Homework: Budget Allocation Policy for Monte Carlo Tree Search

Course: Artificial Intelligence and Business Innovation

1 Introduction

Monte Carlo Tree Search (MCTS) has been developed extensively and applied to various games such as Othello and Go. The term MCTS was introduced by Rémi Coulom in a conference paper presented in 2005/2006 (see also the Wikipedia Monte Carlo tree search entry).

Google DeepMinds AlphaGo employed MCTS to train its two deep artificial neural networks. AlphaGo started training its neural networks using past games, i.e., via supervised learning, and then went on to play against itself (self-play) using reinforcement learning to train its neural networks. In contrast, both AlphaGo Zero and AlphaZero train their respective neural networks by self play only using MCTS for both players (reinforcement learning).

The basic operations of MCTS contain four steps: selection, expansion, simulation and back-propagation. Most bandit-based MCTS algorithms are designed to minimize regret (or maximize the cumulative reward of the agent), whereas in many situations, the goal of the agent may be to efficiently determine the optimal set of actions within a limited sampling budget. Then the tree selection problem at each stage is a statistical Ranking & Selection (R&S) problem, which can balance exploration and exploitation with a limited sampling budget for a tree policy. R&S assumes that we are given a set of bandit machines (often referred to as alternatives in the R&S literature) with unknown reward distributions, and the goal is to maximize the probability of correctly selecting the machine with highest mean reward using a limited sampling budget.

In the lecture, you learned about basic operations of MCTS. In this assignment, you will practice implementing R&S-MCTS to find the best action at the root of tree.

2 Example: Tic-Tac-Toe

You are given 3 files in the handout (https://github.com/gongbozhang-pku/Monte-Carlo-Tree-Search). The 'tic_tac_toe.py' file contains a game named as Tic-Tac-Toe, which is a well-known two-player game on a 3×3 grid with the objective to get 3 marks in a row, column or diagonal, and players alternate between the "X" player who goes first and the "O" player. If each player follows an optimal policy, then the game should always end in a tie (cats game).

There are two board setups: "X" player has already marked in space 0 (setup 1) and space 4 (setup 2). The root nodes and optimal actions for "O" player are illustrated in Figure 1 (a) and

(a) Setup 1 (b) Setup 2

Figure 1: Tic-Tac-Toe board

Figure 1 (b), respectively. In setup 2, the optimal move for "O" player will be marking any of the corner spaces due to symmetry, and we consider it a correct selection if the algorithm returns any one of the optimal moves. For both setups, taking any of the suboptimal actions will end up in losing the game if "X" player plays optimally.

3 Task 1: Implement the MCTS Using Budget Allocation Policy

Let X_a and A_x be the available child states when taking action a and available actions at state x, respectively. Denote by $P_t(x,a)(y)$ the probability of transitioning to state $y \in X_a$ from state $x \in X$ when taking action $a \in A_x$ in stage t, and $R_t(x,a)$ the reward in stage t by taking action a in state x. Define $Q_i(x,a) = \mathbb{E}[R_i(x,a)] + \sum_{y \in X_a} P_t(x,a)(y) V_{i+1}^*(y)$ with $Q_H(x,a) = 0$, where $V_i^*(x)$ is the optimal reward-to-go value function for state x in stage i. Our goal is to identify the optimal action that achieves the highest cumulative reward at the root with initial state x, i.e., find $a_{x_0}^* = \arg\max_{a \in A_{x_0}} Q(x_0, a)$.

