CS 181V: Reinforcement Learning

Lecture 23
April 27, 2020
Neil Rhodes

Images from:

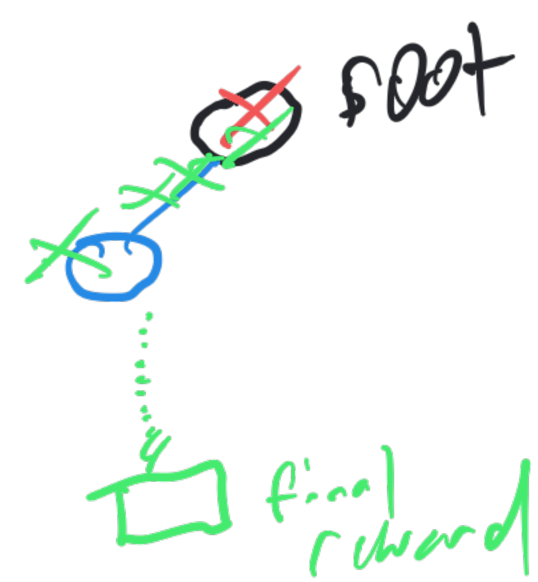
- **David Foster, Applied Data Science**
- ***Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm* by Silver, et al., 2017**
- ***Mastering the game of Go without human knowledge*, Silver, et al., 2017**

Outline

- **Monte Carlo Tree Search refresher**
- DeepMind’s AlphaGo (and successors)

MCTS Refresher

- Monte Carlo Tree Search takes a state, `root`, and rollout policy $\pi_{rollout}$ and produces an improved action



- Algorithm:

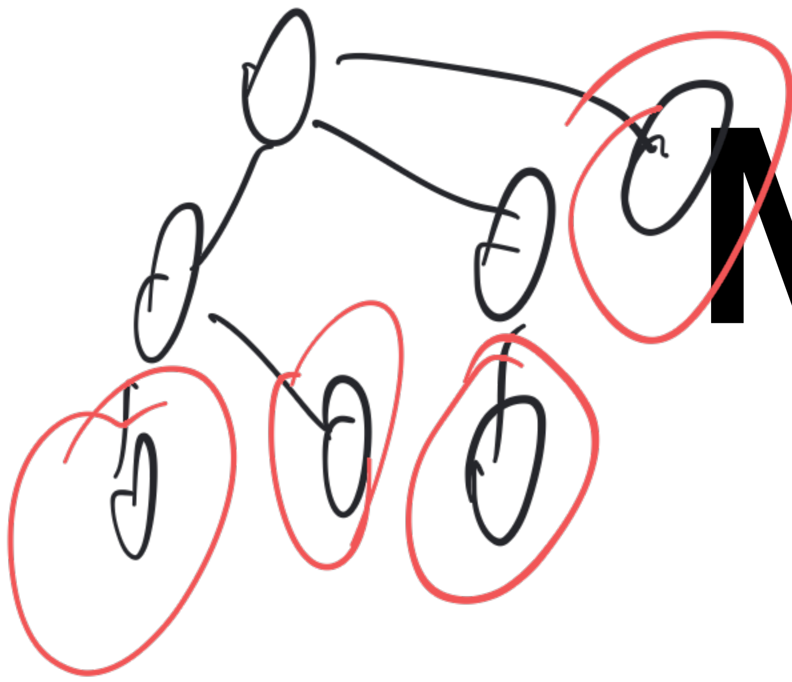
```
repeat many times:
  selected = Select(root)
  expanded = Expand(selected)
  reward = Simulate(expanded, π_rollout)
  Backup(expanded, reward)
Choose root's child action with highest n(v)
```

in expectation $\text{value}(\text{MCTS}(\text{root})) \geq \text{value}(\pi_{\text{rollout}}(\text{root}))$

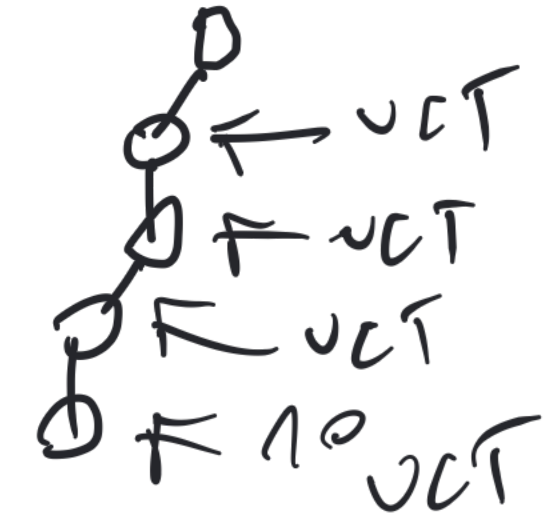
MCTS nodes in tree

- Each node, v stores:
  - $n(v)$: number of times that node has participated in a simulation
  - $q(v)$: total reward attributed to that node during simulations

$\frac{q(v)}{n(v)}$ = avg value for v



MCTS Select



leaf

```
Select(v)
  if node is terminal or has an unexpanded child, return v
  else  return child of v with highest
```

$$UCT(v) = \frac{q(v)}{n(v)} + c\sqrt{\frac{\log n(v.parent)}{n(v)}}$$

exploit
Avg
value

exploration

MCTS Expand

```
Expand(v)
  if node is terminal:
    return v
  c = create random child node of v
  q(c) = n(c) = 0
  return c
```

random unexpanded action

MCTS Simulate

expanded node

```
Simulate(v, π_rollout)
  Choose actions starting from v according
to policy π_rollout until a terminal state is
reached
  return immediate reward leading to that
terminal state
```

MCTS Backup

expanded node

```
Backup(v, reward)
  for all nodes from v up to the root:
    n(v) += 1
    q(v) += reward
```

Outline

- Monte Carlo Tree Search refresher
- **DeepMind's AlphaGo (and successors)**

History

beat prof. players

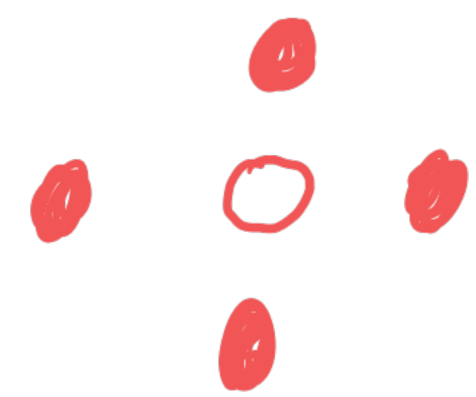
- AlphaGo, 2015
  - Hand-crafted features and trained on human plays
  - Two Neural Networks: One to learn the value function, one to learn policy function

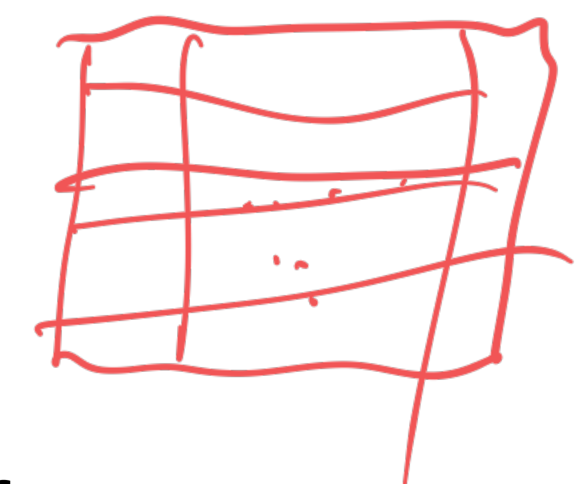
    one learn p : prob of taking action a in state s

    value function : value of state s
  - Policy function trained on large body of human plays

History

zero human preconception all self-play





- AlphaGo Zero, 2017
  - No prior knowledge of Go (no hand-crafted features, no training on human games). All self-play
  - Single Residual Neural Network learns both value and policy functions
  - Keeps best-neural-network-so-far (new champion if beats >55% of games against old champion)

History

/ no more go

not just go

+ chess

+ japanese chess

- AlphaZero, 2017
  - No current best Neural Net. Always uses the latest NN.
  - Same network (other than input and head) with almost same hyper-parameters can learn Go, Chess, and Shogi
    - One hyper-parameter is scaled based on number of possible actions

History

- MuZero, 2019
  - Doesn't know the rules of the game (During MCTS, must use learned representation of the game dynamics)
  - Extended to work with Atari games as well as Go.

