Images from:
- David Foster, Applied Data Science
- Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm by Silver, et al., 2017
- Mastering the game of Go without human knowledge, Silver, et al., 2017
Outline

• Monte Carlo Tree Search refresher

• DeepMind’s AlphaGo (and successors)
MCTS Refresher

- Monte Carlo Tree Search takes a state, root, and rollout policy $\pi_{rollout}$ and produces an improved action.

- Algorithm:

  repeat many times:

  selected = \textbf{Select}(root)
  expanded = \textbf{Expand}(selected)
  reward = \textbf{Simulate}(expanded, $\pi_{rollout}$)
  \textbf{Backup}(expanded, reward)

  Choose root’s child action with highest $n(v)$.
MCTS nodes in tree

• Each node, $v$, stores:

  • $n(v)$: number of times that node has participated in a simulation

  • $q(v)$: total reward attributed to that node during simulations

\[
\frac{q(v)}{n(v)} = \text{avg value for } v
\]
MCTS Select

Select(v)
  if node is terminal or has an unexpanded child, return v
  else return child of v with highest

  \[
  UCT(v) = \frac{q(v)}{n(v)} + c\sqrt{\frac{\log n(v \cdot parent)}{n(v)}}
  \]

exploration
exploitation
leaf

MCTS Expand

Expand(v)
    if node is terminal:
        return v
    c = create random child node of v
    q(c) = n(c) = 0
    return c
MCTS Simulate

Simulate(v, \( \pi_{rollout} \))

Choose actions starting from v according to policy \( \pi_{rollout} \) until a terminal state is reached

return immediate reward leading to that terminal state
MCTS Backup

Backup(v, reward)
for all nodes from v up to the root:
  \( n(v) \) += 1
  \( q(v) \) += reward
Outline

- Monte Carlo Tree Search refresher
- DeepMind’s AlphaGo (and successors)
History

- AlphaGo, 2015
- Hand-crafted features and trained on human plays
- Two Neural Networks: One to learn the value function, one to learn policy function
- Policy function trained on large body of human plays
History

- AlphaGo Zero, 2017

- No prior knowledge of Go (no hand-crafted features, no training on human games). All self-play

- Single Residual Neural Network learns both value and policy functions

- Keeps best-neural-network-so-far (new champion if beats >55% of games against old champion)
History

- AlphaZero, 2017

- No current best Neural Net. Always uses the latest NN.

- Same network (other than input and head) with almost same hyper-parameters can learn Go, Chess, and Shogi

- One hyper-parameter is scaled based on number of possible actions
History

- MuZero, 2019
  - Doesn’t know the rules of the game (During MCTS, must use learned representation of the game dynamics)
  - Extended to work with Atari games as well as Go.
### 3 Policy Networks

<table>
<thead>
<tr>
<th></th>
<th>SL</th>
<th>RL</th>
<th>Rollout</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialized</strong></td>
<td>Random</td>
<td>From SL</td>
<td>Random</td>
</tr>
<tr>
<td><strong>Structure</strong></td>
<td>NN</td>
<td>NN</td>
<td>Linear</td>
</tr>
<tr>
<td><strong>Training</strong></td>
<td>Human games</td>
<td>RL with self-play</td>
<td>Human games</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>Hand-crafted features</td>
<td>Hand-crafted features</td>
<td>Low-level handcrafted features</td>
</tr>
<tr>
<td><strong>Time to select</strong></td>
<td>3 ms</td>
<td>3 ms</td>
<td>2 µs</td>
</tr>
</tbody>
</table>

Given the game state, what is the value of an action?
S → RL network → action for state S → MCTS → better action for S

training example

input S, target better action for S
Hand-crafted Features

• SL/RL example features:
  • Is a move at this point a successful ladder capture
  • How many opponent stones would be captured playing at this point

• Rollout example features:
  • Move matches 3x3 pattern around the move (69938 features)
  • Move saves stones from capture
Value network

- Like SL network, but has only a single output:

  What is the value of a game state?
Overview of Training

- Initialize Neural Network

- Repeat in parallel:
  1. Self-play a game. Do MCTS for each move. MCTS(\(\pi\), s) is a new policy, \(\pi\), more accurate than \(P\).
  2. Create mini batch of 2048 states from the most recent 500,000 games

• Use mini batch to train RL network and value network
How is Policy network used?

- As tree policy in MCTS to guide action selection

- When choosing an action from a node $s_t$ in the MCTS tree:

  $$a_t = \arg\max_a (Q(S_t, a) + U(s_t, a))$$

- bonus (incorporates tree policy and exploration bonus):

  $$U(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$
How is Value network used?

• To help get better simulation results

• After we have expanded node: $n_e$, we do simulation:

  • Do rollout (to obtain win/draw/lose) to estimate value: $v_1(n_e)$

  • Also use value network to estimate value: $v_2(n_e)$

• Use weighted average as value: $\lambda v_1(n_e) + (1 - \lambda)v_2(n_e)$
Each state has edges for all legal actions:
For each edge keep:
  \( N(s, a) \): how many times has this state/action pair been seen
  \( W(s, a) \): total action-value
  \( Q(s, a) \): mean action-value: \( W(s, a)/N(s, a) \)
  \( P(s, a) \): prior probability of selecting that edge

\( \text{Calculated} \)

\( \text{Comes from tree policy (CNN)} \)
MCTS for AlphaGo

- Monte Carlo Tree Search takes a state, root, a tree policy $\pi_{tree}$, a rollout policy $\pi_{rollout}$, a state value approximator and produces an improved action

