Goals

The game of Go has enormous search space and difficult evaluation of board positions and moves. Before AlphaGo state of the art solution for games like Go was using Monte Carlo Tree Search algorithm and this systems played go on a middle adult level. The goal of AlphaGo project was to create a new approach that could beat a performance of existing solutions.

Brief system explanation

A set of models has been trained to be combined lately to modified Monte Carlo Tree Search algorithm.

SL(supervised learing)-policy network

predicts moves of a human player based on online available database of games of experience adult players.
It gets a current board state and predicts a move of human player. It uses 12 layer convolutional network with softmax, output is just a (cell, weight).
An input of this neural net - a vector of cell's feature vectors. Each cell's vector contains information about existing occupation of this cell, information about cell surrounding like a number of empty adjacent points, a potential gain (number of captured stones) and state (number of liberties) if active player or opponent would occupy this cell. So, for each cell we pass information about the cell environment and how the major game parameters would be affected if player or opponent would occupy this cell - a type of very complex evaluation function.
Get accuracy 57%. Which is good, because actual human move is unpredictable and random in some sense and very high value in this accuracy would mean overfitting.

Fast rollout policy

does the same as SL-policy network - predict moves of a human player, trained of the same dataset.
It gets a current board state and predicts a move of human player. Fast rollout policy is less accurate than **SL-policy network**, but uses very quick to compute linear classifier.
As an input it gets a vector of cells, each cell contains vector of features, like the previous model, but features are a bit different and mostly pattern-oriented - does capturing of this cell continue any of known patters and simple things like if move to this cell leads to stones capturing.

RL(reinforcement learning)-policy network

We take our **SL-policy network** as a base and make reinforcement learning.
During learning:

1. We randomly select a player from previous versions of this **RL-policy network** or base **SL-policy network** (to reduce overfitting) as an opponent.
2. We just play a game against it. Reward - we win or loose. Then we update weights of the network using stochastic gradient ascent in the direction that maximises expected outcome (if we loose - decrease probability of predicted steps, if we win - increase).

Value network

estimates if the position good or bad - leading to win or loose and output is between {-1, +1}. So, it's an analogue of evaluation function.
Uses almost the same features as a SL network.
It was impossible to try on positions from human games, because the network tried to capture which specific match this position relates to and tried to estimates matches, not separate positions.
Solution - we contract a new position by playing N moves by **RL-policy network**, make random move and play till the end using **RL-network**. After this we train network on the N + 2 position only (so, first selected by **RL-policy network** after a random move).

Game agent

All this building blocks we use to train modified **Monte Carlo Tree Search (MCTS).**
Each simulation step selects move by maximising:

```
(Q = it's estimated reward, which we learn actually in this simulation) +
(probability of this move by SL-plicy network) / (numbers when we
selected this move / total simulations from this node)
```

When we find new node that we never discover - we start an Expansion as in **MCTS** and

1. We evaluate this new node using **Value network**
2. From this node we play a game till the end. We don't use random choice like in the basic **MCTS**, but use our **Fast rollout policy**. We get final reward of the game.
3. We mix a reward with the estimation from the **Value network** and add the result to all nodes starting from the root till the new expanded node, just take average Q among all simulation.

Simplifying, each simulation

1. goes to the node with the best evaluation(Q), mixed with forcing good nodes by **SL-policy network** selection and stimulation exploration by adding penalty for using moves too often
2. find new position, makes estimation of this new node using (one game executed by **Fast rollout policy** + **Value network** position estimation)
3. Propagate this value to nodes up to the tree

After running simulation we consider a move from the root node which we selected more often during simulation the best one.
Interesting detail - in the final modified **MCTS** we use **SL-policy network** (from experiments it's better than **RL-policy network** here), so we needed to train RL-network only to train **Value network**.

Results

AlphaGo introduces a new approach with deep networks trained by supervised learning and self-play reinforcement learned deep 'policy networks' to select good moves, 'value networks' to evaluate board positions and 'fast rollouts' that could select more or less good moves very quickly. All this parts were combined and applied in different places of Monte Carlo Tree Search and produced a new mixed approach.
AlphaGo system wins in 99.8% of games agains existing Go programs and defeated European human champion by 5:0.