TTIC 31230, Fundamentals of Deep Learning

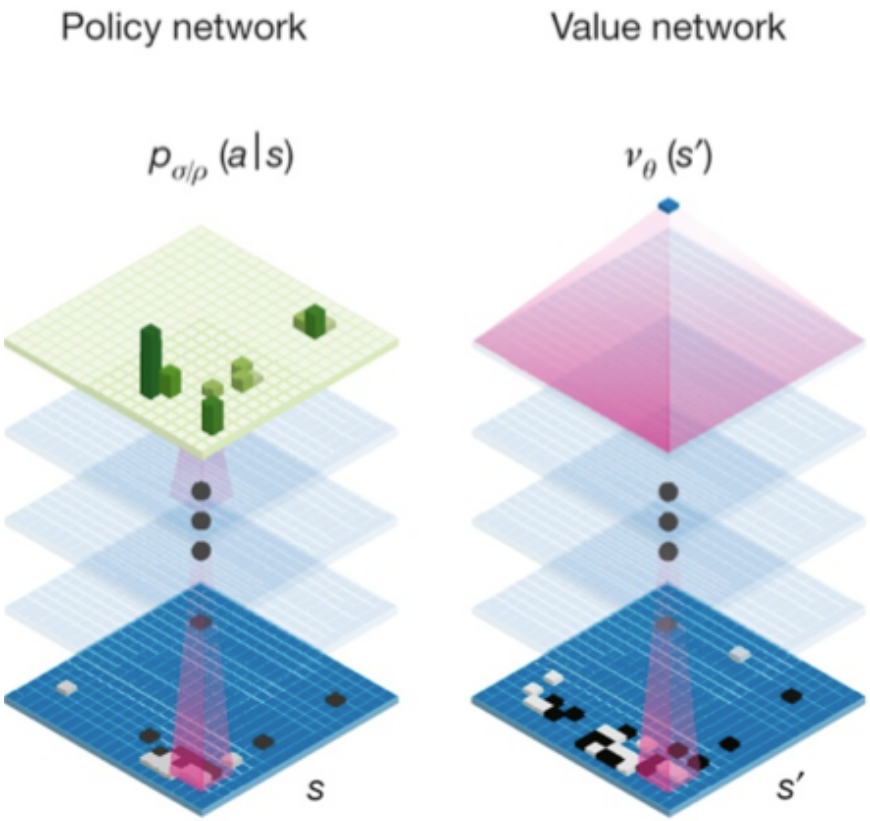
David McAllester, April 2017

AlphaGo

AlphaGo


nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last – a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO

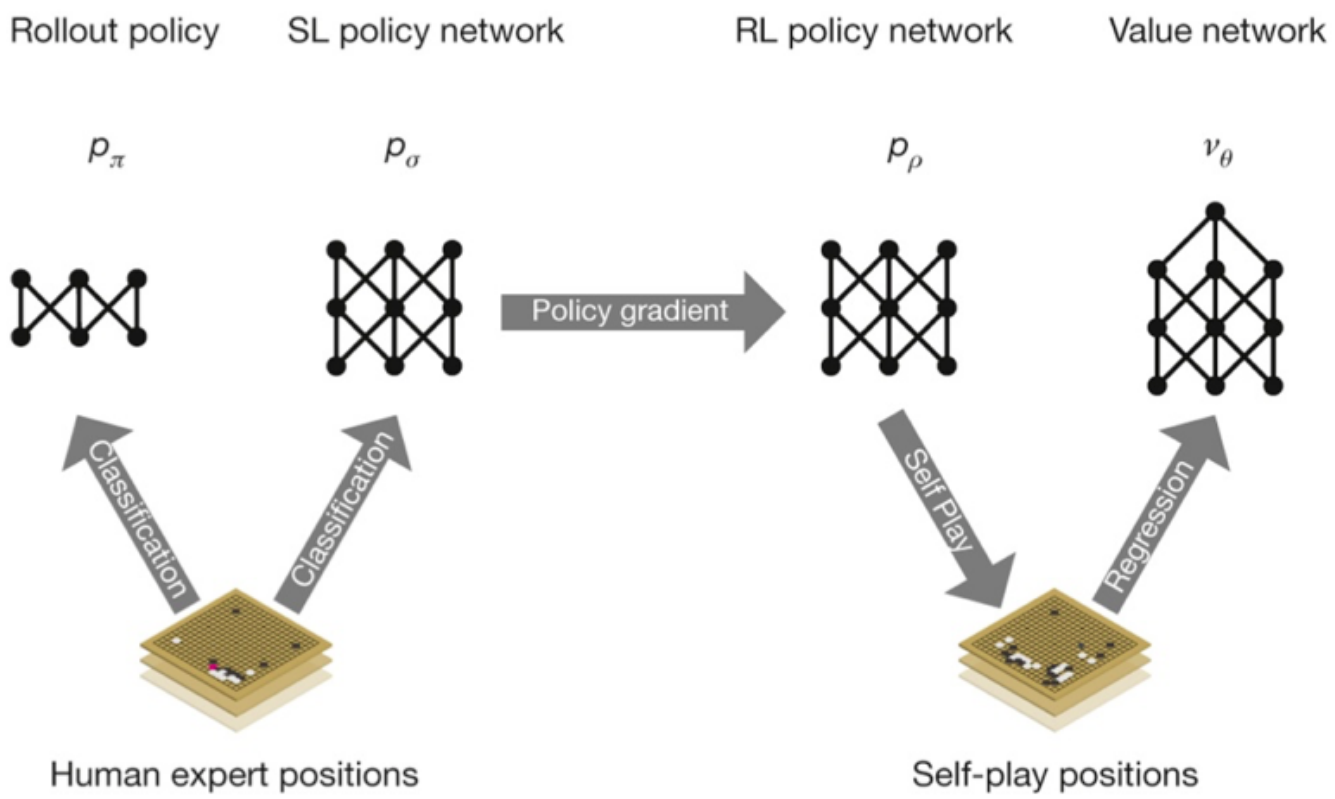
AlphaGo Policy and Value Networks



[Silver et al.]

The layers use 5×5 filters with Relu on 256 channels

AlphaGo Training



Imitation Learning

Policy Gradient and Regression

Board Features

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
Player color	1	Whether current player is black

Imitation Policy Learning

We trained a 13-layer policy network, which we call the SL policy network, from 30 million positions from the KGS Go Server.

The network predicted expert moves on a held out test set with an accuracy of 57.0% using all input features, and 55.7% using only raw board position and move history.

State-of-the-art from other research groups was 44.4% at the date of submission.

Small improvements in move prediction lead to large improvements in probability of win.