- Algorithm:

repeat many times:

expanded = Select(root)
reward = Evaluate(expanded, $\pi_{rollout}$, vapprox)
Backup(expanded, reward)

Choose root’s child action with highest $n(v)$
MCTS: Select

- Different from MCTS we’ve seen:

\[
\text{Select}(n):
\]

\[
\begin{align*}
\text{if } n \text{ is terminal:} &
\text{ return } n \\
\text{child} &\text{ = child with highest } Q+U \text{ (including unexpanded children)} \\
\text{if child is expanded:} &
\text{ return Select(child)} \\
\text{return Expand(child)}
\end{align*}
\]
MCTS Evaluate

\[
\text{Evaluate}(n, \pi_{\text{rollout}}, v_{\text{approx}}) = \begin{cases} \\
\text{Choose actions starting from } n \text{ according to policy } \pi_{\text{rollout}} \text{ until a terminal state is reached} \\
\text{do a rollout} \\
\end{cases}
\]

\(v_1\) is value of that terminal state according to Go rules

\(v_2\) is \(v_{\text{approx}}(n)\)

return \(\lambda v_1 + (1 - \lambda)v_2\)
MCTS: Select

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

\[ Q + U \]

Constant of proportionality decreases over time

\( U \) is upper-confidence bound

AlphaGo
MCTS:
Expand & Evaluate

\[ \text{Value NN} \]
\[ v_\theta \]

\[ \sim p_\pi \]
\[ r \]

\[ \text{Expanded node} \]
MCTS: Backup

For each ancestor edge:

\[ N(s, a) + = 1 \]
\[ W(s, a) + = r \]
Self-play

$\pi_1 \rightarrow s_1 \xrightarrow{a_1 \sim \pi_1} s_2 \xrightarrow{a_2 \sim \pi_2} s_3 \xrightarrow{a_t \sim \pi_t} s_T$

Train Value $NN$ with $(s, z)$ pairs
Train RL $NN$ with $(s, a, r_i)$ pairs
Final result of game
Learn through self-play

- Get rid of handcrafted features
- No network weights pretrained from human games
- One shared network to compute both policy and value
- Simplified MCTS: no rollouts!
AlphaGo Zero: What is a game state for Go?

Current position of black’s stones

19 x 19 x 17 stack

...and for the previous 7 time periods

1 if black stone here
0 if black stone not here

Current position of white’s stones

...and for the previous 7 time periods

All 1 if black to play
All 0 if white to play
The Neural Network

Value head

Policy head

Neural Network

Input (Game state)

Multi-task learning!

AlphaGo

Zero
Value (V) Head

$\mathbb{R} \in [0, 1]$: probability of winning for current player

- 1x1 Conv
- BatchNorm
- Fully-connected to 256 hidden units
- Fully-connected to 1 hidden unit
- sigmoid
Policy (P) Head

- 2 1x1 Conv
- BatchNorm
- Fully-connected
- Softmax
- 19x19+1 probabilities
- 19x19 places to play stone
- 1 way to pass
Overview of Training

• Initialize Neural Network

• Repeat in parallel:

   MCTS(P, s) is a new policy, π, more accurate than P.
   • For each state, record state, π(s), win_lose_or_draw_result

2. Create mini batch of 2048 states from the most recent 500,000 games
   • Use mini batch to train Neural Network

For chess, took 9 hours to run
Total of 700,000 mini batches
Self-play

$\pi_1$ $\pi_2$ $\pi_3$ $\pi_t$
Neural Net Training

\[ L = (z - v)^2 - \pi^T \log p + c\|\Theta\|^2 \]
Neural Net Training

• Update $\Theta$ so that:

  • Given an input state, $s_i$, $f_\Theta$ produces output that is closer to:

    • For value portion: $z$, the actual result of the game
    • For action portion: $\pi_i$, the MCTS-improved policy based on the neural net’s $p_i$
Monte Carlo Tree Search (MCTS)

Each state has edges for all legal actions:
For each edge keep:
  \( N(s, a) \): how many times has this state/action pair been seen
  \( W(s, a) \): total action-value
  \( Q(s, a) \): mean action-value: \( W(s, a)/N(s, a) \)
  \( P(s, a) \): prior probability of selecting that edge
Explore/Exploit

- Two sources to encourage exploration:
  - Dirichlet noise added to top-level $P(s)$ of MCTS
    - Non-zero chance of any move happening
  - Upper-confidence bound when evaluating move in MCTS
    - Encourage actions whose confidence is low

Dirichlet noise: sums to given value (so can be a probability), and, with parameter used in AlphaGoZero, makes most of the probability focused in small area
MCTS: Evaluate

\[(p, v) = f_\theta \left( \begin{array}{c} \text{V} \\ \text{V} \\ \text{V} \end{array} \right) \]
MCTS: Backup

For each ancestor edge:

\[ N(s, a) + = 1 \]
\[ W(s, a) + = V(s_{leaf}) \]

\[ Q \]
\[ V \]

AlphaGo
Zero
After an action is chosen from $\pi$, keep tree starting from resulting state, clear rest of tree.

$\pi(a \mid s_0) = \frac{N(s_0, a)^{1/\tau}}{\sum_b N(s_0, b)^{1/\tau}}$

$\tau$ is a temperature constant: starts near 1 and ends close to zero. For play: close to zero.

MCTS: Play
What Go-specific knowledge is used?

- Network knows input format (19x19 board, etc.)
- Network knows how many possible actions (19x19+1)
- MCTS knows which actions lead to which states
- MCTS knows whether a state is terminal
- MCTS knows how to score a terminal state
- MCTS does dihedral reflection or rotation (takes advantage of symmetry of Go board)