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

Albert Buchard

The game of Go always was considered as a landmark of difficulty for artificial intelligence, both due to the breadth of the search space and the difficulty to approximate the utility value of positions and moves mid-game. At the time of publication, artificial agents were still no match for human professional players. This paper of 2015 by the DeepMind team, published in Nature, presents a novel approach to computer Go, called Alpha Go, that uses a distributed machine learning pipeline composed of two convolutional neural networks, a policy network and a value network. Although the value network has a different single unit output layer, those networks share the same overall structure, and their input layer receives the same information, composed of 19*19 planes of board game one-hot coded informations. The policy networks are trained using supervised and unsupervised (reinforcement) learning, and the value network is trained using only reinforcement learning, during tree search a separate fast rollout policy mitigate the output of the value network by playing fast game until the end at each simulation. The overall value obtained from combining these two networks is used to guide the search of a tree of legal moves using a parallel implementation of Monte Carlo Tree Search (MCTS) over 40 search threads, 48 CPUs and 8 GPUs. Overall, Alpha Go was able to win 99.8% of games against current state of the art Go programs, and won in october 2015 a formal 5-game match against Fan Hui, 2nd dan, and a three time european champion.

The machine learning pipeline used two convolutional neural networks, a policy network able to predict the probability of each legal move, and a value network that is trained to predicts the end game value of a legal choice. While the value network was trained on full games using reinforcement learning, policy networks were trained following a two step process based on supervised learning (predict human moves) followed by an optimization of the final weights by a policy gradient reinforcement learning phase during which the end game value is propagated up the decision tree to update the weight of the first network.

Training of Policy Networks

Supervised learning. First stage is was a 3 week long supervised learning process, training on human decision data, the training set being randomly sampled from a set of 30 Million human action, board state pairs obtained from the KGS Go Server. The data set was augmented to include the eight reflections and rotation for each position.The input layer receives 48 19*19 matrices referred to as "planes". It corresponds to 19*19 board states, that are encoding several input features such as stone colour, liberties, captures, legality, or turns since a stone stone was played (complete list extended data table 2). Those planes are binary matrices and do not hold integer values, such that multiple categories are instead represented by multiple planes (one-hot encoding). The last layer outputs a probability distribution over all legal moves a, I imagine over a 19*19 length vector represented by the units of the last convolutional layer. They train using stochastic gradient ascent, meaning one minibatch of pair at a time, to maximize the likelihood over the action chosen by the human in the training set, and converge on the first set of weights alpha.

They show a strong and quadratic relationship between the classifier accuracy in predicting human choices and the average victory rate, indeed the win rate with a classification accuracy at chance is around 0% and a 9 point increase in training accuracy around 59% brings the win rate up to around 55% against the match version of Alpha Go (Fig. 2a).

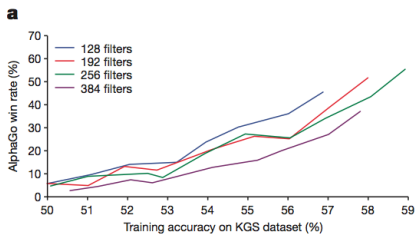

Figure 2 | Strength and accuracy of policy and value networks. **a**, Plot showing the playing strength of policy networks as a function of their training accuracy. Policy networks with 128, 192, 256 and 384 convolutional filters per layer were evaluated periodically during training; the plot shows the winning rate of AlphaGo using that policy network against the match version of AlphaGo. **b**, Comparison of evaluation accuracy between the value network and rollouts with different policies.

Policy gradient reinforcement learning (RL). The second phase of policy training was a one day reinforcement learning process using mini-batches of games played between two alpha go instance, one with the latest parameters and one with parameters sample randomly in the history of used parameters. The network trained is the same as the previous supervised learning phase and the weight are initialized to its final weight values. This process allows for a back-propagation of final endgame reward to the previously learned weight that were based only on single {state,action} pairs during learning, without looking ahead to see whether or not the choice would actually lead to a win. By playing games until the end, this RL algorithm is able to propagate endgame information up the sequence of choices (game are played twice, once to see the result, and once to update the values). Performance of the RL policy network were increased compared to only using the SL network with 80% win average in favor of the two step process (although this advantage was reversed using search) This algorithm, without using any lookahead search, was reported to beat the strongest open source Go program "Pachi" on around 85% of games.

Training of the value network

Definition. The goal of the value network is to predict the value of a move based on the end game result of a game between two player using the same policy network. It determines the outcome of a game from a certain state if both player are

following a specific policy. For most game, evaluating this function is intractable, such as in Go, that has a a extremely vast search space. An approximation of the optimal value function is used in this paper.

Architecture and training. The value network follow also the same CNN architecture as the RL policy network, but the output is a single fully connected unit, hence outputting a single prediction. This network was trained to approximate the value function of the RL policy algorithm. It used the RL network to sample the first moves, pick a random legal move and sample the rest of the game using the RL policy. The error between predicted and observed reward (win or loss) served as objective function for learning.

