Mastering the game of Go with deep neural networks and tree search (Silver et al., 2016)

Florian Brunner

University of Heidelberg

sc223@uni-heidelberg.de

July 4, 2019
The Game of Go

Image 1: [2], Image 2: [3], Image 3: [4]
Timeline

- 1952 – computer masters Tic-Tac-Toe
- 1994 – computer masters Checkers
- 1997 – IBM’s Deep Blue defeats Garry Kasparov in Chess
- 2011 – IBM’s Watson defeats Jeopardy champions
- 2014 – Google algorithms learn to play Atari games
- 2015 – Wikipedia: "Thus, it is very unlikely that it will be possible to program a reasonably fast algorithm for playing the Go endgame flawlessly, let alone the whole Go game."
- 2015 – Google’s AlphaGo defeats Fan Hui (2-dan professional) in Go
"This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away."

– Silver et al., 2016

Figure: David Silver

Image 1: [5], Image 2: [6]
Overview

1. The Game of Go
   - Go Basics
   - Complexity of Go

2. The Architecture of AlphaGo
   - Monte Carlo Tree Search
   - Policy and Value Networks
   - Combining Neural Networks with MCTS
   - Playing Strength Evaluation

3. AlphaGo vs Lee Sedol
Go Basics

Image [7]
Go Basics
Go Basics

Image [7]

Florian Brunner

AlphaGo

July 4, 2019
Go Basics

Image [7]

Florian Brunner

AlphaGo

July 4, 2019
Why is Go so hard?

- Board size usually 19x19
- Almost every move is legal
- Average branching factor of Go: 250
- Amount of possible game states: $10^{171}$ (Chess: $10^{43}$)
Complexity of Go

<table>
<thead>
<tr>
<th></th>
<th>breadth</th>
<th>depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-Tac-Toe</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Checkers</td>
<td>2.8</td>
<td>70</td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
<td>80</td>
</tr>
<tr>
<td>Go</td>
<td>250</td>
<td>150</td>
</tr>
</tbody>
</table>

**Table:** Game tree’s breadths and depths

⇒ For Go: $b^d \approx 10^{360}$
Reducing Search Space

- **Reduce depth:** position evaluation
  - Truncate the search tree at state $s$ and replace subtree below $s$ by an approximate value function $v(s) \approx v^*(s)$

- **Reduce breadth:** sampling actions from a policy
  - Policy $p(a|s)$: probability distribution over possible moves $a$ in state $s$
Monte Carlo Tree Search

- Use Monte Carlo rollouts to estimate the value of each state in a search tree
- Policy during search improved over time by selecting children with higher values
- Policy converges to optimal play asymptotically
Rollout policy $p_{\pi}$

- Training data: 8M board positions from games between human expert players
- Accuracy: 24.2%
- Time required to select an action: 2\(\mu s\)
Features (Rollout Policy $p_\pi$)

<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 <em>nakade</em> 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>

Features used by the rollout policy (first set) and tree policy (first and second set). Patterns are based on stone colour (black/white/empty) and liberties (1, 2, >3) at each intersection of the pattern.
Supervised Learning Policy Network $p_\sigma$

- Training data: 30M board positions from games between human expert players
- Stochastic gradient ascent to maximize likelihood of selecting the same move as the human did
- Architecture: 13-layer network
- Accuracy: 55.7% vs 44.4% (state-of-the-art) (55.7% using board position and move history only)
- Time required to select an action: 3ms

Image: [1]
Reinforcement Learning Policy Network $p_\rho$

- **Goal:** Improve policy by policy gradient reinforcement learning
  Bias towards actually winning games rather than predictive accuracy

- **Architecture:** Identical to SL policy network
  weight initialization $\rho = \sigma$

- **Training:** games between current policy network and a randomly selected previous iteration of itself

- **Reward function only rewards for winning a game**

- **Performance:**
  - 80% of games won against SL policy network
  - 85% of games won against Pachi (using no search at all)
  - state-of-the-art, based on SL of convolutional networks, only won 11% of games against Pachi
Value Network $v_\theta$

- **Goal:** Estimate a value function $v^P(s)$ that predicts the outcome from position $s$
- **Ideally:** optimal value function under perfect play $v^*(s)$
- **Instead:** approximate value function using value network $v_\theta(s)$
- **Architecture:** similar to policy network, however, output is a single prediction instead of a probability distribution
- **Training:** state-outcome pairs $(s, z)$ using SGD and MSE

---

Image: [1]
### Feature Planes (Policy Network and 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>

Feature planes used by the policy network (all but last feature) and value network (all features).
Training the Value Network

- **Naive approach:**
  - Predicting game outcomes from data consisting of complete games
  - Problem: Successive positions are strongly correlated
  - MSE ⇒ Train: 0.19 / Test: 0.37

- **Actual approach:**
  - Generate self-play data set (30M distinct positions)
  - Each position sampled from a separate game
  - Games played between RL policy network and itself until termination
  - MSE ⇒ Train: 0.226 / Test: 0.234
Evaluation Accuracies

![Graph showing evaluation accuracies for different policies.