AlphaGo

SL: human chess games

state input

o o o o 8 chosen action target

3 Policy Networks

supervised learning

trained in standard NN fashion

MCTS

	SL	RL	Rollout
Initialized	Random	From SL	Random
Structure	$\boxed{\text{NN}}$	NN	Linear
Training	Human games	RL with self-play	Human games
Input	Hand-crafted features	Hand-crafted features	Low-level handcrafted features
Time to select action	3 ms	3 ms	2 μs

input target

self-play RL NN + MCTS

Given the game state, what is the value of an action?

(prob dist)

$s \rightarrow$ RL network $\xrightarrow{\text{action}}$ MCTS $\rightarrow$ better action for s

action for state s

training example

input	target
s	better action for s

AlphaGo

Hand-crafted Features

- SL/RL example features:
  - Is a move at this point a successful ladder capture
  - How many opponent stones would be captured playing at this point

19

19

- Rollout example features:
  - Move matches 3x3 pattern around the move (69938 features)
  - Move saves stones from capture

board

AlphaGo

state

Value network

SL/RL network output?

policy

input state (board + features)

prob dist over

$\underbrace{19 \times 19}_{\text{board locations}} + 1$

↑ pass

- Like SL network, but has only a single output:

single # −1 .. 1

0..1

p(current player winning)

0 if anybody's gam

−.7 not good for current player

.7 is good " " "

What is the value of a game state?

AlphaGo

Overview of Training

train SL & rollout policy w/ supervised learning (human games)

- Initialize Neural Network

initialize RL weights from SL

- Repeat in parallel:

2 networks to train
RL NN
state value network

RL policy
rollout policy

1. Self-play a game. Do MCTS for each move.
   MCTS(P, s) is a new policy, π, more accurate than P.

   RL policy network

   chosen action

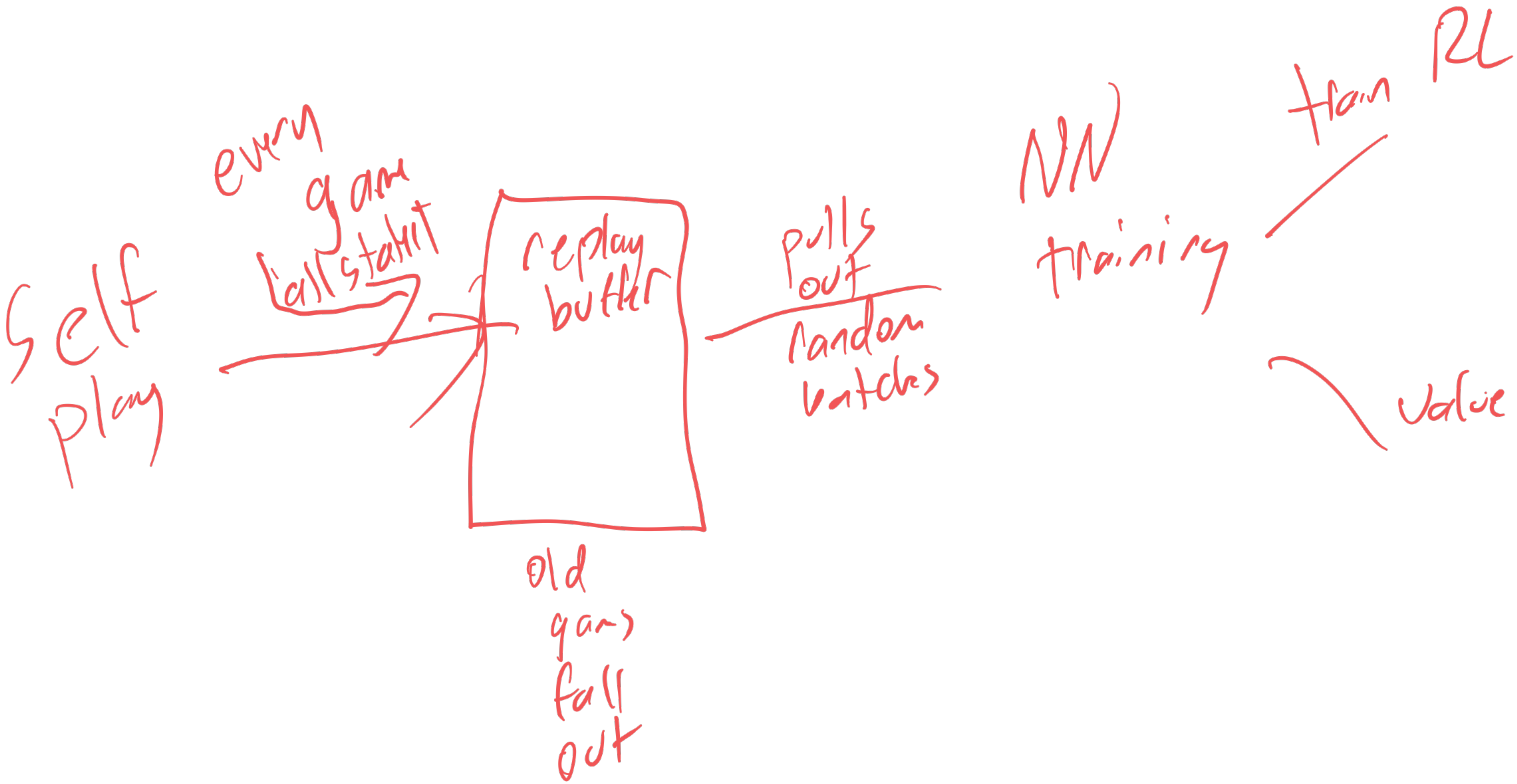
   - For each state, record state, π(s), win_lose_or_draw_result

   state in self-play game

2. Create mini batch of 2048 states from the most recent 500,000 games

   experience replay buffer

   - Use mini batch to train RL network and value network


every game
(all states)
Self play
replay buffer
pulls out
random batches
old games fall out
NN training
tran RL
value

AlphaGo

How is Policy network used?