Let $\hat{Q}(x,a) = R(x,a) + V^*(y)$ be the random cumulative reward by taking action a at state node x, where y is the random state node reached. Define $N(x)$ and $N(x,a)$ the number of visits to node x and (x,a), respectively. We assume $\hat{Q}(x,a)$ is normally distributed with unknown mean $\mu(x,a)$ and known variance $\sigma^2(x,a)$. Consider the non-informative prior for $\mu(x,a)$, and then the posterior distribution of $\mu(x,a)$ given the sample $(\hat{Q}^1(x,a), \hat{Q}^2(x,a) \cdots, \hat{Q}^{N(x,a)}(x,a))$ is $\tilde{Q}(x,a) \sim N\left(\bar{Q}(x,a), \sigma^2(x,a)/N(x,a)\right)$, where $\bar{Q}(x,a) = \sum_{t=1}^{N(x,a)} \hat{Q}^t(x,a)/N(x,a)$ and $\sigma^2(x,a)$ can be approximated by the sample variance $\hat{\sigma}^2(x,a) = \sum_{t=1}^{N(x,a)} (\hat{Q}^t(x,a) - \bar{Q}(x,a))^2/N(x,a)$. A metric for our objective is the probability of correct selection (PCS): $PCS = P[\cap_{a \in A, a \neq \hat{a}_x^*} (\tilde{Q}(x, \hat{a}_x^*) \geq \tilde{Q}(x,a))]$, where $\hat{a}_x^* = \arg\max_{a \in A_x} \bar{Q}(x,a)$. The objective of our tree policy is to maximize PCS.

The objective function and problem formulation are motivated by the R&S literature. For more information about the R&S, please refer to [1, 2, 3]. You can refer to any other sources of reference as well. There are many different budget allocation procedures available in literature to obtain the objective. We will use the randomized policy (with equal probability to mark any feasible space), Upper Confidence Bound Apply to Tree (UCT), Optimal Computing Budget Allocation (OCBA) [1] and Asymptotically Optimal Allocation Procedure (AOAP) [4] for both "X" and "O" players. The following shows one way of solving the problem and skeleton code is provided to you in 'monte_carlo_tree_search_tic_tac_toe.py'.

choose: Choose the best successor (a move) of node.

do_rollout: Make the tree one layer better. Train for one iteration.

_select: Find an unexplored descendent of 'node'.

_expand: Add a node to the tree.

_simulate: Returns the reward for a random simulation (to completion) of 'node'.

_backpropagate: Send the reward back up to the ancestors of the leaf.

_uct_select: UCT. A benchmark for selecting a child of node, balancing exploration & exploitation.

_ocba_select: OCBA. A benchmark for selecting a child of node, balancing exploration & exploitation.

All policies estimate $Q(x, a)$ for each action a by its sample mean, and selects the action that minimizes the sample mean as $\hat{a}_x^*$. Each state-action node should be expanded $n_0 > 1$ times, since we need a sample variance for each state-action node. The process is run for a prespecified N times (number of rollouts or sampling budget) from the root state node x_0, after which a partially expanded tree is obtained and the optimal action $\hat{a}_{x_0}^*$ can be derived. At state node x, the reward function for taking action a is: immediately after taking the action, if "O" player wins the game, $R(x; a) = 1$, if it leads to a draw, $R(x; a) = 0.5$; otherwise ("O" player loses or in any non-terminating state), $R(x; a) = 0$.

After running the three files, the figures can be generated by 'tic_tac_toe_analysis.py'. You may need to adjust the location of the legend for different experiments. There is no one-size-fits-all location option. You are also encouraged to develop your own implementation to solve the problem. **If you are going to use the skeleton code, you have to write the functions '_AOAP_select()', change the policies of "X" and "O" players and change the algorithm parameters to see different performances (i.e., n_0, N, etc.).** The code is well commented but feel free to reach out to the TA in case you have trouble understanding the code.

Algorithms

Two tree policies that optimally allocates a limited computing budget to maximize PCS for selecting the best action are described below.

In this project, the implementation of OCBA-MCTS is given. You are going to use AOAP-MCTS to find the optimal action.

4 Task 2 (optional): Other Games

As an optional part to the assignment, you can change the setup of the board of the Tic-Tac-Toe game or change other games that has a larger scale than the Tic-Tac-Toe (3×3 grid), and test all policies. You can submit the code for the same for extra credit.

Algorithm 1 OCBA-MCTS

Input: state node x.

