# HOW ALPHAGO WORKS - THE FIRST PROGRAM TO BEAT A HUMAN IN THE GAME OF GO

**Karnaukhov A.V. (Russian Federation)**

*Karnaukhov Arseniy Viktorovich – undergraduate Student,*
*DEPARTMENT OF APPLIED MATHEMATICS,*
*NATIONAL RESEARCH UNIVERSITY HIGHER SCHOOL OF ECONOMICS, MOSCOW*

***Abstract:*** *this article analyzes the game of Go that has long been considered the most difficult of the classic computer games because of its huge space of possible positions, as well as the difficulty of evaluating them. The breakthrough in the field of computer Go was made by the AlphaGo program. AlphaGo was the first program to beat a professional player in Go. It was developed by the Google DeepMind in 2015 and described in the article [1]. The purpose of this paper is to give the reader a clear understanding of how the AlphaGo algorithm works and what ideas it uses.*
***Keywords:*** *AlphaGo, network, Go, move, position.*

# КАК РАБОТАЕТ АЛЬФАГО - ПЕРВАЯ ПРОГРАММА, ПОБЕЖДАЮЩАЯ ЧЕЛОВЕКА В ИГРЕ ГО

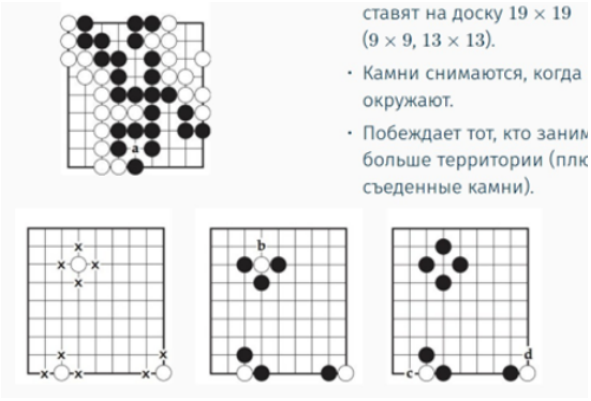
**Карнаухов А.В. (Российская Федерация)**

*Карнаухов Арсений Викторович – студент бакалавриата,*
*кафедра прикладной математики,*
*Национальный исследовательский университет «Высшая школа экономики, г. Москва*

***Аннотация:*** *в данной статье анализируется игра Го, которая долгое время считалась самой сложной из классических компьютерных игр из-за огромного пространства возможных позиций, а также сложности их оценки. Прорыв в области компьютерного го совершила программа AlphaGo. AlphaGo была первой программой, обыгравшей профессионального игрока в го. Она была разработана Google DeepMind в 2015 году и описана в статье [1]. Цель этой статьи — дать читателю четкое представление о том, как работает алгоритм AlphaGo и какие идеи он использует.*
***Ключевые слова:*** *AlphaGo, сеть, Go, ход, позиция.*

## I. About Go and convolutional neural networks

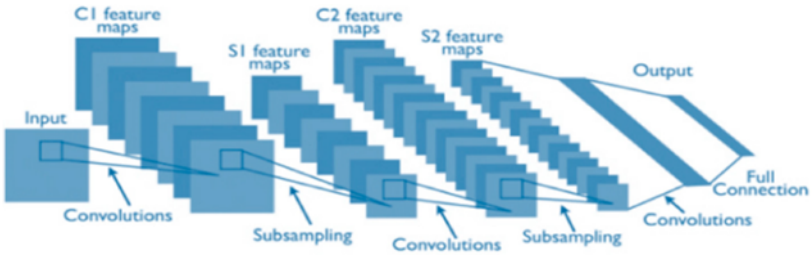
- Briefly about the rules: Stones are placed on a 19 x 19 board (9 x 9, 13 x 13).
- Stones are removed when they are surrounded.
- The one who occupies the most territory (plus the stones eaten) wins.

