David Silver et al. from Google DeepMind

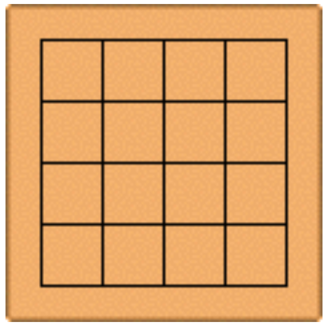
Mastering the game of Go with deep neural networks and tree search

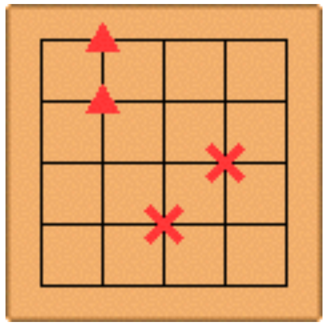
Article overview by
Ilya Kuzovkin

Reinforcement Learning Seminar
University of Tartu, 2016

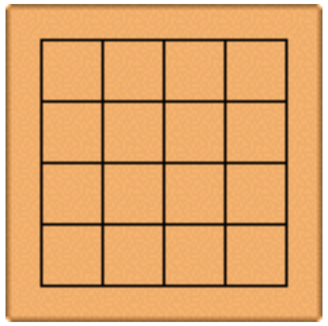
The Game of Go

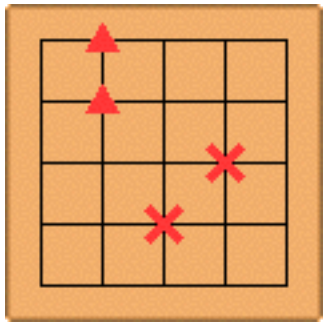
Board



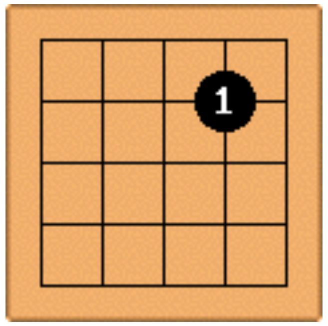


Board

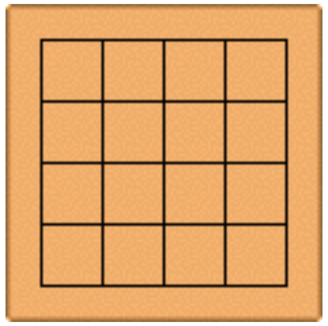


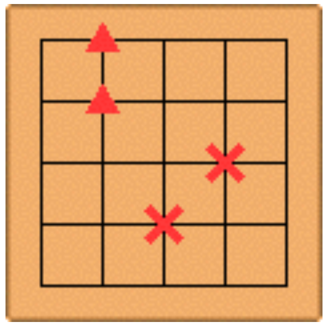


Stones

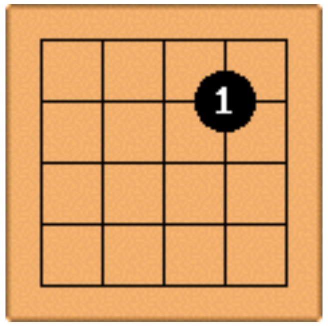


Board

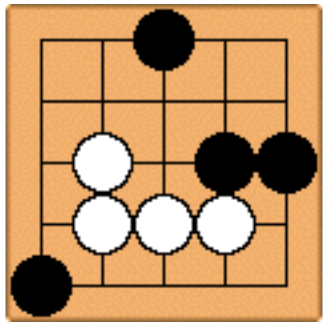




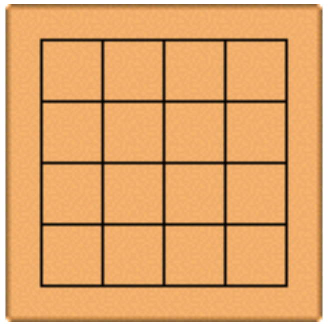
Stones

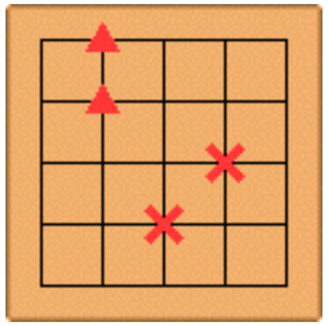


Groups

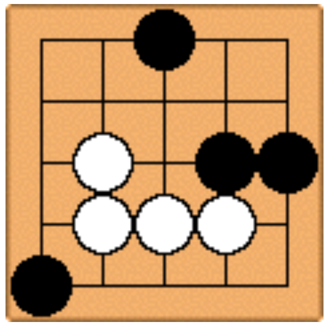


BOARD

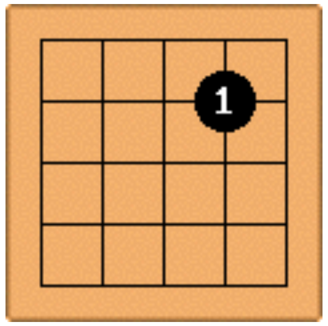




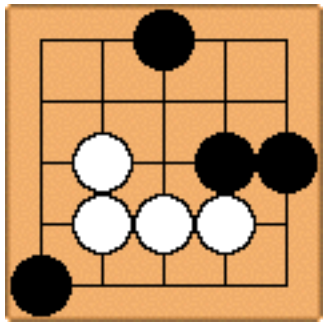
LIBERTIES



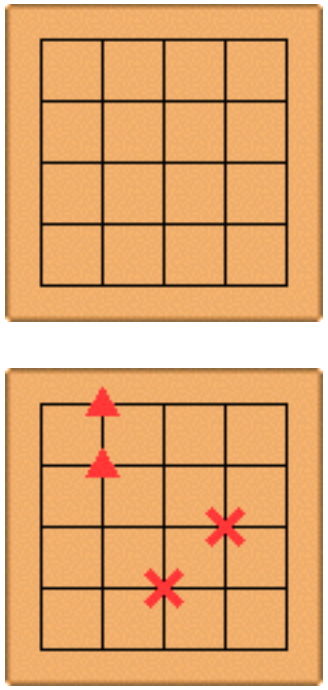
STONES



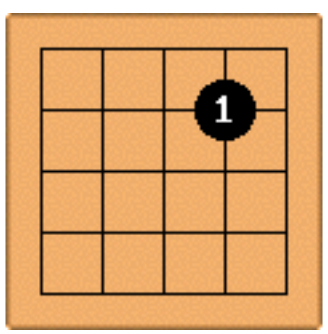
GROUPS



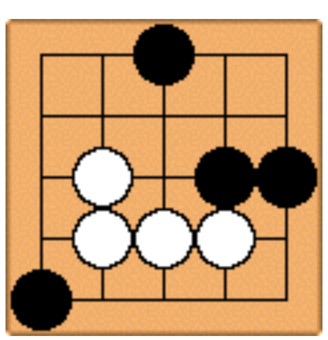
Board



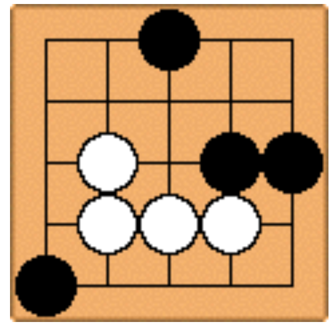
Stones



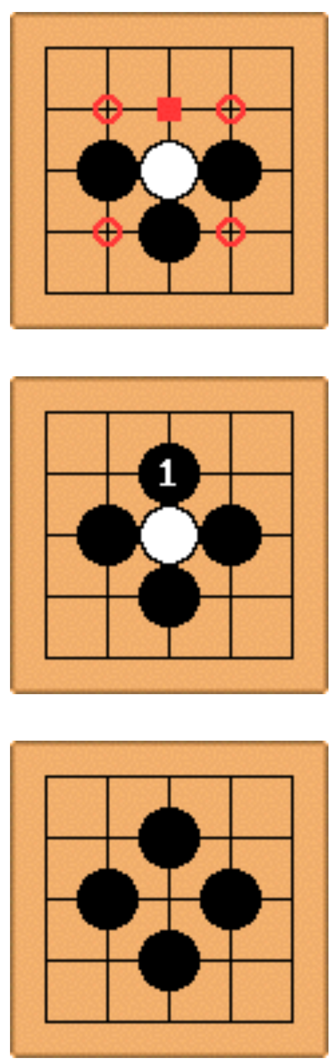
Groups



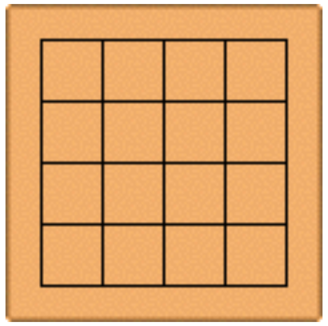
Liberties

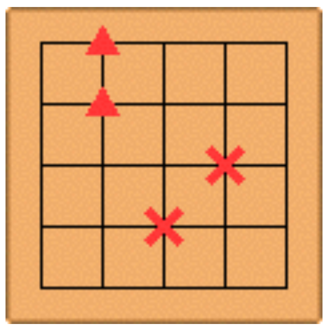


Capture

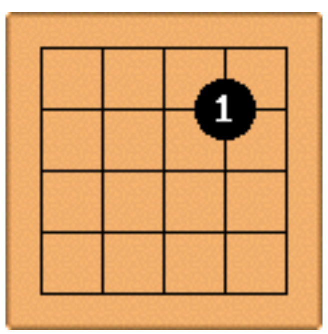


Board

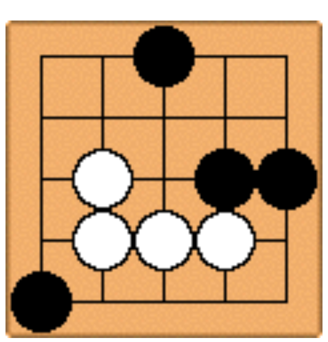




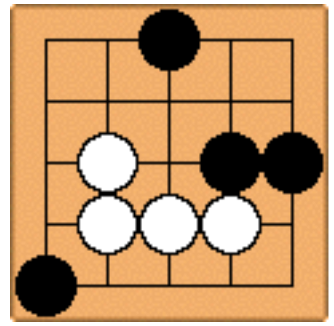
Stones



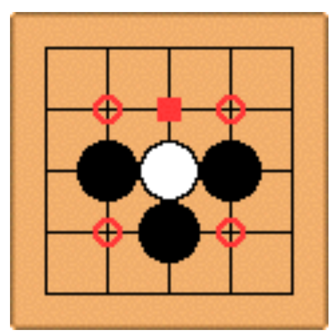
Groups

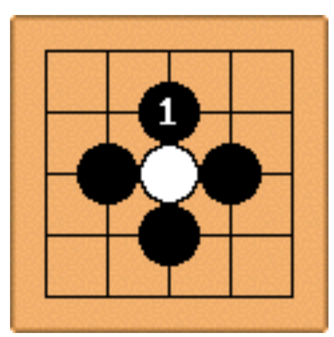


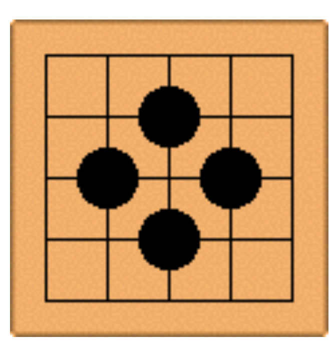
Liberties



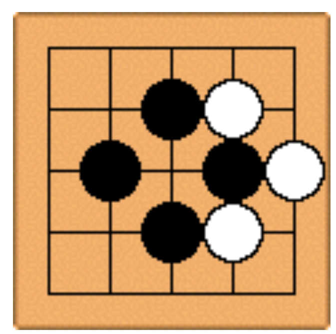
Capture

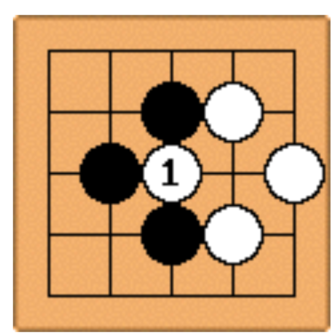


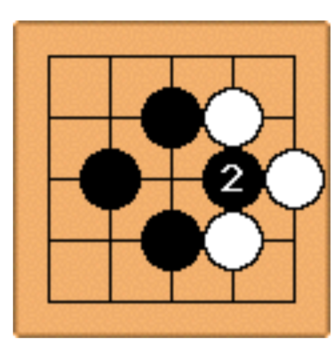




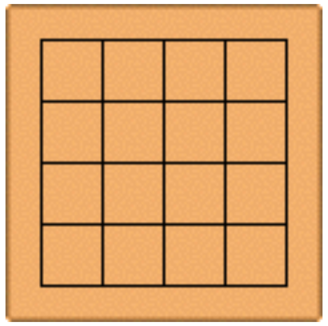
Ko

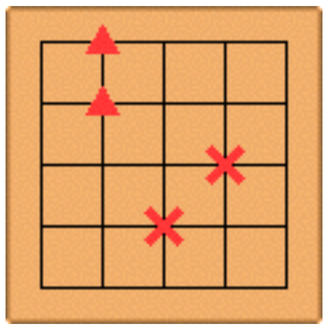




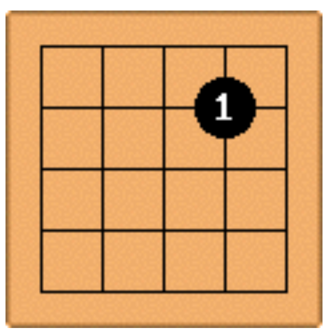


Board

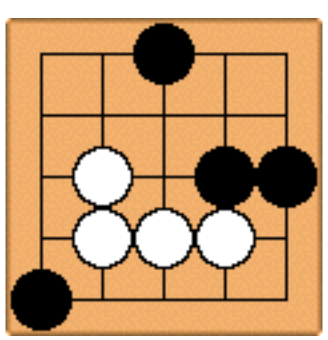




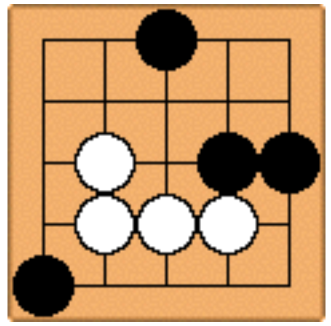
Stones



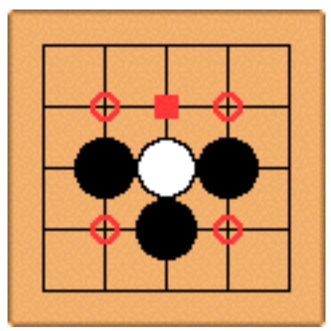
Groups

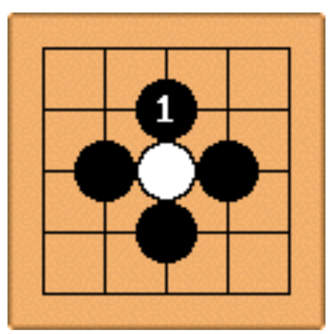


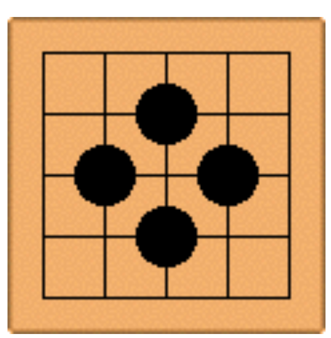
Liberties



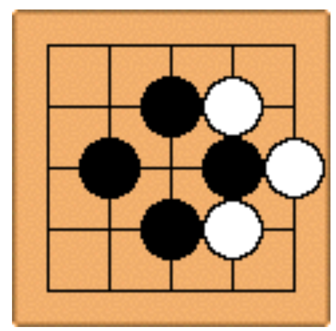
Capture

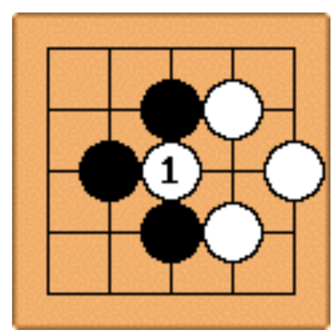


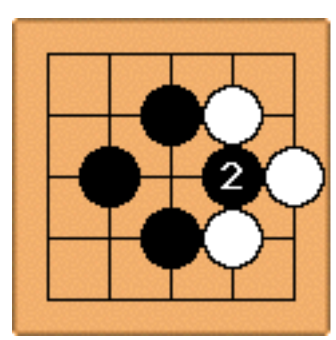




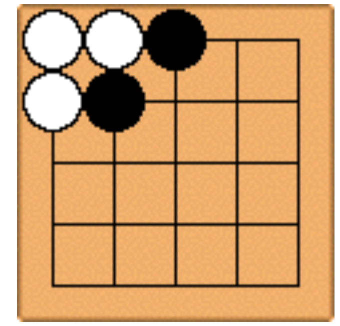
Ko



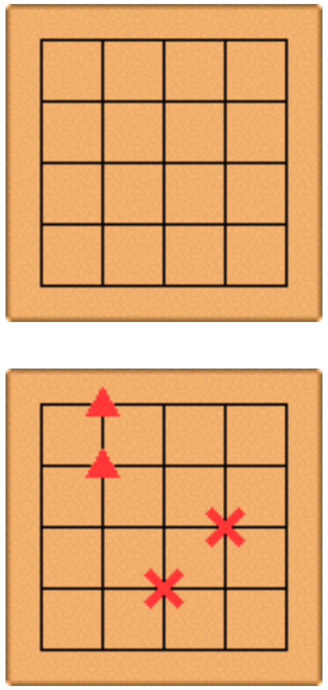




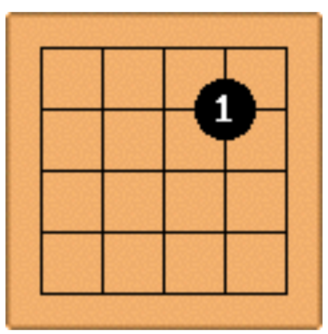
Examples



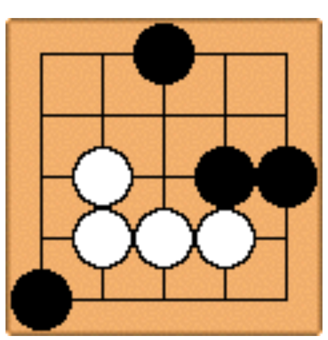
Board



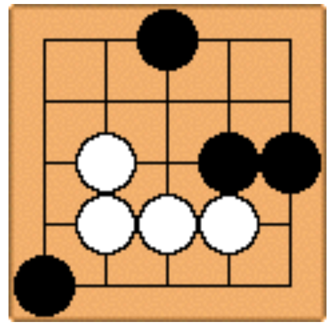
Stones



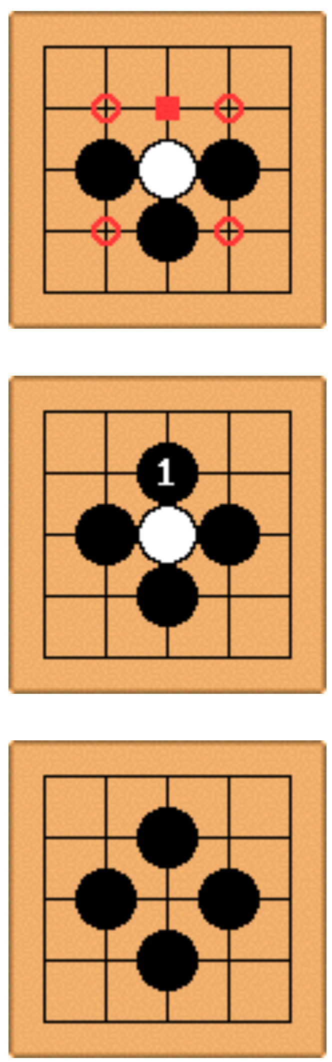
Groups



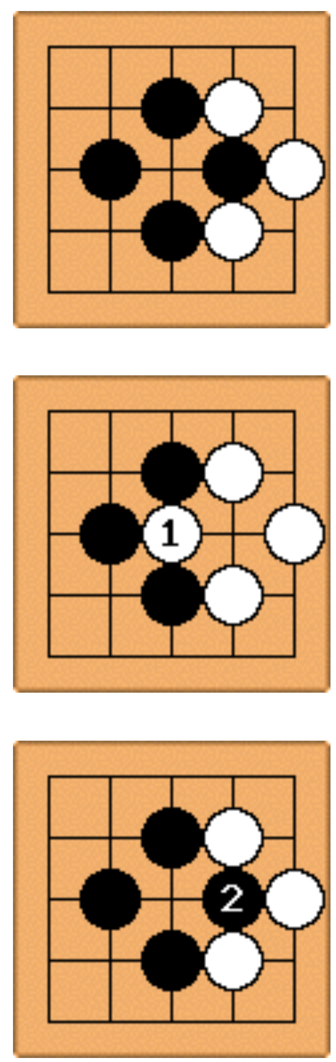
Liberties



