GAME REPORT(1) - CutieGomoku

09016407 Dailin Hu

Generalized gomoku is PSPACE-complete[1], indicating that it can be solved using an amount of memory that is polynomial in the input length or it can be solved in polynomial space and transformed to what's mentioned in polinomial time.

The game Gomoku has a game tree state-space complicity of 10^{105}, and game-tree complicity of 10^{70} [2]. Games with a relatively low game-tree complexity have mainly been solved with **knowledge-based methods**. [2] Thus two different ways of solving Gomoku could be considered. This term, I hope to tackle gomoku from both approaches. One is the knowledge-based method using threat-space search tree, the other is d**eep-reinforcement learning method(mcts+dnn)**, imitating the approaches taken by AlphaZero in the game of Go. The latter would be of major importance, while the knowledge-based method would mainly be considered as a 'control group' for testing the deep-reinforcement learning model

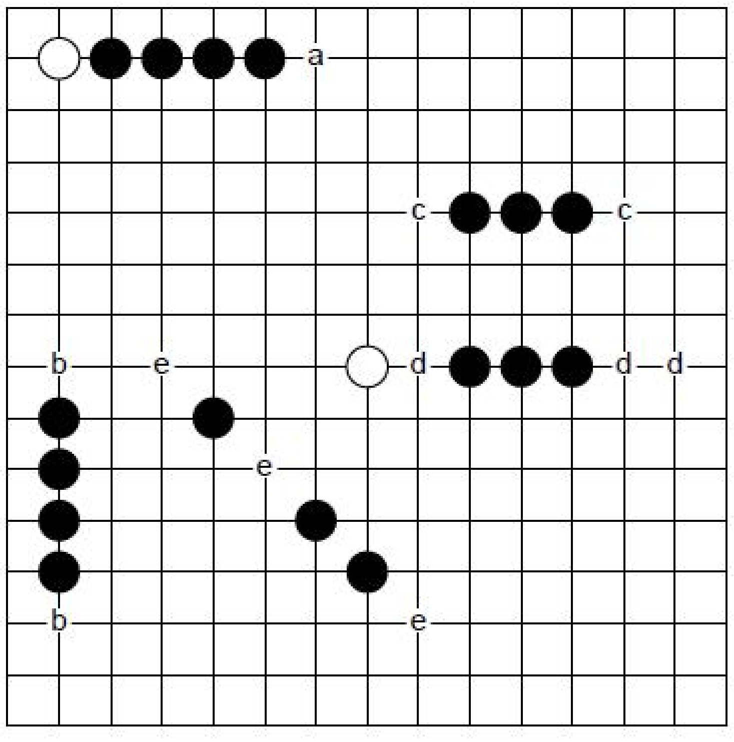
My schedule for the development of cutie-gomoku is as follows:

WEEK	SCHEDULE
1-2	get to know about some possible ways of tackling the game of gomoku
3-4	learn the fundamentals of deep learning
5-6	construct and train the policy model for GOMOKU
7-8	optimizations & further discussion

Knowledge-Based Methods(threat-space search tree)

We consider four types of 'threats' in Gomoku, namely:

1. the *four*
2. the *straight four*
3. the *three*
4. the *broken three*



In general, 3this method is based on human knowledge and empirical research.

Deep-Reinforcement Learning(Monte Carlo Search Tree)

Preliminaries

Markov Decision Process

The environment for GOMOKU is a *Markov Decision Process*(MDP), which is defined as follows:

1. a set of states S
2. a set of possible actions A
3. the conditional distribution P(s'|s,a)P(s'|a) of the next state, gien a current state and an action
4. the reward function of transitioning from state s to s' :R(s,s')
5. the discount factor(the preference for present rewards compared to future rewards)

MDP is memoryless, and states visited before doesn't affect the next transitions and rewards.

Terminology

Final/terminal states: the states with no available actions

Episode: a complete play from initial state to a final state $s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_n$

Cumulative reward: the discounted sum of reward accumulatd throught an episode, $R = \sum_{t=0}^{n} \gamma^t r_{t+1}$

Policy(π): the agent's strategy to choose an action at each state

Objective: train an agent to learn a policy as close as possible to th optimal policy.

How Alpha Zero differs from Apha Go

Alpha-beta pruning

worst case $O(b^d)$

best case $O(\sqrt{b^d})$

for Gomoku, $b \approx 225, d \approx 30$[3]

Unlike Alphago, which required supervised learning in the first stage, Alpha zero is based on an algorithm solely on reinforcment learning.

1. trained by self-play reinforcement learning only
2. use black/white stones & board as input features
3. use a single neural network rather than separate policies
4. a simpler tree search without using MC rollouts

A New RL Algorithm

p ← the vector of move possibilities v ← scalar evaluation estimating the winning probability output:

$(p, v) = f_\theta(s)$ $p_\alpha = Pr(a|s)$

in each position s, MCTS is conducted under the guidance of f_θ

MCTS acts as a policy-improvement operator

General Process

input: raw board position s_t

output: p_t (a probability distribution over moves), v_t(the probability of the current layer winning in position

s_t)

steps:

1. initialized to random weights θ_0
2. at each step, MCTS search $\pi_t = \alpha_{\theta_{i-1}}(s_t)$ is executed using previous network $f_{\theta_{i-1}}$, where $\pi_\alpha \propto N(s,a)^{1/\Gamma}$, where Γ is **a temperature parameter**
3. the game terminate at leafnode, then scored to give a reward $r_t \in -1, 1$
4. the data for each time step: (s_t, π, z_t), where $z_t = \pm r_T$ is the gam winner from the perspective of the current player at step t
5. new network parameters θ_i are trained from data(s, π, z) sampled uniformly among all time-steps of the last iteration(s) of self-play

6. $\boldsymbol{\theta}$ is adjusted by gradient descent on a loss function l that sums over the mean-squared error and cross-entropy losses: $l = (z - v)^2 - \pi^T logp + c||\theta||^2$

---

[1]: Stefan Reisch (1980). "Gobang ist PSPACE-vollständig (Gomoku is PSPACE-complete)". *Acta Informatica*. **13**: 59–66. doi:10.1007/bf00288536.

[2]:van den Herik, H., Uiterwijk, J. W. & van Rijswijck, J. Games solved: now and in the future. Artif. Intell. 134, 277–311 (2002).

[3]: Allis, L. V. Searching for Solutions in Games and Artificial Intelligence. PhD thesis, Univ. Limburg, Maastricht, The Netherlands (1994).