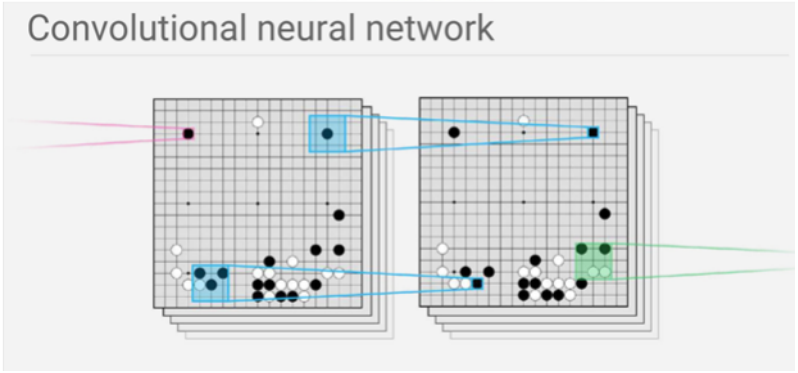



Go is a complex deterministic board game, with complete information, requiring intuition, creativity, and strategic thinking. Computer Go has long been considered a challenge for artificial intelligence.

Approaches that have shown excellent results in other board games have not yielded significant results here. This is partly due to the difficulty of creating a position estimation function, partly due to the very

the depth and width of the tree of possible moves (if in the chess tree the width is ≈35, and the depth is ≈80, then in Go the width is ≈250, and the depth is ≈150).

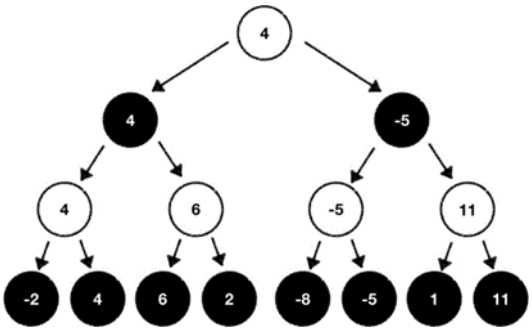



Recently, deep convolutional neural networks have achieved unsurpassed results in areas such as: image classification, face recognition, Atari-like games. Convex networks use multiple layers of neurons arranged in overlapping tiles. This is done to create increasingly abstract, localized representations of an image. The authors use a similar neural network architecture in the AlphaGo algorithm. The network takes a position on the board as a 19×19 image and uses convolutional layers to interpret the position. This approach has achieved unprecedented performance in the task of estimating the position and predicting the professional's move.

## II. Computer Go before AlphaGo

## III.




### 1. Variations of tree search

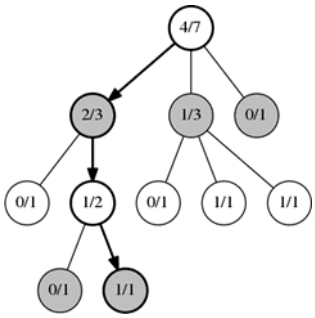
One of the traditional methods is minimax tree search. It consists of modeling all hypothetical moves on the board up to a certain point, and then using an evaluation function to select the move that leads to its maximization. The process is repeated every move. While tree search was very effective in chess, it was much less effective in Go. This happens because of the nature of the game. It is difficult to create an effective board evaluation function, and also because there are numerous of reasonable moves that result in a large branching factor in the tree.

### 2. Playing according to preset patterns and rules

Beginners often learn from game recordings of old games played by experienced players. There are also many books describing the basic principles of Go. Early works often included attempts to teach AI heuristics of human-style Go knowledge.

Two ways were suggested: learn common stone configurations and positions, and concentrate on local battles. However, as was seen later, this approach lacks both quality and quantity of knowledge.

### 3. Monte Carlo tree search (MCTS)



One of the main alternatives to using manually coded patterns is to use Monte Carlo methods. The algorithm creates a list of possible moves and for each move, thousands of games are randomly played on the resulting board. The move leading to the best ratio of random game wins for the current player is declared the best. The advantage of this method is that it requires very little subject matter knowledge or expert input, and the trade-off is increased memory and processor requirements.

