Mastering the game of Go with deep neural networks and tree search

Thomas Mülleder
Graz, 28. January 2020
Content

1. The game of Go
2. Components of AlphaGo
3. The Training Pipeline
4. Integration of the Components
5. Evaluation
6. Conclusion
The game of Go

Thomas Mülleder
Graz, 28. January 2020
The game of Go
Capture of Stones
Capture of Stones
Capture of Groups
The game of Go

Capture of Groups
Capture of Groups
Capture of Groups
Capture of Groups
Capture of Groups
The game of Go

Immortal Groups

[Diagram of a Go board with black and white stones]
Scoring
Search Space

- breath $b = 250$ (# legal moves per game state)
- depth $d = 150$ (# moves until the game is over)
- $b^d$ possible sequences of moves
Search Space - Chess

- breath $b = 35$ (# legal moves per game state)
- depth $d = 80$ (# moves until the game is over)
- Superhuman performance using position evaluation
Existing Go Programs

- Based on MCTS
- Enhanced with shallow policies
- Weak amateur level play
Components of AlphaGo

- Three different policies
  - SL policy $p_\sigma$
  - Fast rollout policy $p_\pi$
  - RL policy $p_\rho$
- A value network
Network Architectures

Policy network: $p_{o/p}(a|s)$

Value network: $v_{\theta}(s')$
Components of AlphaGo

Features used by the SL and RL policy

<table>
<thead>
<tr>
<th>Feature</th>
<th># of planes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stone colour</td>
<td>3</td>
<td>Player stone / opponent stone / empty</td>
</tr>
<tr>
<td>Ones</td>
<td>1</td>
<td>A constant plane filled with 1</td>
</tr>
<tr>
<td>Turns since</td>
<td>8</td>
<td>How many turns since a move was played</td>
</tr>
<tr>
<td>Liberties</td>
<td>8</td>
<td>Number of liberties (empty adjacent points)</td>
</tr>
<tr>
<td>Capture size</td>
<td>8</td>
<td>How many opponent stones would be captured</td>
</tr>
<tr>
<td>Self-atari size</td>
<td>8</td>
<td>How many of own stones would be captured</td>
</tr>
<tr>
<td>Liberties after move</td>
<td>8</td>
<td>Number of liberties after this move is played</td>
</tr>
<tr>
<td>Ladder capture</td>
<td>1</td>
<td>Wheter a move at this point is a successful ladder capture</td>
</tr>
<tr>
<td>Ladder escape</td>
<td>1</td>
<td>Whether a move at this point is a successful ladder escape</td>
</tr>
<tr>
<td>Sensibleness</td>
<td>1</td>
<td>Whether a move is legal and does not fill its own eyes</td>
</tr>
<tr>
<td>Zeros</td>
<td>1</td>
<td>A constant plane filled with 0</td>
</tr>
</tbody>
</table>

Thomas Mülleder
Graz, 28. January 2020
## Components of AlphaGo

### Features used by the Value network

<table>
<thead>
<tr>
<th>Feature</th>
<th># of planes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stone colour</td>
<td>3</td>
<td>Player stone / opponent stone / empty</td>
</tr>
<tr>
<td>Ones</td>
<td>1</td>
<td>A constant plane filled with 1</td>
</tr>
<tr>
<td>Turns since</td>
<td>8</td>
<td>How many turns since a move was played</td>
</tr>
<tr>
<td>Liberties</td>
<td>8</td>
<td>Number of liberties (empty adjacent points)</td>
</tr>
<tr>
<td>Capture size</td>
<td>8</td>
<td>How many opponent stones would be captured</td>
</tr>
<tr>
<td>Self-atari size</td>
<td>8</td>
<td>How many of own stones would be captured</td>
</tr>
<tr>
<td>Liberties after move</td>
<td>8</td>
<td>Number of liberties after this move is played</td>
</tr>
<tr>
<td>Ladder capture</td>
<td>1</td>
<td>Whether a move at this point is a successful ladder capture</td>
</tr>
<tr>
<td>Ladder escape</td>
<td>1</td>
<td>Whether a move at this point is a successful ladder escape</td>
</tr>
<tr>
<td>Sensibleness</td>
<td>1</td>
<td>Whether a move is legal and does not fill its own eyes</td>
</tr>
<tr>
<td>Zeros</td>
<td>1</td>
<td>A constant plane filled with 0</td>
</tr>
<tr>
<td>Player color</td>
<td>1</td>
<td>Whether current player is black</td>
</tr>
</tbody>
</table>
## Features used by the rollout policy

