COMP3702 Tutorial 8

Matt Choy | matthew.choy@uq.edu.au

COMP3702 Tutorial 8

Matt Choy | matthew.choy@uq.edu.au

https://github.com/mattpchoy/comp3702-tutorials

Monte Carlo Tree Search

- MCTS belongs to a family of algorithms 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:
  1. Selection
  2. Expansion
  3. Simulation
  4. Back-Propagation
- At each node, we store the statistics:
  - 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) this is the number of the times this state has been visited with any action performed.

Monte Carlo Tree Search - Selection

- 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)

Monte Carlo Tree Search - Selection

- 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 ϵ and all other actions equally likely (with $\frac{1-\epsilon}{n}$ 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

Monte Carlo Tree Search - Expansion

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

Monte Carlo Tree Search - 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
    - E.g., look 20 steps in the future, discounted by γ
    - This is not necessarily close to the optimum value for the state, but it is a “good enough” estimate
    - It indicates whether the state leads to potentially dangerous states, or states with high 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

Monte Carlo Tree Search - Backpropagation

- We want to use the results from our simulations and update our node statistics and values.
- Update the average total discounted reward and node/action counts for each branch visited.
- Let s_t be the state visited at time step t, and let a_t be the action performed at time step t
- At time step t, the total discounted future reward is given by the following equation:

$$R_t = r(s_t) + \gamma^1 r(s_{t+1}) + \cdots + \gamma^n (s_{t+n}) + \gamma^{n+1} V$$

- For all time steps, we compute the Q-value, and update node statistics.

$$Q(s_t, a_t) \leftarrow \frac{N(s_t) Q(s_t, a_t) + R_t}{N(s_t) + 1}$$

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$

$$N(s_t) \leftarrow N(s_t) + 1$$

MCTS Implementation – Iteratively

- Create a Tree Node class that stores:
  - N(s)
  - Q(s, a) and N(s, a) for each available action
  - Stores a list of child nodes for each available actions
  - Stores a reference to the parent node
- mcts_search(current_state)
  - node ← current_state
  - While the node is not a leaf node, select an action and sample a next state (and set node <- next_state)
  - Expand the leaf node and estimate the value via simulation
    - Create a new TreeNode instance for it
  - 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)
    - We keep repeating this step until we reach the root node.

MCTS Implementation – Recursively

- Dictionaries are used to store node statistics

  $N_S[s]$ → Number of times a state "s" has been visited

  $N_{s,a}[(s,a)]$ → Number of times an action "a" has been performed from state "s"

  $Q_{s,a}[(s,a)]$ → Average reward from performing action a at state s
- mcts_search(current_state)
  - If 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
    - Action can be selected using epsilon-greedy or UCB
  - Sample the outcome of the next state, and set

    $V =$ immediate_reward + $\gamma \times$ mcts_search(next_state)
  - Increment $N_s[s]$ and $N_{s,a}[(s,a)]$ and update $Q_{s,a}[(s,a)]$ using the value V
  - Return the value V (so that the next level above can use the value)

MCTS Implementation

- For both approaches, the mcts_select_action method should:
  - Call mcts_search(current_state) while time / memory / iteration limits are not reached
  - Action ← argmax(Q(s,a)) over all actions
  - Return the action