Research Review: Silver et al., Mastering the game of Go without human knowledge (2017)

Because of its large search space, the game of Go is one of the hardest board games. In 2016 DeepMind's AlphaGo program won a match against Lee Sedol, one the world's strongest players. AlphaGo was described in **Mastering the game of Go with deep neural networks and tree search**. The paper that we review here is about a new version of AlphaGo called AlphaGo Zero. It is called Zero because it uses no knowledge about Go except the rules of the game.

AlphaGo Zero consists of a deep neural network with more than 40 layers. The network uses residual learning, a technique that helps training deep neural networks invented by Microsoft in 2015. Technically, the network is based on Google's TensorFlow. The network is trained with the results from self-play. For self-play a version of Monte Carlo Tree search is used.

Self-play

The neural network can be viewed as a function f_θ that accepts a game position s as input and returns a tuple

$$f_\theta(s) = (\mathbf{p}, v)$$

Thereby $\mathbf{p}$ is a vector of move probabilities. For every legal move from the given game position s, $\mathbf{p}$ contains the probability that this move will be played. v is a value between -1 and +1 that estimates the outcome of the game.

A simple approach for self-play would be to just choose a move according to the move probabilities in $\mathbf{p}$. But stronger play results if the decision does not use $\mathbf{p}$ directly, but a Monte Carlo tree search (MCTS) that is based on $(\mathbf{p}, v)$. This version of MCTS works as follows: first a node in the search tree is selected (looking for a balance between exploration and exploitation). Then this node is expanded, i.e. one of the moves is chosen according to the move probabilities $\mathbf{p}$ for that position. But unlike in standard versions of MCTS, there is no simulation (playout) for the new node. Instead the value v provided by the neural network is used directly to evaluate the new position. Then the result is back-propagated up to the root position. After 1.600 expansions the most promising move is played.

Training of the neural network

In every step of MCTS, the child nodes of the current root node are assigned relative frequencies. We use the variable $\boldsymbol{ß}$ for the vector of the move frequencies from position s, and we use z for the final outcome of the game (either +1 or -1). So we can compare the values for $f_\theta(s) = (\mathbf{p}, v)$ from the neural network with the values $(\boldsymbol{ß}, z)$ from the games. By updating the parameters θ of the neural network in such a way that the difference between $(\mathbf{p}, v)$ and $(\boldsymbol{ß}, z)$ gets smaller, we obtain a new version of the network that has learned from the self-play games. Each training iteration uses 25.000 games of self-play as input.

Differences to the previous AlphaGo version

The most important difference is that AlphaGo Zero uses only reinforcement learning, no supervised learning. Thus it is possible to generalize their approach for use in other domains. In their new paper **Mastering Chess and Shogi by Self-Play with a General Reinforcement**

Learning Algorithm from December 2017 the authors describe an algorithm named AlphaZero that does this with respect to the games of chess and shogi. AlphaZero has beaten current world-class programs in both games.

AlphaGo Zero uses one neural network that computes both the move probabilities $\mathbf{p}$ and the position evaluations v. The previous version used two networks: a policy network and a value network. The approach to use the value v from the network directly for MCTS evaluation (rather than performing a rollout) is also new.