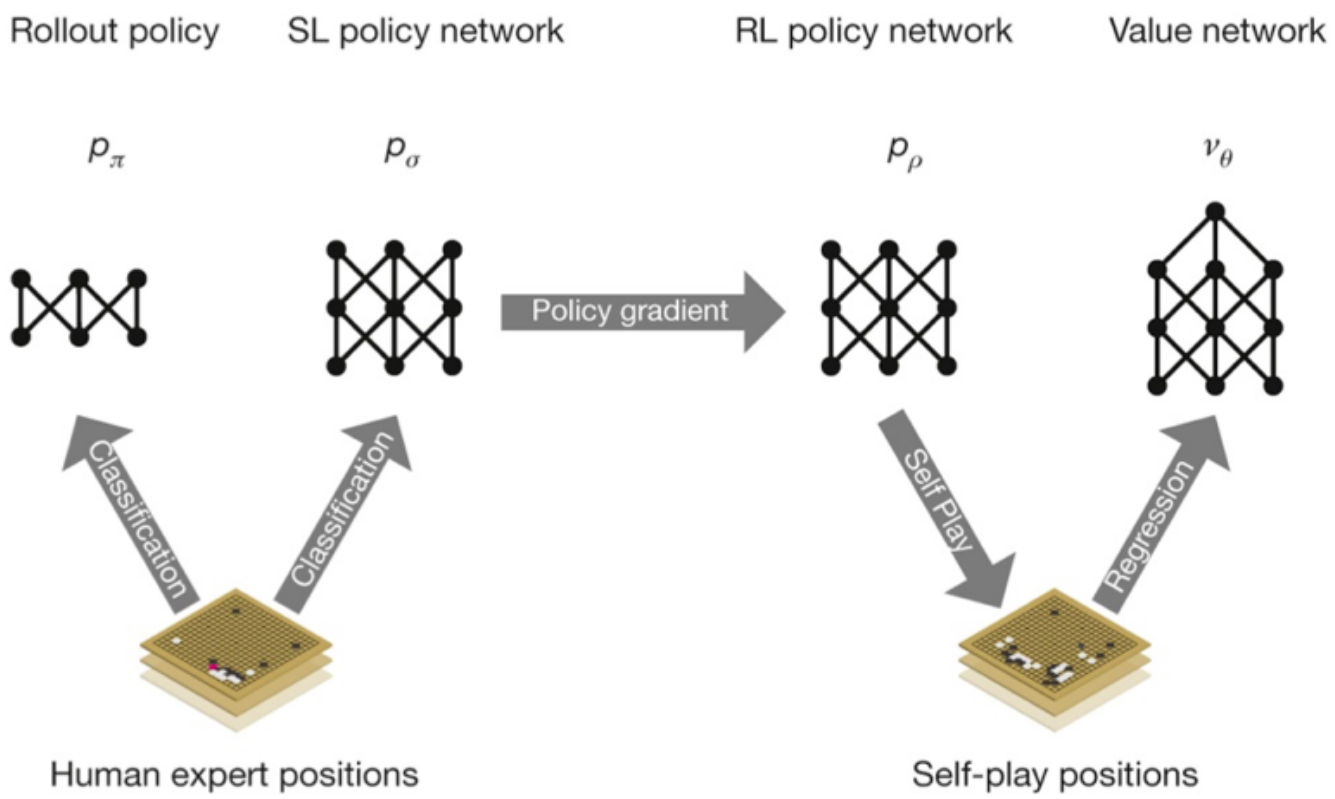
Fast Rollout Policy

Softmax of linear combination of pattern 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 3×3 pattern around move
Self-atari	1	Move allows stones to be captured
Last move distance	34	Manhattan distance to previous two moves
Non-response pattern	32207	Move matches 12-point diamond pattern centred around move

An accuracy of 24.2%, using just 2μs to select an action, rather than 3ms for the policy network.

Policy Gradient Training



Imitation Learning

Policy Gradient and Regression

Policy Gradient for Go

AlphaGo does policy gradient on "self play".

Actually, the program plays games against fixed (not updated) earlier versions of itself.

The paper states that this diversity of opponents prevents overfitting

Machine learning for board games by self play dates back to the Samuel's checkers program in 1959.

Samuel's observed cycles in the feature weights.

Playing against a mixture of earlier versions may prevent cycles.

Policy Gradient for Go

$$\nabla_\Theta \, R(\Theta) = \nabla_\Theta \, \mathrm{E}\left[z \mid \Theta\right] \quad \begin{cases} z = 1 & \text{for a win} \\ z = -1 & \text{for a loss} \end{cases}$$

Let a_i be the program's move and b_i be the opponents response.