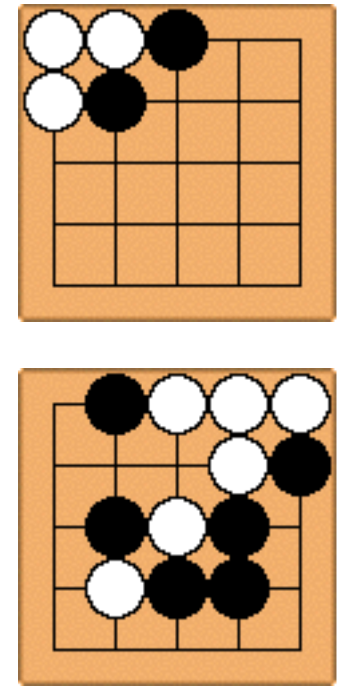
Capture



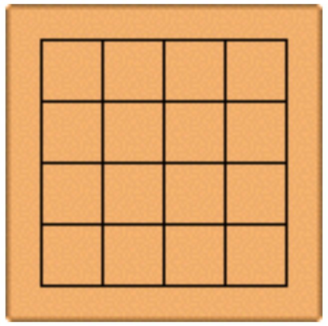
Ko

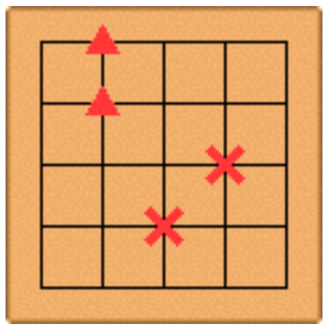


Examples

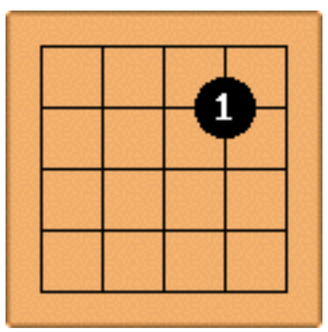


BOARD

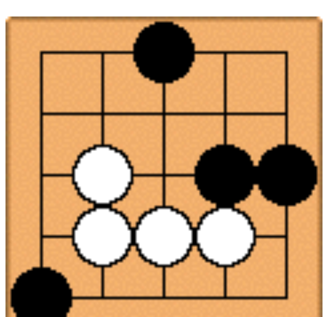




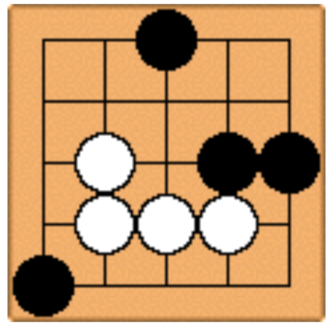
STONES



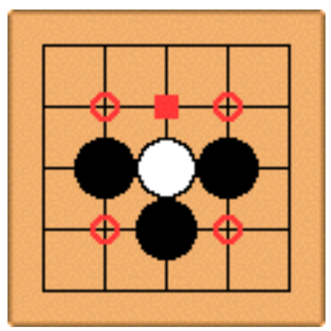
GROUPS

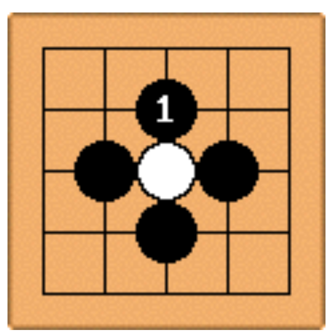


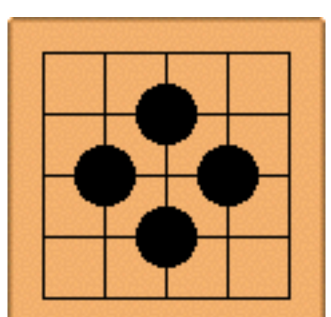
LIBERTIES



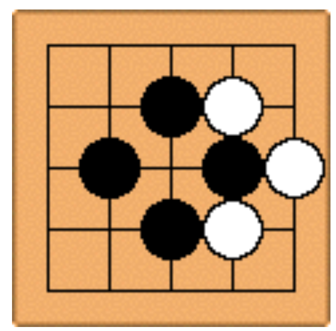
CAPTURE

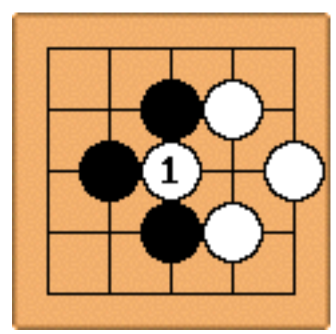


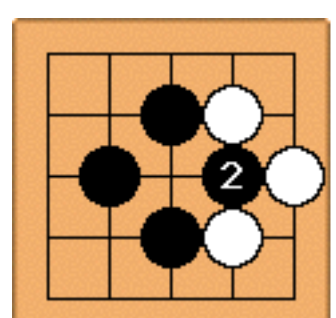




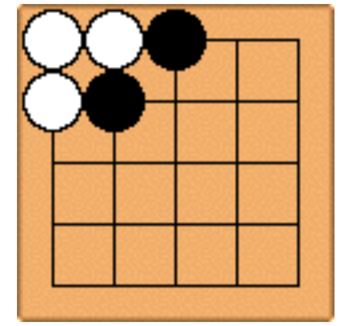
KO

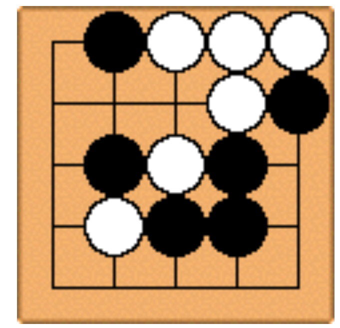


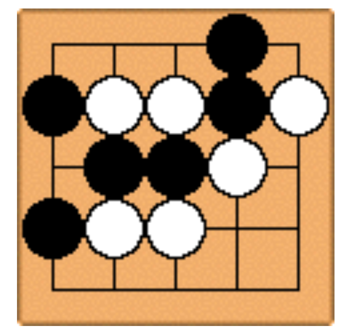




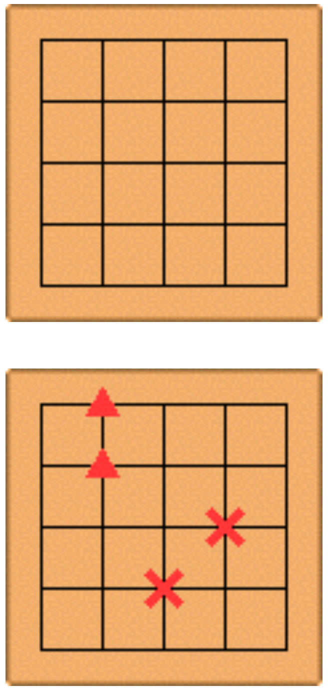
EXAMPLES



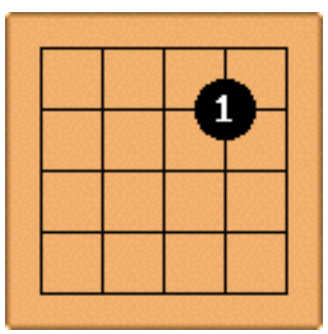




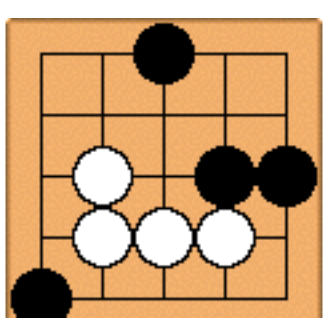
Board



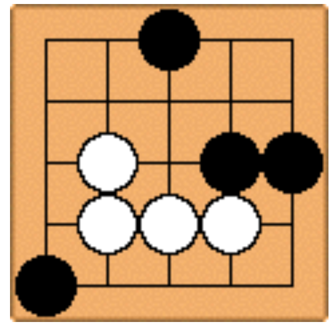
Stones



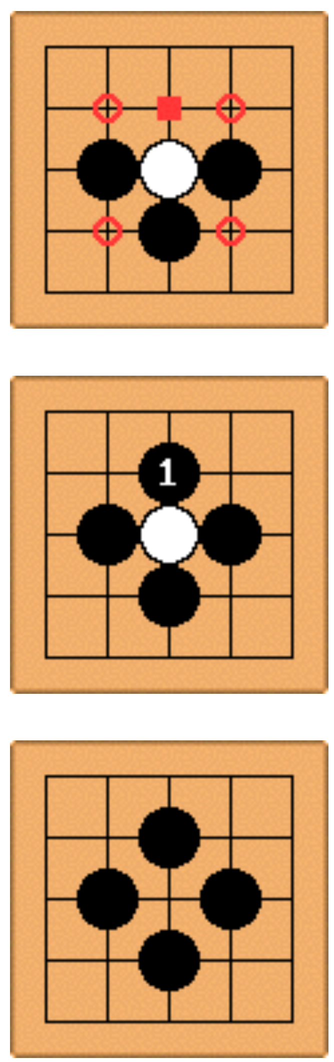
Groups



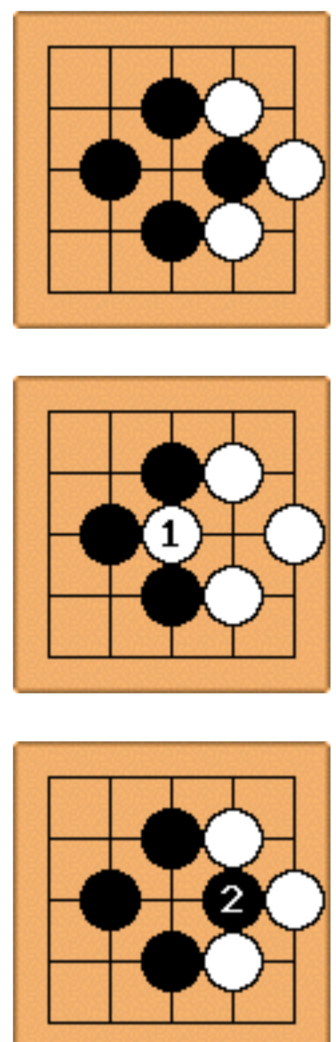
Liberties



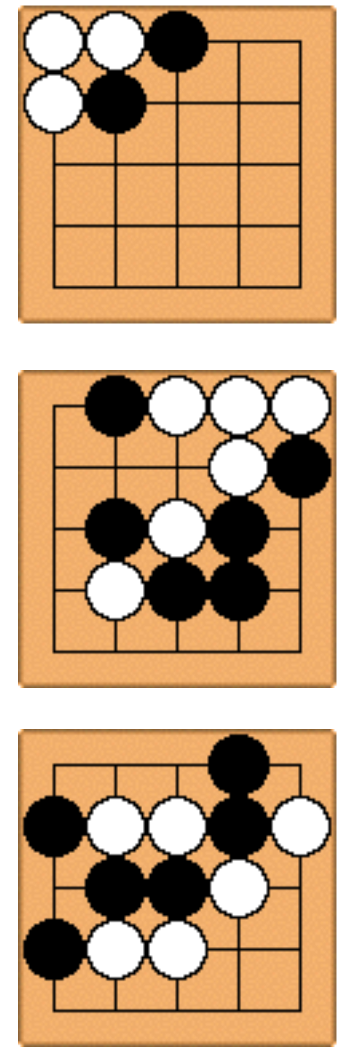
Capture



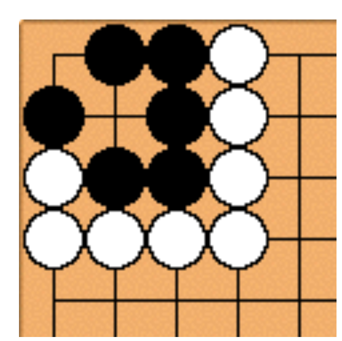
Ko



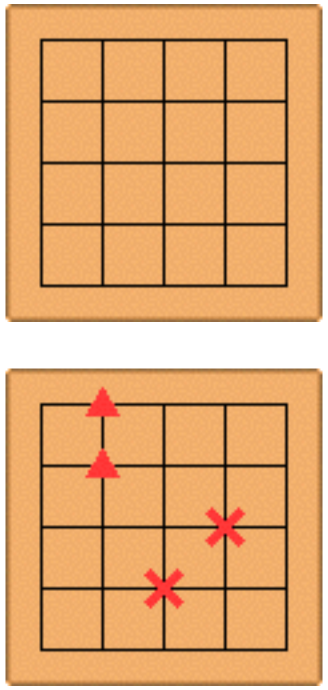
Examples



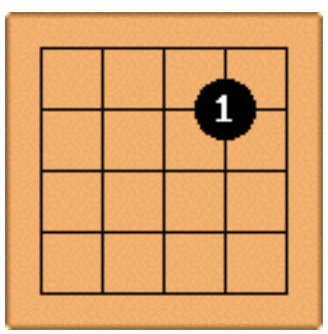
Two eyes



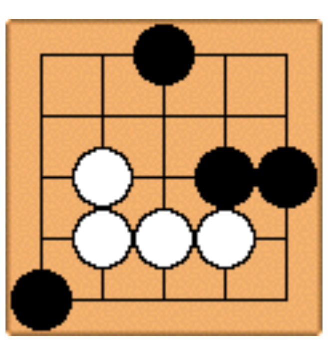
Board



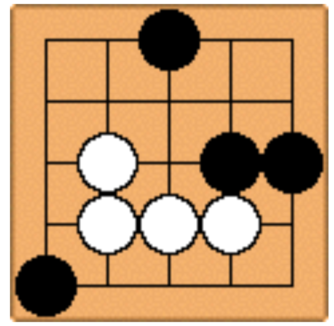
Stones



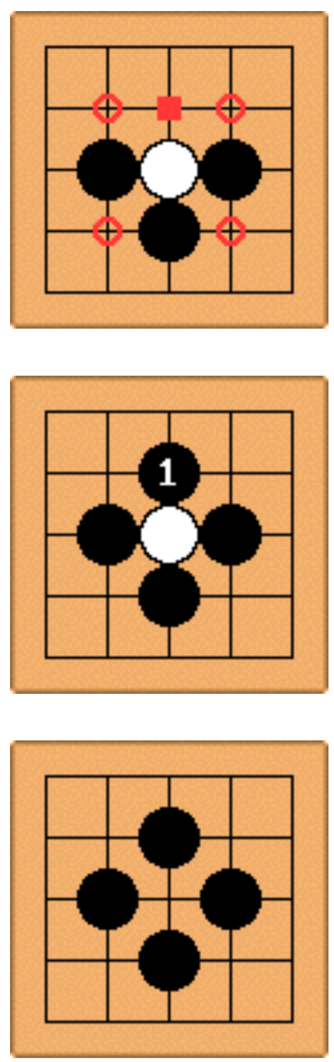
Groups



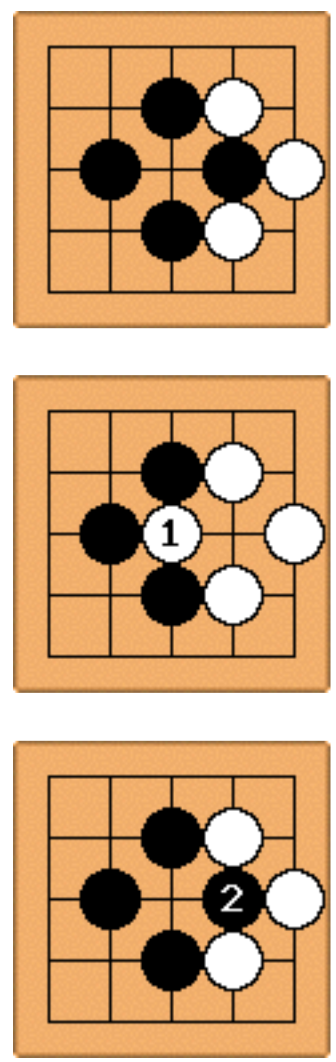
Liberties



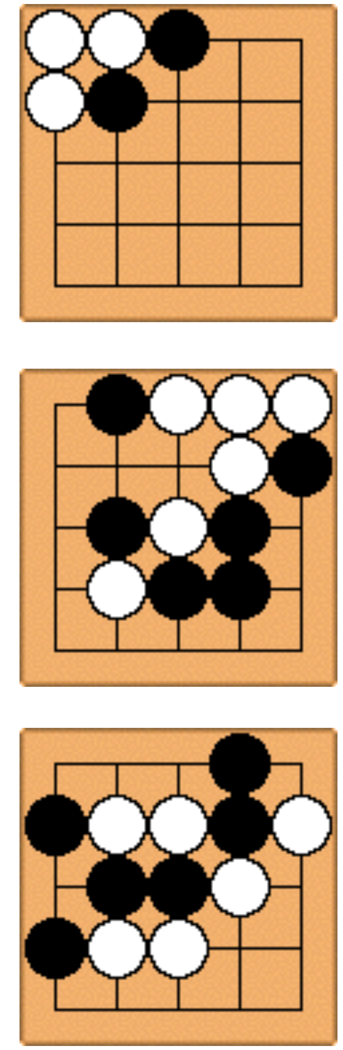
Capture



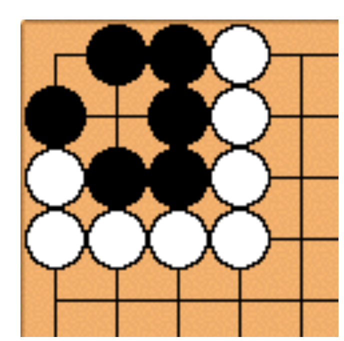
Ko



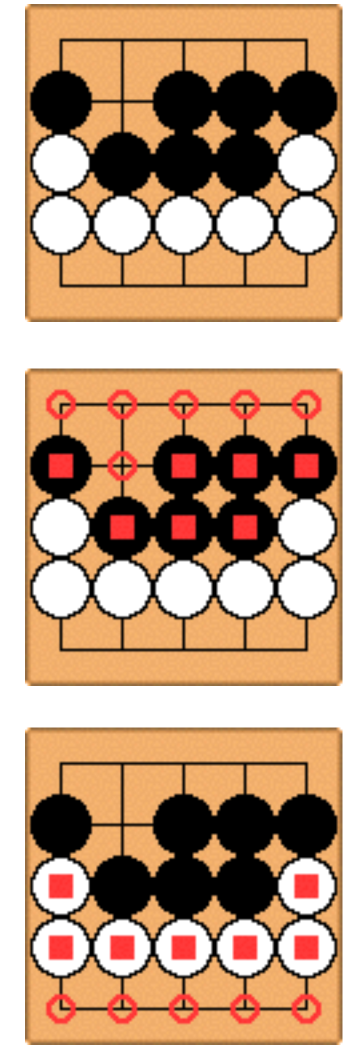
Examples



Two eyes



Final count



AlphaGo

Training

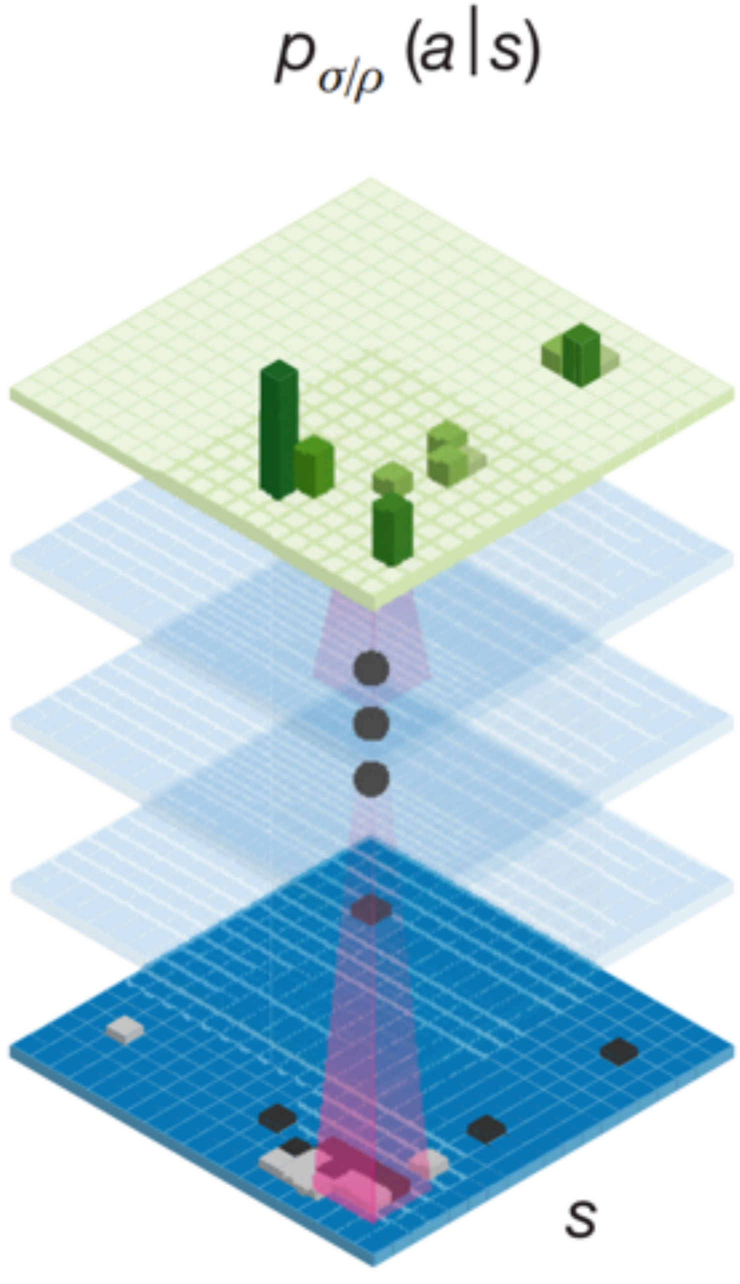
TRAINING THE BUILDING BLOCKS

SUPERVISED CLASSIFICATION	REINFORCEMENT	SUPERVISED REGRESSION
Supervised policy network $p_\sigma(a\|s)$	**Reinforcement** policy network $p_\rho(a\|s)$	**Value** network $v_\theta(s)$
Rollout policy network $p_\pi(a\|s)$		
Tree policy network $p_\tau(a\|s)$		

Supervised

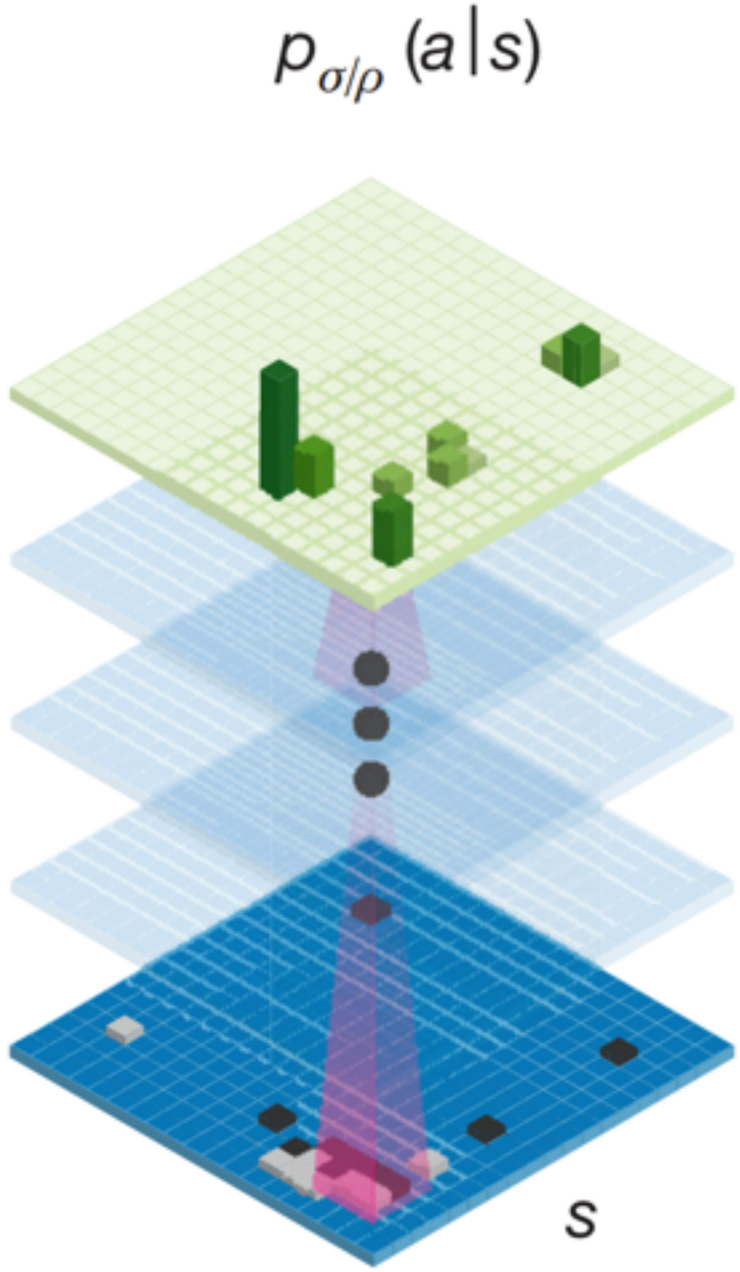
policy network

$p_\sigma(a|s)$



Supervised policy network

$$p_\sigma(a|s)$$



Softmax

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

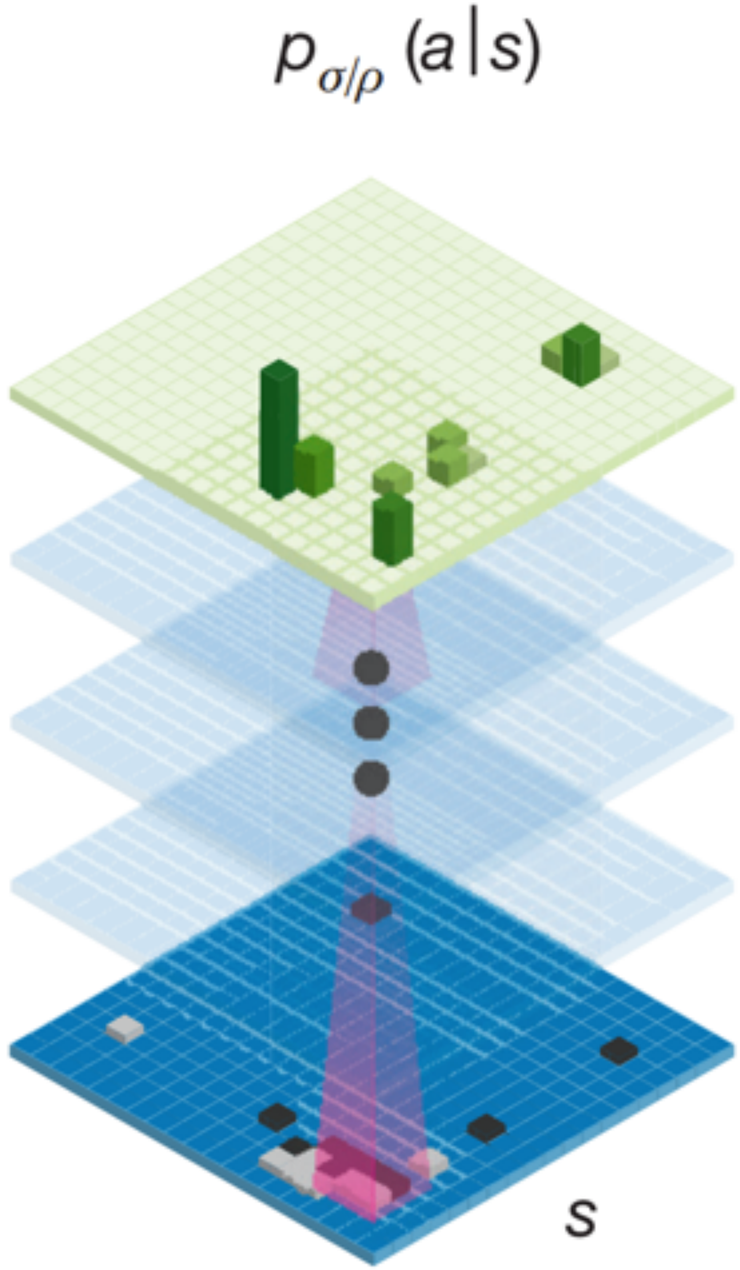
1 convolutional layer 5x5
with *k*=192 filters, ReLU

19 x 19 x 48 input

Supervised policy network

$$p_\sigma(a|s)$$

- 29.4M positions from KGS games between 6 to 9 *dan* players



Softmax

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

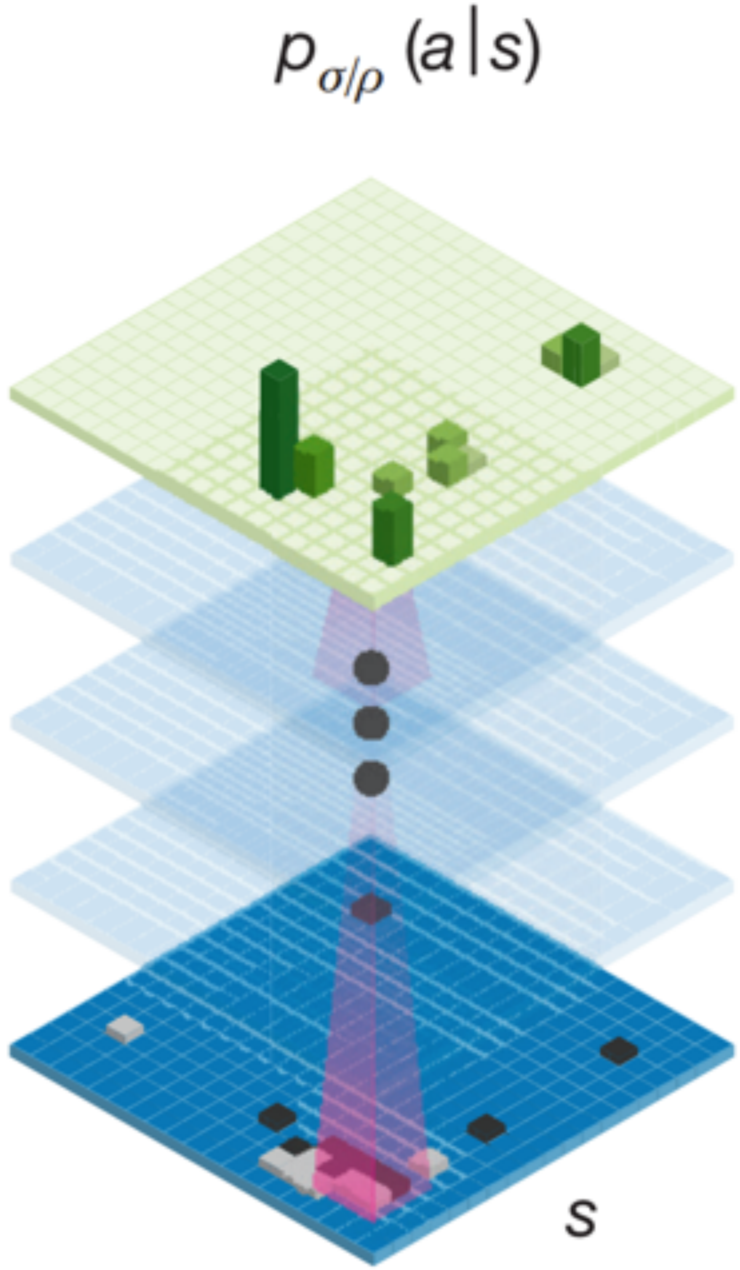
1 convolutional layer 5x5
with *k*=192 filters, ReLU

