A short summary of “Mastering the game of Go with deep neural networks and tree search”

by Rene Kretzschmar

Synopsis

The game of Go has long been perceived as the most challenging classic game for Artificial Intelligence. It has a game tree of b^d possible sequences of moves, where $b \approx 250$ and $d \approx 150$, so exhaustive search is impossible. It's also very difficult to evaluate board positions.

The AlphaGo team introduced a deep learning based approach, where they use different neural networks to select moves and to evaluate board positions. Combined with a MCTS search algorithm, AlphaGo was able to defeated the European Go champion at that time, Fan Hui, by 5 games to 0.

To train their neural networks the team built a training pipeline consisting of several stages of machine learning:

The following chapters give a short summary of each of those stages and the used search algorithm.

First Stage: Supervised learning of policy networks

The team trained a 13-layer policy network on randomly sampled state-action pairs (s, a) out of 30 million positions from the KGS Go Server.

The network structure alternates between convolutional layers with weights and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves. A stochastic gradient ascent is used to maximize the likelihood of a human move selected in a certain state.

The network predicted expert moves with an accuracy of 57.0% when using all input features and even 57.7% using only raw board positions and move history as inputs. That beats the then state of the art 44.4% from other research groups.

Second Stage: Reinforcement learning of policy networks

The trained network has the same structure as in the first stage, but learns reinforced by playing games between the current policy network and randomly selected previous iterations of itself. The randomizing prevents the network from overfitting. The reward function takes the actual outcome of the game into account and returns +1 for winning and -1 for losing. All non terminal states return 0 here.

A stochastic gradient ascent is used to maximize the expected outcome of winning.

This network won 80% of all games against the supervised learning network of stage one.

Final Stage: Reinforcement learning of value networks

This network has a similar structure to the policy network, but outputs a single prediction of the quality of the evaluated position instead of a probability distribution of moves. It uses a stochastic gradient descent in order to minimize the mean squared error (MSE) between the predicted value and the correspondig actual outcome under perfect play. To mitigate the problem of overfitting when trained on the self-similar KGS data, the team generated a self-play data set of 30 million distinct positions, each sampled from a different game. That led to MSEs of 0.226 and 0.234, showing the same accuracy as the policy network by using 15,000 times less computation.

Search with policy and value networks

The search algorithm combines the policy network and the value network in a MCTS algorithm. The game tree is traversed by descending it in complete games, starting at the root.

Each edge (s, a) of the game tree stores three values: the action value $Q(s, a)$, the visit count $N(s, a)$ and the prior probability $P(s, a)$.

When traversing the game tree, at each step t the selected action a_t is the action with the highest sum of $Q(s_t, a)$ and a bonus $u(s_t, a)$. The bonus $u(s_t, a)$ is propotional to $P(s, a)$ but decays with $N(s, a)$ to encourage exploring.

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))$$

$$u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)} \tag{1}$$

When a leaf s is reached, it may be expanded and the new node s is processed once by the policy network, which returns a probability for each legal action a in s. This is stored in $P(s, a)$. The value of the node is then evaluated in two different ways. First, by the value network $v_\theta(s_L)$ and second by running a rollout to the end of the game with a fast rollout policy. The outcome z_L of this rollout and $v_\theta(s_L)$ are then combined as $V(s_L)$ by using a mixing parameter λ.

$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L \tag{2}$$

At the end of the simulation the action values $Q(s, a)$ and the visit counts $N(s, a)$ of all traversed edges are updated. They accumulate the visit count and the mean evaluation of all simulations traversed through them.

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

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{n} 1(s, a, i)V(s_L^i) \tag{3}$$

s_L^i : is the leaf node from the ith simulation

$1(s, a, i)$: indicates whether the edge (s, a) was traversed during the ith simulation

Since it's learning was based on human expert moves, the supervised learning policy network performed better than the reinforcement learning policy network.

It's also worth mentioning, that the computation power of AlphaGo is huge. The final version runs the simulations on 48 CPUs and the policy and value networks on 8 GPUs in parallel with 40 search threats. The team also implemented a distributed system on multiple machines having 1202 CPUs and 176 GPUs in total, but still running only 40 search threats.

Evaluating the playing strength of AlphaGo

The team ran a tournament among variants of AlphaGo and other Go programs, including the strongest *Crazy Stone*, *Zen*, *Pachi* and *Fuego*, with an allowed computation time of 5s.

It turned out that AlphaGo is many *dan* ranks stronger than any previous Go program with a winning rate of 99.8% even in handycapped games. The distributed version was significant stronger, winning 77% of the games against the standalone version and 100% against all other Go programs.

The team also tried variants of the MCTS algorithm by adjusting the mixing parameter λ. Even without rollouts ($\lambda = 0$), AlphaGo performed better than the other Go programs, but a mixed evaluation ($\lambda = 0.5$) worked best.

Finally the distributed version of AlphaGo won 5 games to 0 against Fan Hui, a professional 2 *dan* and the winner of the European Go championships in 2013, 2014 and 2015.

Conclusion

The AlphaGo team achieved something that was estimated to be doable 10 years from now at the earliest. It was only possible with the rise of deep learning in the recent past. If we assume an exponential development in this field, it is unthinkable what actually will be possible in the next 10 years.