Summary: Mastering the game of Go with deep neural networks and tree search

The paper describes the approaches used for developing the AlphaGo engine into the world's strongest Go program, as well as the first program to dominate the best human players. It is structured as follows:

- Core aspects of Go algorithms and the aspects AlphaGo seeks to improve, namely, the use of neural networks in the evaluation of value and policy function
- An explanation of the methods used for the value and policy function
- A discussion of the resulting games of AlphaGo against the existing class-leading software and professional human players

Problem Definition

In Go, the possible consequences of each move are fully known, and a search tree consisting of all possible moves can be constructed. In practice this is impossible, since such a tree for Go would have of order 10^{360}elements. State-of-the-art Go programs attempt to reduce the size of the tree by sampling only a subset of the branches of each state according to a *policy* (i.e., reducing the breadth), and truncating the tree after a certain depth using a *value function* that estimates how favorable the current board is for either player. For policies, in particular, it is important to predict the most likely move of a skilled opponent, as all branches corresponding to 'stupid' moves can safely be ignored. This approach is known as *Monte Carlo Tree Search* (MCTS), and the policy function can be regarded as *importance sampling.*

Traditionally, value functions and policies have been trivial functions such as linear combinations of certain positions on the board. However, the quality of the predicted value (or likely move of the opponent) has previously not been competitive with human players. AlphaGo uses a deep learning approach with convolutional neural networks (CNNs) to improve both the value and policy function.

Deep Learning Improvements

To train appropriate policy and value functions, AlphaGo employs two approaches:

- A *supervised learning* approach, in which data from 30 million positions of past human games stored on the KGS Go server is used for training
- A *reinforcement learning* approach, in which data generated from repeated games of the AlphaGo engine against a prior version itself is used, allowing the generation of arbitrarily large data sets.

For the policy function, the deep learning approach resulted in very convincing results. The accuracy of the expert move prediction increased from the state of the art of 44.4% to 55.7%[1]. However, this on a cursory sight minor increase resulted in significant gains in playing performance: Without employing search at all, AlphaGo was able to achieve an win ratio of 80% against a state-of-the-art program, whereas the previous deep learning approach had only yielded an 11% win rate.

The deep learning network for the value function meanwhile faced the additional obstacle of dealing with a highly correlated training set, as consecutive positions barely differ in terms of features but lead all to the same final result[2]. Consequently, on the initial KGS training set the value function overfitted significantly, resulting in nearly twice the mean squared error on the test set as compared to the training set. This problem was resolved by generating an uncorrelated training set of identical size from computer-computer games using the already trained policy function.

Finally, the trained policy and value functions were combined into a MCTS tree search.

Resulting strength

In evaluations against both state-of-the-art computer players and professional human players, AlphaGo demonstrated a significant increase in playing power, resulting in a 100%-win rate of the distributed version against all tested computer players as well as against a human triple European Go champion.

---

[1] To illustrate the relative unpredictability of Go, one can compare this complexity to that of the MNIST digit recognition problem. While the input complexity (each digit features a 28*28 pixel 'board' with 256 possible grey scale values for each pixel) can be considered comparable, MNIST can be solved by a shallow fully-connected network trained on a comparably tiny set of 60,000 labeled digits to an accuracy of over 96%.

[2] For the policy function obviously the same correlation exists in the input features, however it does not matter because the predicted variable is not shared.