19 x 19 x 48 input

Supervised policy network

$$p_\sigma(a|s)$$

- 29.4M positions from KGS games between 6 to 9 *dan* players



$$\Delta\sigma = \frac{\alpha}{m}\sum_{k=1}^{m}\frac{\partial \log p_\sigma(a^k|s^k)}{\partial\sigma}$$

Softmax

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

1 convolutional layer 5x5
with *k*=192 filters, ReLU

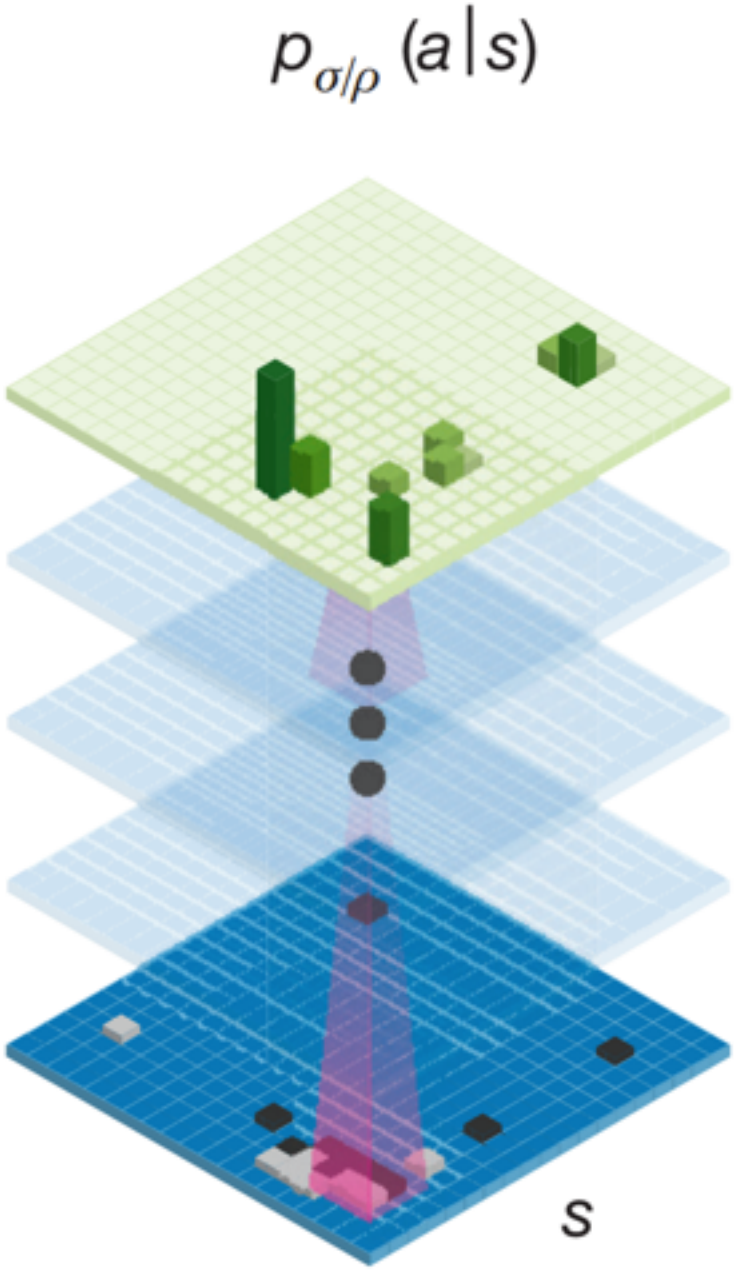
19 x 19 x 48 input

- stochastic gradient ascent
- learning rate α = 0.003, halved every 80M steps
- batch size *m* = 16
- 3 weeks on 50 GPUs to make 340M steps

Supervised policy network

$p_\sigma(a|s)$

- 29.4M positions from KGS games between 6 to 9 *dan* players
- Augmented: 8 reflections/rotations
- Test set (1M) accuracy: 57.0%
- 3 ms to select an action



$$\Delta\sigma = \frac{\alpha}{m} \sum_{k=1}^{m} \frac{\partial \log p_\sigma(a^k|s^k)}{\partial \sigma}$$

Softmax

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

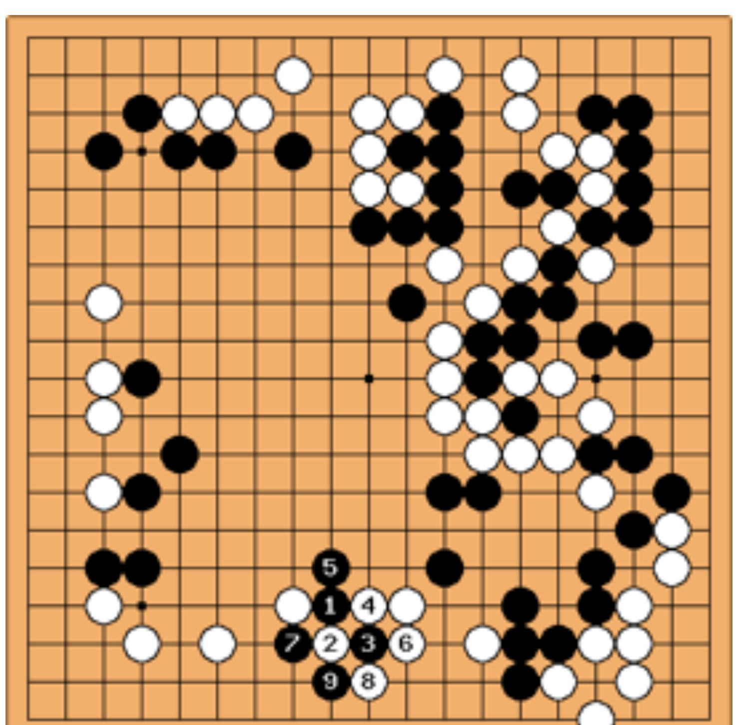
1 convolutional layer 5x5
with *k*=192 filters, ReLU

19 x 19 x 48 input

- stochastic gradient ascent
- learning rate α = 0.003, halved every 80M steps
- batch size *m* = 16
- 3 weeks on 50 GPUs to make 340M steps

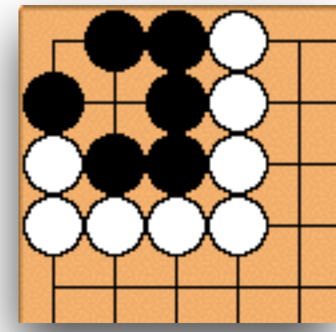
19 X 19 X 48 INPUT

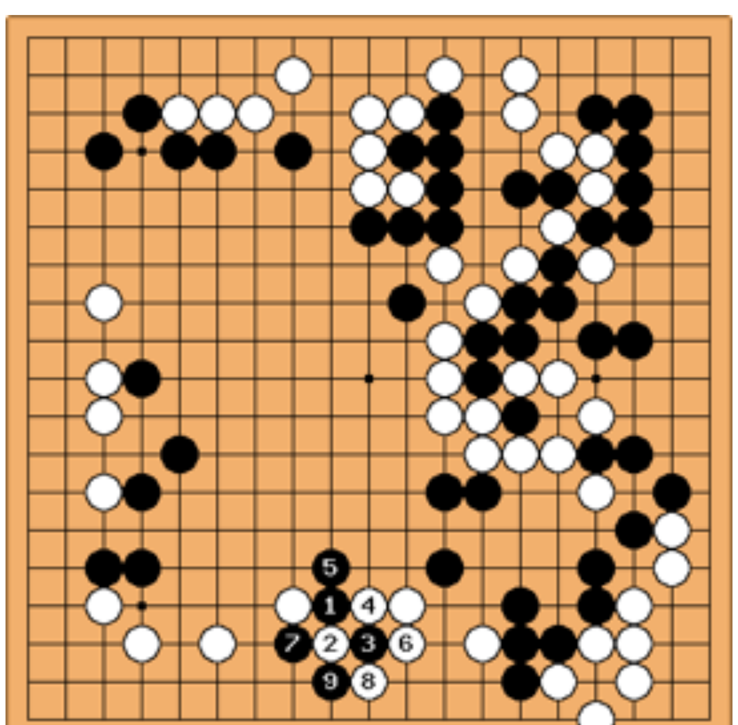
Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari 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



19 X 19 X 48 INPUT

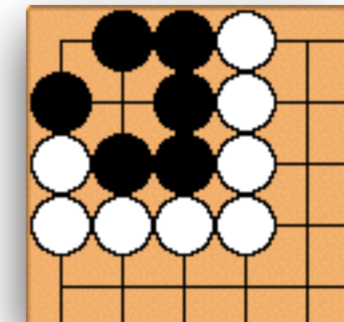
Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari 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

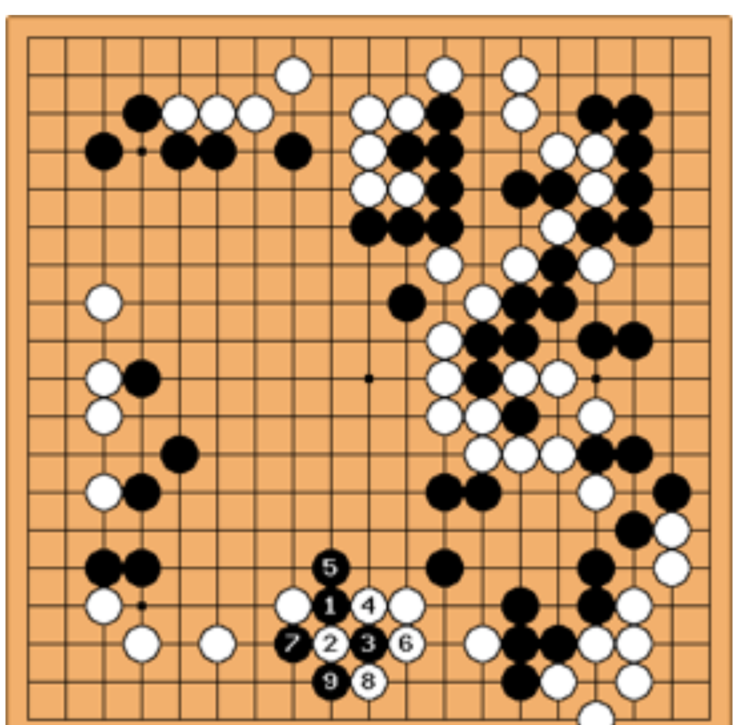


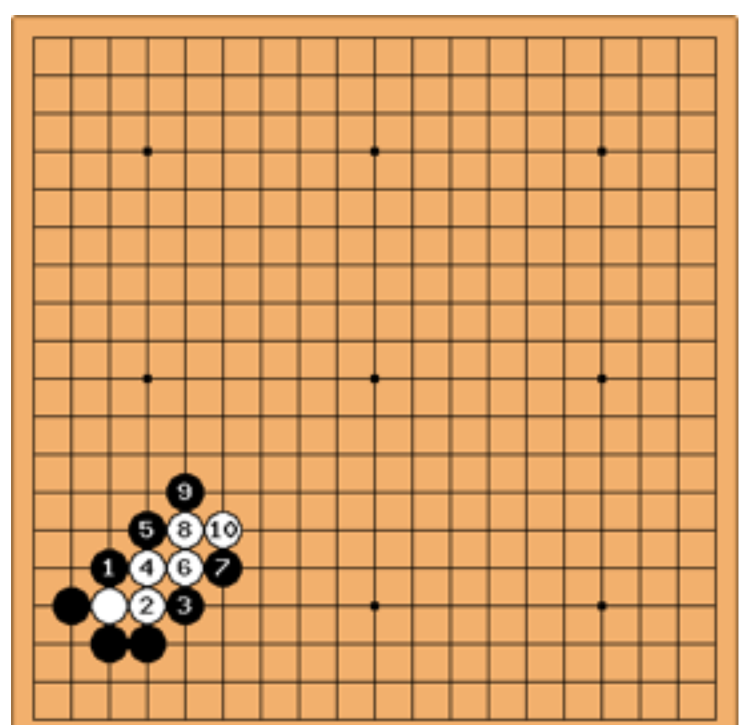


19 X 19 X **48** INPUT

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

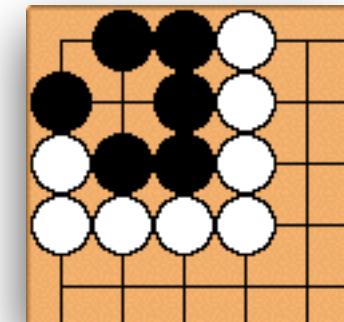


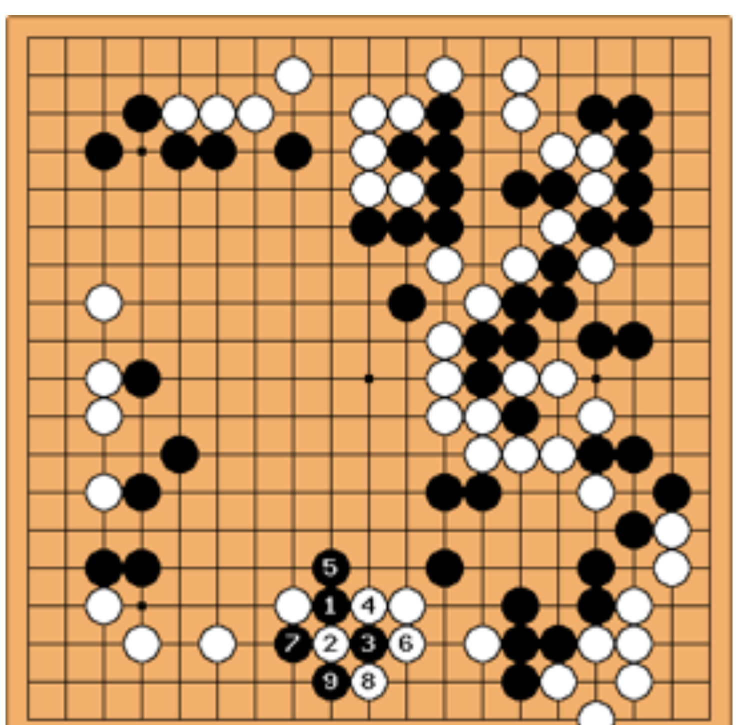


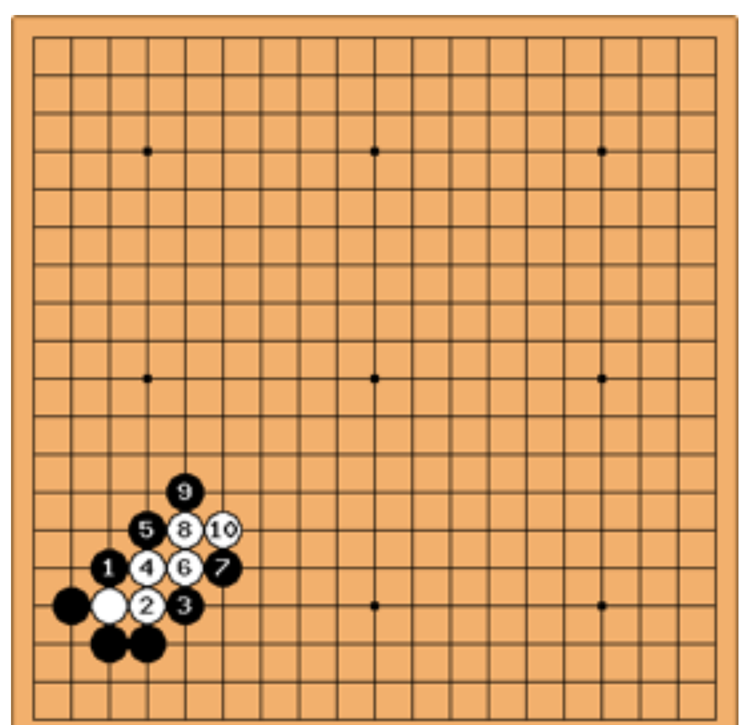


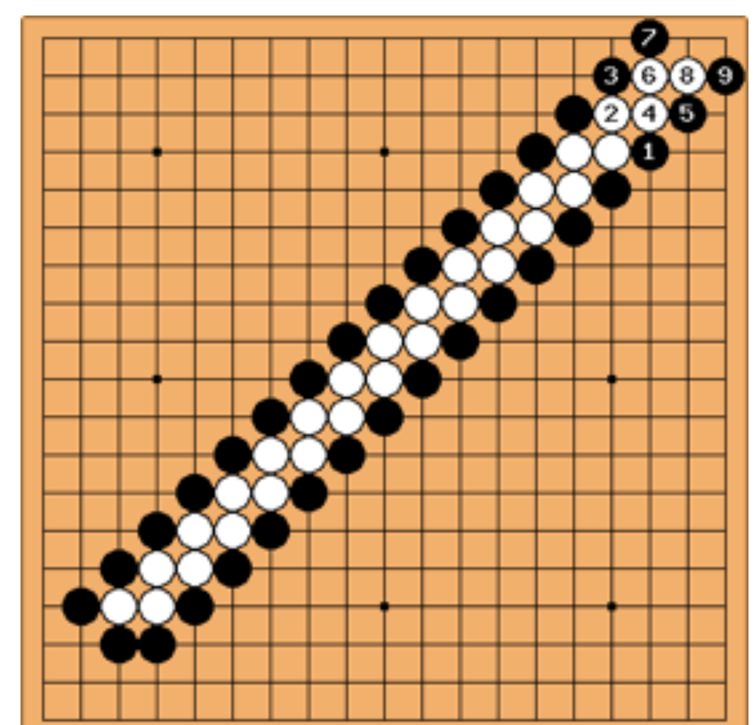
19 X 19 X 48 INPUT

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









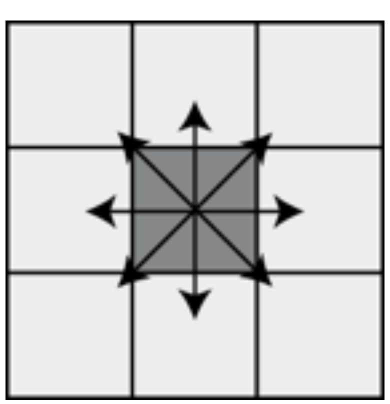
Rollout policy $p_\pi(a|s)$

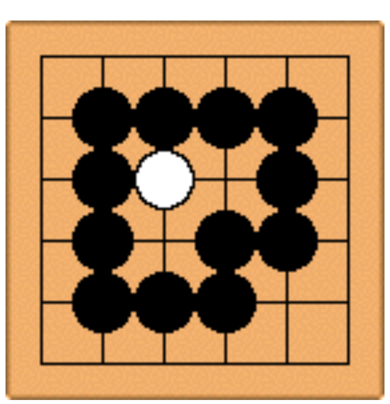
- Supervised — same data as $p_\sigma(a|s)$
- Less accurate: 24.2% (vs. 57.0%)
- Faster: 2μs per action (1500 times)
- Just a linear model with softmax

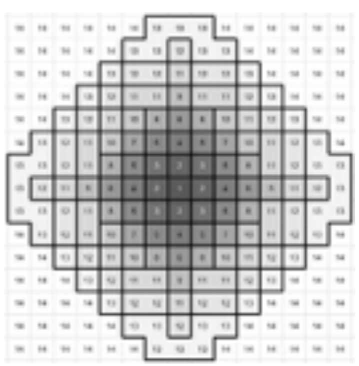
Rollout policy $p_\pi(a|s)$

- Supervised — same data as $p_\sigma(a|s)$
- Less accurate: 24.2% (vs. 57.0%)
- Faster: 2μs per action (1500 times)
- Just a linear model with softmax

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





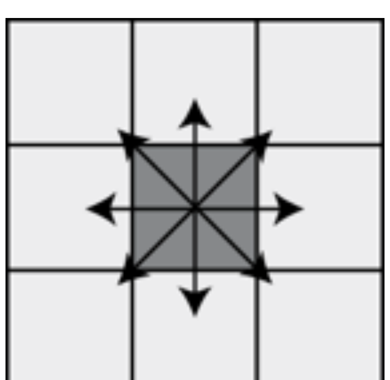


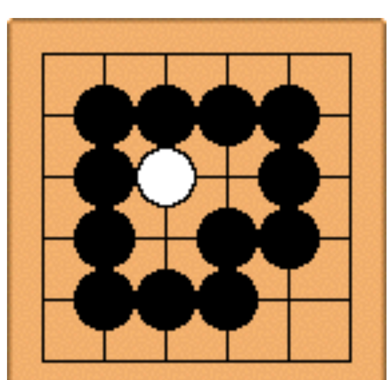
Rollout policy $p_\pi(a|s)$

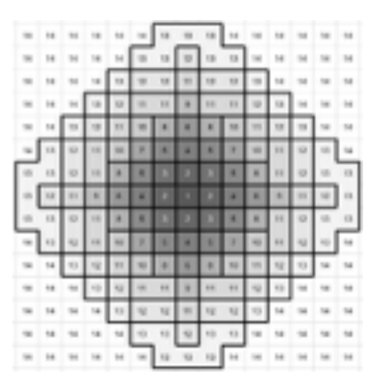
Tree policy $p_\tau(a|s)$

- Supervised — same data as $p_\sigma(a|s)$
- Less accurate: 24.2% (vs. 57.0%)
- Faster: 2μs per action (1500 times)
- Just a linear model with softmax

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







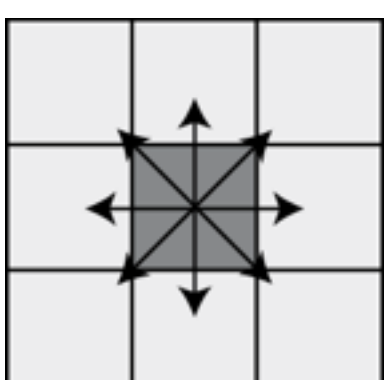
Rollout policy $p_\pi(a|s)$

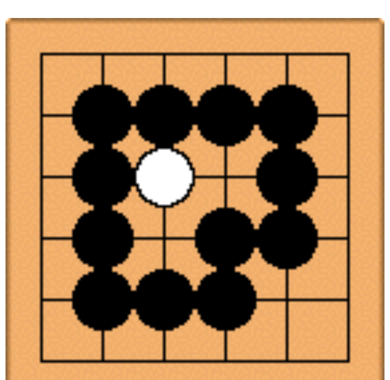
- Supervised — same data as $p_\sigma(a|s)$
- Less accurate: 24.2% (vs. 57.0%)
- Faster: 2μs per action (1500 times)
- Just a linear model with softmax

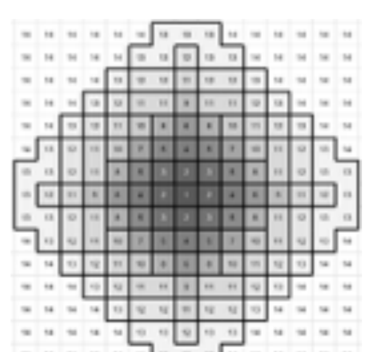
Tree policy $p_\tau(a|s)$

- "*similar to the rollout policy but with more 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

