kptan86@gmail.com

Research trend - Algorithm Cocktails

Monte Carlo Tree Search (MCTS)

- An exhaustive "full" tree search would solve the decision problem

Example: Tic-Tac-Toc

- Maximum of 9 steps
- Maximum of 9 nodes

Pseudo-code for full search tree on tic-tac-toc

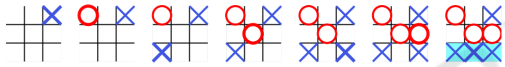
```
for step in [1..9]:
  new_layer = List()
  player *= -1
  for node in last_layers:
    for i in [1..9]:
      state = node.state
      state[i].set_color(player)

      if is_valid(state) {
          _node = new node(state)
          node.add_child(_node)
          new_layer.add(_node)
      }
    }
  }
  layers.add(new_layer)
}
```

- Generates a tree of 9 layers with 549,946 leaf nodes

kptan86@gmail.com

Tic-Tac-Toc full tree



```
[---][---][---]
--[B--][---][---]
----[BW-][---][---]
------[BWB][---][---]
--------[BWB][W--][---]
----------[BWB][WB-][---]
------------[BWB][WBW][---]
--------------[BWB][WBW][B--]
--------------[BWB][WBW][-B-]
----------------[BWB][WBW][WB-]
------------------[BWB][WBW][WBB]
----------------[BWB][WBW][-BW]
------------------[BWB][WBW][BBW]
--------------[BWB][WBW][--B]
------------[BWB][WB-][W--]
--------------[BWB][WBB][W--]
----------------[BWB][WBB][WW-]
------------------[BWB][WBB][WWB]
```

Monte Carlo Tree Search (MCTS)

Core idea:

- An exhaustive "full" tree search would solve the decision problem
  - (*except that it is computationally prohibitive*)
- Expand tree only up to a few layers (steps)
- Identify only the more **valuable** nodes to sample, criteria being
  - [**exploitation**] known to give higher estimated reward, or
  - [**exploration**] higher uncertainty of the upper reward bound
- Given the resources (e.g. time), sample and refine the estimated value of each move
- Finally, choose the move with the highest refined estimated value

Monte Carlo Tree Search (MCTS)

Core idea:

- An exhaustive "full" tree search would solve the decision problem
  - (except that it is computationally prohibitive)
- Expand tree only up to a few layers (steps)
- Identify only the more **valuable** nodes to sample, criteria being
  - [**exploitation**] known to give higher estimated reward, or
  - [**exploration**] higher uncertainty of the upper reward bound
- Given the resources (e.g. time), sample and refine the estimated value of each move
- Finally, choose the move with the highest refined estimated value

Components

1. Estimator on the reward
2. Estimator on the upper reward uncertainty bound
3. A way to sample and refine the estimated value

kptan86@gmail.com

Monte Carlo Tree Search (MCTS)

Estimator of reward

The reward estimator updates as simulation proceeds, and usually takes the form of

$$Q(s, a) = \frac{w(s, a)}{n(s, a)}$$

where s is current state, a is the move action, $w(s, a)$ is the winning count and $n(s, a)$ is the observation count.

Estimator of uncertainty bound

- This purpose of this term is to encourage sampling on less visited nodes.
- It should scales with total number of samples N, and inversely with current node visit count n_i, e.g. $\mu(s, a) = c\frac{N^\alpha}{n^\beta(s,a)}; N = \sum_b n(s, b)$
- A commonly used expression is $\mu(s, a) = c\sqrt{\frac{\log N}{n(s,a)}}$
- it is possible to use improve the estimator by assigning a *prior* on this bound *if we have some knowledge on the context* (s, a).

A way to sample and refine the estimated value

- Usually the expression of uncertainty bound is unchanged $\mu(s, a)^{\text{new}} = \mu(s, a)^{\text{old}}$
- Estimation of reward is updated with a **simulation**
  - simulate the final outcome (win/loss) after choosing the move
    $$Q^{\text{new}}(s, a) = \frac{w(s, a) + v}{n(s, a) + 1}; v = \begin{cases} 1 & \text{if win} \\ 0 & \text{if loss} \end{cases}$$
  - *In case without any prior knowledge*, a "rollout" (random playout of the game) can be used for the simulation

Monte Carlo Tree Search (MCTS)

