Deep Reinforcement Learning for the Game of Simulated Curling

Kyowoon Lee[1,2,a)] Sol-A Kim[1,2,b)] Jaesik Choi[2,c)] Seong-Whan Lee[3,d)]

Abstract: Recently, deep reinforcement learning have achieved super-human performance in the deterministic games with discrete action spaces, such as Atari games and Go. However, it is still not clear how to utilize deep neural networks for discrete actions in complex games in which a minute change of an action could alter the outcome of the games dramatically. To solve this issue, we incorporates a deep neural network for learning game strategy with a kernel-based Monte Carlo tree search for finding actions from continuous space especially for the game of curling. Without any hand-crafted feature, we train our network in supervised learning manner and then reinforcement learning. Recently, our framework outperforms existing programs equipped with several hand-crafted features for curling and won in the Game AI Tournaments (GAT-2018).

1. Background

1.1 Monte Carlo Tree Search

Monte Carlo Tree Search is a search algorithm to explore game tree in finite-horizon sequential decision-making settings. It iteratively simulate executions (edges) from the current state to a terminal state, trying to approximate the winning percentage or expected value of the simulated states (nodes). For each simulation, it selects successive child nodes based on a *selection function*. When a node is visited whose immediate children are not in the tree, the node is *expanded* by creating a new child node. Then, simulation begins at expanded node and continues until the end of the game, selecting actions based on the *rollout policy*. The rollout statistics are updated in a backward pass.

One of the commonly used MCTS methods is Upper Confidence Bounds Applied to Trees (UCT) [1], which uses Upper Confidence Bound (UCB) selection function. Each node maintains the mean of the rewards received for each action $\bar{v_a}$ and the number of times each action has been used n_a. Then, it decides what action with high mean of rewards and few simulations based on the size of the one-sided confidence interval on the reward computed based on the Chernoff-Hoeffding bound; $\arg\max_a \bar{v}_a + C\sqrt{\frac{\log \sum_b n_b}{n_a}}$.

1.2 Kernel Regression

Kernel regression is a non-parametric method for non-linear regression where the target value is estimated using a weighted average of the values of all points in the data set. A set of identical weighted function is called the kernel and further denoted K. Given a choice of kernel K, and a data set $(x_i, y_i)_{i=0}^n$, kernel regression is defined as follow:

$$E[y|x] = \frac{\sum_{i=0}^n K(x, x_i) y_i}{\sum_{i=0}^n K(x, x_i)} \tag{1}$$

One typical kernel function is the Gaussian probability density function which is defined as follows:

$$K(\mathbf{x}, \mathbf{x}') = \frac{\exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Sigma^{-1} (\mathbf{x} - \mathbf{x}')}{\sqrt{(2\pi)^k |\Sigma|}} \tag{2}$$

2. Deep Reinforcement Learning in Continuous Action Spaces

2.1 The Policy-Value Network

Our policy and value networks are trained by a unified network. By sharing the front layers, the network could be trained faster than training two networks individually [4]. The input of the policy-value network is 32x32 discretized image (the position of stones in the curling sheet) and 29 features planes; the colour of stones (3 planes), a constant plane filled by 1s (1 plane), the turn number (8 planes), a flag whether a grid inside of the house is occupied by any stone or not (1 plane), and the orders to tee (16 planes). The hidden layers which are shared have one convolutional layer and eight residual blocks [3]. Each residual block has two convolutional layers with batch normalization and the result is added by the input of the block. For each convolution, 3x3 filter is used and ReLU is used for the activation function.

After the shared layers, the network is divided by the policy head and the value head. The policy head has two convolutional layers and outputs 32x32x2 discretized image which is the probability distribution of actions for the best

[1] Contributed equally to this work
[2] Department of Computer Engineering, Ulsan National Institute of Science and Technology, Ulsan, Republic of Korea
[3] Department of Brain and Cognitive Engineering, Korea University, Seoul, Republic of Korea
[a)] leekwoon@unist.ac.kr
[b)] sol-a@unist.ac.kr
[c)] jaesik@unist.ac.kr
[d)] sw.lee@korea.ac.kr

1: Use *Selection* on the current state s_t to obtain a state $s_{t'}$ to be used in expansion step.
2: From $s_{t'}$, run *Expansion* to add a new action $a_{t'}{}'$ and generate initial actions $A_{t'+1}$.
3: Use *Simulation* to get the next state $s_{t+1'}$ generated by $a_{t'}{}'$.
4: $s_{t+1'}$ is evaluated by value network $\mathbf{v}_\theta$ and updated in backward pass.
5: Get back to step 1, until the number of iterations reaches to a predefined playout numbers.

Algorithm 1: KR-DL-UCT

shot given the input state. The policy network $\mathbf{p}_\theta$, including the shared layers, have 21 convolutional layers.