P(s,a) is based on NN policy

- As tree policy in MCTS to guide action selection
- When choosing an action from a node s_t in the MCTS tree:

select phase

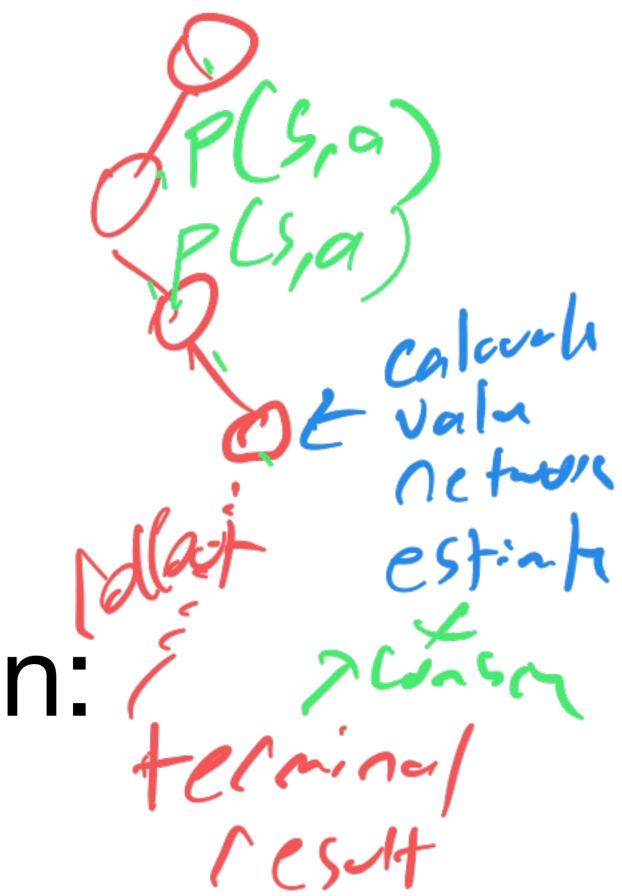
$$a_t = \text{argmax}_a(\overbrace{Q(S_t, a)}^{\text{MC approx}} + U(s_t, a))$$

- bonus (incorporates tree policy and exploration bonus):

$$U(s, a) \propto \overbrace{\boxed{\frac{P(s, a)}{1 + N(s, a)}}}^{\text{bonus}}$$

gives bonus if:
tree policy likes this action
or
we haven't tried this action much

AlphaGo How is Value network used?



- To help get better simulation results
- After we have expanded node: n_e, we do simulation:
    - Do rollout (to obtain win/draw/lose) to estimate value: $v_1(n_e)$ use rollout policy for rollout
    - Also use value network to estimate value: $v_2(n_e)$
- Use weighted average as value: $\lambda v_1(n_e) + (1 - \lambda) v_2(n_e)$

AlphaGo

Monte Carlo Tree Search (MCTS)

Each state has edges for all legal actions:

For each edge keep:

***N(s, a):* how many times has this state/action pair been seen**

***W(s, a)*: total action-value**

Q(s, a)*: mean action-value: *W(s, a)/N(s, a) Calculated

***P(s, a)*: prior probability of selecting that edge**

comes from tree policy (NN)

AlphaGo

MCTS for AlphaGo

(different from std MCTS)

- Monte Carlo Tree Search takes a state, `root`, a tree policy π_{tree}, a rollout policy $\pi_{rollout}$, a state value approximator and produces an improved action

- Algorithm:

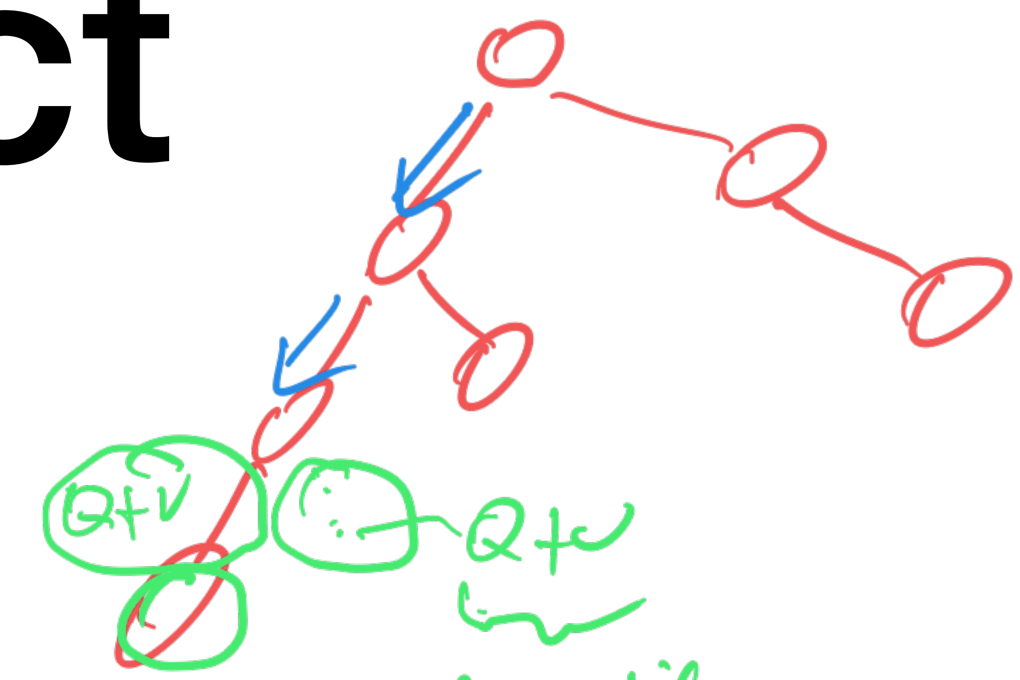
← n is a hyper-param

```
repeat many times:
  expanded = Select(root)
  reward = Evaluate(expanded, π_rollout, vapprox)
  Backup(expanded, reward)
Choose root's child action with highest n(v)
```

← π_{tree}

AlphaGo

MCTS: Select



simplify to constant · P(s,a)

- Different from MCTS we've seen:

```
Select(n):
  if n is terminal:
    return n
  child = child with highest Q+U (including
    unexpanded children)
  if child is expanded:
    return Select(child)
  return Expand(child)
```

Same as normal MCTS

AlphaGo

MCTS Evaluate

```
Evaluate(n, π_rollout, vapprox)
  Choose actions starting from n according
to policy π_rollout until a terminal state is
reached
  v_1 is value of that terminal state
according to Go rules
  v_2 is vapprox(n)
  return λv_1 + (1 − λ)v_2
```

