AlphaGo

Jonathan Hui

https://medium.com/@jonathan_hui/alphago-how-it-works-technically-26ddcc085319
Go – surround territory on 19x19 board
Learn policy

- \( a = f(s) \)
- \( a = \) where to play (19*19)
- \( s = \) description of board
<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>
Train “SL policy network”

- 13 layers of convolutional filters and rectifiers
  - softmax classifier
- Train using moves for 30 million board positions
- 50 GPUs for 3 weeks
Approximate using rollout policy net

- SL policy network (55.7% accuracy) 3 ms
- Rollout policy network (24.2% accuracy) 2 μs
  - Also trained on human expert positions
**RL policy network**

- Initialize with SL policy network
- Train using self-play with older network
- Let $z_t = 1$ if we win game or $-1$ if we lose

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

**Policy gradient RL**

\[
\Delta \sigma \propto \frac{\partial \log p_\sigma(a | s)}{\partial \sigma}
\]

**DL backpropagation**
Value network

- Estimate value of board state under the policy followed by the policy network
- **Input:** one board from each self-play game
- **Output:** $z$ (win/loss for that game)
- again, 50 GPUs for one week.
Monte Carlo Tree Search

- Need to trade off exploration and exploitation
  - But can’t afford to do a full search
- Use
  - predictions from the policy and value networks
  - how many times we have picked the move
  - simulated game results with wins.
Monte Carlo Tree Search
Selection

\[ a_t = \underset{a}{\text{argmax}} (Q(s_t, a) + u(s_t, a)) \]

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

\[ P(s, a) = p_{\sigma}(a|s) \]

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

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

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

- \( p_\sigma(a|s) \): From the policy network: how good to take action \( a \).
- \( v_\theta(s_L) \): From the value network: how good to be in positions \( s_L \).
- \( N(s, a) \): How many times have we select action \( a \) so far.
- \( z_L \): the previous simulated game result.
Expansion

- \( Q \) from RL value net
  - more accurate
  - use for exploitation

- \( P \) from SL policy network
  - more diverse
  - use for exploration (\( u \))
Evaluation

- Simulate the rest of the game using **Monte Carlo Rollout** starting from the leaf node
- Sample moves using the rollout policy.
  - Use the fast (but inaccurate) rollout net
    - 1500x faster
- Predicts a win or a loss $z_L$
Backup

- Update $Q$ with

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

$$V(s_L) = (1 - \lambda)\nu_\theta(s_L) + \lambda z_L$$

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{n} 1(s, a, i) V(s^i_L)$$
Picking the next move

- Could use $Q(s,a)$ but don’t
- Use the move that was most often picked for the current board position
  - Leads to increasing exploitation over time
Monte Carlo Tree Search

Diagram:

- **Selection**
  - $Q + u(P)$
  - $Q + u(P)$
  - $Q + u(P)$

- **Expansion**
  - $P_a$ (expanded)
  - $P$ (expanded)
  - $P$ (expanded)

- **Evaluation**
  - $v_\theta$ (expanded)
  - $v_\theta$ (expanded)
  - $r$ (expanded)

- **Backup**
  - $v_\theta$ (backup)
  - $v_\theta$ (backup)
  - $v_\theta$ (backup)
  - $r$ (backup)
AlphaGo take-away

- **Boot-strap**
  - Start with policy learned from human play

- **Self-play**

- **Speed matters**
  - Rollout network
  - Monte Carlo search

- **It still helps to have fast computers**
  - 100 GPU weeks