Monte Carlo Tree Search

2/19/18
The Monte Carlo Tree Search Algorithm

Figure from Chaslot (2006)
Selection
• Used for nodes we’ve seen before.
• Pick according to UCB.

Expansion
• Used when we reach the frontier.
• Add one node per playout.
Simulation

• Used beyond the search frontier.
• Don’t bother with UCB, just play randomly.

Backpropagation

• After reaching a terminal node.
• Update value and visits for all states visited in selection and expansion phases.
function MCTS_sample(node)
    if all children expanded:
        # selection
        next = UCB_sample(node)
        outcome = MCTS_sample(next)
    else:
        # expansion
        next = random unexpanded child
        create node for next, add to tree
        # simulation
        outcome = random_playout(next.state)
        # backpropagation
        update_value(node, outcome)
MCTS Helper Functions

function UCB_sample(node):
    weights = []
    for child of node:
        w = child.value
        w += C*sqrt(ln(node.visits) / child.visits)
        add w to weights
    distribution = normalize weights to sum to 1
    return child sampled according to distribution
MCTS Helper Functions

function random_playout(state):
    while state is not terminal:
        state = make a random move from state
    return outcome

function update_value(node, outcome):
    #combine the new outcome with the average value
    node.value *= node.visits
    node.visits++
    node.value += outcome
    node.value /= node.visits
Selection  Expansion  Simulation  Backpropagation
Selection  Expansion  Simulation  Backpropagation
Selection  Expansion  Simulation  Backpropagation
Selection  Expansion  Simulation  Backpropagation

\[
C = 5.0
\]

\[
v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}
\]

\[
w_i = v_i + 5\ln(3)\]

weights = [7.24, 5.24, 6.24]
distribution = [.39, .28, .33]
\[ v_i + C \times \sqrt{\frac{\ln(N)}{n_i}} \]

\[ w_i = v_i + 5 \times \ln(4)^{0.5}/n_i^{0.5} \]

Weights: [7.89, 5.89, 6.45]
Distribution: [.39, .29, .32]

Weights: [7.24, 5.24, 6.24]
Distribution: [.39, .28, .33]
UCB Distribution Example

weights = [2.13, 2.48, 1.96, 2.43]

probs = [0.24, 0.28, 0.22, 0.27]

\[ v_i + C \times \sqrt{\frac{\ln N}{n_i}} \]

\[ C = 2 \]
How do we pick a move at the root?

MCTS builds a tree, with visits and values for each node. How can we use this to pick a move?

- Pick the highest-value move.
- Pick the most-visited move.
- Can we do both?
  - Use some weighted combination.
  - Keep simulating until they agree.
Generalizing MCTS Beyond UCT

The tree policy returns a child node in the explored region of the tree.

**UCT** uses a tree policy that draws samples according to UCB.

The default policy returns a value estimate for a newly expanded node.

**UCT** uses a default policy that completes a uniform random playout.

Figure from Chaslot (2006)
Alternative tree policies

Requirement: The tree policy needs to trade off exploration and exploitation.

- Epsilon-greedy: pick a uniform random child with probability $\epsilon$ and the best child with probability $(1-\epsilon)$.
  - We’ll see this again soon.

- Use UCB, but seed the tree within initial values.
  - From previous runs.
  - Using a heuristic.

- Other ideas?
Alternative default policies

Requirement: The default policy needs to run quickly and return a value estimate.

• Use the board evaluation heuristic from bounded minimax.

• Run multiple random rollouts for each expanded node.

• Other ideas?
Exercise: extend MCTS to these games

How can MCTS handle non-zero-sum games?

How can MCTS handle games with randomness?
Non-Zero-Sum Games

**Key idea:** store a value tuple with the average utility for each player.

- Each node now stores visits, children, and one value for each player.

- The agent who’s making a decision will compute UCB weights using only *their component* of the value tuple.
Randomness in the Environment

This is what Monte Carlo simulations were made for!

- Whenever we hit a move-by-nature in the game tree, sample from nature’s move distribution.

- We still need to track value and visits for the nature node, so that the parent can make its choices.