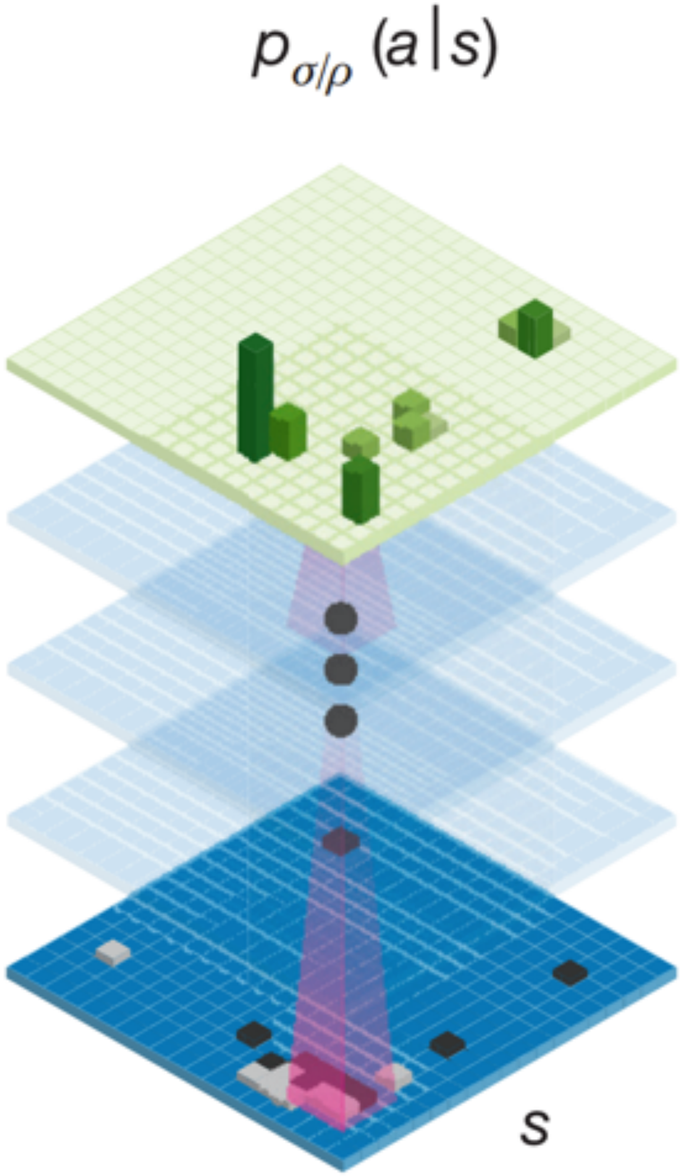






Reinforcement policy network

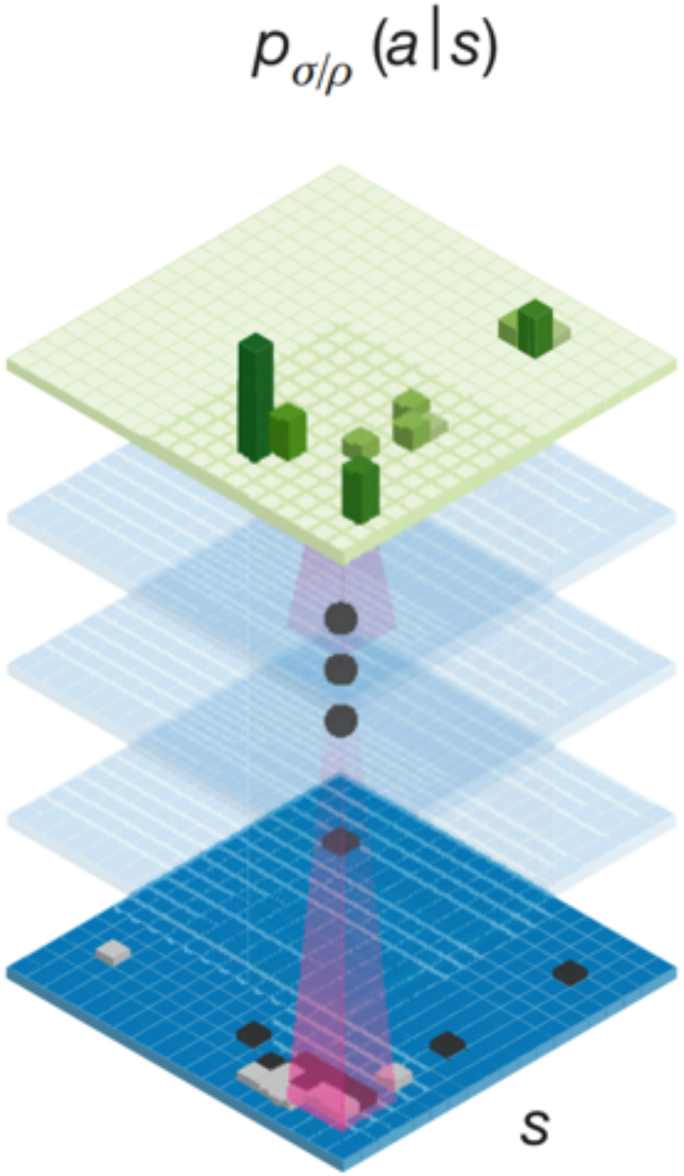
$$p_\rho(a|s)$$



Same architecture
Weights ρ are initialized with σ

Reinforcement policy network

$$p_\rho(a|s)$$

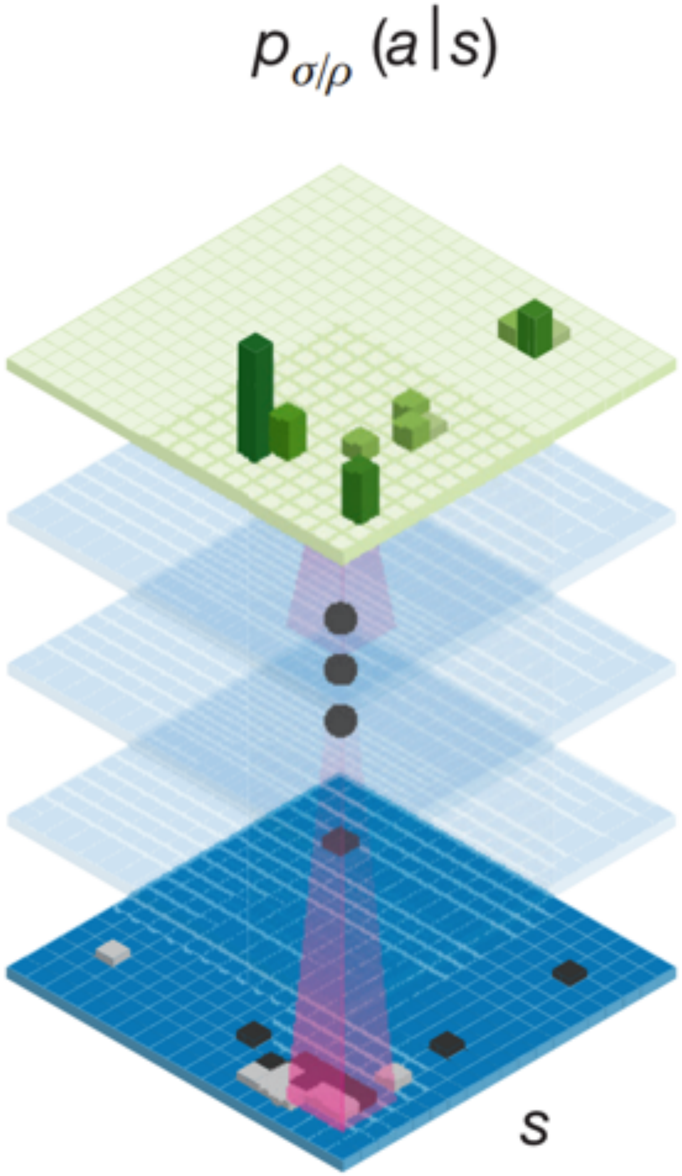


Same architecture
Weights ρ are initialized with σ

- Self-play: current network vs. randomized pool of previous versions

Reinforcement policy network

$p_\rho(a|s)$



Same architecture
Weights ρ are initialized with σ

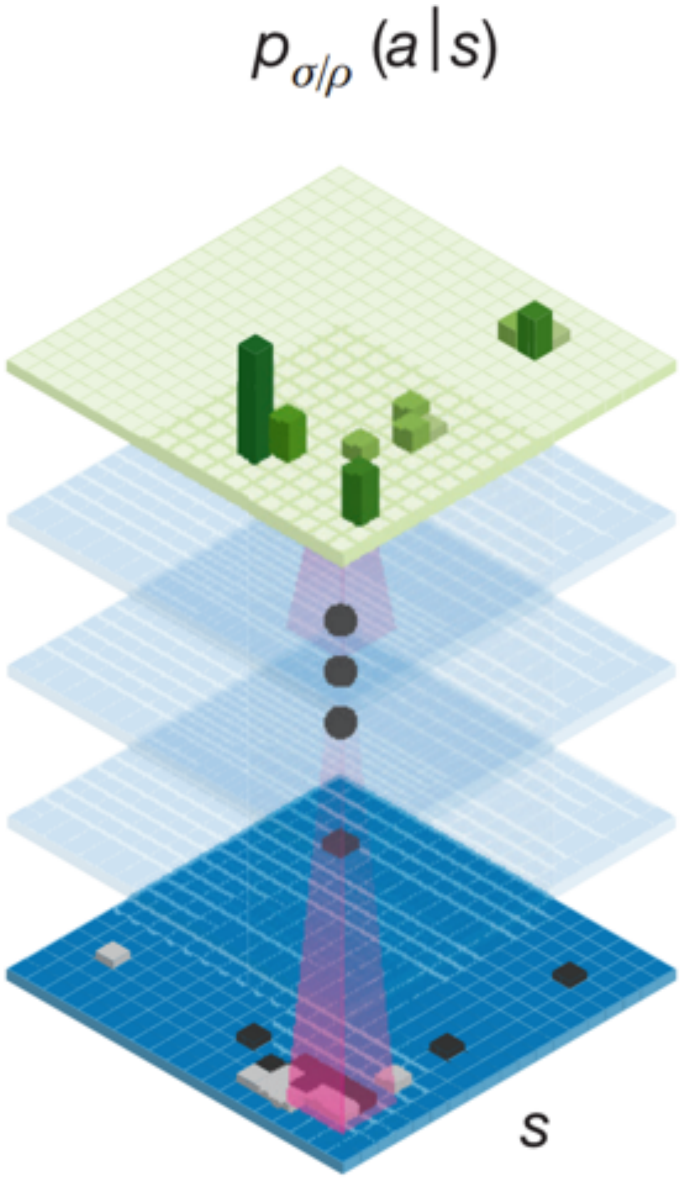
- Self-play: current network vs. randomized pool of previous versions

$$\Delta\rho = \frac{\alpha}{n} \sum_{i=1}^{n} \sum_{t=1}^{T^i} \frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial \rho} (z_t^i - v(s_t^i))$$

- Play a game until the end, get the reward $z_t = \pm r(s_T) = \pm 1$

Reinforcement policy network

$p_\rho(a|s)$



Same architecture
Weights ρ are initialized with σ

- Self-play: current network vs. randomized pool of previous versions

$$\Delta\rho = \frac{\alpha}{n} \sum_{i=1}^{n} \sum_{t=1}^{T^i} \frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial \rho}(z_t^i - v(s_t^i))$$

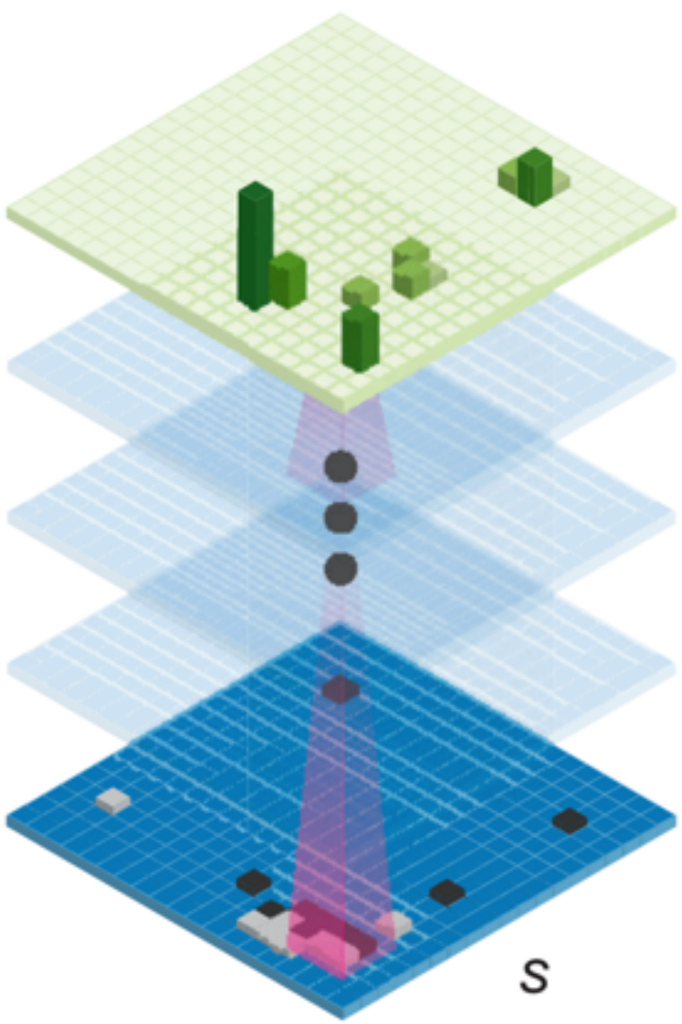
- Play a game until the end, get the reward $z_t = \pm r(s_T) = \pm 1$
- Set $z_t^i = z_t$ and play the same game again, this time updating the network parameters at each time step t

Reinforcement policy network

$p_\rho(a|s)$

- Self-play: current network vs. randomized pool of previous versions

$p_{\sigma/\rho}\,(a|s)$



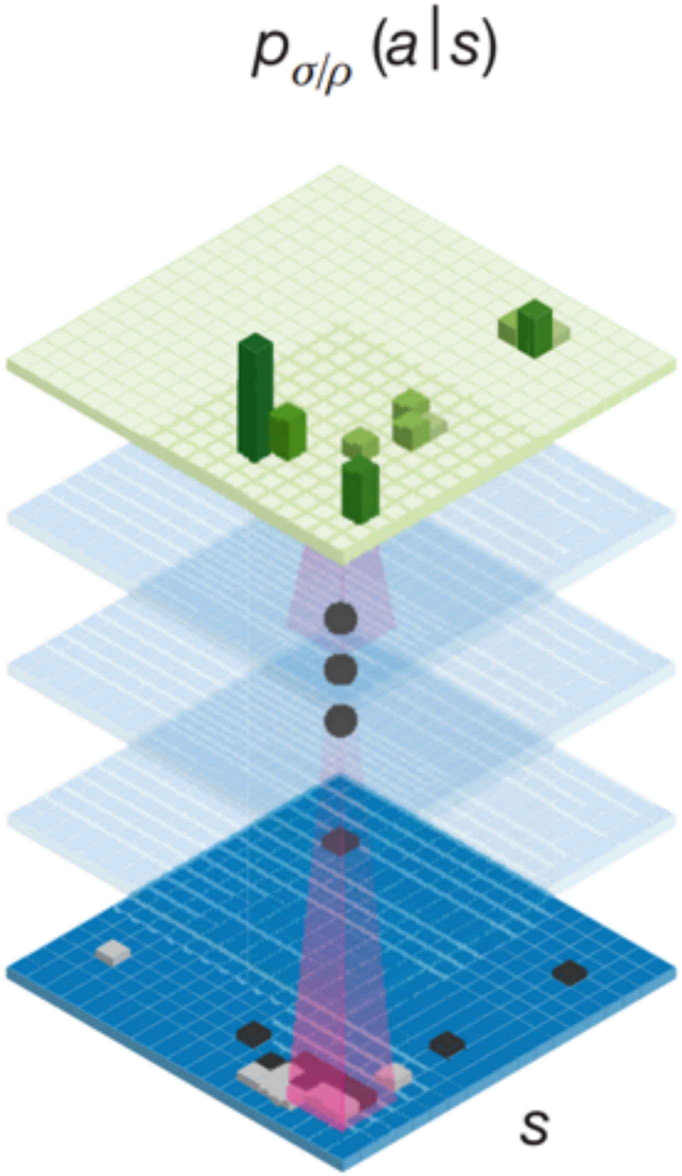
Same architecture
Weights ρ are initialized with σ

$$\Delta\rho = \frac{\alpha}{n}\sum_{i=1}^{n}\sum_{t=1}^{T^i}\frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial\rho}(z_t^i - v(s_t^i))$$

- Play a game until the end, get the reward $z_t = \pm r(s_T) = \pm 1$
- Set $z_t^i = z_t$ and play the same game again, this time updating the network parameters at each time step *t*
- $v(s_t^i)$ = …
  - 0 "*on the first pass through the training pipeline*"
  - $v_\theta(s_t^i)$ "*on the second pass*"

Reinforcement policy network

$p_\rho(a|s)$



Same architecture
Weights ρ are initialized with σ

- Self-play: current network vs. randomized pool of previous versions

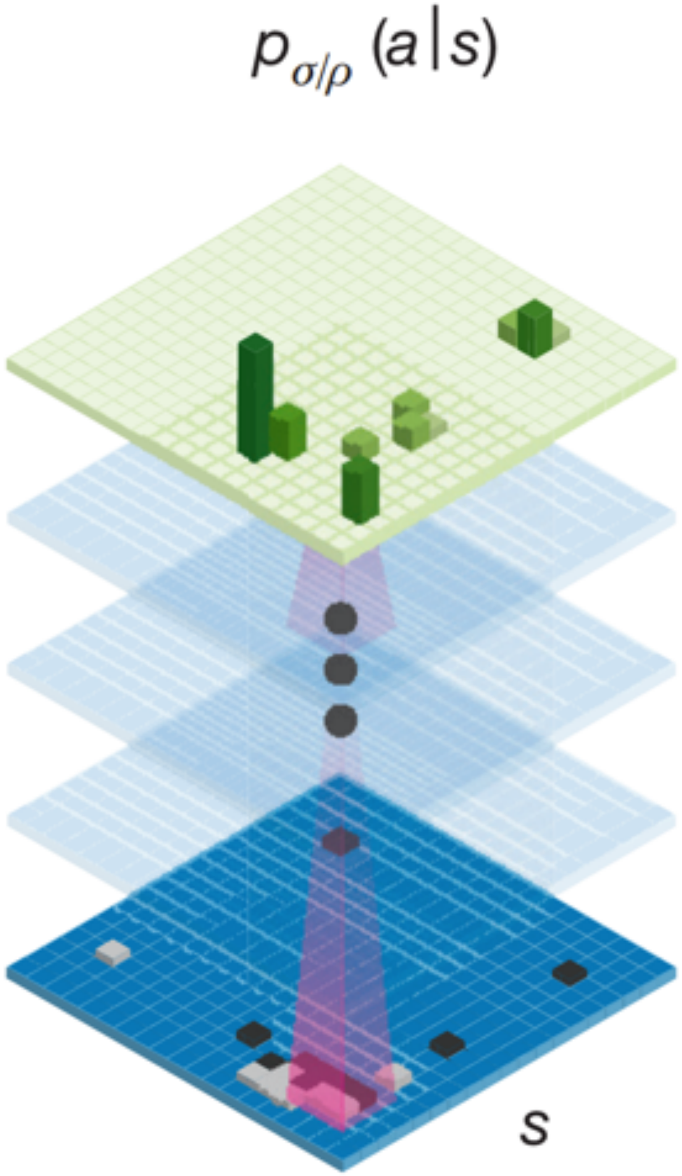
$$\Delta\rho = \frac{\alpha}{n} \sum_{i=1}^{n} \sum_{t=1}^{T^i} \frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial \rho} (z_t^i - v(s_t^i))$$

- Play a game until the end, get the reward $z_t = \pm r(s_T) = \pm 1$
- Set $z_t^i = z_t$ and play the same game again, this time updating the network parameters at each time step *t*
- $v(s_t^i)$ = …
  - 0 "*on the first pass through the training pipeline*"
  - $v_\theta(s_t^i)$ "*on the second pass*"
- batch size *n* = 128 games
- 10,000 batches
- One day on 50 GPUs

Reinforcement policy network

$p_\rho(a|s)$

- Self-play: current network vs. randomized pool of previous versions
- 80% wins against Supervised Network
- 85% wins against *Pachi* (no search yet!)
- 3 ms to select an action



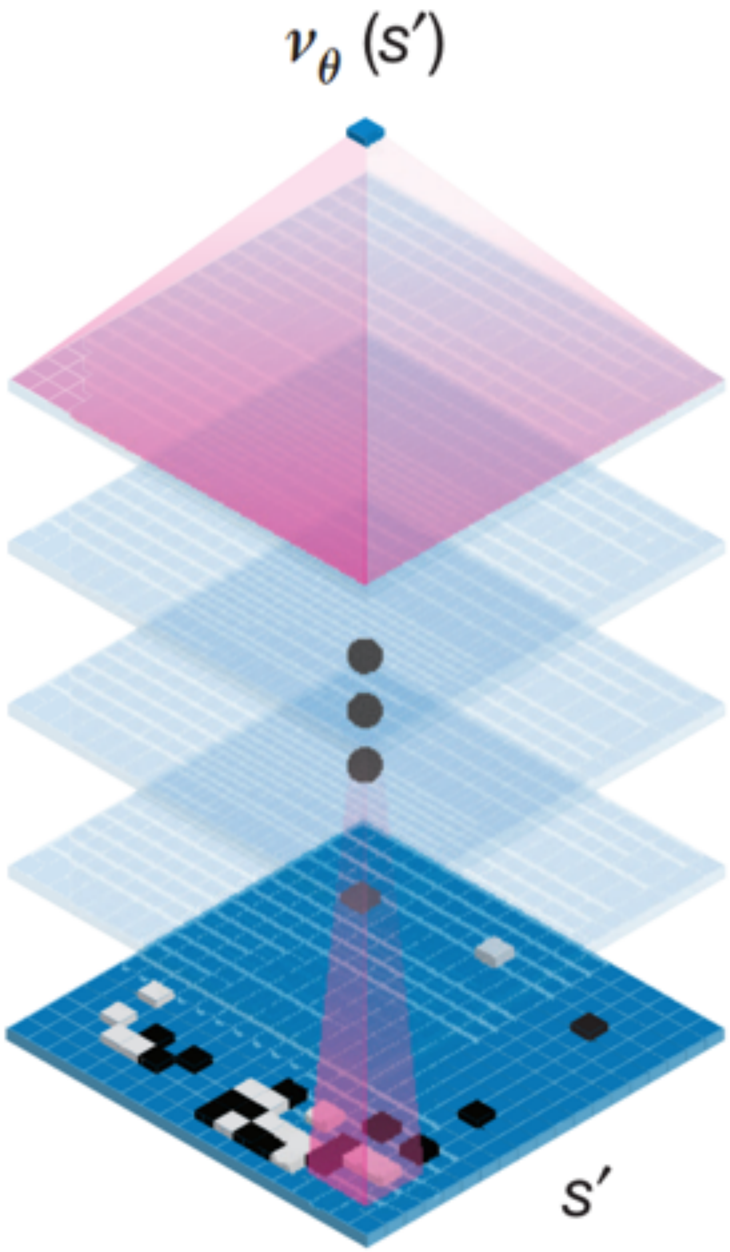
$$\Delta\rho = \frac{\alpha}{n}\sum_{i=1}^{n}\sum_{t=1}^{T^i}\frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial\rho}(z_t^i - v(s_t^i))$$

- Play a game until the end, get the reward $z_t = \pm r(s_T) = \pm 1$
- Set $z_t^i = z_t$ and play the same game again, this time updating the network parameters at each time step *t*
- $v(s_t^i)$ = …
  - 0 "*on the first pass through the training pipeline*"
  - $v_\theta(s_t^i)$ "*on the second pass*"
- batch size *n* = 128 games
- 10,000 batches
- One day on 50 GPUs

Same architecture
Weights ρ are initialized with σ

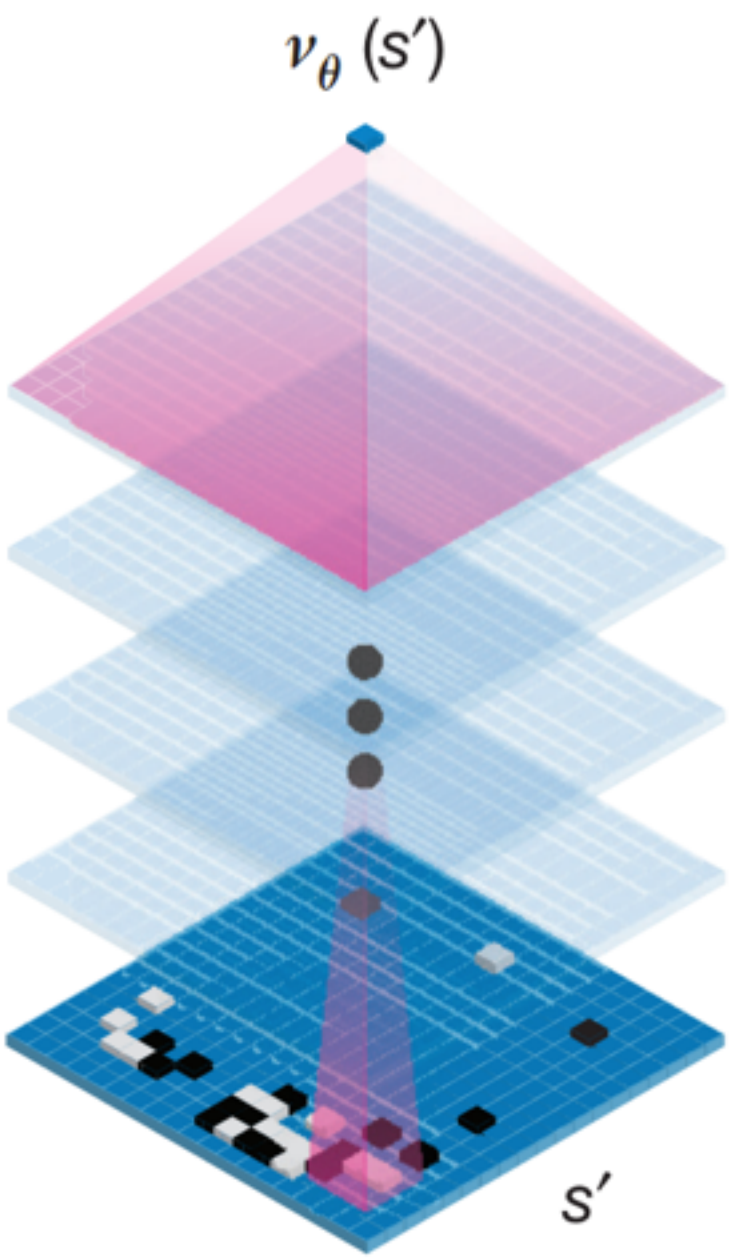
Value network

$v_\theta(s)$



Value network

$v_\theta(s)$



Fully connected layer
1 tanh unit

Fully connected layer
256 ReLU units

1 convolutional layer 1x1
ReLU

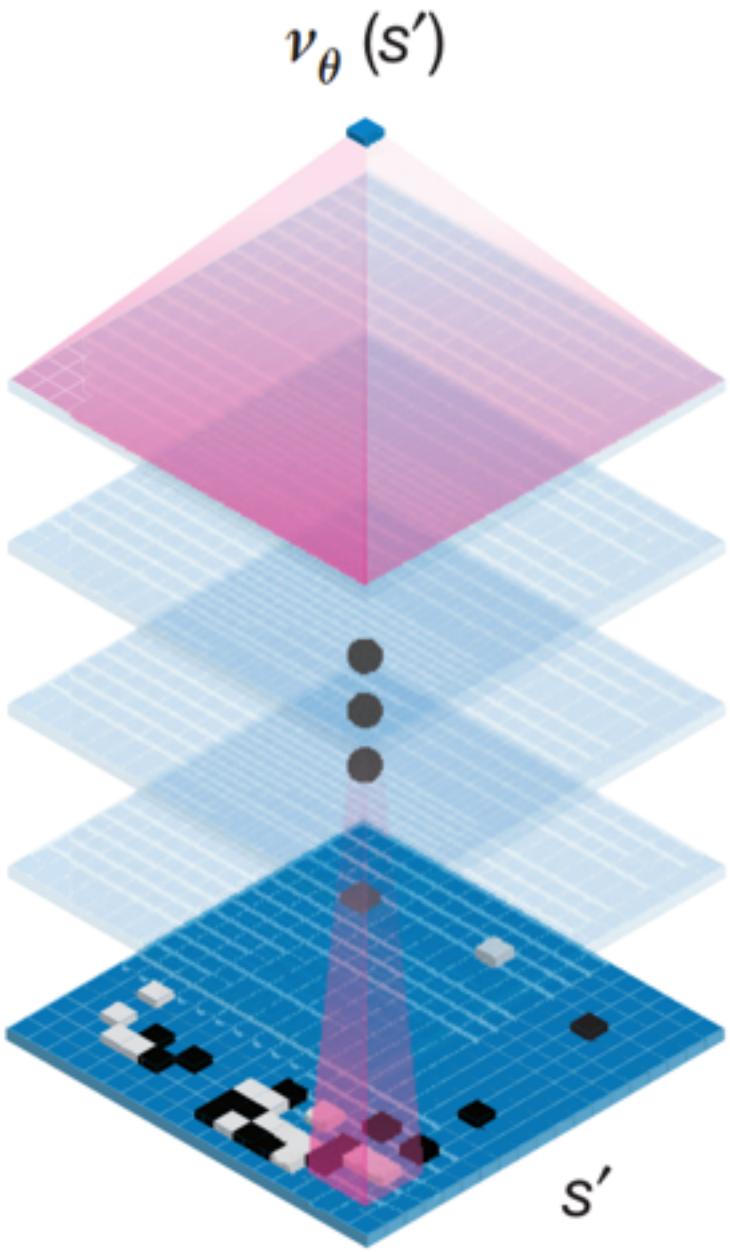
11 convolutional layers 3x3
with k=192 filters, ReLU