Rollout policy. They used also a separate policy they referred to as the fast rollout policy. This algorithms allows to mitigate the output of the value network by combining it linearly with the result of a rollout game played until the end using this rollout policy. The rollout policy evaluate action value faster based on local patterns around a certain move, as well as certain handcrafted pattern characteristic of Go rules. It is trained using supervised learning on data obtained from Tygem server. The algorithm caches the best moves in a hash table that allows for fast recovery of the best action if a previously learned state is re-encountered.

Search and decision

They used an approach called Monte Carlo Tree Search to explore the tree, making two approximations, the first being that the value obtain through sampling represented the simulation policy, and that the value of that simulation policy actually approximated the value of a minimax optimal action value function.

The actual search of the tree is keeps a memory of all its previous and relevant searches, and expands the tree at the leaves, sampling from those with the highest probably, while using a distributed approach. For each {state,action} pair the search algorithm stores and updates a set of six values:

$$\{P(s,a),\ N_v(s,a),\ N_r(s,a),\ W_v(s,a),\ W_r(s,a),\ Q(s,a)\}$$

P is the prior, N the number of time the node has been visited both forth evalue and policy network, W is the value associated with the node both for the policy and value network, and Q is the overall value.

$$Q(s,a) = (1-\lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{\bar{W}_r(s,a)}{N_r(s,a)}$$

Nodes are visited a maximum amount of time n_{thr} ($n_{thr} = 40$ in the match implementation) before setting succeeding states as leaves. The values W_r and W_n of each branch are updated end of each simulation.

Although the value guides the selection of actions during search, it is actually the number of node visits that will determine the action alpha Go will take.

Extended Data Table 7 | Results of a tournament between different variants of AlphaGo

Short name	Policy network	Value network	Rollouts	Mixing constant	Policy GPUs	Value GPUs	Elo rating
α_{rvp}	p_σ	v_θ	p_π	$\lambda = 0.5$	2	6	2890
α_{vp}	p_σ	v_θ	–	$\lambda = 0$	2	6	2177
α_{rp}	p_σ	–	p_π	$\lambda = 1$	8	0	2416
α_{rv}	$[p_\tau]$	v_θ	p_π	$\lambda = 0.5$	0	8	2077
α_v	$[p_\tau]$	v_θ	–	$\lambda = 0$	0	8	1655
α_r	$[p_\tau]$	–	p_π	$\lambda = 1$	0	0	1457
α_p	p_σ	–	–	–	0	0	1517

Evaluating positions using rollouts only (α_{rp}, α_r), value nets only (α_{vp}, α_v), or mixing both (α_{rvp}, α_{rv}); either using the policy network p_σ(α_{rvp}, α_{vp}, α_{rp}), or no policy network (α_{rvp}, α_{vp}, α_{rp}), that is, instead using the placeholder probabilities from the tree policy p_τ throughout. Each program used 5 s per move on a single machine with 48 CPUs and 8 GPUs. Elo ratings were computed by BayesElo.

Results

They report that a complete implementation of their algorithm performs better in tournament against alpha go implementation only using parts of the pipeline, with an Elo rating of 2890 for the full implementation against 2177 for the best diminished version (extended data table 7).

They also report that the full implementation of alpha Go running on a single machine is able to beat all the state of the art Go programs 99.8% of the times. Starting with a 4 stone handicap, single machine Alpha Go was still able to win 77%, 86 and 99% of handicap games against three programs, Crazy Stone, Zen and Pachi respectively. The distributed version was stronger, wining 77% of games against single machine alpha Go and 100% of games against other programs.

In October 2015, distributed alpha Go running 40 threads was able to beat a professional Go player, Fan Hui, three times europeen champions, 5 games to 0.

Discussion

What is worth noting is that this algorithm take advantage not only of the renewal of neural networks (CNNs), but also on the implementation of state of the art algorithms to tackle more classic AI problems such as tree search (MCTS), or rollout policies, while heavily relying on cutting edge distributed computation capabilities. This reinforce the idea that the future of AI does not lie in one specific field, or on one specific technology, or on supervised over unsupervised learning, but will most probably be enriched by advances across a very large set of fields, old and new.

Conclusion

Relying on an innovative approach combining two convolutional neural networks, a policy and value network, as well as a distributed Monte Carlo tree search algorithm, Alpha Go is a new Go playing program developed by the Google Deepmind team that was able in 2015 to achieve human level abilities at playing Go, and beat 5 games to none the current European Go champion Fan Hui. After publication of the article, in Mars 2016, Alpha Go won 4-1 against Lee Sedol, 9th dan, considered one of the best player of Go in the world. This win confirmed that Alpha Go could achieve at least human level ability and most probably super-human ability in the game of Go, an event which was predicted to happen in the next decade.