<table>
<thead>
<tr>
<th>Feature</th>
<th># of patterns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>1</td>
<td>Whether move matches one or more response pattern features</td>
</tr>
<tr>
<td>Save atari</td>
<td>1</td>
<td>Move saves stone(s) from capture</td>
</tr>
<tr>
<td>Neighbour</td>
<td>8</td>
<td>Move is 8-connected to previous move</td>
</tr>
<tr>
<td>Nakade</td>
<td>8192</td>
<td>Move matches a nakade pattern at captured stone</td>
</tr>
<tr>
<td>Response pattern</td>
<td>32207</td>
<td>Move matches 12-point diamond pattern near previous move</td>
</tr>
<tr>
<td>Non-response pattern</td>
<td>69338</td>
<td>Move matches $3 \times 3$ pattern around move</td>
</tr>
</tbody>
</table>
## Components of AlphaGo

### Features used by the tree policy

<table>
<thead>
<tr>
<th>Feature</th>
<th># of patterns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>1</td>
<td>Whether move matches one or more response pattern features</td>
</tr>
<tr>
<td>Save atari</td>
<td>1</td>
<td>Move saves stone(s) from capture</td>
</tr>
<tr>
<td>Neighbour</td>
<td>8</td>
<td>Move is 8-connected to previous move</td>
</tr>
<tr>
<td>Nakade</td>
<td>8192</td>
<td>Move matches a nakade pattern at captured stone</td>
</tr>
<tr>
<td>Response pattern</td>
<td>32207</td>
<td>Move matches 12-point diamond pattern near previous move</td>
</tr>
<tr>
<td>Non-response pattern</td>
<td>69338</td>
<td>Move matches $3 \times 3$ pattern around move</td>
</tr>
<tr>
<td>Self-atari</td>
<td>1</td>
<td>Move allows stones to be captured</td>
</tr>
<tr>
<td>Last move distance</td>
<td>34</td>
<td>Manhattan distance to previous two moves</td>
</tr>
<tr>
<td>Non-response pattern</td>
<td>32207</td>
<td>Move matches 12-point diamond pattern centred around move</td>
</tr>
</tbody>
</table>
The Training Pipeline

- Rollout policy
- SL policy network
- RL policy network
- Value network

$p_\pi$, $p_\sigma$, $p_\rho$, $v_\theta$

Policy gradient

- Classification
- Classification
- Self Play
- Regression

Human expert positions
Self-play positions

Neural network
Data
Training the SL and fast rollout policy

- 30 million training pairs
- Maximize the likelihood of the human move
The Training Pipeline

Training the RL policy

- Initialized with the weights of the SL policy
- 10000 minibatches of 128 self-play games
- Trained using REINFORCE
Training the value network

- 30 million board positions drawn from unique games
- Minimize the mean squared error
Integration of the Components

Asynchronous Policy and Value MCTS
Edge Statistics

- Prior probability $P(s, a)$
- Number of leaf evaluations $N_v(s, a)$
- Number of rollout rewards $N_r(s, a)$
- Total action value MC estimates $W_v(s, a)$, $W_r(s, a)$
- Combined mean action value for that edge $Q(s, a)$
### Selection

- **Exploitation:**
  \[
  Q(s, a) = (1 - \lambda) \frac{W_v(s,a)}{N_v(s,a)} + \lambda \frac{W_r(s,a)}{N_r(s,a)}
  \]

- **Exploration:**
  \[
  u(s, a) = c_{puct} P(s, a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)}
  \]
Integration of the Components

Expansion

- Initialize:
  \[ P(s', a) = p_{\text{tree}}(s', a) \]

- Update asynchronously:
  \[ P(s', a) = p_\sigma(s', a) \]
Evaluation

\[ v_{\theta} \left( \begin{array}{c} \end{array} \right) \sim p_\pi \]
\[ r \left( \begin{array}{c} \end{array} \right) \]
Integration of the Components

Backup

- \( N(s, a) = \sum_{i=1}^{n} 1(s, a, i) \)
- \( Q(s, a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s, a, i) V(s_{L}^{i}) \)
- \( V(s_{L}) = (1 - \lambda)v_{\theta}(s_{L}) + \lambda z_{L} \)
Evaluation

- Win rate of 99.8% against other Go programs
- Above 95% win rate against versions using only position evaluations or rollouts
- Won 5 out of 5 games against Fan Hui
Conclusion

- Strength mainly attributed to the intelligent selection of positions to explore
- Efficient reduction of search breath using policy networks
- Combination of value network and MC rollouts for leaf node evaluation