AIND Research Review: Alpha Go

Stefan Heidekrüger

June 28, 2017

In Silver et al. (2016), the authors describe the functionality, methodology and achievments of Google DeepMind's *Alpha Go* Go-game playing agent (as of 2015). In essence, Alpha Go combines previously the methods of *position evaluation (PE)* and *Monte Carlo Tree Search (MCTS)* —both of which had previously been applied to Ga playing. Alpha Go's novelty lies in applying deep learning methods in order to find good heuristics for either of these methods, namely an approximate *value function* $v \approx v^*$ in PE, and a succesfull *policy* $p(a|s)$ for MCTS. The resulting agent vastly outperformed any previous computer Go agent and for the first time could match expert level human players. In this review, we will very briefly outline the methods employed and the macro strategies of how the authors trained their heuristics.

Monte Carlo Tree Search

MCTS is a sophisticated method of selectively building a search tree in game playing. While the traditional methods of minimax expansion simply build the *entire* tree up to a certain depth that's dictated by time constraints (possibly with pruning, such as α-β-pruning as applied in the lectures) and then evaluates these intermediate positions through a heuristic value function (see PE), MCTS selectively grows a few branches very deep - all the way to endgame, at the expense of less horizontal completeness. Simulated end-states are then sampled (hence Monte Carlo) in order to derive a winning probability for each legal move. In order to so, MCTS requires an appropriate sampling *policy* to determine which branches to expand; given a current game state s and the set of legal moves $\mathcal{A}$, a policy p is a probability distribution on $\mathcal{A}$:

$$p : \mathcal{A} \to [0,1], a \mapsto p(a|s)$$

A sucessfull policy should find the right balance between *exploration*—exploring parts of the tree with little known information about them —and *exploitation*—focussing on the most promising actions to maximize value. Alpha Go's policy network is found in two steps:

- First, an "Expert Policy" p_σ is trained through supervised learning on a database of human expert games. The corresponding NN architecture alternates between convolutional and relu layers (with a final softmax).
- Given p_σ trained as above, the authors then train a second policy p_ρ through reinforcement learning, by letting the current iteration of the game agent play agains previous iterations. The NN evaluating p_ρ has the same architecture as above, with only the weights changed by the RL step. (The final version of Alpha Go replaces p_ρ with a less accurate but much more efficient approximation p_π.)

Position Evaluation

Due to the complexity of Go, it's prohibitive to expand branches until end-game, even when using MCTS. Therefore Alpha Go requires a position evaluation function v in order to score the leaves of the tree expansion and propagate the scores back up the tree. The authors use reinforcement learning in order to learn an approximation v_θ to the true value function v^* of the subgames. This is achieved through self-play of the agent using the previously trained policy p_ρ

The final, now world-famous agent Alpha Go finally plays by using p_π for tree rollout and v_θ for leaf evaluation.

References

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.