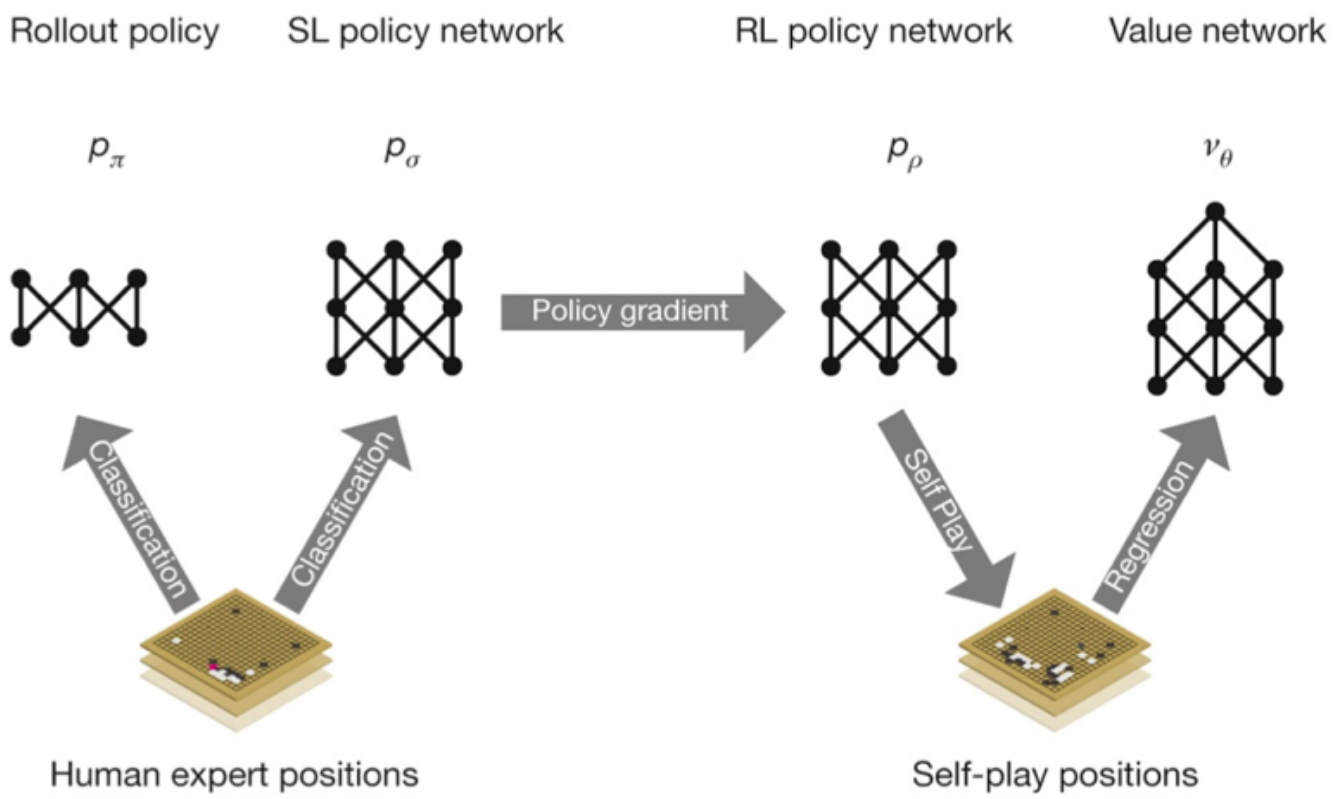
An episode is a series of moves $a_1, b_1, a_2, b_2, \ldots, a_N, b_N$.

$$\Theta_\pi \texttt{ += } z \, \nabla_{\Theta_\pi} \, \ln \pi(a_t | s_t; \Theta_\pi)$$

When played head-to-head, the policy gradient trained RL policy network won more than 80% of games against the immitation trained SL policy network.

Using no search at all, the RL policy network won 85% of games against the previous leading go program.

Value Network Regression



Imitation Learning

Policy Gradient and Regression

Regression Training of Value Function

Using self-play of the final RL policy we generate a database of 30 million pairs (s, z) where s is a board position and $z \in \{-1, 1\}$ is an outcome and each pair is from a different game.

We then train a value network by regression.

$$\Theta^* = \operatorname*{argmin}_{\Theta} \mathrm{E}_{(s,z)} \left[(V(s, \Theta) - z)^2 \right]$$

$$\Theta \texttt{ -= } \eta (V(s, \Theta) - z) \nabla_{\Theta} V(s, \Theta)$$

Monte Carlo Tree Search (MCTS)

In actual play a move selected by tree search.

Each board state data structure s has a list of edge data structure (s, a) for each legal action from s.

The edge data structure (s, a) may or may not point to a new state data structure. (We have "dangling" edges.)

Monte Carlo Tree Search (MCTS)

Each state s in the tree has a value

$$s.V = (1 - \lambda)V(s) + \lambda z$$

where $V(s)$ is the value network value and z is value of a rollout from s using the fast rollout policy.

Each edge (s, a) has the following properties

- $(s, a).P$ is the **imitation-trained** policy value for a at s.
- $(s, a).N$ is count of the number of traversals of (s, a).
- $(s, a).V$ is the sum over traversals of (s, a) of $s_L.V$ where s_L is the leave state reached in that traversal.

Traversing the Tree

We traverse the tree by starting at the root and, recursively, from state s traversing the edge (s, a^*) where

$$a^* = \underset{a}{\operatorname{argmax}} \ \frac{(s,a).V}{(s,a).N} + \gamma \frac{(s,a).P}{1 + (s,a).N}$$

The paper does not explain the case $(s,a).N = (s,a).V = 0$.

If the selected edge (s,a) is "dangling" (there is no next state attached) then a new state (and all of its dangling edges) may be created.

Once the search is deemed complete, the most traversed edge from the root is selected as the move.

END