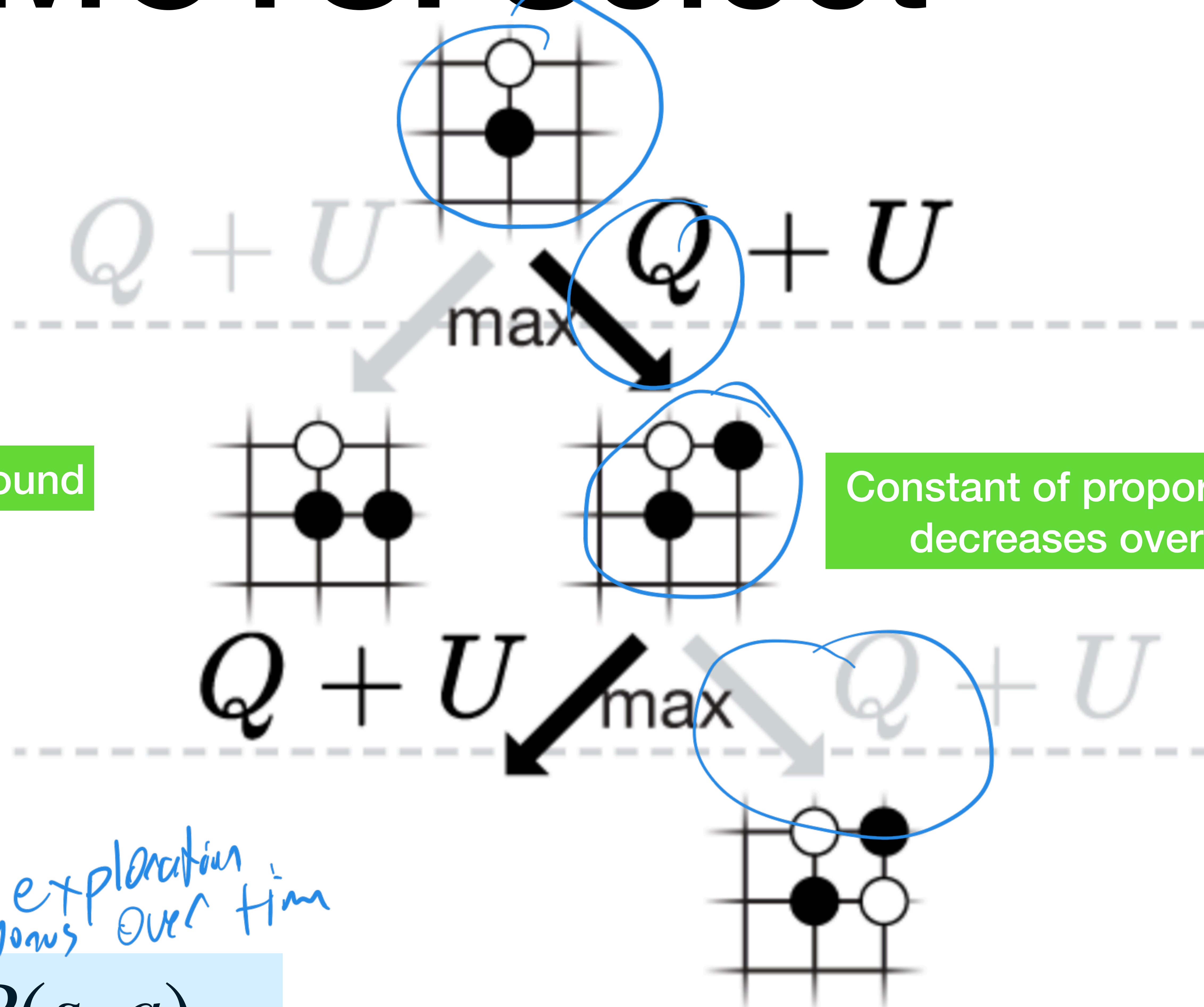
do a rollout

– NN

$$\lambda v_1 + (1-\lambda) v_2$$

AlphaGo

MCTS: Select



U Is upper-confidence bound

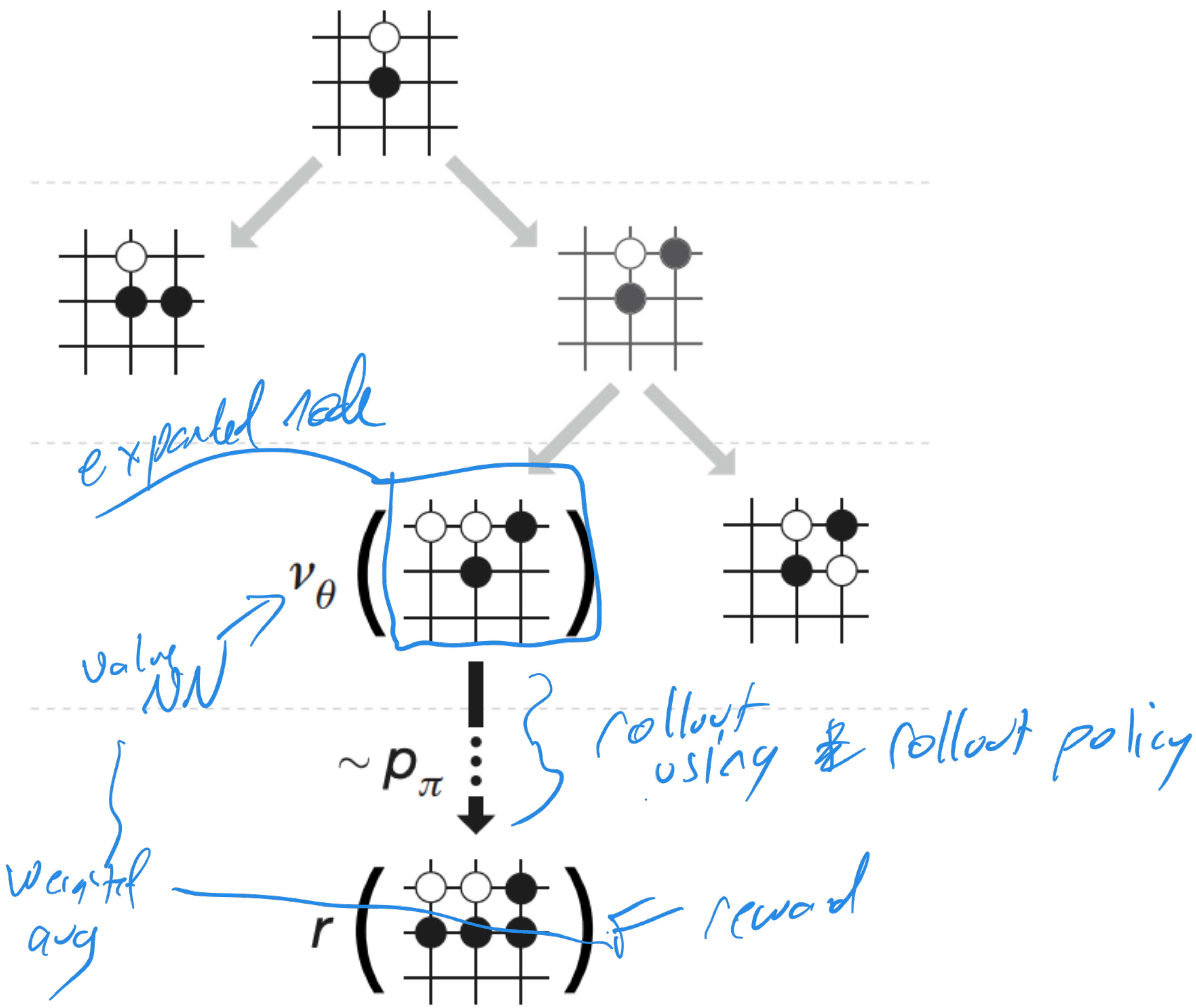
Constant of proportionality decreases over time

