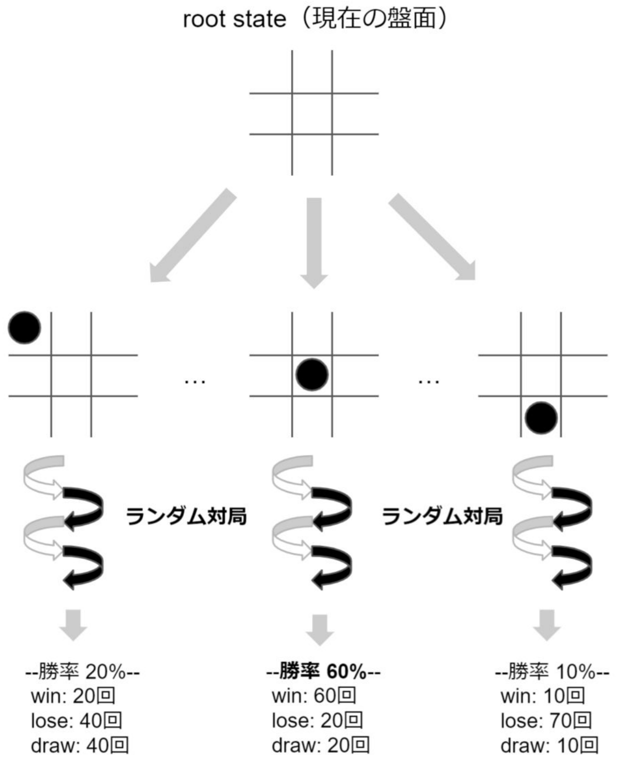
If we are on the play,
we have nine options.
Which action
should we take?



Pure Monte Carlo Tree Search - Part2

In Pure-MCTS, we run countless matches that start with these 9 actions.
For example, we run 100 random matches and get the following results.

From the diagram on the right, we can see that placing a stone in the center to get the highest chance of winning.



Pure Monte Carlo Tree Search - Part3

The problem of Pure-MCTS is that it requires the same number of random matches for every action.
A certain number of trials is necessary to ensure reliable board state evaluation, but complex games such as Go require a huge number of plays.

We want to find actions that will take good evaluation values in a finite amount of play.

Upper confidence bound applied to Trees - Part1

“Upper confidence bound applied to trees” select a certain action by using the following formula.

wi is the number of wins for a certain action.
ni is the number of attempts for a certain action.
Ni is the number of attempts for all actions.

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

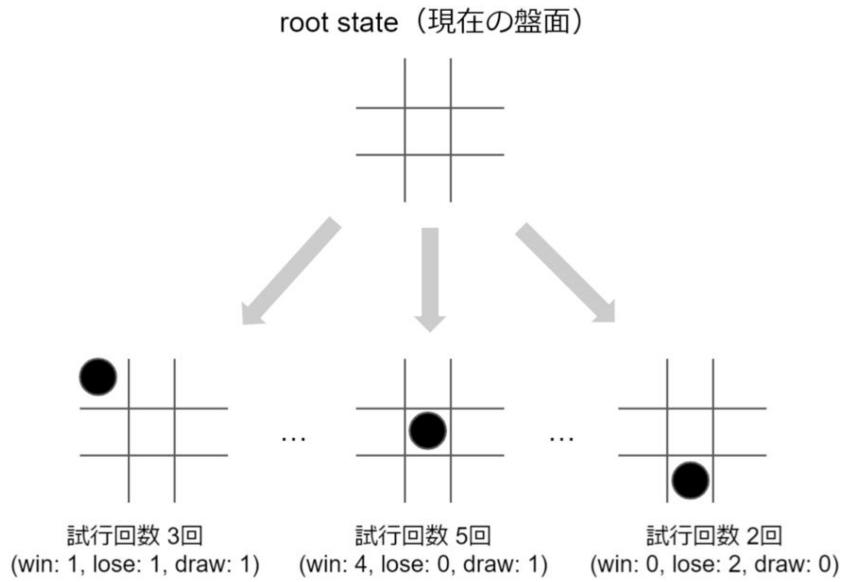
The term on the left is the win rate,
and the term on the right is larger
when a certain action is attempted fewer times.

In other words, this algorithm select an action that have a high win rate,
but are not played often.

Upper confidence bound applied to Trees - Part2

We have just finished
10 rounds of tic-tac-toe.

The action with the highest score is the center one 1.47, so we will play this position from now on.


root state（現在の盤面）
…
…
試行回数 3回
(win: 1, lose: 1, draw: 1)
試行回数 5回
(win: 4, lose: 0, draw: 1)
試行回数 2回
(win: 0, lose: 2, draw: 0)

UCTスコア (c=1.0)	1/3 + √(ln10/3)	4/5 + √(ln10/5)	0/2 + √(ln10/2)
	= 0.33 + 0.87	= 0.8 + 0.67	= 0 + 1.07
	= **1.2**	= **1.47**	= **1.07**

N look ahead MCTS - Part1

We add N lookahead to Pure-MCTS.
In the previous MCTS, we only do 1 lookahead, but deeper lookahead increases the reliability of the play.

