AlphaGo Paper Summary

Why is Google so interested in the game Go? Well, this game seemed like an impossible game for many AI agents for decades. It has stumped many agents in history such as Agent Smith from the Matrix, the Terminator, C-3PO, etc. Because of the game's enormous search space, many agents had difficulty evaluating board positions and selecting moves. Google, being the hero of robots and AI agents, sent an agent out to a journey to redeem the name of Artificial Intelligence in the Go community. It went by the name, AlphaGo.

The root cause of this problem is evaluating the game's search space. The agent must compute these games recursively, thus having a search space complexity of b^d. In the game Go, the branching factor (**b**) is approximately 250 and the depth (**d**) is 150, meticulous search is ridiculous. However, the search can be reduced by the following principles, (1) depth search may be reduced by shortening the search tree at a certain state and replacing the subtree below that state with an estimated value to predict the outcome and (2) the scope of the search can be reduced by trying actions from a probability distribution of possible moves of a certain state.

The strongest Go programs at the time uses a search method called the Monte Carlo Tree Search (MCTS) to reduce the search space. MCTS uses what's called 'rollouts' to estimate the value of each state in the tree. As more simulations are ran, the bigger the tree gets the appropriate values become more accurate. The policy used for this becomes better over time and eventually converges to an optimal value. These policies are then used to narrow search of high probability actions and to sample these actions during rollouts. However, this isn't enough to beat a human player in Go.

Strengthening this tree search required the use of deep convolutional neural layers, mainly used in the visual domain, in conjunction with MCTS. They passed an image of the game board to the neural network and used convolutional layers to represent a position. These networks are then trained using several stages of machine learning.

In the first stage of the training, a faster rollout was trained in exchange for accuracy and the neural network was trained using Supervised Learning (SL) to provide fast and efficient

learning updates with immediate feedback and high-quality gradients. The SL is used to predict expert moves in Go using a policy network that alternated between weighted layers and rectifier nonlinearities. This policy network was trained on randomly tested state-action pairs in order to increase the likelihood of predicting a human move in a given state.

The second stage of the training uses Reinforcement Learning (RL) to improve the current SL policy network by playing the policy against its previous versions. It randomly selects from a pool of its previous versions to stabilize the training and prevent overfitting to the current policy.

In the final stage of the training, the focus was on position evaluation. A value network is trained by regression to forecast whether the current player wins in positions from the self-play data set generated from the previous training.

To test this the performance of AlphaGo, it was entered in a tournament against several other Go programs including the strongest commercial programs. The results are astounding, AlphaGo only lost one game out of 495 games against other Go programs. And even with a handicap, AlphaGo won over 70% of its games. Because of this, AlphaGo was played against a European Go champion for 5 matches. HOLY ----! It won all 5 games against a pro?!?!?!, this was the first time a computer program defeated a human player which was supposedly about a decade away. That's pretty amazing!