reduce exploration over time
grows

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

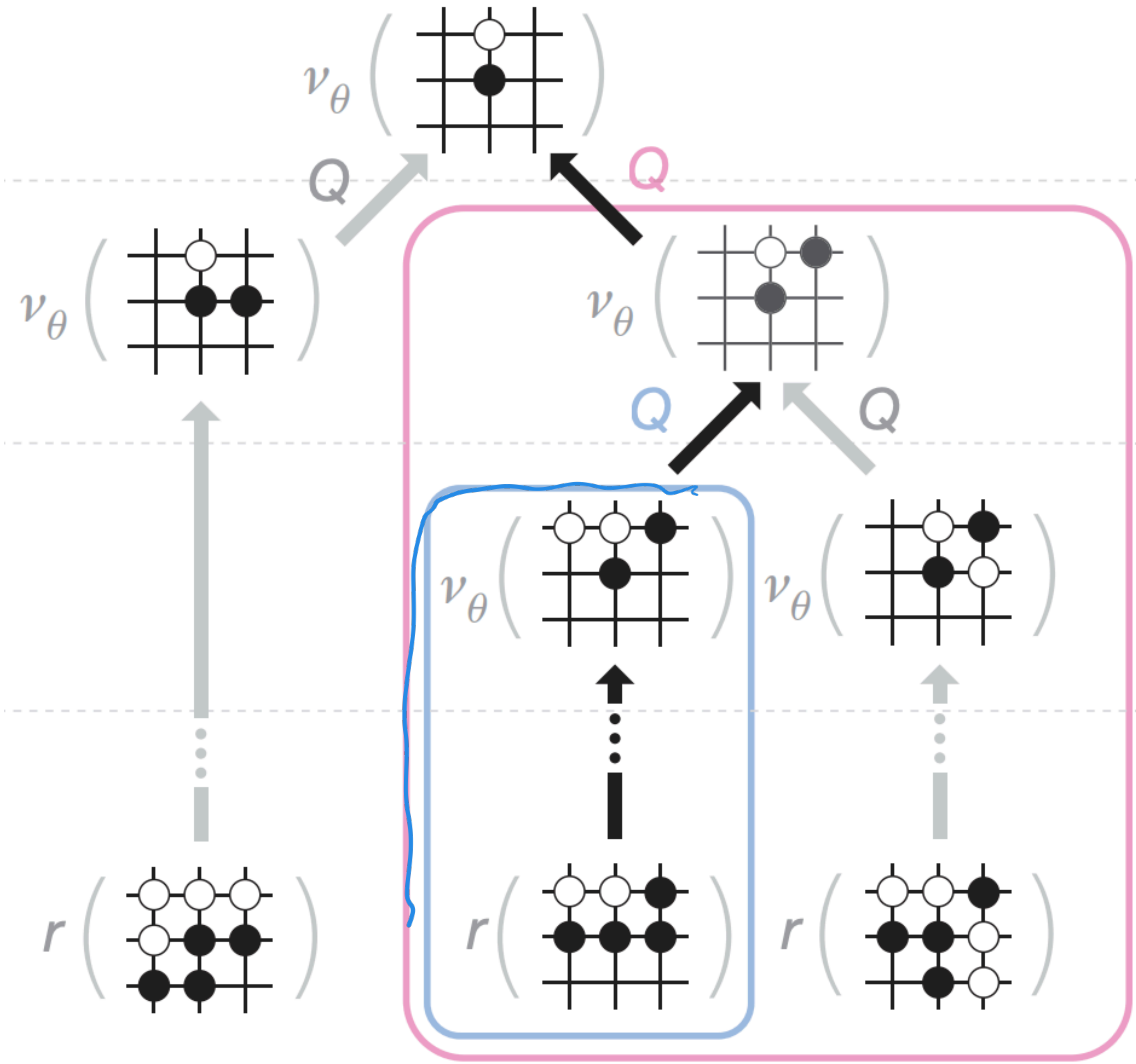
AlphaGo

MCTS:
Expand & Evaluate



AlphaGo

MCTS: Backup



For each ancestor edge:

