AlphaGo: Combining Searching, Simulation and Prediction

陈云川
2016.04.06
“AlphaGo's algorithm uses a Monte Carlo tree search to find its moves based on knowledge previously "learned" by machine learning, specifically by an artificial neural network (a deep learning kind) by extensive training, both from human and computer play.”

– https://en.wikipedia.org/wiki/AlphaGo
Outline

- Monte Carlo Tree Search (MCTS)
- Combination of MCTS and CNN
Monte Carlo Tree Search
Formal Definition of a Game

• A game may be described by the following components

  • $S$: the set of states, where $s_0$ is the initial state
  • $S_T \subseteq S$: the set of terminal states
  • $k$: The number of players
  • $A$: the set of actions
  • $f$: the state transition function
  • $R$: the utility function
  • $\rho$: player about to act in each state

Games with two players that are zero sum, perfect information, deterministic, discrete, and sequential are described as combinatorial games.
Search: basic tech. to make machine play

- Exhaustive search (minimax: $O(b^d)$)
- Heuristic Search
  - depth ($d$) of the search may be reduced by position evaluation
    - look only at a certain number of moves ahead
  - breadth ($b$) of the search may be reduced by sampling actions from a policy
    - define a heuristic value function to evaluate the proposal move
General MCTS Algorithm

- **Selection**: A child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached.
- **Expansion**: One (or more) child nodes are added to expand the tree, according to the available actions.
- **Simulation**: A simulation is run from the new node(s) according to the default policy to produce an outcome.
- **Backpropagation**: The simulation result is back-propagated through the selected nodes to update their statistics.

```
function MCTSSearch(s₀)
    create root node v₀ with state s₀
    while within computational budget do
        vₙ ← TREEPOLICY(v₀)
        Δ ← DEFAULTPOLICY(s(vₙ))
        BACKUP(vₙ, Δ)
    return a(BESTCHILD(v₀))
```

- **Tree policy**: Select or create a leaf node from the nodes already contained within the search tree.
- **Default policy**: Play out the domain from a given non-terminal state to produce a value estimate (simulation).
Tree Policy

- UCT (base on Upper Confidence Bound)

$$\text{UCT} = \bar{X}_j + 2C_p \sqrt{\frac{2 \ln n}{n_j}}$$

**function** TREEPOLICY($v$)

while $v$ is nonterminal do

if $v$ not fully expanded then

return EXPAND($v$)

else

$v \leftarrow$ BESTCHILD($v$, $C_p$)

return $v$

**function** BESTCHILD($v$, $c$)

return $\arg \max_{v' \in \text{children of } v} \frac{Q(v')}{N(v')} + c \sqrt{\frac{2 \ln N(v)}{N(v')}}$
Backpropagation

• Each node is associated with two values
  • $N(v)$: visited times
  • $Q(v)$: state value

• Algorithm

```
function BACKUP(v, Δ)
    while v is not null do
        $N(v) \leftarrow N(v) + 1$
        $Q(v) \leftarrow Q(v) + \Delta(v, p)$
        $v \leftarrow$ parent of $v$
```

```
function MCTSSEARCH($s_0$)
    create root node $v_0$ with state $s_0$
    while within computational budget do
        $v_i \leftarrow$ TREEPOLICY($v_0$)
        $Δ \leftarrow$ DEFAULTPOLICY($s(v_i)$)
        BACKUP($v_i, Δ$)
    return $a(\text{BESTCHILD}(v_0))$
```
Next Move Selection

- **max child**: select the root child with the highest reward;

- **robust child**: select the most visited root child;

- **max-robust child**: select the root child with both the highest visit count and the highest reward; if none exists, then continue searching until an acceptable visit count is achieved;

- **secure child**: select the child which maximizes a lower confidence bound.
Characteristic of MCTS

- **Aheuristic**: No domain-specific knowledge are required, making it readily applicable to any domain that may be modeled using a tree.\(^1\)

- **Anytime**: MCTS backpropagates the outcome of each game immediately

- **Asymmetric**: The tree selection allows the algorithm to favor more promising nodes

---

1. Although MCTS can be applied in its absence, significant improvements in performance may often be achieved using domain-specific knowledge.
Asymmetric tree growth
Supervised Learning (SL) of Policy Networks

- Given 30 million positions from the KGS Go Server, How can we design a neural network to predict experts’ move?

- **Answer**: A network alternates between convolutional layers with rectifier nonlinearities. (13 convolutional layers). Outputs a probability distribution over all legal moves.

- **Result**: Accuracy:
  - 57.0% using all input features
  - 55.7% using only raw board position and move history as inputs
Reinforcement Learning (RL) of Policy Networks

• This stage of the training pipeline aims at improving the policy network by policy gradient RL

• Create a pool of SL policy networks

• We play games between the current policy network and a randomly selected previous iteration of the policy network

  • Reward \( r(s_t) = \)
    • 0 for all non-terminal time steps \((t<T)\)
    • +1 for winning and -1 for losing \((t=T)\)

  • Outcome \( z_t = \pm r(s_T) \)
Reinforcement Learning of Policy Networks (cont.)

• Weights are then updated at each time step $t$ by stochastic gradient ascent in the direction that maximizes expected outcome

$$\Delta \rho \propto \frac{\partial \log p_{\rho}(a_t|s_t)}{\partial \rho} z_t$$

• Effectiveness

  • RL policy network won more than 80% of games against the SL policy network

  • RL policy network won 85% of games against *Pachi*——the strongest open-source Go program, ranked at 2 amateur *dan* on KGS
Overall Networks
Reinforcement learning of value networks (VN)

• This stage of the training pipeline focuses on position evaluation

\[ v^p(s) = \mathbb{E}[z_t | s_t = s, a_t...T \sim p] \]

• We approximate the value function using a value network \( v_\theta(s) \) with weights \( \theta \).

• The reference values for the VN are obtained by simulating the RL policy network.

\[ \Delta \theta \propto \frac{\partial v_\theta(s)}{\partial \theta} (z - v_\theta(s)) \]
Results of Position Value Estimation
Searching with policy and value networks

- AlphaGo combines the policy and value networks in an MCTS algorithm that selects actions by lookahead search.
Action Value Updating

\[ a_t = \arg\max_a (Q(s_t, a) + u(s_t, a)) \]

\[ u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)} \]

\[ V(s_L) = (1 - \lambda) v_\theta(s_L) + \lambda z_L \]

\[ N(s, a) = \sum_{i=1}^{n} 1(s, a, i) \]

\[ Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{n} 1(s, a, i) V(s_L^i) \]
Overall Pipeline

$p_\pi$ $p_\sigma$ $p_\rho$ $\nu_\theta$

Classification

Policy gradient

Self Play

Regression
Tournament evaluation of AlphaGo
Thanks!