Alpha Go, Neural Networks and tree search

The main issue that computers face when trying to solve any game of perfect information (games where reward or loss for end states and moves can be accurately determined) is the explosive rise of the number of nodes (in a minimax search tree) that the agent might have to explore. The total number of nodes represented by b^d for an 5x5 isolation board is ≅ 68,719,476,736 (8^12). An average processor is going to take a lot of time processing all these nodes therefore effective techniques like alpha beta pruning and iterative deepening have been developed to return a good move in a reasonable amount of time. But while these techniques have efficiently defeated human opponents in games like Chess and Isolation, the game of Go with total number of nodes approximately equal to 250^150 along with the complexity of the game cannot be defeated using these techniques.

First off, Alpha Go uses the Monte Carlo tree search (MCTS) to effectively predict the relevant values over many rollouts. The policy which chooses the actions during search is also improved over each rollout by selecting children with higher values. Ideally, this would make the agent play optimally. The strongest Go programs have been traditionally based on MCTS, further enhanced by policies that are trained to predict experts human moves. Alpha Go uses the same technique in the first stage of its execution whereby it uses Supervised Learning (Convolutional Neural Networks) to predict expert human moves in the game of Go. This provides the program with solid and reliable weights (the correct determination of which is one of the most important aspect of a neural network). The CNN at this stage was trained using 30 million positions from the KGS Go Server to predict expert human moves with a good accuracy.

In the second stage the training pipeline uses Reinforcement Learning with weights equal to the weights of the CNN trained in the first stage (again this means that weights selected are optimal). The policy network in this stage is made to play against its own previous plays which are selected randomly from a pool. This is to prevent overfitting which allows the neural network to generalise its weights for more unforeseen situations. Using this technique allowed the RL policy network to win 80% of games against SL network trained in the first stage and 85% of games against Pachi, the 2nd strongest amateur Go program.

The final stage of the training pipeline focuses on position evaluation, estimating a value function that predicts the outcome from the position of games by using policy for both players. This neural network has a similar architecture to the policy network trained in the second stage, with the main difference being that rather then outputting a probability distribution, this network outputs a single prediction. When Alpha Go was trained on the KGS dataset it memorised the results of the games rather than generalising to new positions (overfitting) getting a mean squared error of 0.37 on the testing dataset and 0.19 on the training dataset. The Deepmind team had to generate a complete dataset on their own consisting of 30 million distinct positions, each sampled from a separate game to get rid of this problem.

Alpha Go combines all of the above (policy and value) networks in an MCTS algorithm that attempts to foresee the effects of choosing a move and determines the best move. Each edge in the search tree stores an action value (utility), visit count and prior probability. The tree is traversed starting from the root state and at each time step an action is selected from the current time state, which maximises action value plus a bonus that is directly proportional to the prior probability but decays with repeated visits to encourage the program to visit more areas of the tree. When the traversal reaches a leaf node, that leaf node is expanded and processed by the SL policy network. The output (probability distribution) from this network is stored as prior probabilities for each legal action. The leaf node is evaluated using two different ways, first by the value network and secondly by the rollout policy, these evaluations are combined into a leaf evaluation for the current leaf node. At the end of this process, the action values and visit counts of all traversed edges are updated. Finally once the search is complete, the algorithm chooses the most visited move from the root position.

When AlphaGo was pitted against the strongest prior programs that played Go, it defeated all of them and played against the European Champion and won 5-0 which was the first time a computer Go program defeated a human professional player, a feat that was believed to be at least a decade away. The combination of neural networks and tree search in Alpha Go has achieved one of artificial intelligence’s “grand challenges” which makes it an exemplary agent that is able to make very complex decisions, solve an extremely huge search space and compute optimal solutions that apparently could not be directly computed. The Deepmind team hopes that this would enable artificially intelligent agents to achieve human like performance or better in the future on similarly extremely complex and intractable problems.