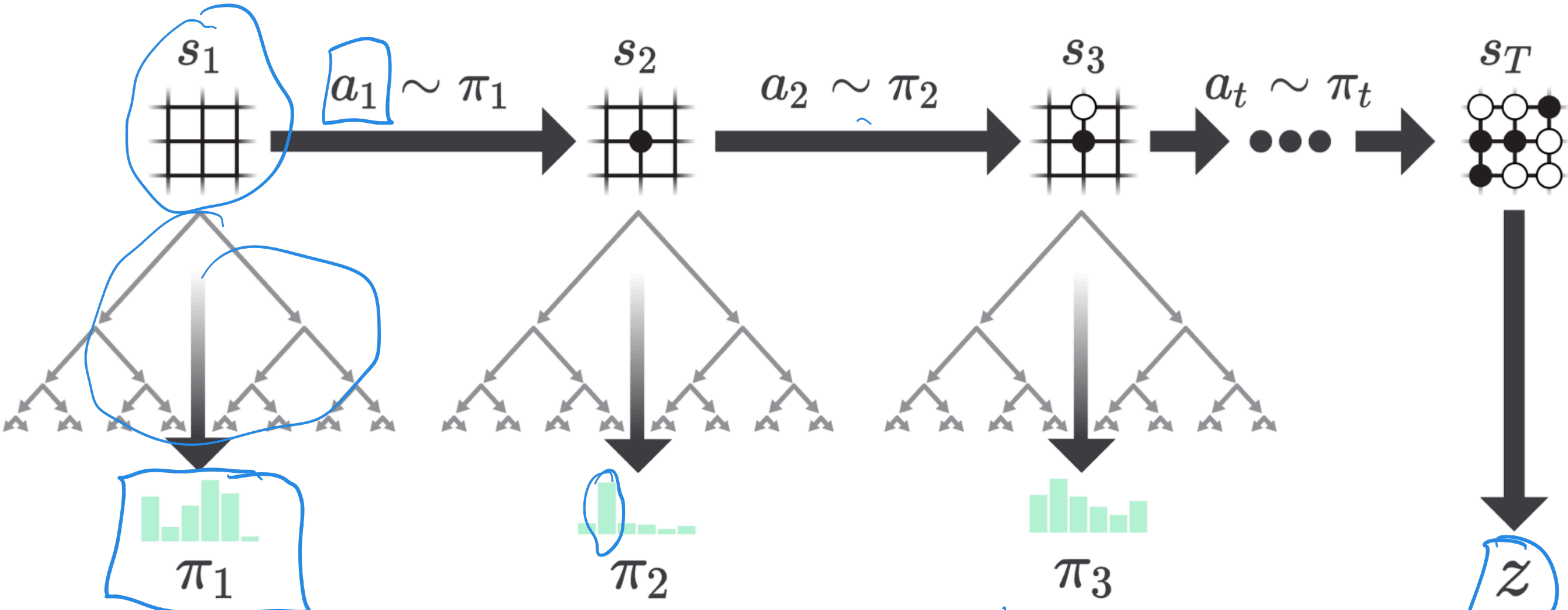
$N(s,a) += 1$

$W(s,a) += r$

weighted reward

AlphaGo

Self-play



train Value NN w/ (s_i, z) pairs

train RL NN w/ (s_i, a_i) pairs

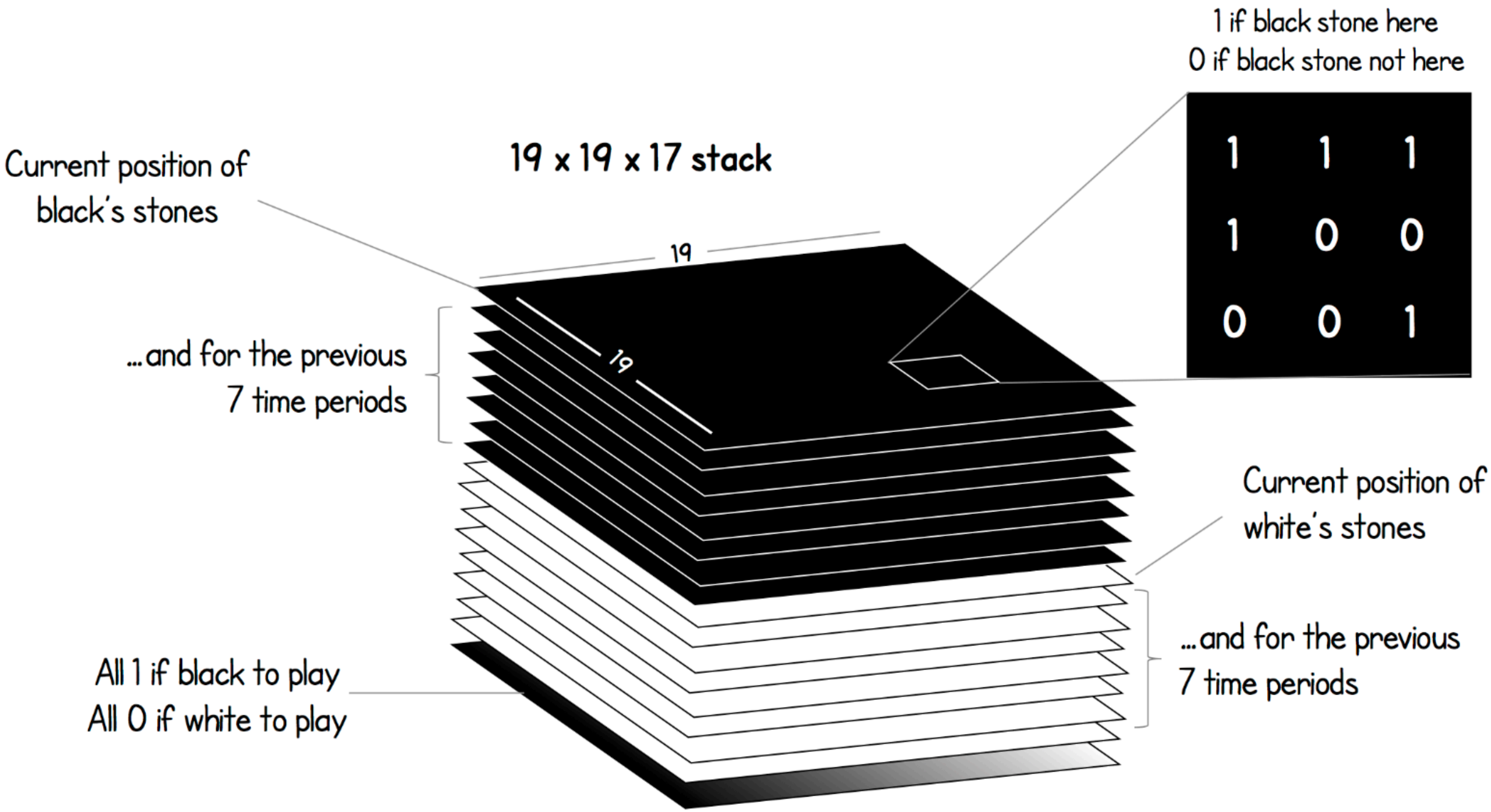
final result of game

AlphaGo Zero

Learn through self-play

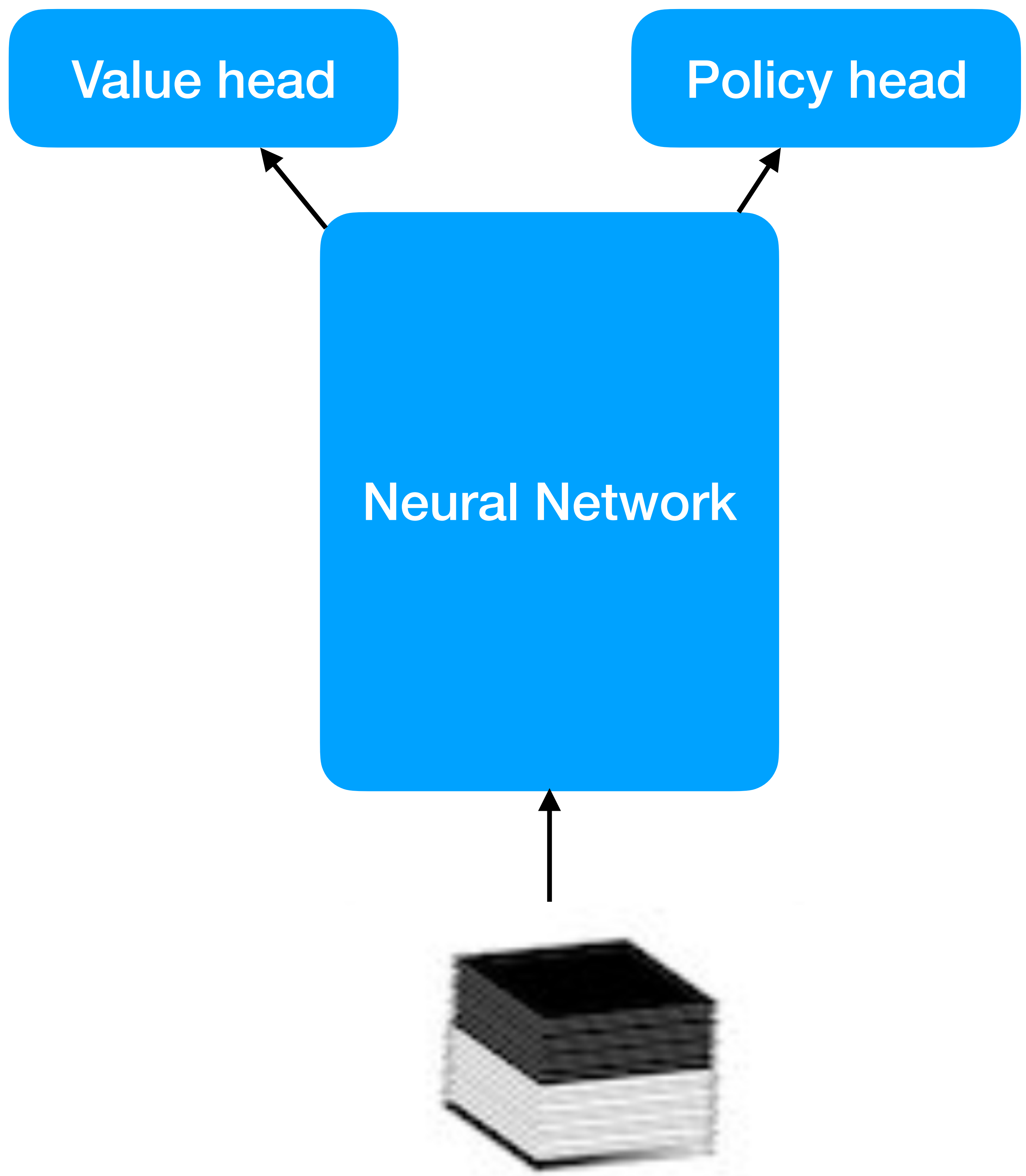
- Get rid of handcrafted features
- No network weights pretrained from human games
- One shared network to compute both policy and value
- Simplified MCTS: no rollouts!

AlphaGo Zero: What is a game state for Go?



