Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

by Silver et al
Published by Google Deepmind

Presented by Kira Selby
Background

- In March 2016, Deepmind’s AlphaGo program was the first computer Go program to defeat a top professional player.

- In October 2017, Deepmind published AlphaGo Zero – a version of AlphaGo trained solely through self-play, with no human input.

- AlphaZero generalizes this algorithm to the games of chess and shogi, achieving world-class performance in each through only self-play.
Chess

- Possibly the most popular and widely-played strategy game in the world
- Chess is turn-based, asymmetric, completely observable, and played on an 8x8 board
- Each player controls 16 pieces (8 pawns, 2 knights, 2 bishops, 2 rooks/castles, 1 king and 1 queen)
- Players take turns moving one of their pieces to attempt to “capture” the opposing pieces
- The game is won through “checkmate” – i.e. placing your opponent in a situation where their king cannot avoid capture
Shogi

- Similar in concept to chess – 9x9 board where players take alternating turns, each moving one of their pieces
- Pieces are similar in concept, but somewhat different in details
- Captured pieces may be returned to the board under the opponent’s control

Since then, chess engines have grown rapidly in skill, and now far exceed human players in skill.

Shogi engines first defeated top human professionals in 2013.

AlphaGo was the first computer go engine to defeat a Go professional.
Go vs Chess and Shogi

- Go has a far larger action space than Chess or Shogi due to its 19x19 board
  - 32490 possible opening moves for Go vs 400 for Chess
  - $10^{174}$ board configurations for Go vs $10^{120}$ for Chess

- Go is also more uniquely suited to an RL approach – simple rules, highly symmetric board (data augmentation), all interactions are local (CNNs), rules are translationally invariant (CNNs)
Go vs Chess and Shogi

- Existing top chess and shogi engines rely on traditional algorithms
  - Handcrafted features
  - Alpha-beta search
  - Memorized “books” of openings and endgames

- Chess and Shogi are naturally more amenable to these methods than Go due to their smaller search space
Alpha-Beta Search

- Variant of general minimax search
  - Minimize max-imum reward of opponent’s actions
- ABS works by “pruning” away nodes which are provably worse than other available nodes
- Constrains search space to minimize computation
- Requires an explicit evaluation function for each state
Handcrafted Features

- Chess and shogi engines use a variety of complicated techniques to store and evaluate board positions.

- They use handcrafted features such as specific point values for each piece in each game stage, metrics to determine the game stage, heuristics to evaluate piece mobility or king safety, etc…
Handcrafted Features

\[ P_{(\text{pawn})} = 1 \quad BB_{(\text{bishop pair})} = +\frac{1}{2} \quad R_{(\text{rook})} = 5 \quad B_{(\text{bishop})} = \]

\[ 3\frac{1}{4} \quad N_{(\text{knight})} = 3\frac{1}{4} \quad Q_{(\text{queen})} = 9\frac{3}{4} \]
Opening and Endgame

- Retrograde analysis is performed to analyze all endgame positions of e.g. 6 piece or less (for chess)

- The results are then compressed and stored in a database that the program can access to achieve perfect knowledge of endgame play

- The program is also given access to “books” of common opening positions with evaluation metrics for each position

- Opening positions are so open-ended that it is difficult for engines to evaluate them accurately without a book
AlphaZero is trained solely through reinforcement learning, starting from a *tabula rasa* state of zero knowledge.

It is given the base rules of the game, but no handcrafted features aside from the base state, and no prior experience or training data aside from self-play.
AlphaZero Algorithm

- AlphaZero uses a form of policy iteration through a Monte Carlo Tree Search (MCTS) combined with a Deep Neural Network (DNN)

- **Policy Evaluation**: Simulated self-play with MCTS guided by a DNN which acts as policy and value approximator

- **Policy Improvement**: The search probabilities from the MCTS are used to update the predicted probabilities from the root node
The algorithm is guided by a DNN: \((p, v) = f(s)\)
- \(p\) represents the probability of each mode being played, where 
  \(v\) represents the probability of winning the game

This represents both the value function \(Q(s,a)\) and the policy 
\(\pi(s,a)\)
DNN Architecture

- The network consists of “blocks” of convolutional layers with residual connections. Each “block” consists of:
  - Two 256x3x3 convolutional layers, each followed by batch normalization and ReLU activation
  - A skip connection to add the block input to the output of the convolutional layers, followed by another ReLU activation

- The network consists of 20 blocks, followed by a “policy head” and “value head”, which map to their respective target spaces
Feature Representation

- The input and outputs of the neural network take the form of $F$ stacks of $NxN$ planes
  - $N$ is the board size for the game
  - $F$ is the number of features for that game (can differ for inputs and outputs)

- Each input plane is associated with a particular piece, and has a binary value for that piece’s presence or absence

- Each output plane is associated with a particular type of action (e.g. move N one space, move SW three spaces), with the $NxN$ values representing probabilities for the piece at that square to take that move
## Feature Representation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Go Planes</th>
<th>Chess Feature</th>
<th>Chess Planes</th>
<th>Shogi Feature</th>
<th>Shogi Planes</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 stone</td>
<td>1</td>
<td>P1 piece</td>
<td>6</td>
<td>P1 piece</td>
<td>14</td>
</tr>
<tr>
<td>P2 stone</td>
<td>1</td>
<td>P2 piece</td>
<td>6</td>
<td>P2 piece</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Repetitions</td>
<td>2</td>
<td>Repetitions</td>
<td>3</td>
</tr>
<tr>
<td>Colour</td>
<td>1</td>
<td>Colour</td>
<td>1</td>
<td>Colour</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total move</td>
<td>1</td>
<td>Total move</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>count</td>
<td>1</td>
<td>count</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P1 castling</td>
<td>2</td>
<td>opponent</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P2 castling</td>
<td>2</td>
<td>P1 prisoner</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>count</td>
<td>1</td>
<td>count</td>
<td>7</td>
</tr>
</tbody>
</table>