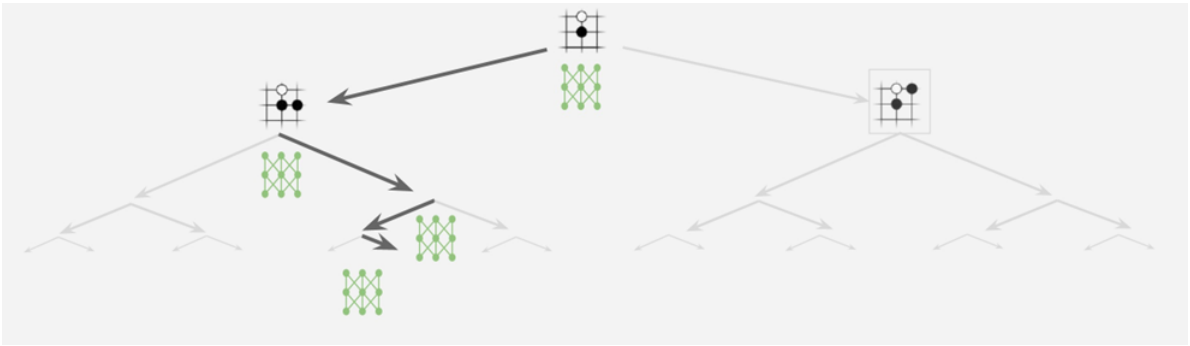
**4. The use of machine learning**

The level of knowledge-based systems is closely tied to the knowledge of their programmers and associated subject matter experts. This limitation makes it difficult to create really strong AIs. Another way is to use machine learning techniques. In them, the only thing programmers need to program are rules and simple algorithms to evaluate how to analyze the value of a position. During the learning process, the machine itself generates its own understanding of patterns, heuristics, and game strategies.

## III. How AlphaGo works

AlphaGo is the first computer program to defeat a professional Go player, the first to defeat the World Go Champion, and arguably the strongest Go player in history. AlphaGo algorithm is based on a Monte Carlo method, which additionally uses convolutional neural networks to optimize the tree width and depth.

### 1. Shrink the tree in width



**Idea:**

Train a neural network that will predict the most likely moves of professional players by their position on the board. Then at each moment of branching the tree, we can consider only the most likely moves, which greatly reduces the width of the search.

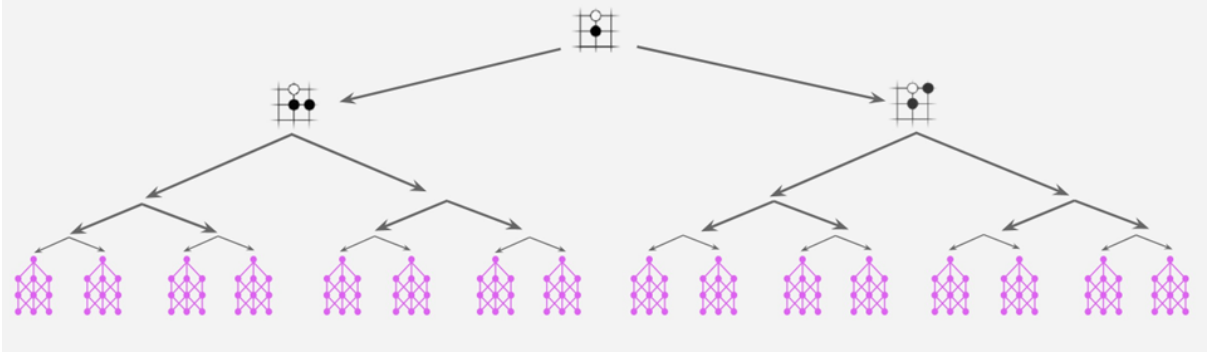
**Implementation:**

| 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 |

30 million games of strong players (KGS 5+ dan) are taken for training. On this data we train a 12-layer convolutional neural network, which the authors call SL-policy network. The input is a tensor of size 19×19×48. 19×19 is the size of the game board, 48 is the number of features after encoding them using One-Hot Encoding method:

The output is an array of size 19×19 - for each square the probability is predicted that the professional player goes exactly there. It is trained using stochastic gradient descent, maximizing likelihood. The accuracy of the predictions of this network is 57.0% (the prediction is chosen as the move with the highest probability). Interestingly, if we leave only the "Stone color" trait, the accuracy will be 55.7%. As the authors note, even a small increase in accuracy greatly affects the final strength of the game. The previous accuracy result in this area was 44.4%.