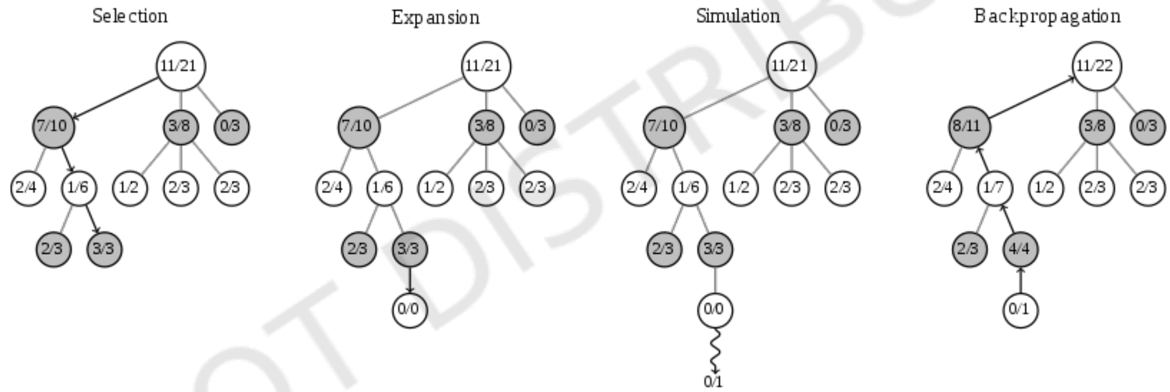
Vanilla implementation

1. At every iteration we compute *UCT* for every move a of the current state s as
   $$U(s, a) = Q(s, a) + \mu(s, a) = \frac{w(s, a)}{n(s, a)} + \mu(s, a)$$
2. We then **select** the move a_t that maximizes the $U(s, a)$ and transverse to the next (leaf) node s_L.
   $$a_t = \underset{a}{\text{argmax}}\, U(s, a)$$
3. (We may optionally **expand** the leaf node. The number of level expansion is called the "depth" of the tree)
4. Start a **simulation** from s_L until end state
5. $Q(s, a)$ (and hence $U(s, a)$) value is updated by **backpropagation** through the node trail based on the simulation outcome v
   $$Q^{\text{new}}(s, a) = \frac{w(s, a) + v}{n(s, a) + 1}; v = \begin{cases} 1 & \text{if win} \\ 0 & \text{if loss} \end{cases}$$
6. Repeat the process as resources allow

kptan86@gmail.com

Monte Carlo Tree Search (MCTS)

Schematic diagram for MCTS



AlphaZero

There are two opportunities for function approximator in MCTS algorithm:

1. Estimator of uncertainty bound $\mu(s, a)$
2. Simulation $v(s)$

Estimator of uncertainty bound

- AlphaZero uses the expression:

$$\mu(s, a) = \frac{c\sqrt{N}}{1 + n(s, a)} \cdot \boxed{P(s, a)}$$

  - The first term encourages sampling on less visited nodes
  - The second term is a *prior* modifier of the action selection

A way to sample and refine the estimated value

- In AlphaZero, the final outcome is directly predicted from a neural network
  - No explicit simulation is performed
    - (explicit simulation was performed in AlphaGo implementation)

$$Q^{\text{new}}(s, a) = \frac{w(s, a) + \boxed{v(s)}}{n(s, a) + 1}$$

The boxed terms are obtained from a neural network

kptan86@gmail.com

AlphaZero

AlphaGo networks

- In AlphaGo implementation, two neural networks were built:
  - $P(s, a)$ is obtained from a "policy network"
  - $v(s)$ is obtained from a "value network"
- Policy network $\text{net}_{\text{policy}}$ takes the current game state (19x19=361* positions with 3 possible values {black, white, empty}) as input, and computes a 362 (=361 positions + pass) real-valued vector as output.
- Value network $\text{net}_{\text{value}}$ takes the same input and computes a real-value vector as output.

$$S = \{0, 1, -1\}$$
$$\text{net}_{\text{policy}} : S^{361} \rightarrow R^{362}$$
$$\text{net}_{\text{value}} : S^{361} \rightarrow R$$

AlphaZero network

- In AlphaZero implementation, **the two networks are merged into one**.
- The network takes the same input, but output a 363 real-valued vector (361 positions + pass + value output)

$$\text{net}_{\text{zero}} : S^{361} \rightarrow R^{363}$$

- The loss function l combines mean-squared error from value network and cross-entropy losses from policy network

$$l = (z - v)^2 - \pi^T \log P + c||\theta||^2$$

where (z, π) are the actual outcome and probability of moves from training data respectively

(** in actual implementation, 7 steps are used in order to resolve the three-way knot scenario in Go*)

AlphaZero

Reinforcement Learning

- MCTS algorithm could generate gameplays by self-play
- Gameplays are then utilized as training data

kptan86@gmail.com

AlphaZero

Other details

- Artificially encourage more explorations
  - Stochastic policy with temperature control
  - Dirichlet prior
- Symmetry
  - Rotational and reflection invariance are utilized
- Neural network model succession
  - A new network is pitched against its successor
  - New network takes over if winning rate > threshold (55% in AlphaZero)
- Asynchronous
  - self-play, neural network training and model succession are performed in parallel
- Hardware / Distributed computing
- ...

DO NOT DISTRIBUTE