1 convolutional layer 5x5
with k=192 filters, ReLU

19 x 19 x 49 input

Value network $v_\theta(s)$

- Evaluate the value of the position *s* under policy *p:*
$$v^p(s) = \mathbb{E}[z_t | s_t = s, \ a_{t\ldots T} \sim p]$$
- Double approximation $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$



Fully connected layer
1 tanh unit

Fully connected layer
256 ReLU units

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

1 convolutional layer 5x5
with *k*=192 filters, ReLU

19 x 19 x 49 input

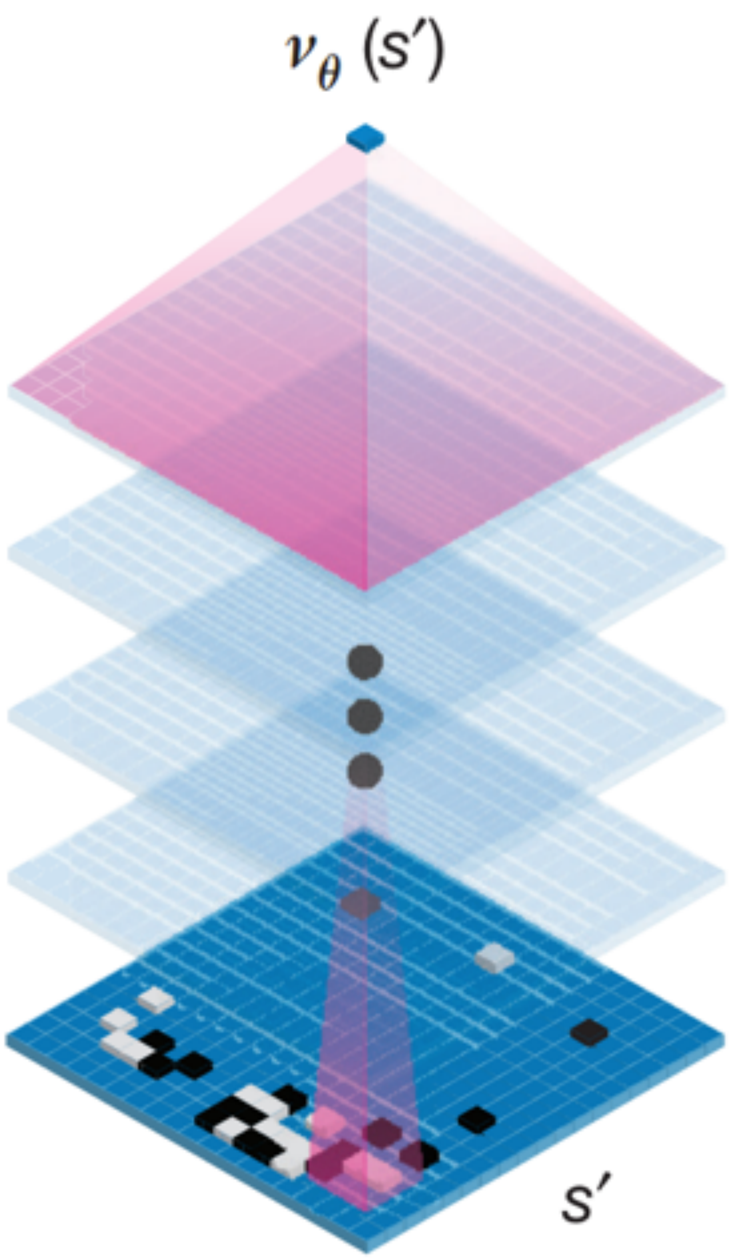
Value network $v_\theta(s)$

- Evaluate the value of the position *s* under policy *p:*

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t\ldots T} \sim p]$$

- Double approximation $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$

$$\Delta\theta = \frac{\alpha}{m} \sum_{k=1}^{m} (z^k - v_\theta(s^k)) \frac{\partial v_\theta(s^k)}{\partial \theta}$$



Fully connected layer
1 tanh unit

Fully connected layer
256 ReLU units

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

1 convolutional layer 5x5
with *k*=192 filters, ReLU

19 x 19 x 49 input

- Stochastic gradient descent to minimize MSE

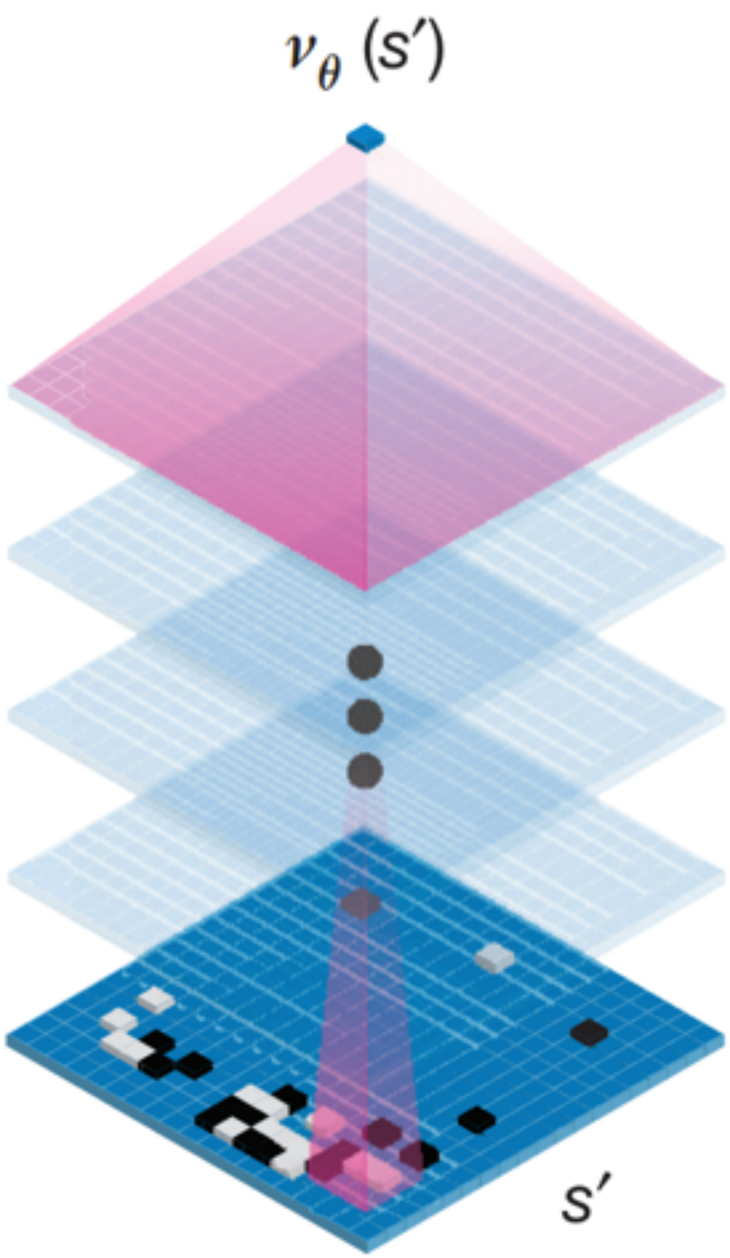
Value network $v_\theta(s)$

- Evaluate the value of the position *s* under policy *p:*

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \dots T} \sim p]$$

- Double approximation $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$

$$\Delta\theta = \frac{\alpha}{m} \sum_{k=1}^{m} (z^k - v_\theta(s^k)) \frac{\partial v_\theta(s^k)}{\partial \theta}$$



Fully connected layer
1 tanh unit

Fully connected layer
256 ReLU units

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

1 convolutional layer 5x5
with *k*=192 filters, ReLU

19 x 19 x 49 input

- Stochastic gradient descent to minimize MSE
- Train on 30M state-outcome pairs (s, z), each from a unique game generated by self-play:

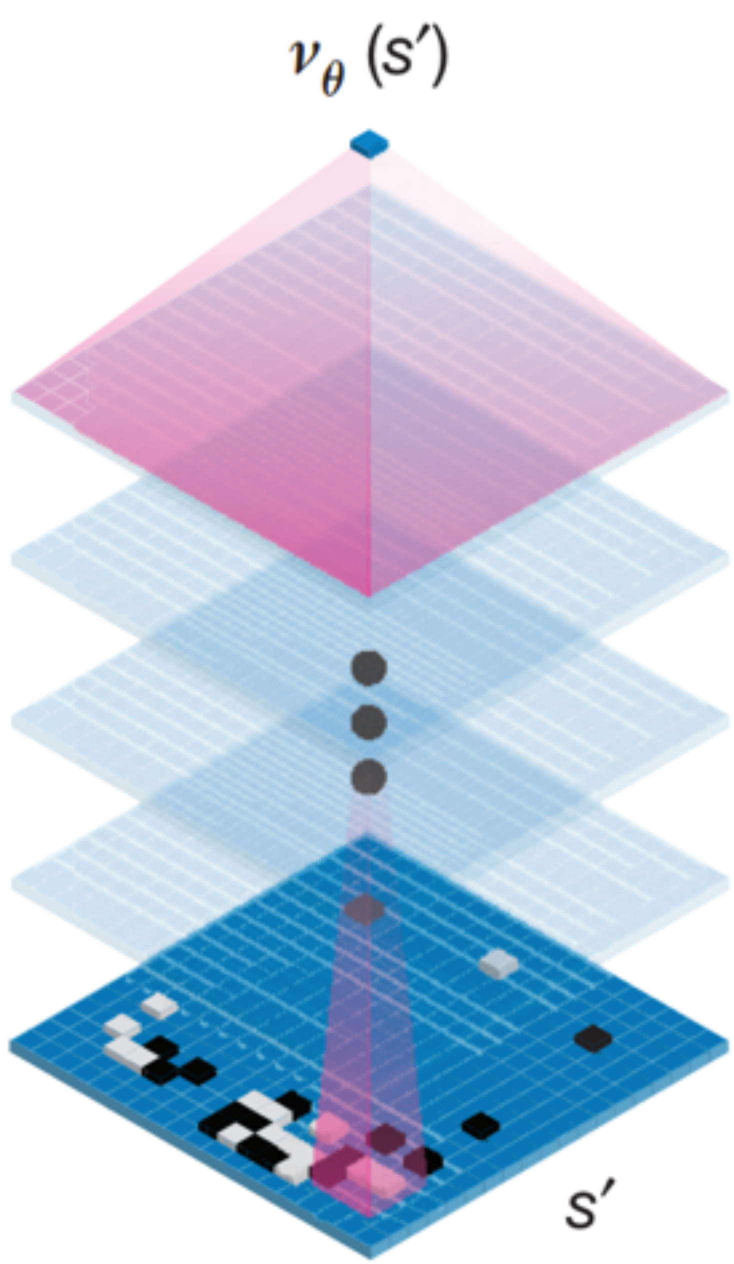
Value network $v_\theta(s)$

- Evaluate the value of the position *s* under policy *p*:

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \dots T} \sim p]$$

- Double approximation $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$

$$\Delta\theta = \frac{\alpha}{m} \sum_{k=1}^{m} (z^k - v_\theta(s^k)) \frac{\partial v_\theta(s^k)}{\partial \theta}$$



Fully connected layer
1 tanh unit

Fully connected layer
256 ReLU units

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with k=192 filters, ReLU

1 convolutional layer 5x5
with k=192 filters, ReLU

19 x 19 x 49 input

- Stochastic gradient descent to minimize MSE
- Train on 30M state-outcome pairs (s, z), each from a unique game generated by self-play:
  - choose a random time step u
  - sample moves t=1…u-1 from SL policy
  - make random move u
  - sample t=u+1…T from RL policy and get game outcome z
  - add (s_u, z_u) pair to the training set

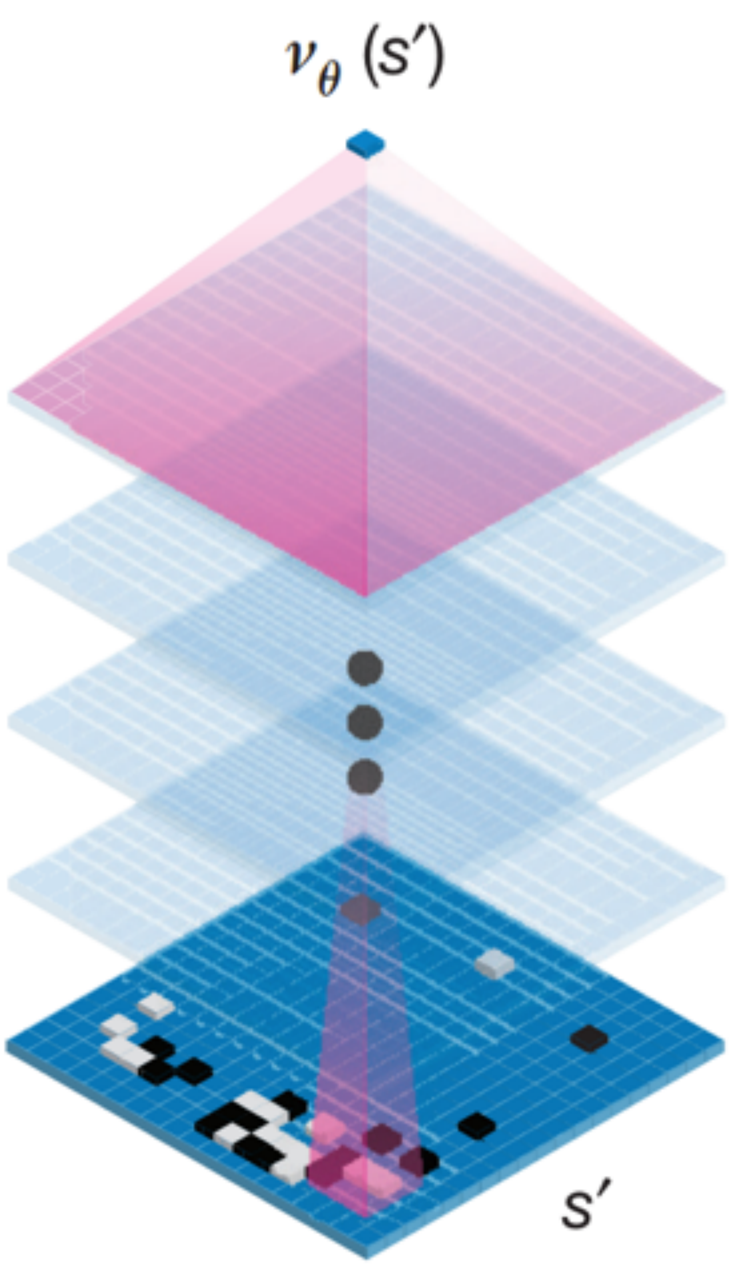
Value network $v_\theta(s)$

- Evaluate the value of the position *s* under policy *p*:

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t\dots T} \sim p]$$

- Double approximation $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$

$$\Delta\theta = \frac{\alpha}{m} \sum_{k=1}^{m} (z^k - v_\theta(s^k)) \frac{\partial v_\theta(s^k)}{\partial \theta}$$



Fully connected layer
1 tanh unit

Fully connected layer
256 ReLU units

1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

1 convolutional layer 5x5
with *k*=192 filters, ReLU

19 x 19 x 49 input

- Stochastic gradient descent to minimize MSE
- Train on 30M state-outcome pairs (s, z), each from a unique game generated by self-play:
  - choose a random time step *u*
  - sample moves *t*=1…*u*-1 from SL policy
  - make random move *u*
  - sample *t*=*u*+1…T from RL policy and get game outcome *z*
  - add (s_u, z_u) pair to the training set
- One week on 50 GPUs to train on 50M batches of size *m*=32

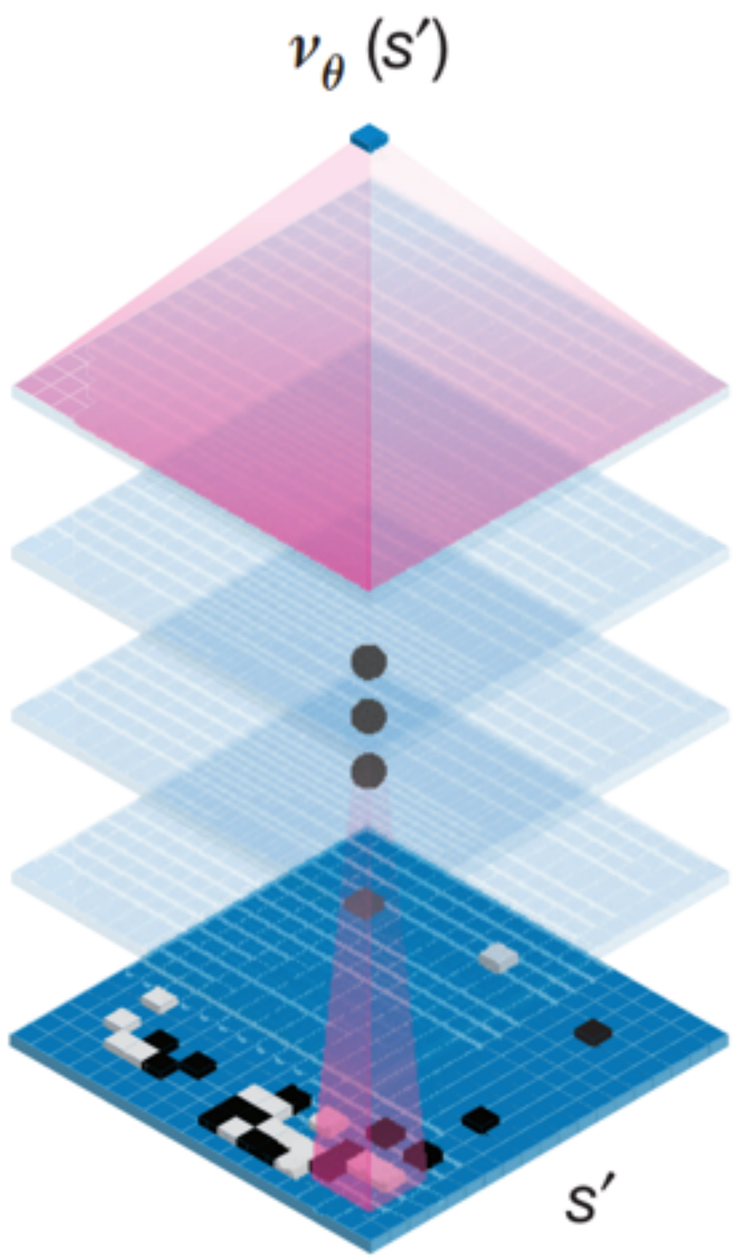
Value network $v_\theta(s)$

- Evaluate the value of the position *s* under policy *p:*

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t\ldots T} \sim p]$$

- Double approximation $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$
- MSE on the test set: 0.234
- Close to MC estimation from RL policy; 15,000 faster

$$\Delta\theta = \frac{\alpha}{m} \sum_{k=1}^{m} (z^k - v_\theta(s^k)) \frac{\partial v_\theta(s^k)}{\partial \theta}$$



Fully connected layer
1 tanh unit

Fully connected layer
256 ReLU units

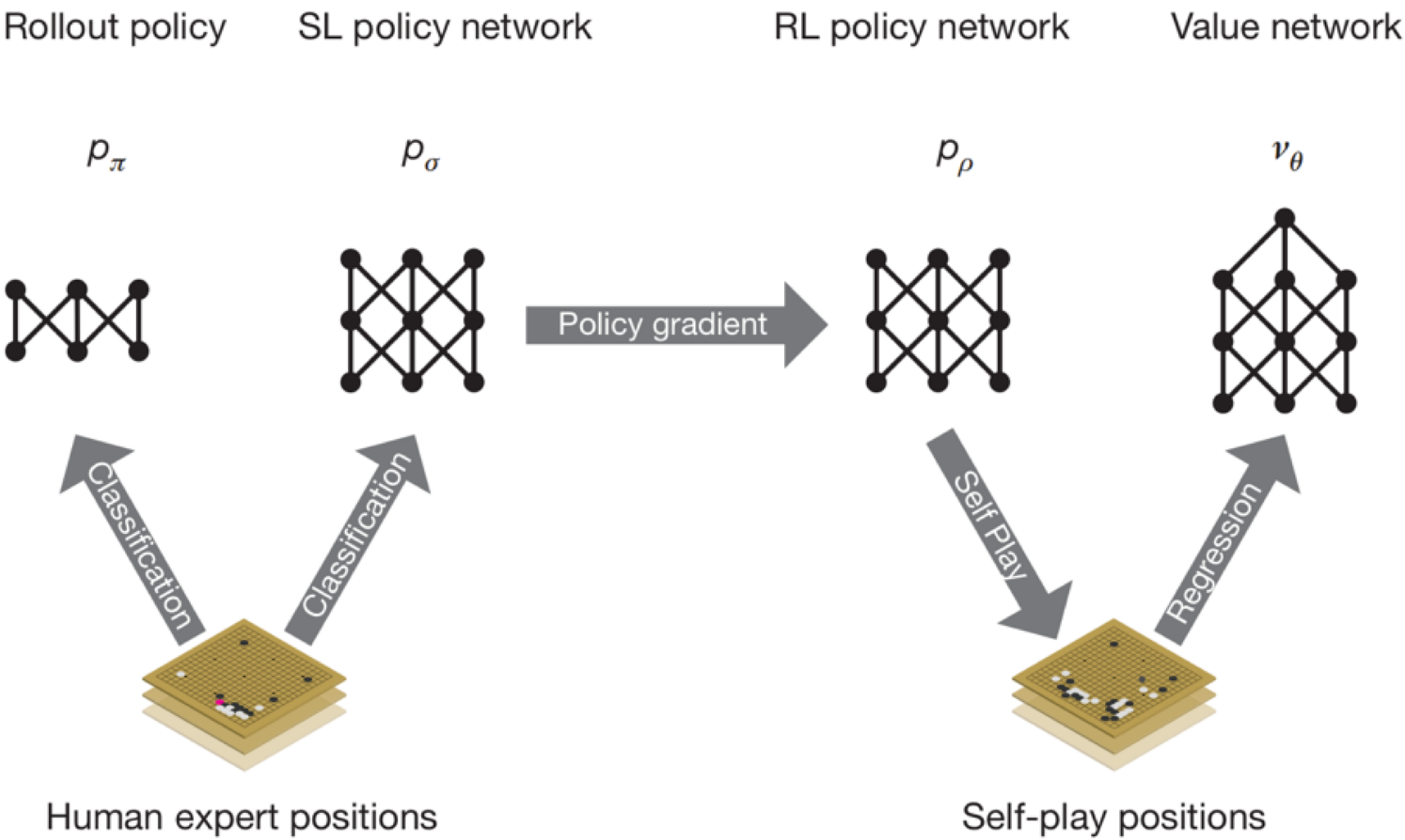
1 convolutional layer 1x1
ReLU

11 convolutional layers 3x3
with *k*=192 filters, ReLU

1 convolutional layer 5x5
with *k*=192 filters, ReLU

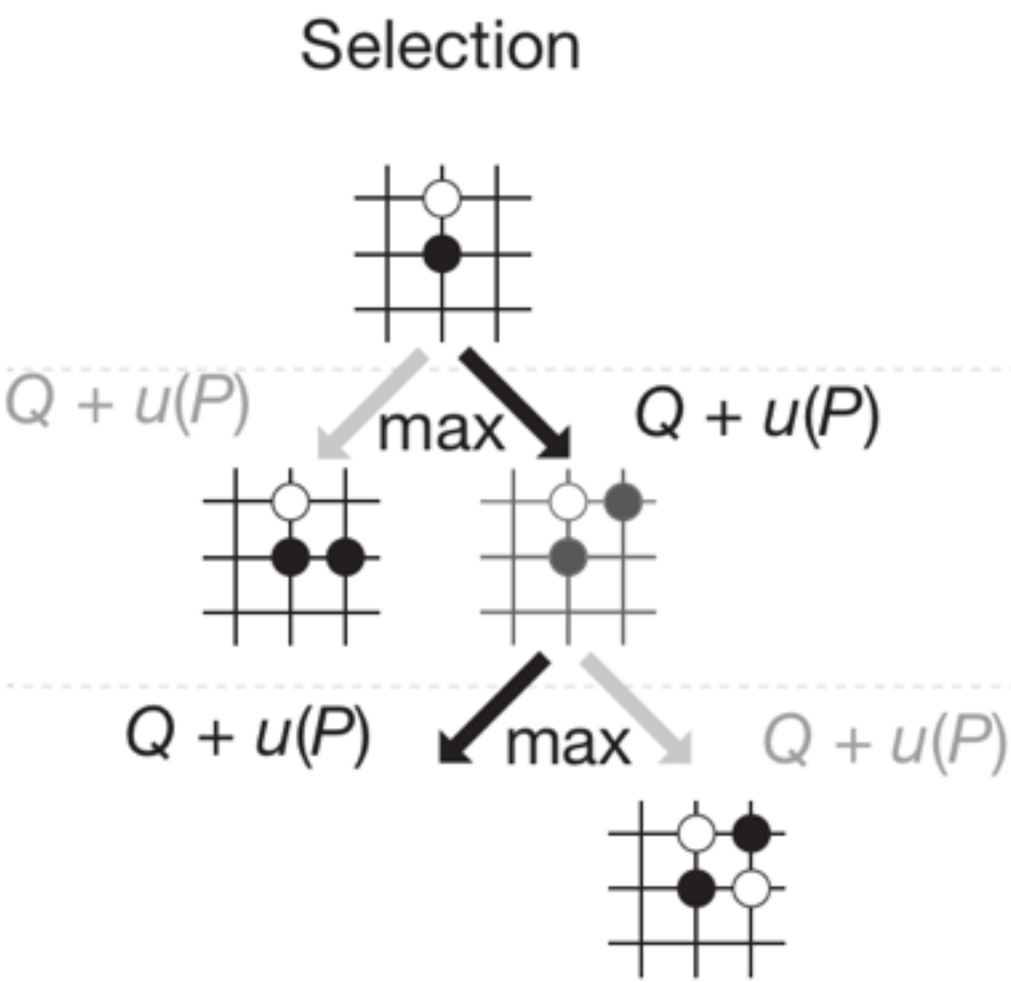
19 x 19 x 49 input

- Stochastic gradient descent to minimize MSE
- Train on 30M state-outcome pairs (s, z), each from a unique game generated by self-play:
  - choose a random time step *u*
  - sample moves *t*=1…*u*-1 from SL policy
  - make random move *u*
  - sample *t*=*u*+1…T from RL policy and get game outcome *z*
  - add (s_u, z_u) pair to the training set
- One week on 50 GPUs to train on 50M batches of size *m*=32