### 2. Reduce the tree in depth



**Idea:**

1. Train a neural network that will give the probability of winning this game based on the position given to it.
2. Let's train a very fast model, which will predict the most probable move of a pro.

Now, coming to a new node that we don't know anything about yet, we can quickly estimate the outcome of the game from it using neural network 1, as well as playing the position to the end of the game using fast model 2.

**Implementation:**

The network architecture is similar to the SL-policy network (the same 12-layer convolutional neural network), except that at the input it receives one more additional attribute - whether the player plays black, and the output is one number - the probability of victory. In the article, this network is called the value network.

If to teach the value network on the same 30 million games as SL, the problem of overlearning appears (many positions belong to a game with the same outcome and the network starts to learn to remember the game).

To solve this problem, the authors use the following strategy: the SL-policy network is copied and called RL-policy network, then RL starts playing a game with a randomly chosen its previous version (the first game is played with itself). At the end of the game weights of RL network are updated so that the network learns to win instead of predicting a professional's move (the gradient sign depends on game result), the updated RL is added to the pool of its versions. This approach results in a neural network that already wins the best Go programs 80% of the time.

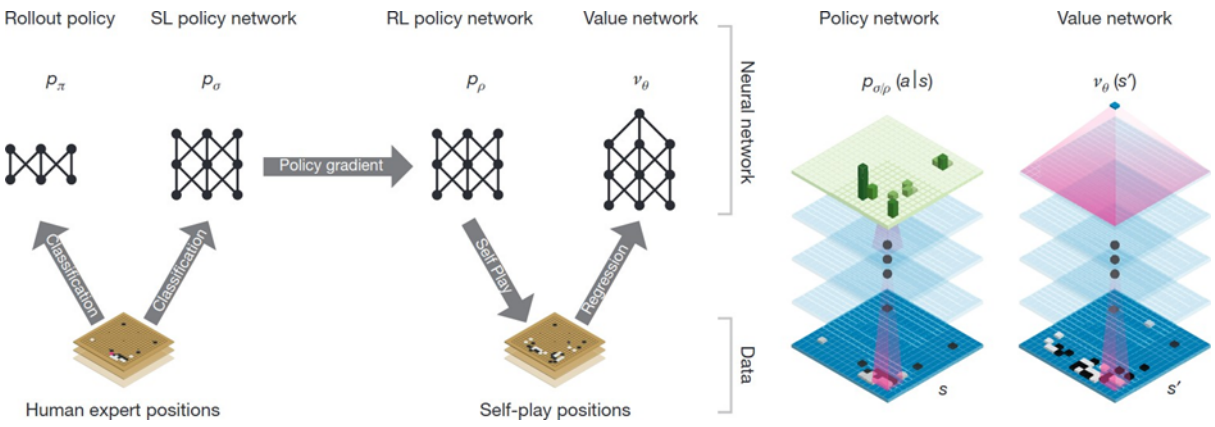
Each game played between RL networks takes a random position and sends it to the training dataset for the value network. So, this whole self-game process is only needed to collect data to train the value network.

2. A very fast model is trained, softmax linear regression, which the authors call fast rollout policy, on the same 30 million batches as SL. This network has a not very good accuracy of 24.2%, but it is 1500 times faster than SL. It uses a lot of preset patterns as features:

| 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 $3 \times 3$ pattern around move |

As a bottom line, one can estimate a random position as follows: take the value network estimate with a weight of λ, and play that position to the end of the game using fast rollout policy, and take the bottom line with a weight of (1 - λ).