Identify $\hat{a}_x^* = \arg\max_a \bar{Q}(x,a)$

$\delta_x(\hat{a}_x^*, a) \leftarrow \bar{Q}(x,\hat{a}_x^*) - \bar{Q}(x,a)$, $\forall a \neq \hat{a}_x^*$

Compute new sampling allocation $(\tilde{N}(x,a_1), \tilde{N}(x,a_2), \cdots, \tilde{N}(x,a_{|A|}))$ by solving equations:

$$\frac{\tilde{N}(x,a_{n+1})}{\tilde{N}(x,a_n)} = \left(\frac{\sigma(x,a_{n+1})/\delta_x(\hat{a}_x^*,a_{n+1})}{\sigma(x,a_n)/\delta_x(\hat{a}_x^*,a_n)}\right)^2,\ \forall a_n, a_{n+1} \neq \hat{a}_x^*,\ a_n, a_{n+1} \in A_x\ ,$$

$$\tilde{N}(x,\hat{a}_x^*) = \sigma(x,\hat{a}_x^*)\sqrt{\sum_{a\in A, a\neq\hat{a}_x^*} \frac{(\tilde{N}(x,a))^2}{\sigma^2(x,a)}}\ ,$$

$$\sum_{a\in A}\tilde{N}(x,a) = \sum_{a\in A} N(x,a) + 1\ .$$

$\hat{a} \leftarrow \arg\max_{a\in A}(\tilde{N}(x,a) - N(x,a))$.

Return $\hat{a}$

Algorithm 2 AOAP-MCTS

Input: state node x.

Identify $\hat{a}_x^* = \arg\max_a \bar{Q}(x,a)$

$\delta_x(\hat{a}_x^*, a) \leftarrow \bar{Q}(x,\hat{a}_x^*) - \bar{Q}(x,a)$, $\forall a \neq \hat{a}_x^*$

Compute following equations:

$$V_{a_x^*} = \min_{a_n \neq \hat{a}_x^*} \frac{(\delta_x(\hat{a}_x^*, a_n))^2}{\tilde{\sigma}^2(x,\hat{a}_x^*) + \sigma^2(x,a_n)}\ ,$$

$$V_{a_n} = \min\left\{\frac{(\delta_x(\hat{a}_x^*,a_n))^2}{\sigma^2(x,\hat{a}_x^*)+\tilde{\sigma}^2(x,a_n)}, \min_{a_{n+1}\neq\hat{a}_x^*, a_n} \frac{(\delta_x(\hat{a}_x^*,a_{n+1}))^2}{\sigma^2(x,\hat{a}_x^*)+\sigma^2(x,a_{n+1})}\right\}, \forall a_n \neq \hat{a}_x^*\ ,$$

$$\tilde{\sigma}^2(x,a) = \sigma^2(x,a)/(N(x,a)+1)\ .$$

$\hat{a} \leftarrow \arg\max_{a\in A} V_a$.

Return $\hat{a}$

5 Submission

Your submission will include a small report and all of your codes. The report should cover the followings:

- A short report explaining the code briefly and answering the following questions:

  Under different parameter settings and the policy of the "X" player, which policy (randomized, UCT, OCBA, AOAP) of the "O" player has a better performance?

- Screenshots of your results from "tic_tac_toe_analysis.py" file. If you opt to conduct the optional part, proper screenshots should be included.

  The submitted code should be properly formatted and commented.

References

[1] Chun-Hung Chen and Loo Hay Lee. *Stochastic simulation optimization: an optimal computing budget allocation*, volume 1. World scientific, 2011.

[2] L Jeff Hong, Weiwei Fan, and Jun Luo. Review on ranking and selection: A new perspective. *Frontiers of Engineering Management*, 8(3):321–343, 2021.

[3] Yijie Peng, Chun-Hung Chen, Edwin KP Chong, and Michael C Fu. A review of static and dynamic optimization for ranking and selection. In *2018 Winter Simulation Conference (WSC)*, pages 1909–1920. IEEE, 2018.

[4] Yijie Peng, Edwin KP Chong, Chun-Hung Chen, and Michael C Fu. Ranking and selection as stochastic control. *IEEE Transactions on Automatic Control*, 63(8):2359–2373, 2018.