The value head has one convolutional layer and two more fully connected layers. The last layer outputs the probability distribution of the expected score in the range of [-8,8] after the activation with the softmax function. The value network $\mathbf{v}_\theta$ have 20 convolutional layers and two fully connected layers including the shared layers.

2.2 Continuous Action Search

Algorithm 1 shows the our algorithm Kernel Regression Deep Learning Upper Confidence bound applied to Trees (KR-DL-UCT) written with general subroutines *Selection*, *Expansion* and *Simulation*. First, subroutine *Selection* selects action based on *UCT* selection function with kernel regression $\mathbb{E}(\bar{v}_a|a)$ and kernel density estimation $W(a)$ [2].

$$a_t = arg\max_{a\in A_t} \mathbb{E}[\bar{v}_a|a] + C\sqrt{\frac{\log \Sigma_{b\in A_t} W(b)}{W(a)}} \qquad (3)$$

As long as child node exists, it iteratively selects child node while $\sqrt{\sum_{a\in A_t} n_a} < |A_t|$ which is called as *progressive widening*, adding additional actions only after the quality of the best available action is estimated sufficiently well. This procedure enables the algorithm to use a kind of information sharing among visited nodes.

Second, subroutine *expansion* adds a new node by exploring the uncertain area. That is, among the sampled actions a from the selected action $a_{t'}$ within the similarity γ, an action $a'_{t'}$ which has the lowest estimated visit number $W(a)$ is selected to expand a node; $a'_{t'} \leftarrow arg\min_{K(a_{t'},a)>\gamma} W(a)$. The policy network is used to initialize actions with the policy $\pi_{a|s_{t+1}}$ for the new expanded nodes including the root.

$$A_{t'+1} \leftarrow \cup_{i=1}^{k} \{a_{init}^{(i)}\} \text{ s.t } a_{init}^{(i)} \sim \pi_{a|s_{t'+1}} \qquad (4)$$

$$\pi_{a|s_{t'+1}} = \frac{\mathbf{p}_\theta(a|s_{t'+1})^{1/\tau}}{\Sigma_b \mathbf{p}_\theta(b|s_{t'+1})^{1/\tau}} \qquad (5)$$

To prevent the biased sampling, the temperature parameter τ is applied. Using the unbiased estimator $\pi_{a|s_{t+1}}$, the k number of actions are initialized.

Third, subroutine *simulation* requests the curling simulator to generate the next state (s_{t+1}) by delivering a stone (a_t) given the current state (s_t).

Program	Winning Percentage
GCCSGAT'17	74.0 ± 6.22%
AyumuGAT'16	66.5 ± 6.69%
AyumuGAT'17	62.3 ± 6.87%
JiritsukunGAT'16	86.3 ± 4.88%
JiritsukunGAT'17	55.5 ± 7.04%

Table 1 The 8-end game results of KR-DL-UCT against other programs with exchanging the opening player per game. The matches are held by following the rules of the lastest GAT competition.

3. Results

3.1 Dataset

Our dataset is divided into two parts. One is from the play data of the first grade program AyumuGAT'16 on the 2016 Game AI Tournament (GAT) digital curling field. The other is the self-play data within our program to improve the performance of our policy-value network.

3.2 Settings

We validate our proposed model with the high performing programs in 2016 and 2017 GAT. All the games follows the rules of 2017 GAT. Each match is scored by 200 games; 100 games as a first player and 100 games as a second player.

To consider the multi ends strategy, we constructs a winning rate table by analyzing the statistical result of our dataset 3.1. When evaluating actions at the selection stage of our algorithm, the expected score distribution from the value network $\mathbf{v}_\theta$ and the number of remaining ends are converted to the the expected winning rate.

3.3 Results

Our experimental results competing the existing programs are in **Table 1**. GCCSGAT'17 is the program based on the implementation of a strategy book, and both AyumuGAT'16 and '17 are MCTS based algorithms with hard coded default policy. Our proposed algorithm KR-DL-UCT significantly outperforms them. Both Jiritsukun'16 and '17 are game tree based search algorithms, and Jirisukun'17 uses the deep neural networks as evaluation functions. It shows the striking performance along with other programs, but KR-DL-UCT still outperforms it.

Acknowledgements

A full version of this paper is available at [5]. This work was supported by Institute for Information and communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2017-0-01779 and No.2017-0-00521).

References

[1] Kocsis, L. and Szepesvári, C. Bandit Based Monte-Carlo Planning. 2006
[2] Yee, T., Lisý, V. and Bowling, M. Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty. 2016
[3] He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. 2016

[4] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. 2017

[5] Lee, K., Kim, S., Choi, J. and Lee, S. Deep Reinforcement Learning in Continuous Action Spaces; a Case Study in the Game of Simulated Curling. 2018