AlphaGo Zero

The Neural Network



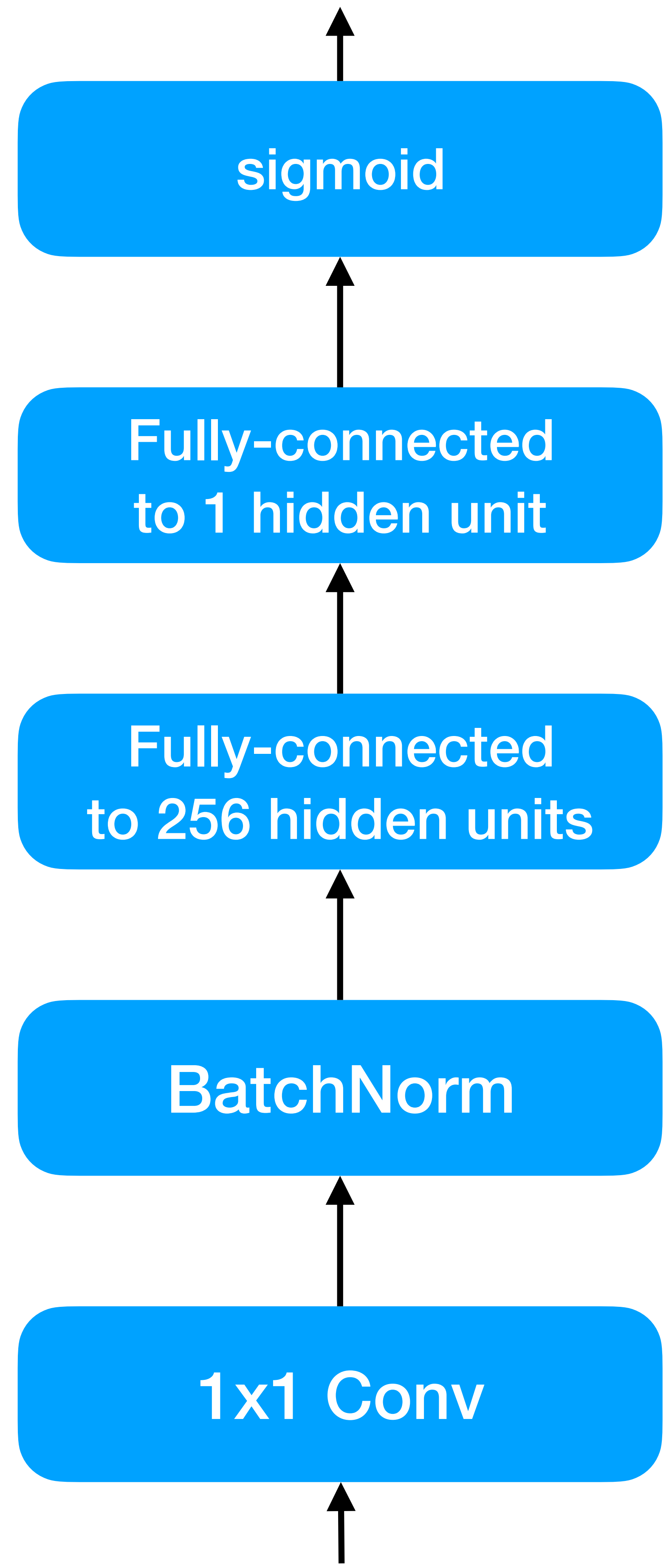
Input (Game state)

Multi-task learning!

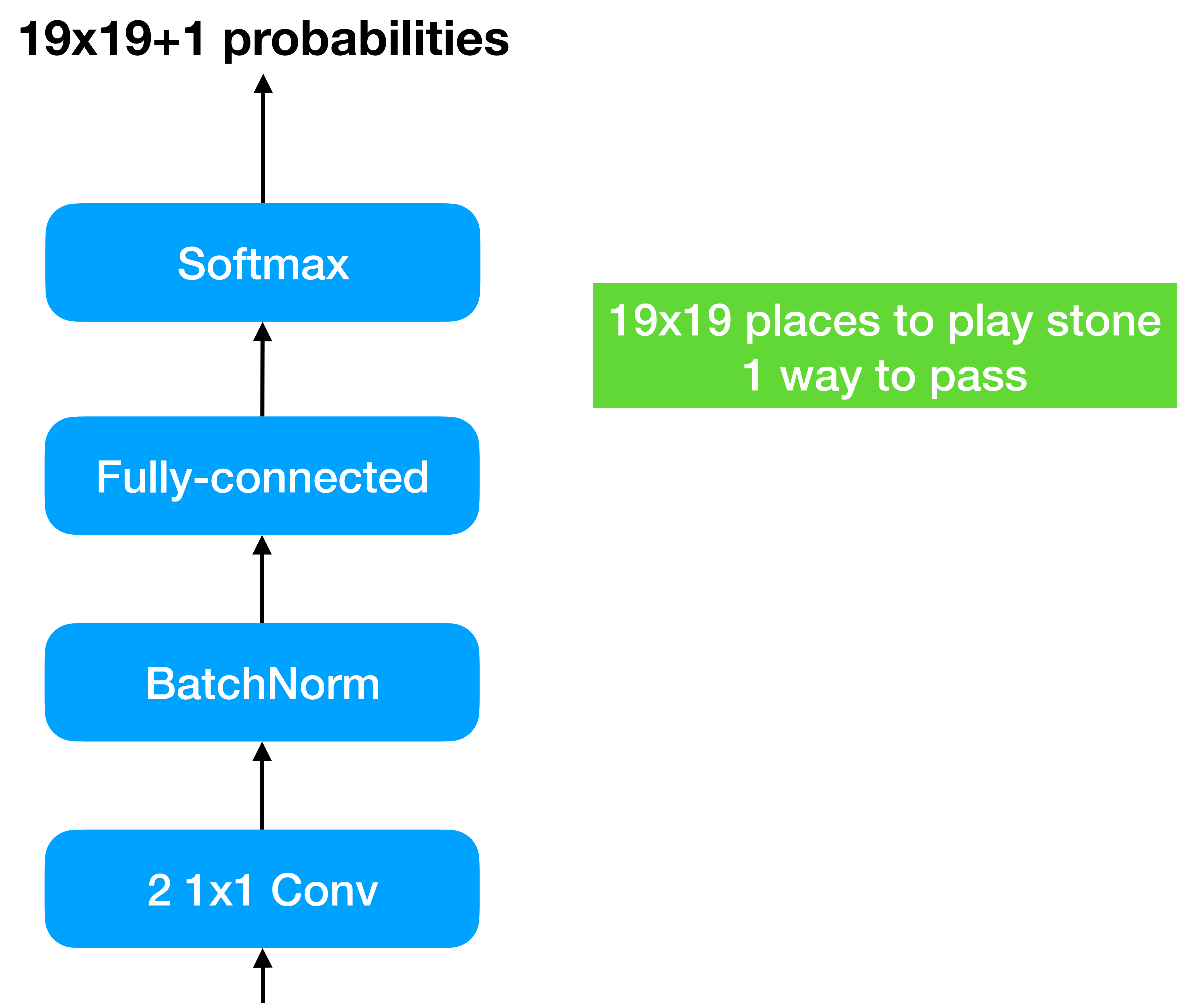
AlphaGo Zero

Value (V) Head

$\mathbb{R} \in [0, 1]$: **probability of winning for current player**



Policy (P) Head



AlphaGo Zero

Overview of Training

- Initialize Neural Network

- Repeat in parallel:

1. Self-play 44 million games (800 Monte Carlo Tree Search [MCTS] simulations/move).
   MCTS(P, s) is a new policy, π, more accurate than P.

