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

Solving the game of Go has been a long standing challenge in the realm of artificial intelligence. As a two-person game of perfect information, it can in principle be solved exactly. However, due to the game's large branching factor ($b \approx 250$) and depth ($d \approx 150$), searching through the entire space of possible moves ($b^d \approx 10^{359}$) is unfeasible. Instead, one could perform a depth-limited search by approximating thse optimal value function $v^*(s)$ of any given state of the game *s* by some approximate value function that truncates the search tree at the given state *s*, i.e. $v^*(s) \approx v(s)$. Further, one could prune the search tree and limit the breadth of the search by sampling random actions *a* given a state of the game *s* via some policy (conditional) probability distribution p(a|s). This is the approach used in AlphaGo.

The architecture of AlphaGo consists of several neural networks, each designed to perform a specific task. In the first stage, a supervised machine learning approach is adopted, and a convolutional neural network with weights σ is trained on expert human moves to produce a probability distribution p_σ (a|s) of likely actions given a state of the game. For the training of this neural net, random state-action pairs (s, a) are sampled as inputs, and the minimum of the objective function is sought using stochastic gradient ascent, which maximizes the log-likelihood probability distribution as a function of the weights. This 13-layer policy neural network was able to predict expert human moves with an accuracy of up to 57.0% on a test set. Meanwhile, a smaller and less accurate, but faster, rollout policy network p_□ (a|s) was similarly trained for the purposes of quickly generating state-action pairs (s, a) that could be fed into the convolutional policy network for its training.

The next stage in the pipeline consists of a reinforcement learning policy network p_ρ (s|a), with an identical structure to the supervised learning policy network p_σ (a|s), and with the weights initialized to the learned weights in the previous stage of the pipeline, ρ=σ. Games are then played between the current policy network and some randomly selected previous iteration of the same network, using a reward function *r(s)*, which is set to zero for all time steps before the terminal time step, i.e. $r(s_t) = 0$ for all $t < T$, and non-zero only at the terminal time step T. The outcome at a given time-step t is given in terms of the terminal reward via $z_t = \pm r(s_T)$, and equal to +1 if the current player at time step t wins at the terminal time step T, and -1 if that player loses. Using this prescription, the weights of the reinforcement learning network are then updated at each time step t using stochastic gradient ascent to maximize the log-likelihood probability distribution times the reward function, log p_ρ (s|a) * $z_t$. Upon sampling moves *a* from the learned probability distribution p_ρ (s|a), this network was found to win 85% games against Pachi, the strongest open-source Go program at the time of writing of the paper.

The last stage in the training pipeline consists of learning an estimated reward or value function $v^p(s)$ that predicts $z_t$ (defined above) given the state of the game at any given time step t, $v^p(s) = E[z_t \mid s_t = s]$ using the previously learned policy to generate actions $a_{t \dots T} \sim p$. We can then approximate the value function by a value network v_θ (s), with an architecture similar to that of policy network, and with weights θ learned to minimize the mean square error between $z$ and v_θ (s).

AlphaGo then combines all the above in a Monte Carlo Tree Search (MCTS) which involves selecting an action at each time step that maximizes a linear combination of a back-propagated action value, using the value network v_θ (s) and the outcome at a leaf node $z_L$ (see definition of $z_t$ above), and another term that encourages more probable actions using the learned prior distribution P(s, a) = p_σ (a|s), but penalizes the number of visit counts of each (s, a) pair to encourage exploration of the search tree. Playing the game of Go in this manner, AlphaGo became the first computer program in Oct 2015 to beat a human Go master in a 5-0 win.