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

Christopher Rivera

Despite its simple rules, the ancient game of Go is incredibly complex. This game, which consist of players taking turns placing stones to capture territory, has a game search depth and branching factor that far exceeds Chess by (Chess: d~70, b~35, Go: d~150, b~250). Due to its complexity, it is difficult to computationally play Go using adversarial search algorithms such as minimax with alpha beta pruning. Furthermore, unlike Chess, it is difficult to develop a scoring function that can assess the strength of a particular position. Rather than relying on adversarial search, state of the art professional GO programs have employed Monte Carlo Tree Search (MCTS) algorithms that simulate the expected outcome of a Go game from a given state by performing many fast Monte Carlo simulations of games in order to choose the next action (this is called rollouts). While these programs are fast and can beat the average player, they poise little challenge to expert players. In their paper, *Mastering the game of Go with deep neural networks and tree search (Nature 2016),* David Silver et al train and engineer an AI system, named Alpha Go, that employs deep neural networks, reinforcement learning and MCTS to play Go with the objective of playing at the grand master Go level.

To do this, they train four machine-learning algorithms. The first algorithm, which they refer as a supervised policy network, consists of a convolutional neural network that has been trained using 19x19 'images' of a 30 million Go positions from expert games from the KGS go data base. The objective of this network is to predict how an expert human player would play his/her next step given the current state of the board using a softmax function, i.e. develop a policy p(als). They show that these policy networks can reasonably accurately classify the most probable expert player move given a board state and that Alpha go models that employ the policy networks to choose plays win more games when they use more accurate policy networks Figure 2a. Next, they develop a reinforcement policy network model, which initially uses the network architecture and parameters of the supervised policy, to play Go games. To train this model, they use it to play games against other alpha go players with the reward function of +1 if the player wins and -1 if the player looses. Following the outcome of the game, back propagation is used to update the model weights depending on the outcome. Unlike the supervised policy network, which is focused on choosing a play that is most like a human expert players move, this network chooses a play that will

maximize its ability to win. They show that this model is able to beat a good computer player (Pachi) 85% without any search. They also train a fast policy classifier that takes in an engineered vector representation of the Go state to predict the next move. Although less 'accurate' than the convolutional network, this model is much faster. Finally, they train a 4th convolutional value network with the goal of predicting that a given game state will lead to a win, unlike the other models which are classifiers, this model estimates the expected value of a position as a regression problem using the function $v^p(s)=\mathrm{E}[z_t|s_t=s, a_{t...T}\sim p]$.

To develop the system known as Alpha Go, they combine three of these algorithms together with MCTS. While playing a Go game, this algorithm represents the game as tree with nodes and leafs as board states and edges as transitions between the states. The edges store two values, a p(als) and the number of times that the edge has been visited. Alpha Go chooses it next move given a game state by first selecting among possible potential moves. It uses it RL policy network to estimate that probability of choosing that particular move and uses its value network to estimate the value of a given state. It uses MCTS with the fast policy algorithm to choose next play to simulate many games originating from a given possible state. Using statistics of win probabilities collected from MCTS, Alpha Go computes a joint value function $V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_{L,}$ that combines the estimate value from the value network ($v_\theta(s_L)$) and the simulate outcomes generated by the MCTS search. The algorithm ultimately uses a policy function $a_t=\mathrm{argmax}(Q(s_t,a)+u(s_t,a))$, which is function of the other models predictions, to choose the next action.

Using a tournament against computer players (figure 4a) they show that Alpha Go is able to easily outperform other computer players that rely on MCTS alone. They also examine the importance of the individual parts of the algorithm by developing algorithms that use 1, 2 or 3 of the models. Although single and dual model alpha Go programs are decent, expert play only occurs when the program uses all 3 models (figure 4b). They also do experiments demonstrating that increasing the computational resources available to Alpha Go improves play efficiency.

Finally, they demonstrate the Alpha Go plays at a 'super' human level by competing it with a Human Go European Champion. Alpha Go is able to beat the player in 8/10 matches.

In summary, the Google Brain team lead by David Silver engineers and trains an AI system that uses deep learning models and MCTS to play the difficult game of GO at super human levels and 'solving' a grand challenge of AI.