COMP3702

Tutorial 8: MCTS

Semester 2, 2022

Aryaman Sharma (aryaman.sharma@uq.edu.au)

The University of Queensland
School of Information Technology and Electrical Engineering

Monte Carlo Tree Search

MCTS

- Algorithm for fast planning in online settings.
- The longer MCTS is allowed to run for, the higher quality the resulting policy is (on average).
- Rather than iterating over every connection in the state graph representation, we build a tree (subset of state graph), and prioritise the important states.
- The MCTS algorithm, consists of four components:
  - Selection
  - Expansion
  - Simulation
  - Back-Propogation
- For each node we store
  - **Q(s, a) for each action in A** – this is the average reward for (s, a) over all of our trials.
  - **N(s, a) for each action in A** – this is the number of times action a has been performed from state s.
  - **N(s)** his is the number of the times this state has been visited with any action performed.

MCTS

Given that we are at a current state, how do we choose what action to perform?

- Aim to compromise between:
  - Exploration (visiting under-explored branches) and
  - Exploitation (Visiting branches with higher average reward)
- Selection strategies:
  - **Random Choice** – if any actions have never been tried, choose an untried action at uniform random (tends to have better performance than trying to choose actions in a fixed order).
  - **Epsilon-Greedy** – Choose the highest Q-action with probability and all other actions equally likely (with probability, where n is the number of actions that can be performed).
  - **UCB** – Compute a confidence interval for the true average reward, based on the number of trials and choose the action with highest UCB

MCTS-Expansion

- Convert a leaf node into a non-leaf node.
- When a leaf node is reached:
    - Set (initialise the node count to 1)
    - Estimate V (the future expected value of the state) via simulation

MCTS-Simulation

- Estimate the future expected value of a state without building up a tree.
- Random Roll-out Choose actions at random until some maximum horizon is reached, keeping a running total of the reward.
- Can average this over a number of random rollouts.
- Can use a heuristic to choose actions during roll-out rather than choosing purely at random.
- Return the estimated value V

MCTS-Backpropagation

- We want to use the results from our simulations and update our node statistics and values.
- Essentially we update out Q values.

MCTS-Iterative

- Create a Tree Node class that stores:
  - N(s)
  - Q(s,a) and N(s,a) for each available action
  - Stores a lit of child nodes for each available action
  - Stores a reference to the parent node
- mcts_search(current_state)
  - Node current_state
  - While the nodes is not a leaf node, select and action and sample a next state(and set node $<-$ next_state)
  - Expand the leaf node and estimate the value via simulation (Create new tree node instance)
  - While the node doesn't have a parent, update Q(s,a), N(s, a) and N(a)
    - And set Node $<-$ node.parent
    - This is our backpropagation step (where we move backward in time, and update values)
    - Do until we reach the root node

MCTS-recursive

- Dictionaries are used to store node statistics
  - Number of times a state “s” has been visited
  - Number of times an action “a” has been performed from state “s”
  - Average reward from performing action a at state s
- mctsSearch(currentState)
  - f the current state is a leaf node, estimate the value V from simulation and return the value (recursion base case)
  - Otherwise, select an action for the current state using our dictionaries
  - Sample the outcome of the next state, and set (immediateReward + mctsSearch(nextState))
  - Increment and and update using the value V
  - Return the value V (so that the next level above can use the value)

MCTS

- For both approaches, the selectAction metohod should:
  - Call mcts while time/memeory limits are not reached
  - Action Argmax(Q(s,a)) over all actions
  - return action