**Table S1:** Input features used by AlphaZero in Go, Chess and Shogi respectively. The first set of features are repeated for each position in a $T = 8$-step history. Counts are represented by a single real-valued input; other input features are represented by a one-hot encoding using the specified number of binary input planes. The current player is denoted by P1 and the opponent by P2.
### Feature Representation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Chess</th>
<th>Shogi</th>
<th>Planes</th>
<th>Planes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queen moves</td>
<td>56</td>
<td>Queen moves</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>Knight moves</td>
<td>8</td>
<td>Knight moves</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Underpromotions</td>
<td>9</td>
<td>Promoting queen moves</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Promoting knight moves</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Drop</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>73</strong></td>
<td><strong>Total</strong></td>
<td></td>
<td><strong>139</strong></td>
</tr>
</tbody>
</table>

Table S2: Action representation used by *AlphaZero* in Chess and Shogi respectively. The policy is represented by a stack of planes encoding a probability distribution over legal moves; planes correspond to the entries in the table.
The MCTS procedure acts as **policy improvement**:  
- The tree is iteratively expanded to explore the most promising nodes  
- At each stage of expansion, the values \((p, v)\) are updated based on the child node’s values  
- The network parameters \(\theta\) are updated to minimize the difference between the predicted values at the root node and the updated values based on the MCTS  
- This acts as a policy improvement operator
The training is performed over mini-batches of 4096 states from the buffer of self-play games generated at that iteration.

Parameters are updated through a combined loss function with a squared error over the value $v$ and cross-entropy of probabilities $p$ with respect to the search probabilities $\pi$.

\[ l = (z - v)^2 - \pi^T \log p + c \| \theta \|^2 \]
Differences from AlphaGo Zero

◆ Symmetry:
  ◆ Data Augmentation using 8-fold symmetry of Go board – does not hold for Chess or Shogi

◆ Draws:
  ◆ AlphaGo Zero maximized the binary probability of victory, whereas AlphaZero optimizes expected outcome, including draws (as Go does not allow draws)

◆ Updates:
  ◆ AlphaGo Zero replaces old player with new player after 55% win rate achieved, whereas AlphaZero updates continuously
Training

<table>
<thead>
<tr>
<th></th>
<th>Chess</th>
<th>Shogi</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mini-batches</td>
<td>700k</td>
<td>700k</td>
<td>700k</td>
</tr>
<tr>
<td>Training Time</td>
<td>9h</td>
<td>12h</td>
<td>34h</td>
</tr>
<tr>
<td>Training Games</td>
<td>44 million</td>
<td>24 million</td>
<td>21 million</td>
</tr>
<tr>
<td>Thinking Time</td>
<td>800 sims</td>
<td>800 sims</td>
<td>800 sims</td>
</tr>
<tr>
<td></td>
<td>40 ms</td>
<td>80 ms</td>
<td>200 ms</td>
</tr>
</tbody>
</table>

Table S3: Selected statistics of AlphaZero training in Chess, Shogi and Go.
# Results

<table>
<thead>
<tr>
<th>Game</th>
<th>White</th>
<th>Black</th>
<th>Win</th>
<th>Draw</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>AlphaZero</td>
<td>Stockfish</td>
<td>25</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Stockfish</td>
<td>AlphaZero</td>
<td>3</td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>Shogi</td>
<td>AlphaZero</td>
<td>Elmo</td>
<td>43</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Elmo</td>
<td>AlphaZero</td>
<td>47</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Go</td>
<td>AlphaZero</td>
<td>AG0 3-day</td>
<td>31</td>
<td>–</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>AG0 3-day</td>
<td>AlphaZero</td>
<td>29</td>
<td>–</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 1: Tournament evaluation of AlphaZero in chess, shogi, and Go, as games won, drawn or lost from AlphaZero’s perspective, in 100 game matches against Stockfish, Elmo, and the previously published AlphaGo Zero after 3 days of training. Each program was given 1 minute of thinking time per move.
Results

Figure 1: Training AlphaZero for 700,000 steps. Elo ratings were computed from evaluation games between different players when given one second per move. a Performance of AlphaZero in chess, compared to 2016 TCEC world-champion program Stockfish. b Performance of AlphaZero in shogi, compared to 2017 CSA world-champion program Elmo. c Performance of AlphaZero in Go, compared to AlphaGo Lee and AlphaGo Zero (20 block / 3 day) (29).
Results

Figure 2: Scalability of AlphaZero with thinking time, measured on an Elo scale. 

a Performance of AlphaZero and Stockfish in chess, plotted against thinking time per move.

b Performance of AlphaZero and Elmo in shogi, plotted against thinking time per move.
After training for 44 million games of self-play, AlphaZero achieves state of the art play for Chess, winning 28 of its 100 games against Stockfish and drawing the other 72.

Similarly, with 24 million games of self-play it defeated Shogi engine Elmo 90-2-8.

AlphaZero uses no human-provided knowledge and trains solely through self-play, with only the form of the input/output features changing between games (to represent each game’s rules).
Many chess experts criticised the AlphaZero vs Stockfish match as unfair or deceptive.

Stockfish was arguably handicapped by:
- Not having access to an openings book
- Playing with fixed time controls rather than total time per game
- Using a year-old version
- Suboptimal choices for hyperparameters
“God himself could not beat Stockfish 75 percent of the time with White without certain handicaps”

- GM Hikaru Nakamura