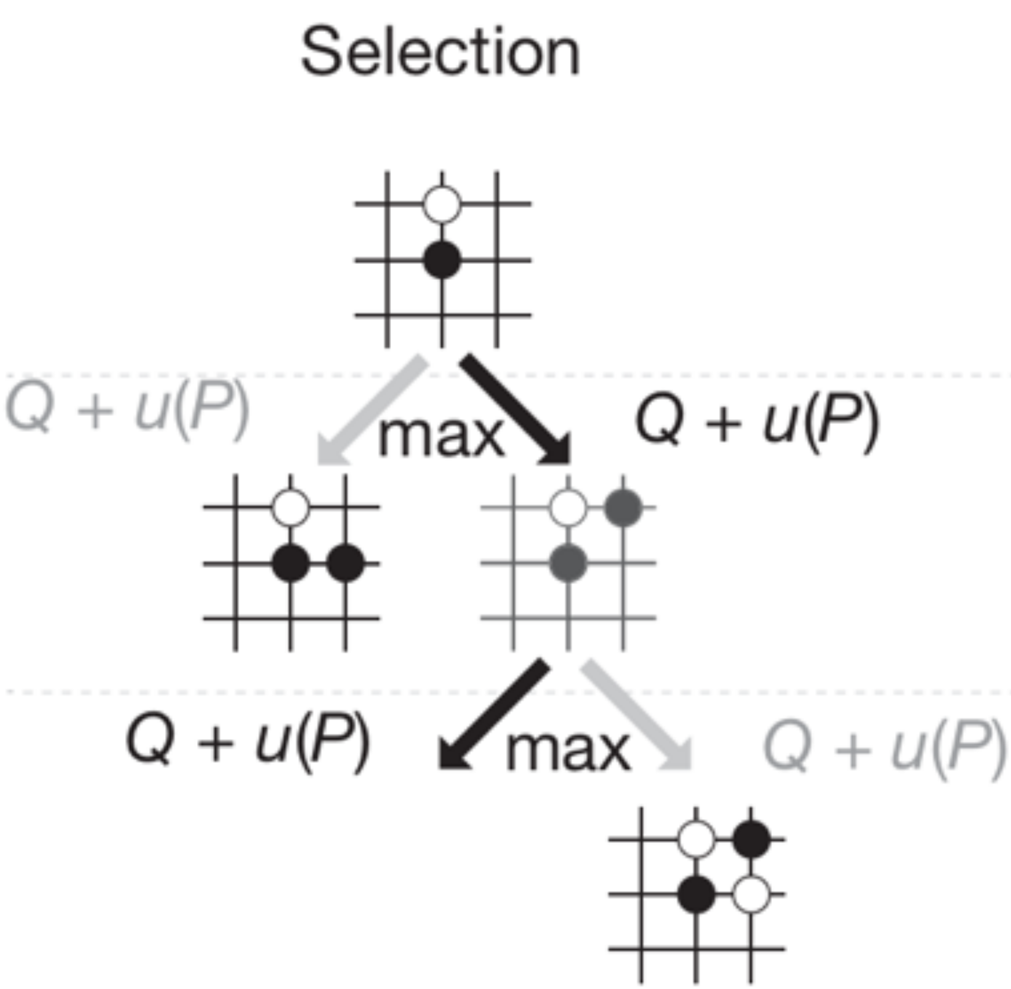

Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Human expert positions
Self-play positions

AlphaGo
Playing

APV-MCTS Asynchronous Policy and Value MCTS



APV-MCTS Asynchronous Policy and Value MCTS

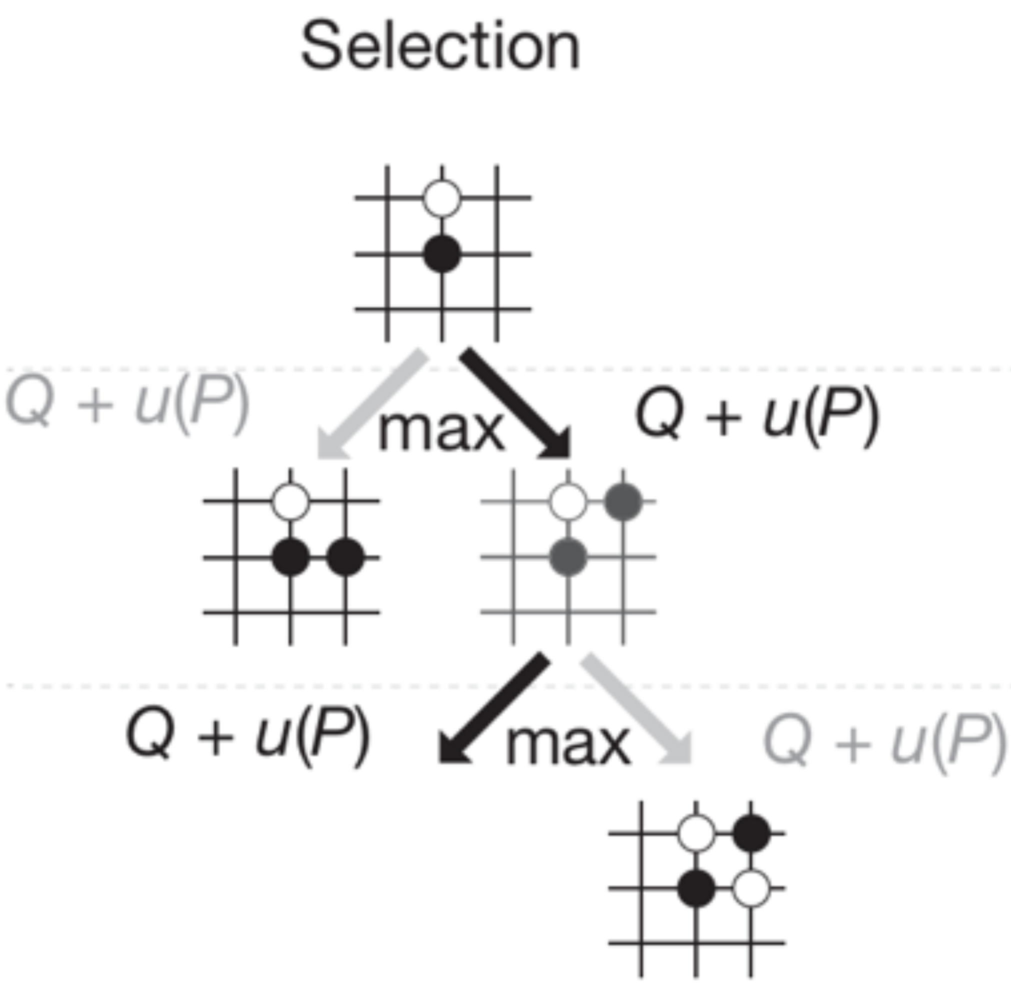


Each node *s* has edges (*s*, *a*) for all legal actions and stores statistics:

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

$P(s,a)$	$N_v(s,a)$	$N_r(s,a)$	$W_v(s,a)$	$W_r(s,a)$	$Q(s,a)$
Prior	Number of evaluations	Number of rollouts	MC value estimate	Rollout value estimate	Combined mean action value

APV-MCTS ASYNCHRONOUS POLICY AND VALUE MCTS



Each node *s* has edges (*s*, *a*) for all legal actions and stores statistics:

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

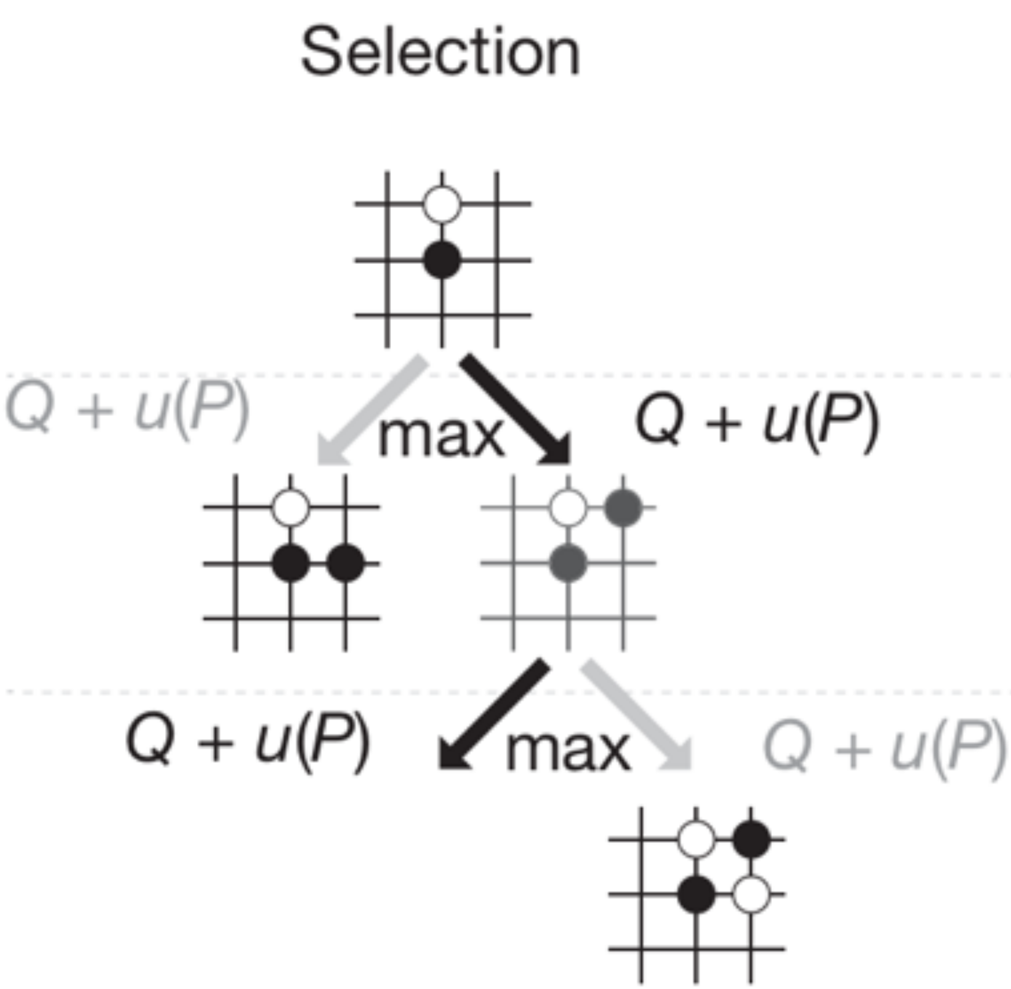
$P(s,a)$	$N_v(s,a)$	$N_r(s,a)$	$W_v(s,a)$	$W_r(s,a)$	$Q(s,a)$
Prior	Number of evaluations	Number of rollouts	MC value estimate	Rollout value estimate	Combined mean action value

Simulation starts at the root and stops at time *L*, when a leaf (unexplored state) is found.

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

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

APV-MCTS Asynchronous Policy and Value MCTS



Each node *s* has edges (*s*, *a*) for all legal actions and stores statistics:

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

Prior	Number of evaluations	Number of rollouts	MC value estimate	Rollout value estimate	Combined mean action value

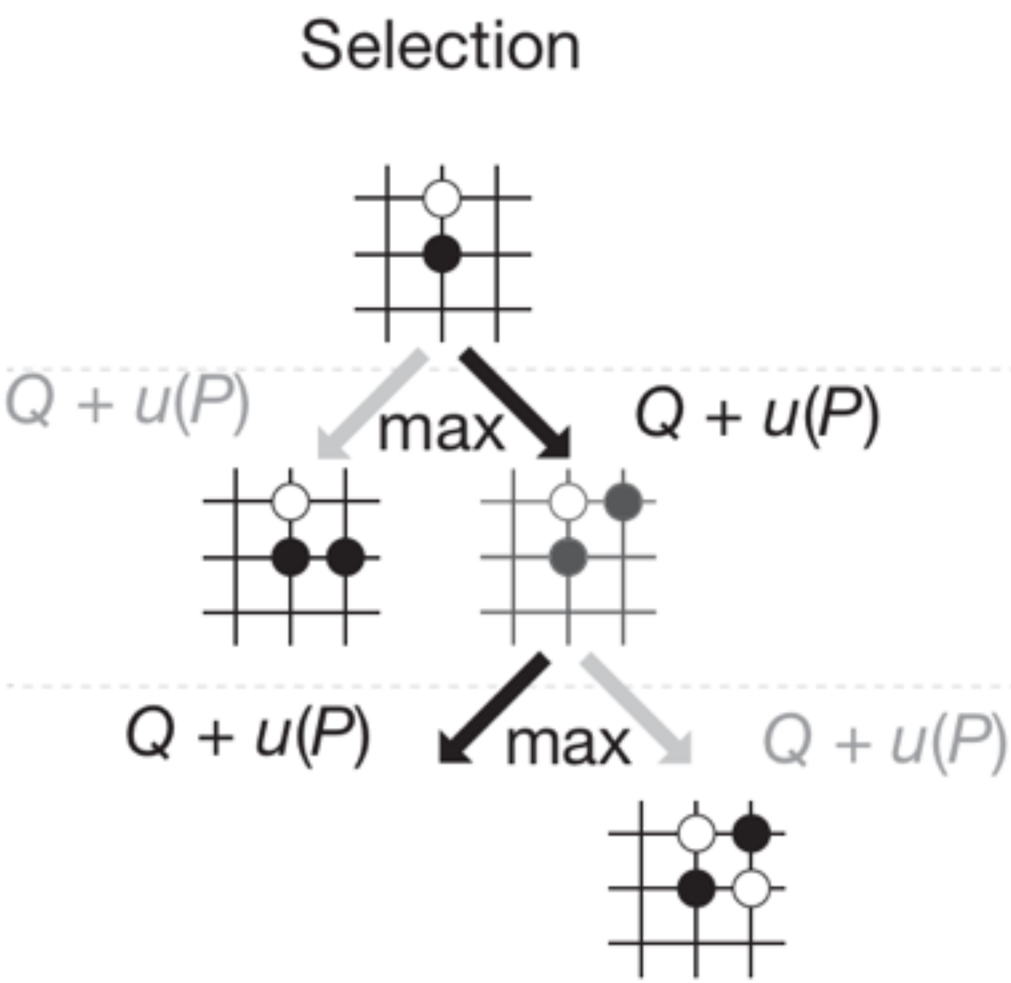
Simulation starts at the root and stops at time *L*, when a leaf (unexplored state) is found.

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

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

Position s_L is added to evaluation queue.

APV-MCTS Asynchronous Policy and Value MCTS



Each node s has edges (s, a) for all legal actions and stores statistics:

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

$P(s,a)$	$N_v(s,a)$	$N_r(s,a)$	$W_v(s,a)$	$W_r(s,a)$	$Q(s,a)$
Prior	Number of evaluations	Number of rollouts	MC value estimate	Rollout value estimate	Combined mean action value

Simulation starts at the root and stops at time L, when a leaf (unexplored state) is found.

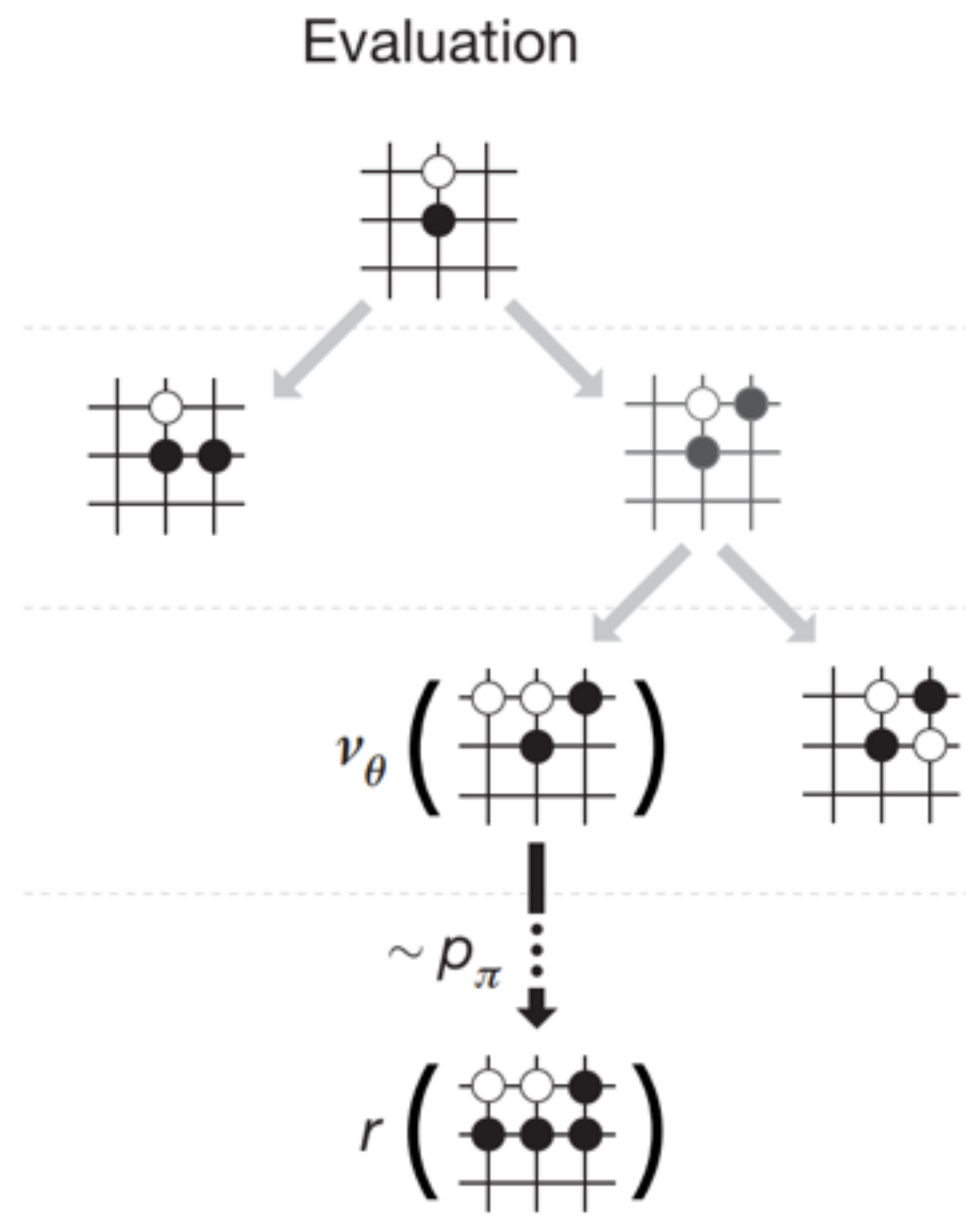
$$a_t = \operatorname{argmax}_a \left(Q(s_t, a) + u(s_t, a)\right)$$

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

Position s_L is added to evaluation queue.