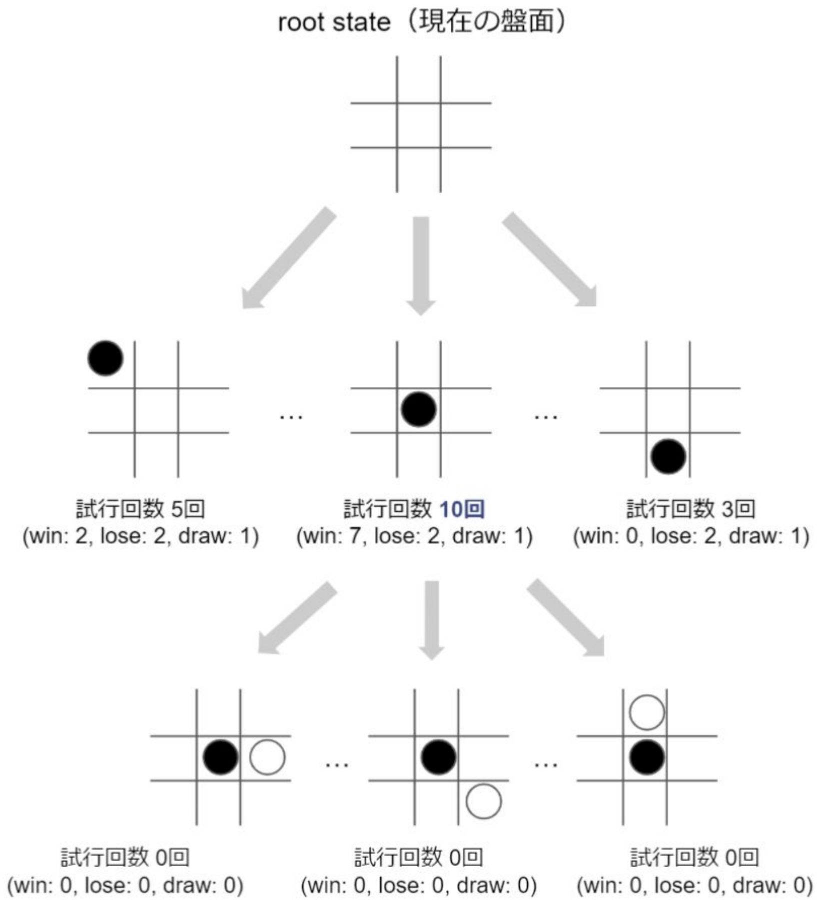
However, lookahead is very costly, so we want to limit it to promising actions.
Therefore, we will only consider the next state for which the number of attempts exceeds a certain value.

N look ahead MCTS - Part2

For example, if we set the threshold to 10, it will look like this!

Only the middle action has been attempted more than 10 times, so we consider the next board state.

This allows us to allocate more think times to promising actions and do more deep lookahead on actions that are promising enough.


root state（現在の盤面）
試行回数 5回
(win: 2, lose: 2, draw: 1)
…
試行回数 10回
(win: 7, lose: 2, draw: 1)
…
試行回数 3回
(win: 0, lose: 2, draw: 1)
試行回数 0回
(win: 0, lose: 0, draw: 0)
…
試行回数 0回
(win: 0, lose: 0, draw: 0)
…
試行回数 0回
(win: 0, lose: 0, draw: 0)

MCTS + Episode memory - Part1

MCTS is an efficient algorithm, but it forces us to learn from scratch every time.
We can use the history of past MCTS to find out and eliminate bad actions.

For example, if we play 100 games, we will collect 100 data about the early game.
Then, we can distinguish between good and bad actions.
We should not try actions that are clearly unpromising based on past history.

MCTS + Episode memory - Part2

In AlphaZero, they build neural networks that approximate past MCTS results.

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

Cpuct is high parameter and P(s, a) reflects the history of MCTS.

In action selection, they choose the action that maximizes the sum of utilization Q(s, a) and exploration U(s, a).

$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t,a) + U(s_t,a)\right)$$

Neural network input

For example, in the game of Go, we input 17 boards.
Each board has 19 * 19 pieces.

Prepare board with our own stones valued at 1 and everything else valued at 0.
Also prepare board with opponent’s stones valued at 1 and everything else is 0.
These board is prepared for 8 steps. ((1 + 1) * 8 = 16 boards)

For the remaining board, if our color is black, prepare a board filled with 1.
And if our color is white, prepare a board filled with 0.

Go		Chess		Shogi	
Feature	Planes	Feature	Planes	Feature	Planes
P1 stone	1	P1 piece	6	P1 piece	14
P2 stone	1	P2 piece	6	P2 piece	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P2 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 castling	2		
		P2 castling	2		
		No-progress count	1		
Total	17	Total	119	Total	362

Table S1: Input features used by *AlphaZero* in Go, Chess and Shogi respectively. The first set of features are repeated for each position in a $T = 8$-step history. Counts are represented by a single real-valued input; other input features are represented by a one-hot encoding using the specified number of binary input planes. The current player is denoted by P1 and the opponent by P2.

Neural network output

(a, b) represent the action to be taken.
Each element has a probability of placing a stone in the corresponding location.

Summary of AlphaZero

AlphaZero uses MCTS to select promising actions in a small number of trials. MCTS has evolved step by step, and now its accuracy has been greatly improved by approximating neural networks.

AlphaZero reflects the results of past MCTS and can search game trees more efficiently.

Importantly, all data to train the network is generated by self play, no humans required!