- Uniform random rollout policy
- Fast rollout policy
- Value network
- SL policy network
- RL policy network

Mean squared error on expert games vs. move number.

Image: [1]
Putting It All Together

Rollout policy  SL policy network  RL policy network  Value network

$p_\pi$  $p_{\sigma}$  $p_\rho$  $v_{\theta}$

Policy gradient

Human expert positions  Self-play positions

Data

Neural network

Image: [1]
Action selection at timestep $t$

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

$$u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$
Searching with Policy and Value Networks

Leaf evaluation

\[ V(S_L) = (1 - \lambda)v_\theta(S_L) + \lambda z_L \]
Searching with Policy and Value Networks

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

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

Image: [1]
AlphaGo’s Playing Strength

Image: [1]

Florian Brunner

AlphaGo

July 4, 2019
Example: How AlphaGo Selects Its Moves

Image: [1]
Why Use Policy and Value Networks?

- Value network and policy network work hand in hand

- Value network alone:
  - Would have to exhaustive compare the value of all children
    ⇒ Policy network predicts best move, narrows the search space

- Policy network alone:
  - Unable to directly compare nodes in different parts of the tree
  - Value network gives an estimate of winner as if the game was played according to policy network
    ⇒ Values direct later searches to moves that are actually evaluated to be better
Why Combine Neural Networks with MCTS?

- How does MCTS improve a Policy Network?
  - Recall: MCTS (Pachi) won 15% of games against Policy Network
  - Policy Network is just a *prediction*
  - MCTS and Monte Carlo rollouts help the policy adjust towards moves that are actually evaluated to be good

- How do Neural Networks improve MCTS?
  - The Slow Policy guides tree exploration more intelligently
  - The Fast Policy guides simulations more intelligently
  - Value Network and Simulation Value are complementary
AlphaGo vs Lee Sedol

Image: [8]
WHO WOULD WIN?

A highly intelligent world-class Go champion with years of experience who won 18 international awards

A poorly understood pile of linear algebra
Image: [9]
"It’s not a human move, I’ve never seen a human play this move. So beautiful. Beautiful. Beautiful."

– Fan Hui (2p)
AlphaGo Documentary

Image: [12]
Thank you for your attention!
[1] Silver et al. (2016)  
Mastering the game of Go with deep neural networks and tree search  
URL: https://vk.com/doc-44016343_437229031?dl=56ce06e325d42fbc72

URL: https://upload.wikimedia.org/wikipedia/commons/e/e3/Korean_Game_from_the_Carpenter_Collection%2C_ca._1910-1920.jpg

[3] Woman playing Go  

[4] Go Board  
URL: https://i1.wp.com/cdn0.vox-cdn.com/thumbor/cxHFEPUtYJkaAz2Uf0dV5qLtc90=/cdn0.vox-cdn.com/uploads/chorus_asset/file/6160055/akrales_160307_0970_a_0127.0.png
References II

[5] **AlphaGo Logo**
URL: https://blog.talla.com/hs-fs/hubfs/AlphaGo.png?width=3000&name=AlphaGo.png

[6] **David Silver**
URL: https://amp.businessinsider.com/images/56dfdf0cdd089521638b4689-750-562.png

What did AlphaGo do to beat the strongest human Go player?

[8] **Alpha Go vs Lee Sedol**
URL: https://compote.slate.com/images/9f656d7e-720a-4b84-aeca-154b07213300.jpg

[9] **Move 37**
https://qph.fs.quoracdn.net/main-qimg-6e771c6719fc2fda77bc1b68119cb756
[10] Fan Hui
https://media.wired.com/photos/592722acaf95806129f51b6c/master/pass/GW20160132503.jpg

https://qph.fs.quoracdn.net/main-qimg-04274753a6dc479b197000895a39df47

[12] AlphaGo Documentary
https://cdn-images-1.medium.com/max/1200/1*sf4ZeTwBq1061U4W49NBdQ.png