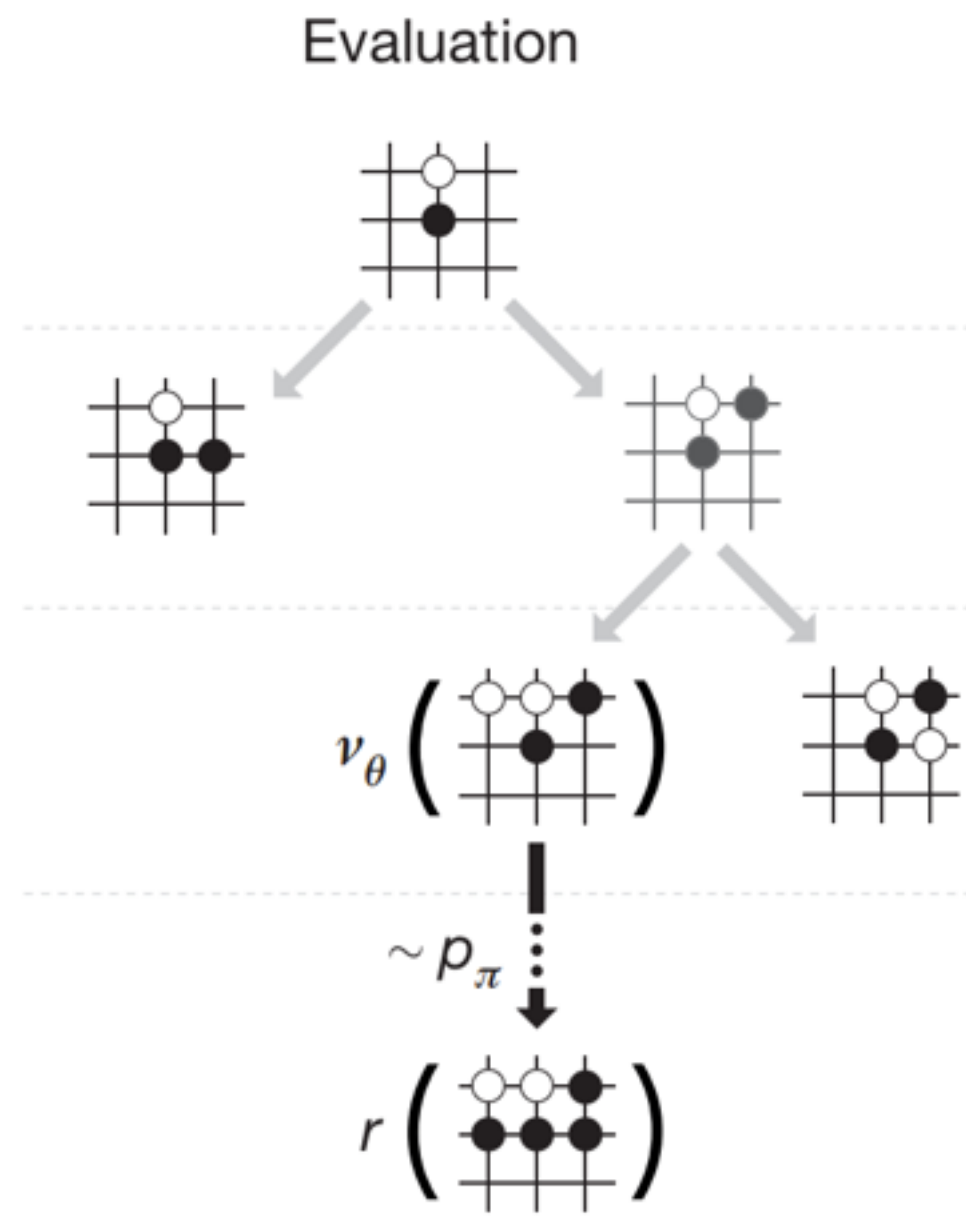
Bunch of nodes selected for evaluation…

APV-MCTS Asynchronous Policy and Value MCTS

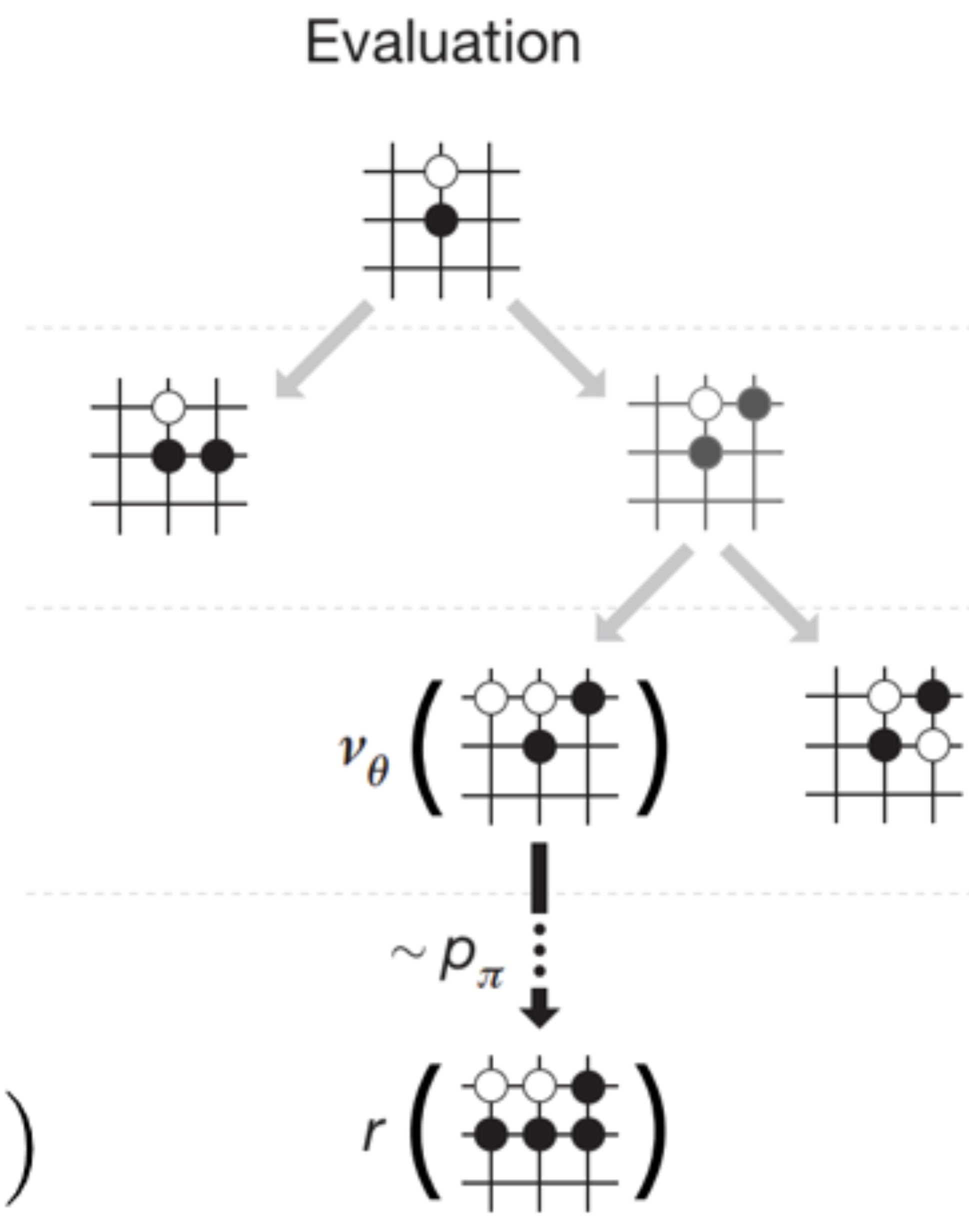


APV-MCTS Asynchronous Policy and Value MCTS

Node s is evaluated using the value network to obtain $v_\theta(s)$



APV-MCTS Asynchronous Policy and Value MCTS



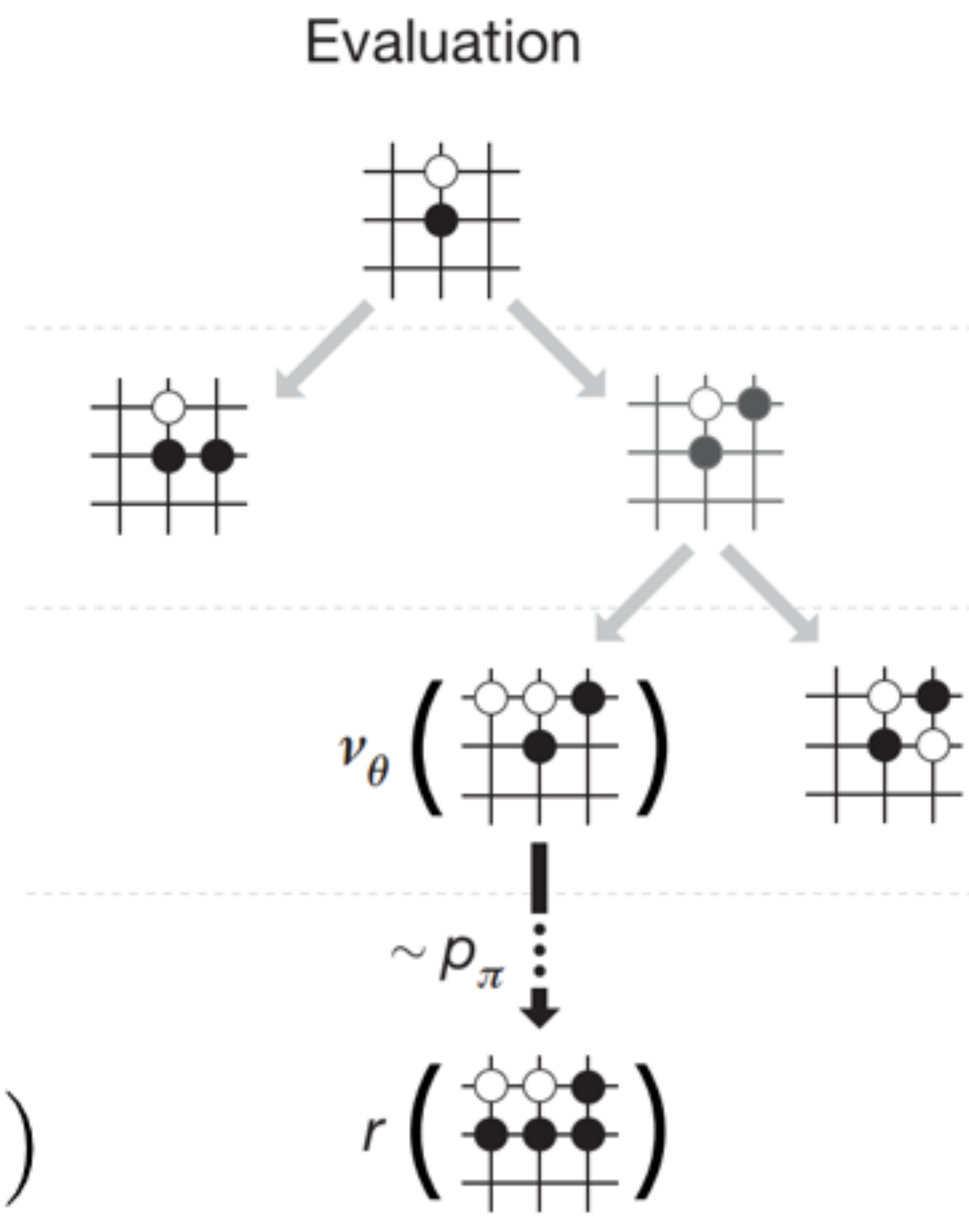
Node *s* is evaluated using the value network to obtain

$$v_\theta(s)$$

and using rollout simulation with policy p_π till the end of each simulated game to get the final game score.

$$z_t = \pm r(s_T)$$

APV-MCTS ASYNCHRONOUS POLICY AND VALUE MCTS



Node s is evaluated using the value network to obtain

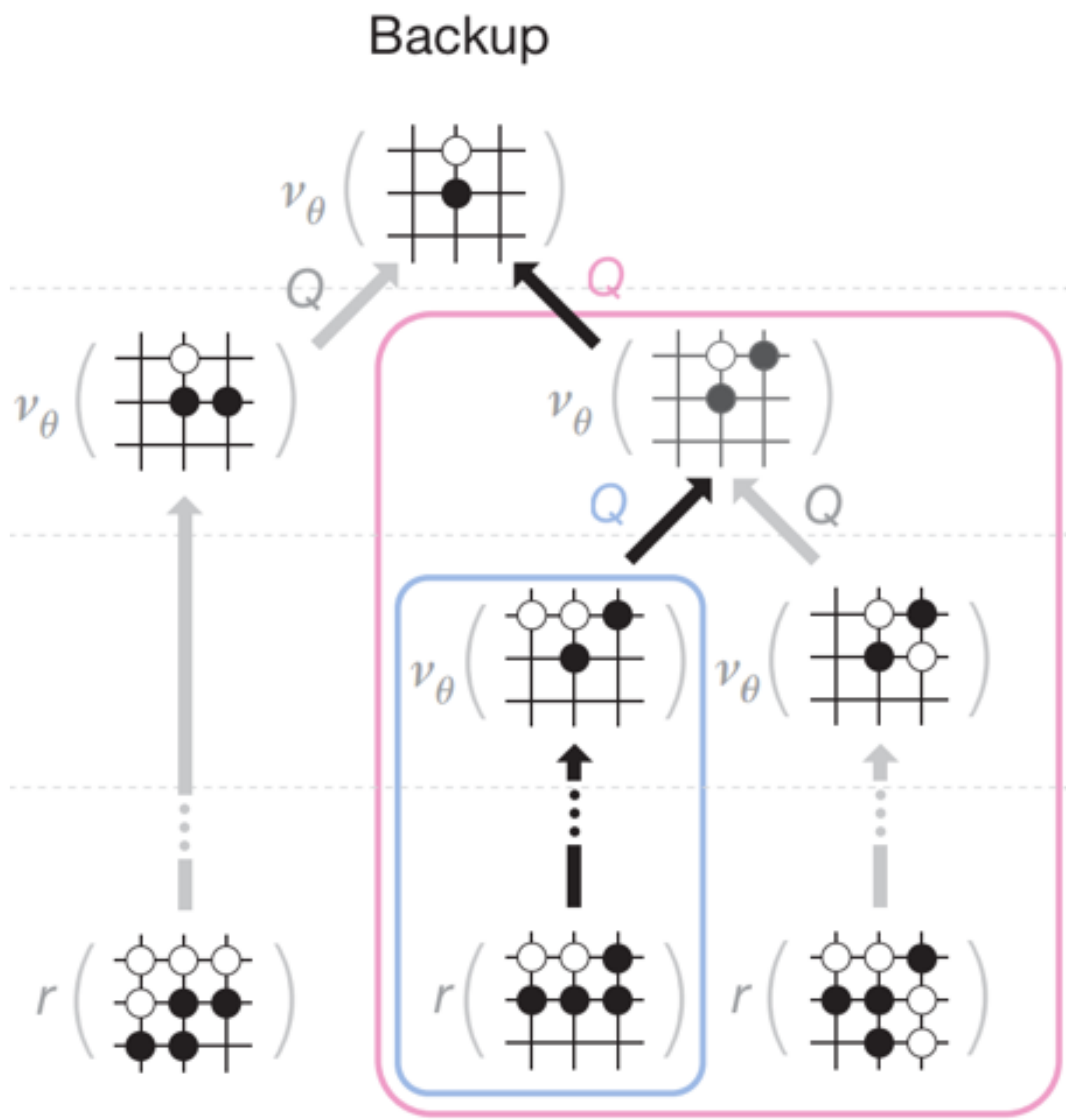
$$v_\theta(s)$$

and using rollout simulation with policy p_π till the end of each simulated game to get the final game score.

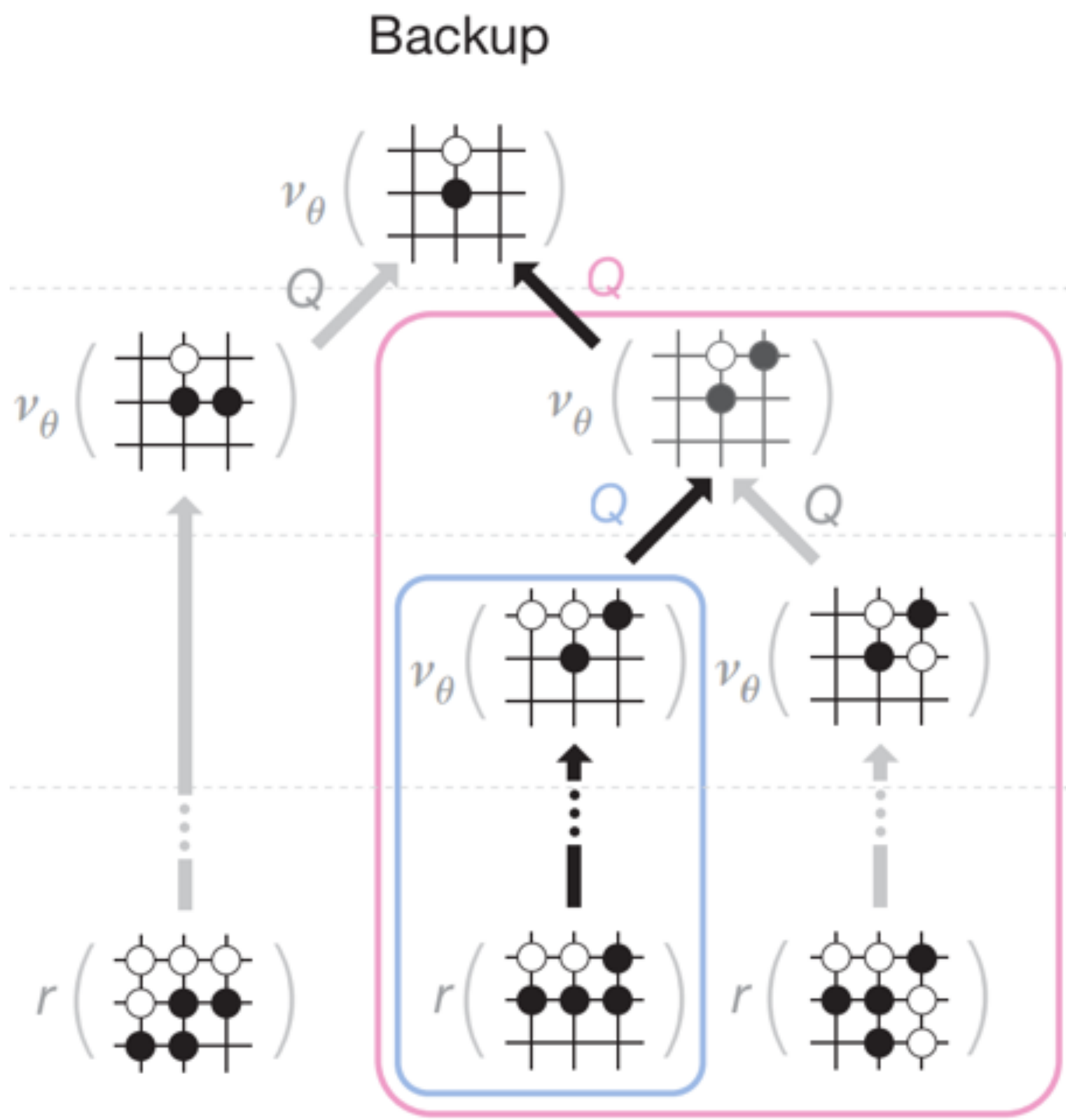
$$z_t = \pm r(s_T)$$

Each leaf is evaluated, we are ready to propagate updates

APV-MCTS Asynchronous Policy and Value MCTS



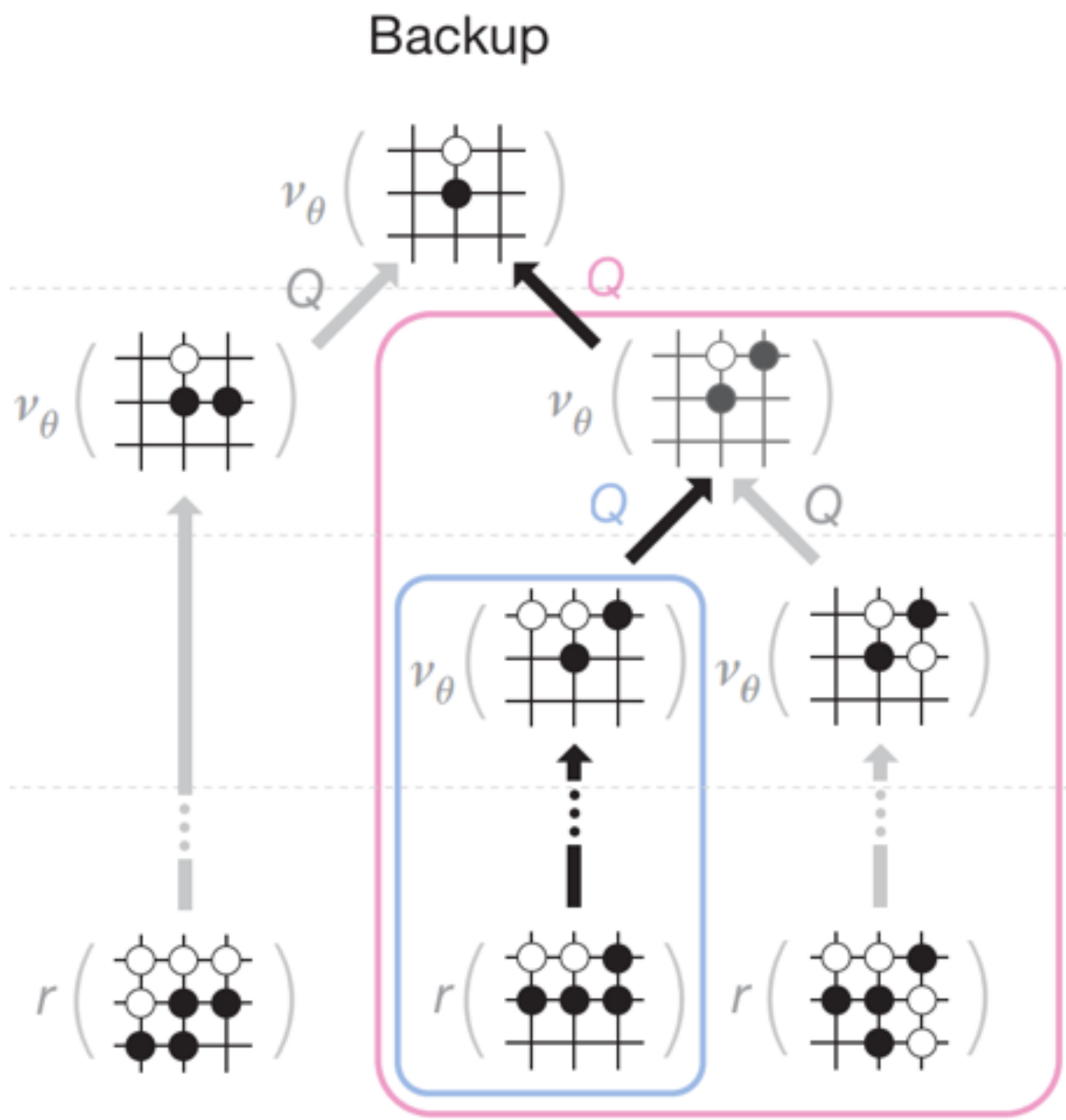
APV-MCTS Asynchronous Policy and Value MCTS



Statistics along the paths of each simulation are updated during the backward pass though $t < L$

$$W_v(s_t, a_t) \leftarrow W_v(s_t, a_t) + v_\theta(s_L)$$
$$W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) + z_t$$

APV-MCTS Asynchronous Policy and Value MCTS



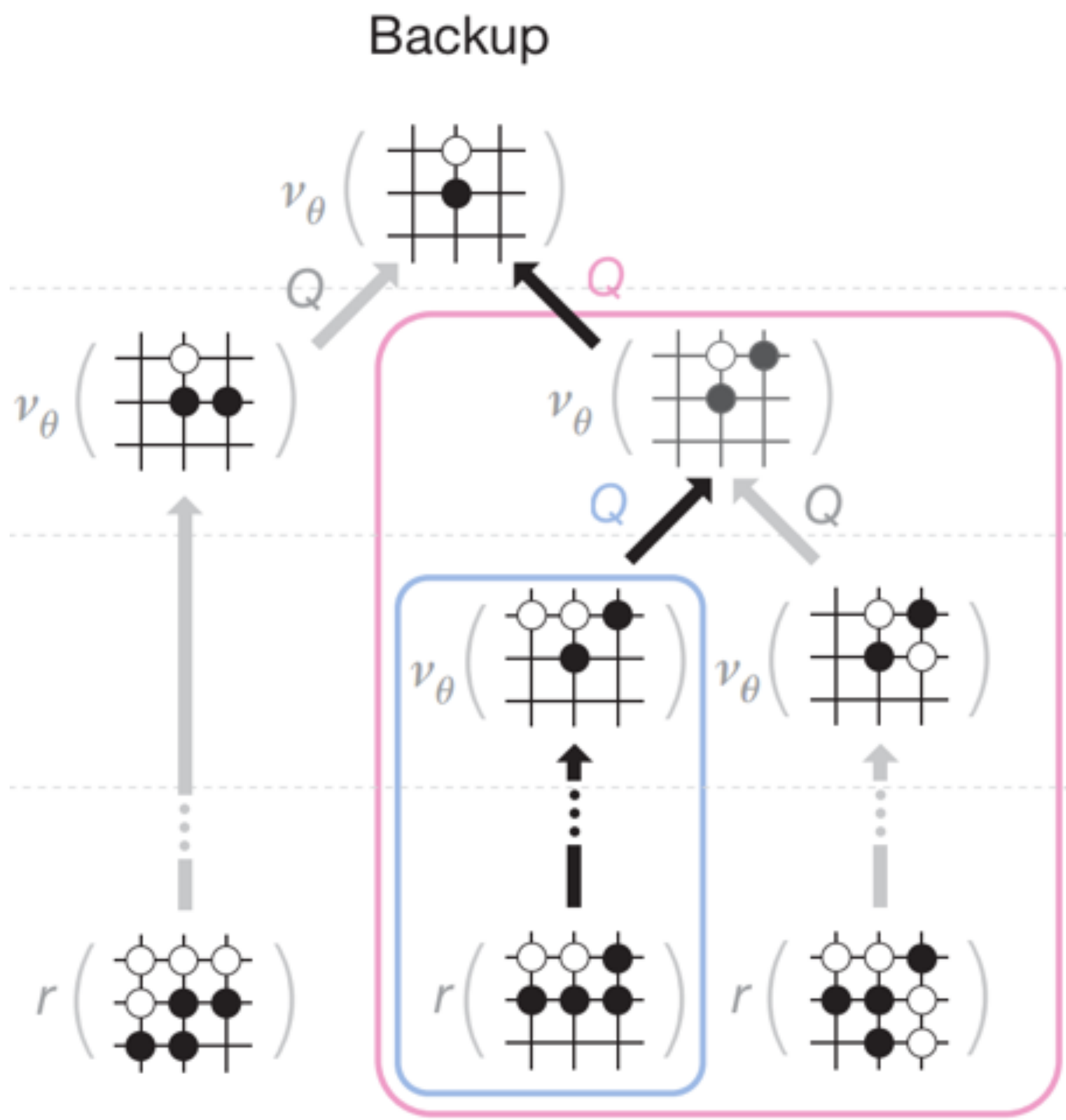
Statistics along the paths of each simulation are updated during the backward pass though $t < L$

$$W_v(s_t, a_t) \leftarrow W_v(s_t, a_t) + v_\theta(s_L)$$
$$W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) + z_t$$

visits counts are updated as well

$$N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) + 1$$
$$N_v(s_t, a_t) \leftarrow N_v(s_t, a_t) + 1$$

APV-MCTS Asynchronous Policy and Value MCTS



Statistics along the paths of each simulation are updated during the backward pass though $t < L$

$$W_v(s_t, a_t) \leftarrow W_v(s_t, a_t) + v_\theta(s_L)$$
$$W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) + z_t$$

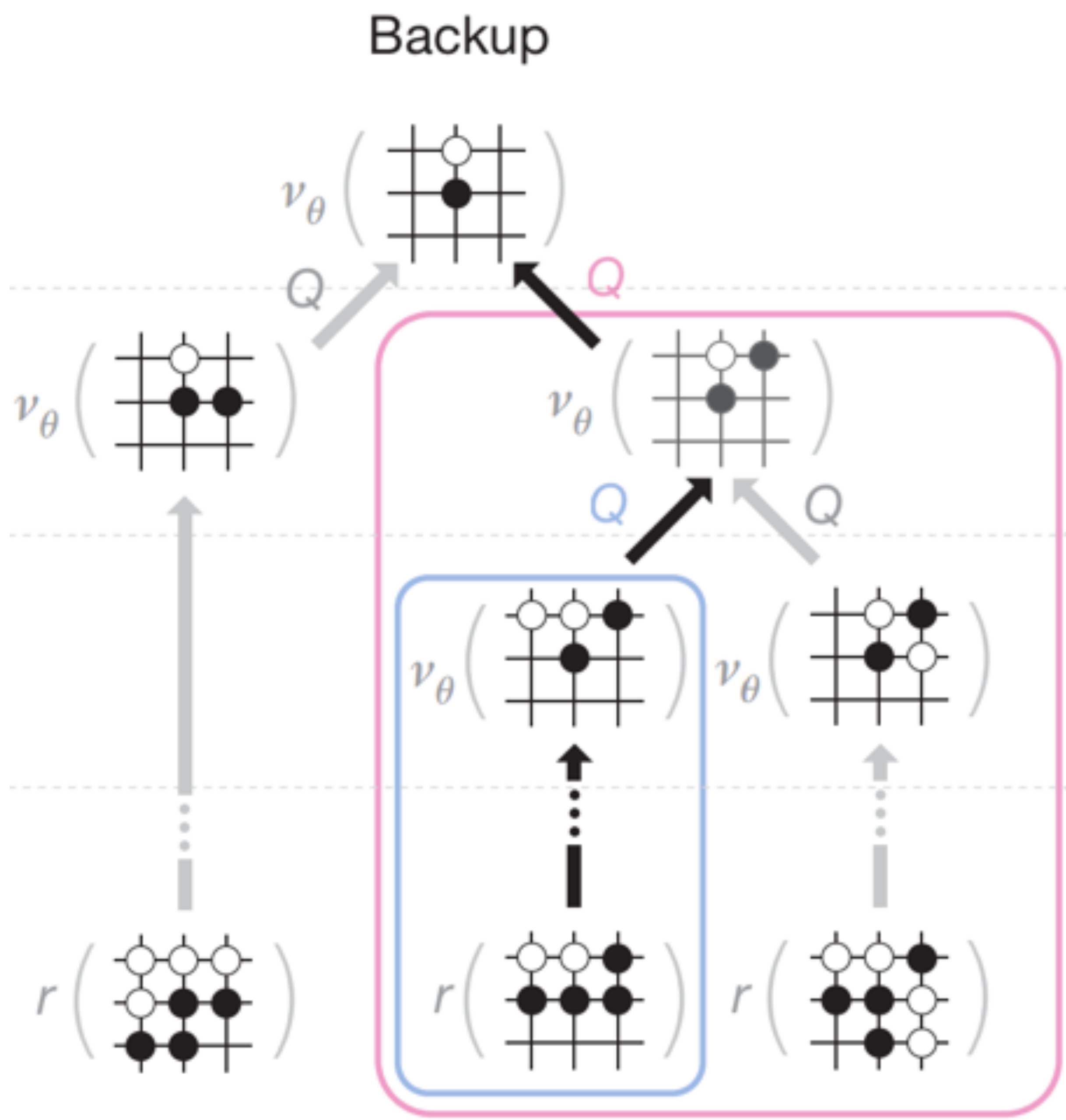
visits counts are updated as well

$$N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) + 1$$
$$N_v(s_t, a_t) \leftarrow N_v(s_t, a_t) + 1$$

