Summary of AlphaGo paper

Introduction:

The paper describes how Google deepmind developed a system to consistently beat a human player in a game of go, a feat that has never been done before. The paper talks about the novel techniques that were used to build AlphaGo.

Prior to AlphaGo, game-playing agents used evaluation heuristics and sampling actions using Monte Carlo rollouts, to reduce the depth and width of the search tree. AlphaGo uses Monte Carlo Tree Search (MCTS) along with policy network and value network to choose an appropriate move. A policy network is used to sample actions and value network is used to evaluate positions.

In this summary we will see how the policy and value networks were trained and how they are used in combination with MCTS.

Components of AlphaGo:

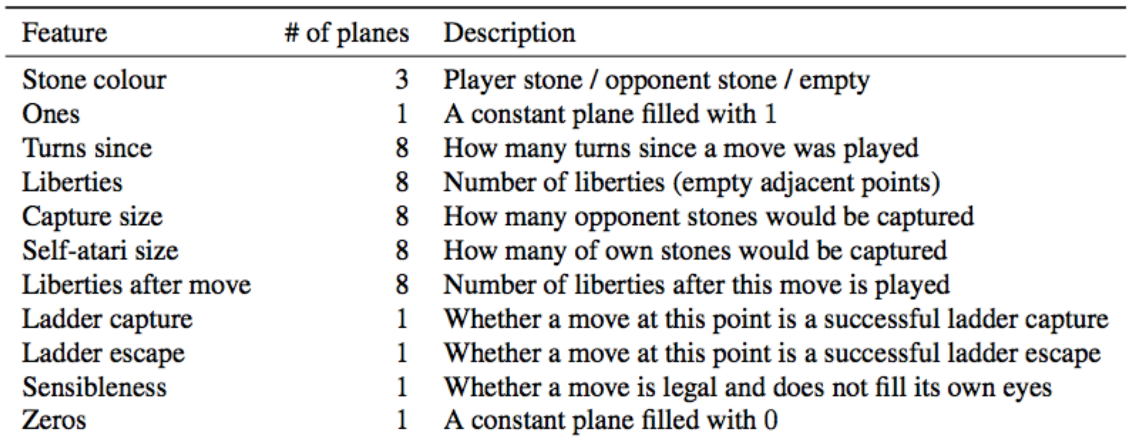
1. Supervised Learning (SL) of Policy Network
2. Reinforcement Learning (RL) of Policy Network
3. Reinforcement Learning of Value Network
4. AlphaGo MCTS to search Policy Network and Value Network

Supervised Learning (SL) of Policy Network:

The policy networks are trained using SL. The policy networks were trained to predict KGS 6-9 dan player's moves. The first policy network P_σ(a/s) was trained on 28 million positions from the KGS server and tested on 1million positions and was 13 layered. The input to the network was in the form of (s, a) where 's' is the state of the game board and 'a' is the action. Stochastic gradient ascent was used to increase the likelihood of mimicking a human move 'a' given state 's'.

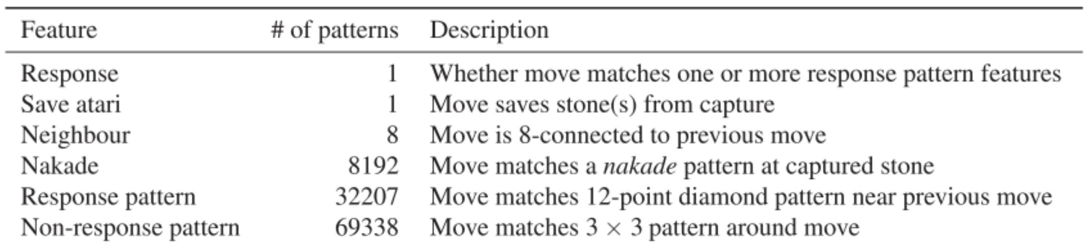
$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

This policy network had an accuracy of 55.7% as opposed to the best at that time which was 44.4%. The input features for this neural network is shown below.



Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0

There was also another policy network that was trained for the purpose of rollouts in MCTS; this was the P_π(a/s), which was a linear softmax of features. This was trained on 8 million positions from tygem server. The input features for this policy network is shown below.



Feature	# of patterns	Description
Response	1	Whether move matches one or more response pattern features
Save atari	1	Move saves stone(s) from capture
Neighbour	8	Move is 8-connected to previous move
Nakade	8192	Move matches a *nakade* pattern at captured stone
Response pattern	32207	Move matches 12-point diamond pattern near previous move
Non-response pattern	69338	Move matches 3 × 3 pattern around move

Reinforcement Learning (RL) of Policy Network:

The second step was to improve the policy network built from the previous step to optimize for winning as opposed to mimicking the human players. The new policy network is P_ρ. To achieve this games were played between the policy network and a randomly selected version of a previous policy network. The weights of the policy network were updated at each iteration. Stochastic gradient ascent was used to maximize wins.

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t | s_t)}{\partial \rho} z_t$$

Here Z is the terminal reward, -1 for loosing and +1 for winning. The policy network from RL won 85% against SL.

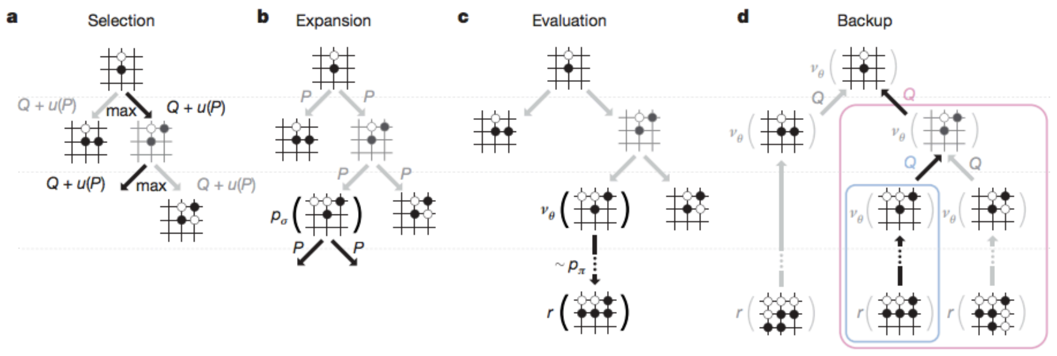
Reinforcement Learning (SL) of Value Network:

In an ideal case there would be a value function that gives the exact value based on the game board, but since search to end game is not feasible and a linear formula for calculating evaluation heuristic for a go game board isn't good enough. Estimation for the value function of the policy network (V^p(s)) had to be calculated. This was done by training a network on regressing (s, z) using stochastic gradient descent to minimize MSE between predicted ($V_\theta(s)$) and outcome z.

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

This gives a value network $V_\theta(s)$.

AlphaGo MCTS:



1. Selection: The selection policy selects an action based on the formula

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))$$

$$u(s,a) \propto \frac{P(s,a)}{1 + N(s,a)}$$

Starting with Q = 0, N = 0, P: Prior probability calculated using P_σ. The selection step goes to a depth d.

2. Evaluation: After reaching node of depth d, the node is evaluated based on 2 criteria, rollout till end game using P_π(a/s) and value network V_θ(s). The final metric is a linear combination of the previous 2 criteria.

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

Z: outcome of rollout (-1 for loss and +1 for win)

3. Backup: The value from the above step will be used to update Q based on the formula.

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$
$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

Once the search is completed for breadth d and depth d, AlphaGo selects the move, which was visited the most from the root.

Conclusion:

AlphaGo defeated Crazy Stone, Zen, Pachi, Fuego, which are all, go playing agents. AlphaGo also defeated Fan Hui a 2-dan player in the game of Go. It should be noted that AlphaGo did not store any moves but learnt from game play. This demonstrates the power of neural networks.