**3. General scheme of the described neural networks**

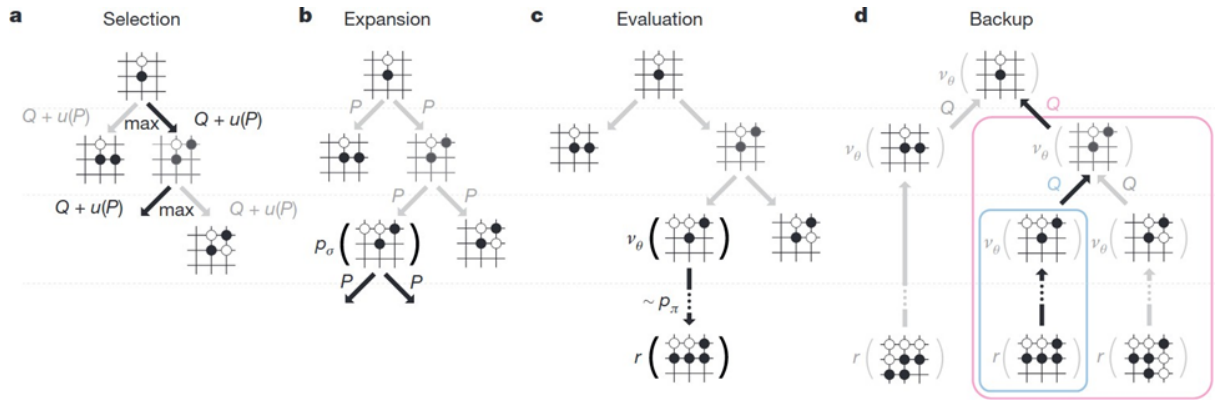



To summarize, we have:

- Fast rollout policy (on the picture Rollout policy) is a very fast model that predicts the professional's move in a given position. It is used to play the position to the end of the game to find out the result.

- SL policy network - says on this position with what probability each
Possible move will be made by the professional player. It is needed for choosing the direction of descent in a tree.
- RL policy network - needed only to prepare training data for the value
network. It is obtained from the SL policy network by playing with itself.
- Value network - by position tells us what chances we have to win the game.

**4. Tree search**




tree of positions is built, with the current one at the root. For each vertex the value Q is stored - the average of all the results of parties that are reachable from that vertex - essentially how much the given position leads to victory. This tree goes through many iterations of the following algorithm:

At each iteration, we go from the root and go down to where more Q + u(P). u(P) is a special additive that encourages exploration of new paths in the tree. u(P) is the greater the probability of that move in the SL-policy network prediction, and the smaller the more often we have walked through that node. When we reach a new leaf, it is evaluated through the value network and fast rollout policy. The resulting score updates all nodes on the path from leaf to root.

As a result, the best move is the node which was used most often (this is more stable than choosing the maximum by Q).

**5. Result**

Algorithm iterations during tree search are run in parallel, on multiple graphics cards (176 GPUs in AlphaGo, which played at the championship). As a result, we have a top-1 Go algorithm - AlphaGo which has beaten all top Go programs with the score 494 out of 495 games, and has also beaten multiple European champion Fan Hui (professional 2 dan) with the score 5:0.

**Conclusion**

The authors have succeeded for the first time in developing a program for playing Go, AlphaGo, based on a combination of deep neural networks and tree search, which plays at the level of the strongest human players. It was also the first to develop effective move selection and position estimation functions in Go, based on deep neural networks.

A further line of work was AlphaGo Master and AlphaGo Zero, which cemented the computer's breakthrough in the game of Go by beating professional players (including top-1 Lee Sedol) 60-0.

***References / Список литературы***


1. *Silver David et al.* Mastering the game of Go with deep neural networks and tree search. [Electronic Resource]. URL: https://doi.org/10.1038/nature16961/ (date of access: 07.11.2022).