Finally overall evaluation of each visited state-action edge is updated

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

APV-MCTS Asynchronous Policy and Value MCTS



Statistics along the paths of each simulation are updated during the backward pass though $t < L$

$$W_v(s_t, a_t) \leftarrow W_v(s_t, a_t) + v_\theta(s_L)$$
$$W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) + z_t$$

visits counts are updated as well

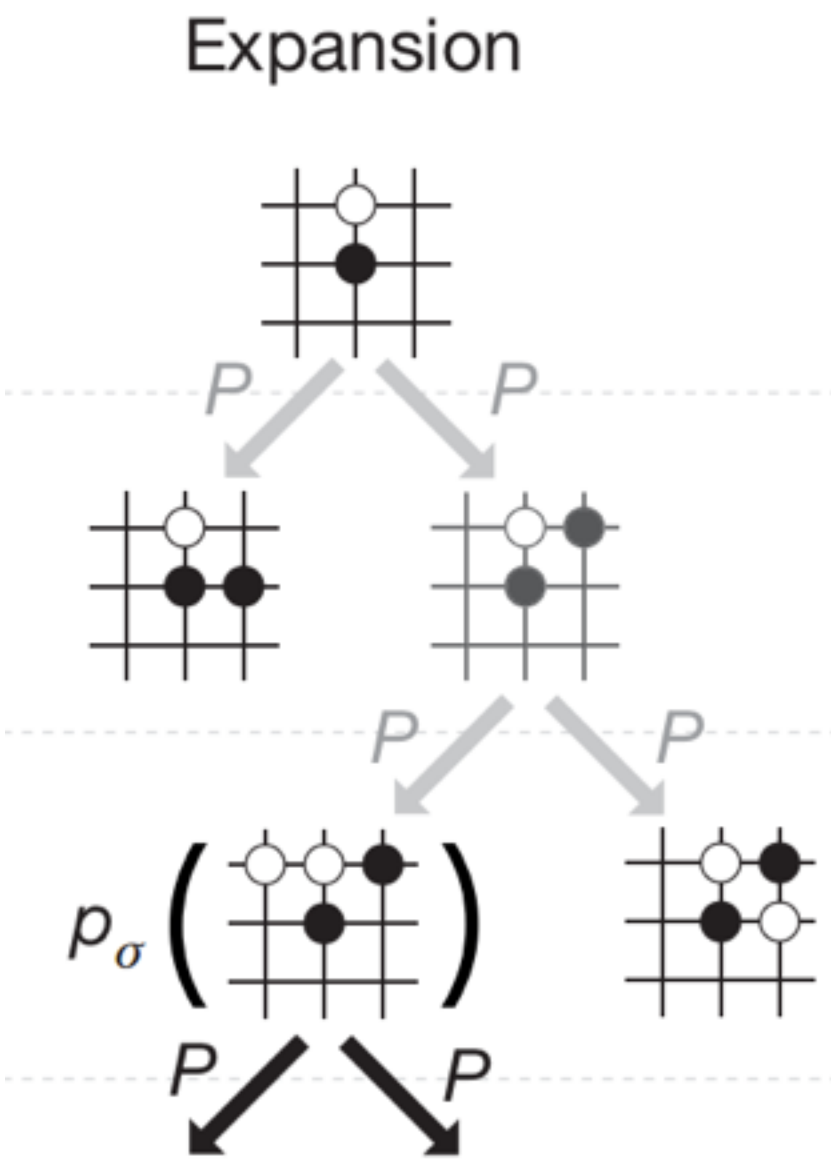
$$N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) + 1$$
$$N_v(s_t, a_t) \leftarrow N_v(s_t, a_t) + 1$$

Finally overall evaluation of each visited state-action edge is updated

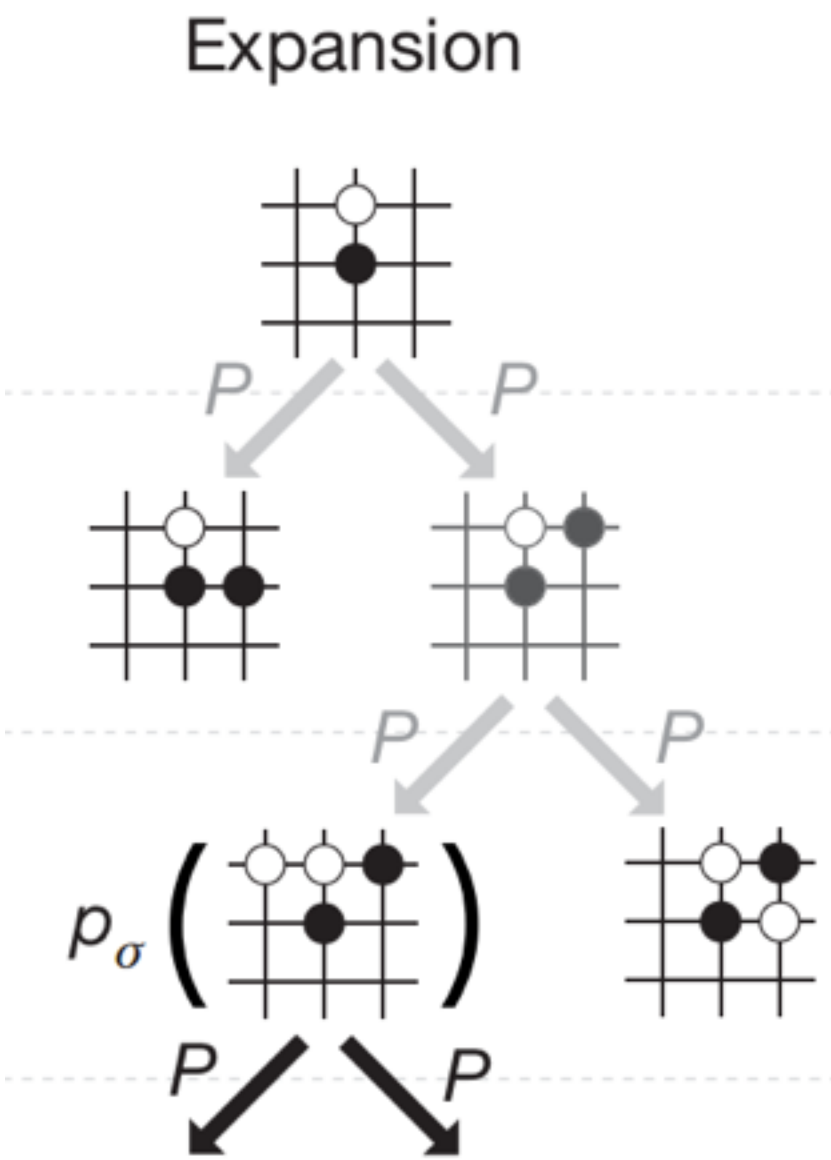
$$Q(s,a) = (1-\lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{W_r(s,a)}{N_r(s,a)}$$

Current tree is updated

APV-MCTS Asynchronous Policy and Value MCTS

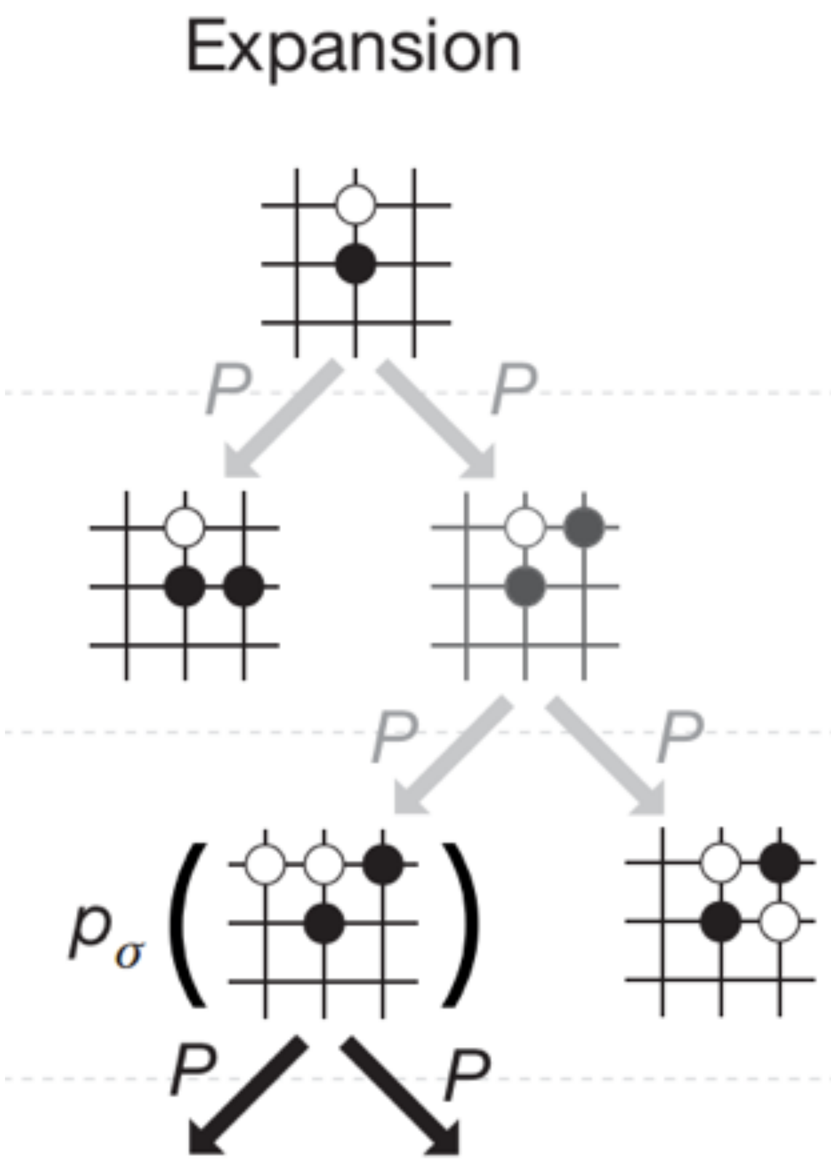


APV-MCTS Asynchronous Policy and Value MCTS



Once an edge (*s*, *a*) is visited enough (n_{thr}) times it is included into the tree with *s*’

APV-MCTS Asynchronous Policy and Value MCTS



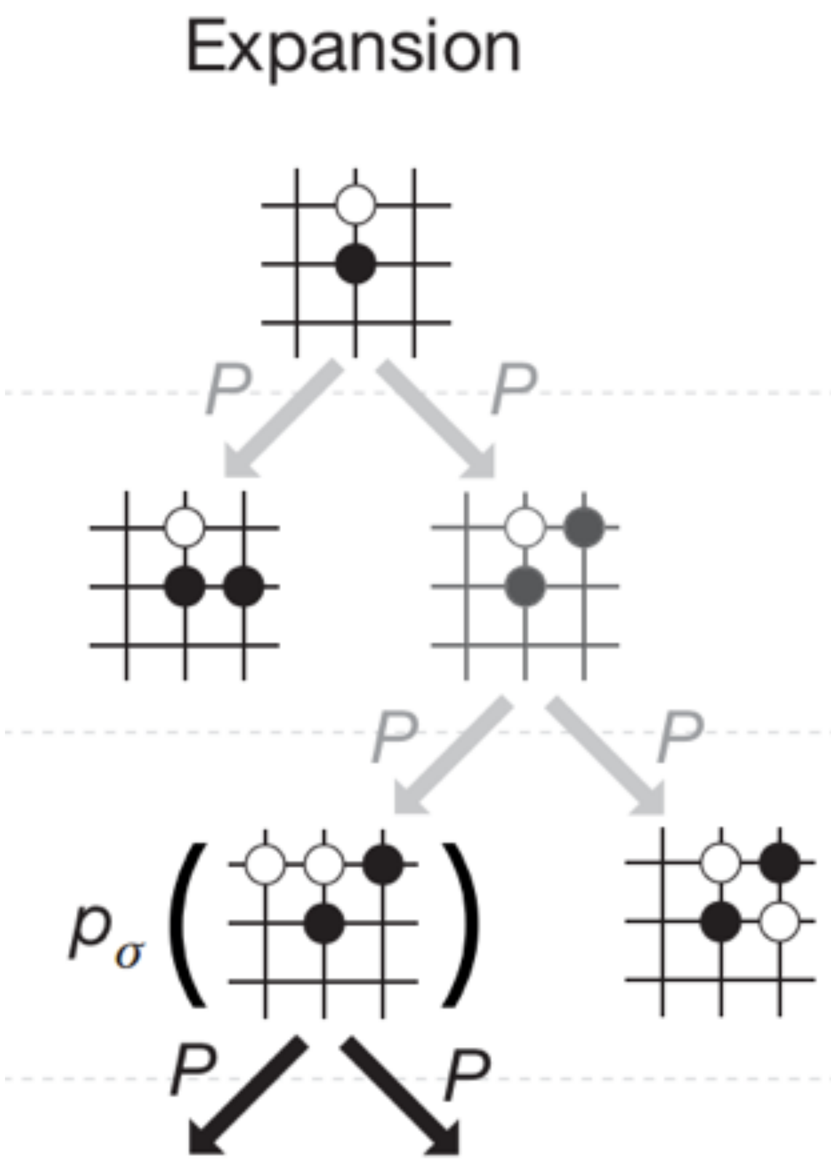
Once an edge (s, a) is visited enough (n_{thr}) times it is included into the tree with s'

It is initialized using the tree policy $p_\tau(a|s')$ to

$\{N(s', a) = N_r(s', a) = 0,\ W(s', a) = W_r(s', a) = 0,\ P(s', a) = p_\tau(a|s')\}$

and updated with SL policy: $P(s', a) \leftarrow p_\sigma(a|s')$

APV-MCTS Asynchronous Policy and Value MCTS



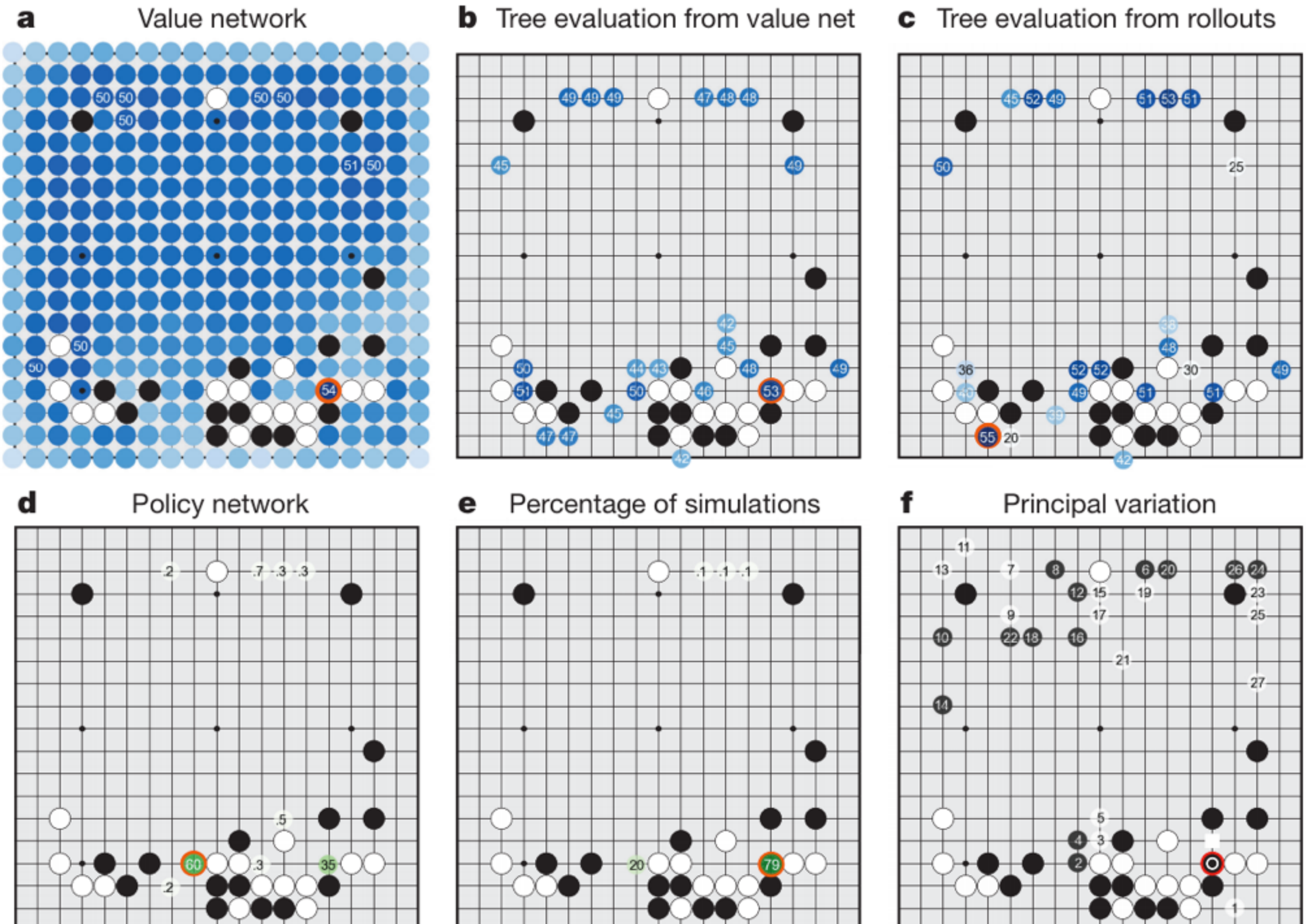
Once an edge (*s*, *a*) is visited enough (n_{thr}) times it is included into the tree with *s*'

It is initialized using the tree policy $p_\tau(a|s')$ to

$\{N(s', a) = N_r(s', a) = 0,\ W(s', a) = W_r(s', a) = 0,\ P(s', a) = p_\tau(a|s')\}$

and updated with SL policy: $P(s', a) \leftarrow p_\sigma(a|s')$

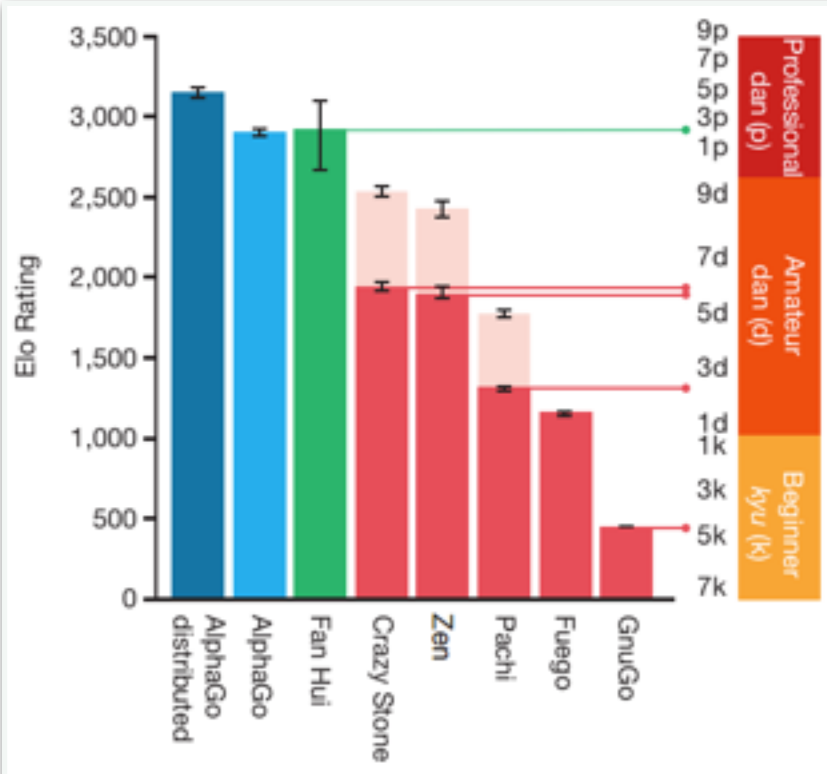
Tree is expanded, fully updated and ready for the next move!



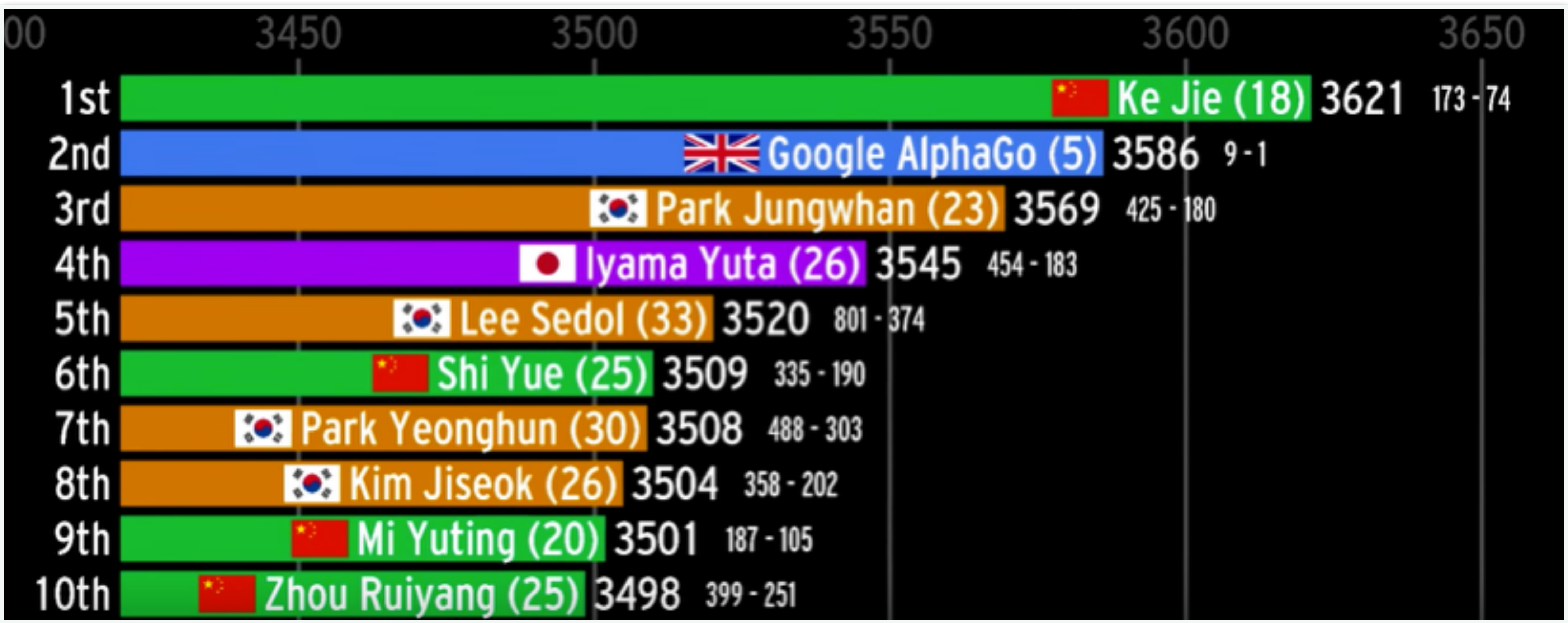
a Value network
b Tree evaluation from value net
c Tree evaluation from rollouts
d Policy network
e Percentage of simulations
f Principal variation

AlphaGo

Winning


Elo Rating
3,500
3,000
2,500
2,000
1,500
1,000
500
0
AlphaGo distributed
AlphaGo
Fan Hui
Crazy Stone
Zen
Pachi
Fuego
GnuGo
9p
7p
5p
3p
1p
9d
7d
5d
3d
1d
1k
3k
5k
7k
Professional dan (p)
Amateur dan (d)
Beginner kyu (k)


AlphaGo


00
3450
3500
3550
3600
3650
1st Ke Jie (18) 3621 173 - 74
2nd Google AlphaGo (5) 3586 9 - 1
3rd Park Jungwhan (23) 3569 425 - 180
4th Iyama Yuta (26) 3545 454 - 183
5th Lee Sedol (33) 3520 801 - 374
6th Shi Yue (25) 3509 335 - 190
7th Park Yeonghun (30) 3508 488 - 303
8th Kim Jiseok (26) 3504 358 - 202
9th Mi Yuting (20) 3501 187 - 105
10th Zhou Ruiyang (25) 3498 399 - 251