Andrew Sutton, 10 August 2017

LITERATURE REVIEW: "**MASTERING THE GAME OF GO WITH DEEP NEURAL NETWORKS AND TREE SEARCH**", SILVER ET AL., NATURE VOL 529, (JAN 2016)

The paper discusses the application of a machine learning strategy to pay the game Go. Go is played on a 19 x 19 board, with players alternately placing pieces. Game rules dictate that pieces may be gained from an opponent, or lost to an opponent based upon the game mechanics. Since each player can view the entire board, Go is a game of perfect information where there is no information hidden between the players.

The concept of an '*optimal value function'* represents the outcome of a game (from the current state), assuming perfect play by all players. This indicates that each player should act optimally to maximize their own value, or equivalently their own likelihood of winning the game. The game of Go allows players to place pieces in the available squares. The number of available moves for Go, from a cited reference, is the breadth $b \approx 250$ whilst the approximate depth, the expected number of moves until end-game, is $d \approx 150$. As the total search space scales as bd, an exhaustive search of a game of Go until end-game is not considered feasible.

Instead, the optimal value function for a given state may be approximated by performing a limited-depth search and replacing the value function of each sub-tree at search termination by an *approximate value function* which attempts to predict the game outcome based upon the state at search termination. Additionally, low-value sub-trees may be pruned in early stage of the search using a 'probabilistic search' combined with an appropriate 'search policy', a rule which determines that sub-tree sampling probability.

An example policy is the '*Monte Carlo rollouts'*, which samples a large number of game sequences by exclusive application of a policy-rule. The result of the Monte-Carlo game sequence generated provides a measure of the position evaluation, i.e. an approximation to the ideal value function. *Monte-Carlo Tree Search* applies Monte-Carlo rollout to estimate the value or each state in a state-tree.

In comparison, the AlphaGo program uses several deep convolutional neural networks to evaluate the current game position and provide policy decisions. A convolutional neural network applies convolution operations between an input (the board state) and a set of learned convolution kernals, followed by a decision rule ('rectifiers') to generate intermediate state information. Cascades of convolution-decision layers allows the network to 'learn' high-level relationships via a series of simple operations. In the case of AlphaGo's networks, the board state is represented as a 19x19 state 'image'.

AlphaGo fuses the policy networks with Monte-Carlo Tree Search, where a policy network and value network select candidate and final actions by search. To assist the MCTS process, several neural networks are trained and applied:

- A Supervised Learning (SL) policy network: this network learned to predict the moves of expert human players.
- A 'fast policy' network: a faster network trained with reduced complexity to provide the policy information for rollout sampling of the search tree.
- A Reinforcement Learning (RL) policy network: this network improves the SL network based upon self-play, competitions between the updated network and previous versions of the network. The RL network re-orients the cost function from prediction of expert moves (in the SL case), to winning games.
- A Value-network, a network to predict the winner of the games based upon simulation of RL games. This network estimates the 'optimal value function'. The paper notes that training the value-network based upon complete games reduces performance as each state of the game is highly-correlated with previous states. This leads to over-fitting of the network. Instead, intermediate/distinct game-state data from games between RL-policy networks was used to train the value network.

At each time step /turn, AlphaGo simulates the game from the current state. At each step of the simulation, a candidate action is selected for evaluation as the action that maximizes the value, which has been generated and stored on previous evaluations, plus a bonus term that adds an additional value proportional to the prior probability of selection that action. The bonus term includes a decay component that reduces the selection prior based upon the number of visits: the probability of selecting high-prior probability sub-trees is reduced with regular visitation, allowing exploration of other 'less-likely' game sub-trees. This prior is supplied by evaluation of the SL policy network. The value of each node is a fusion of two value estimates:

- The value of the current game state is predicted using the value network, and
- The value of the current state is estimated using random-rollout, using the fast rollout policy network.

Thus, the network searches the sub-trees associated with high-value moves, based upon prior evaluation, while also expanding the search tree.

The policy and value networks require significant computational complexity due to the number of convolutional layers. The paper demonstrates that distributed/parallel implementation of AlphaGo improved performance.

Results Discussion

The paper presents results for AlphaGo, which consistently won games against other Go programs (Figure 4a). Similarly, Figure 4b demonstrates the relative contribution of the difference networks towards the final AlphaGo performance by inclusion of combinations of a) fast-rollout Policy network, b) Value network, and c) Policy network. The relative combinations/permutations of the network shows the merit of each complexity increase and associated interplay. AlphaGo demonstrates the first implementation of an effective move selection and position evalution based upon neural networks, trained from expert human player data and self-play between algorithm versions, via Reinforcement Learning. The AlphaGo algorithm minimizes the number of individual states that must be visited in the search tree based upon decisions provided by the policy networks.