   - For each state, record state, π(s), win_lose_or_draw_result

2. Create mini batch of 2048 states from the most recent 500,000 games

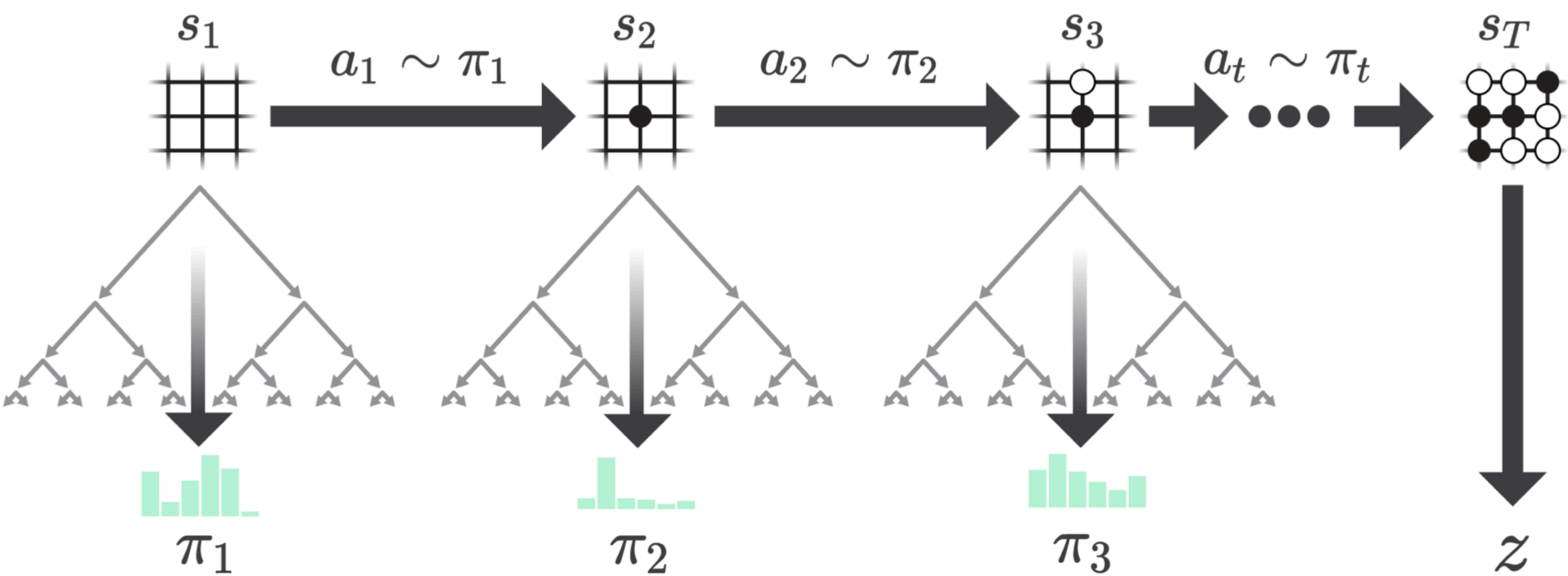
   - Use mini batch to train Neural Network

For chess, took 9 hours to run

Total of 700,000 mini batches

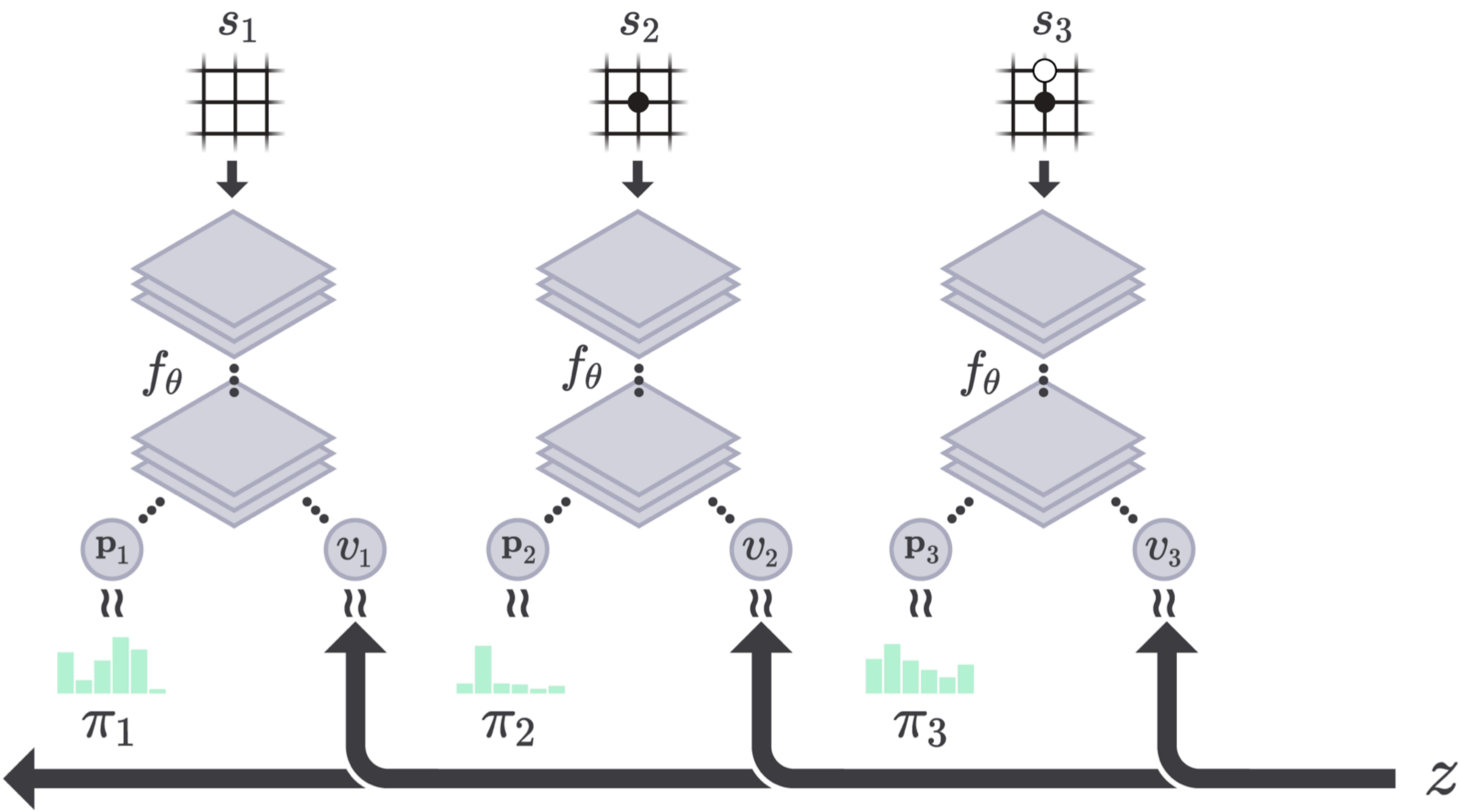
AlphaGo Zero

Self-play



AlphaGo Zero

Neural Net Training



$$L = (z - v)^2 - \pi^T \log p + c\|\Theta\|^2$$

AlphaGo Zero

Neural Net Training

- Update Θ so that:
  - Given an input state, s_i, f_Θ produces output that is closer to:
    - For value portion: z, the actual result of the game
    - For action portion: π_i, the MCTS-improved policy based on the neural net's p_i

AlphaGo Zero

Monte Carlo Tree Search (MCTS)

Each state has edges for all legal actions:
For each edge keep:

- ***N(s, a):*** **how many times has this state/action pair been seen**
- ***W(s, a)*****: total action-value**
- ***Q(s, a)*****: mean action-value:** ***W(s, a)/N(s, a)***
- ***P(s, a)*****: prior probability of selecting that edge**

AlphaGo Zero

Explore/Exploit

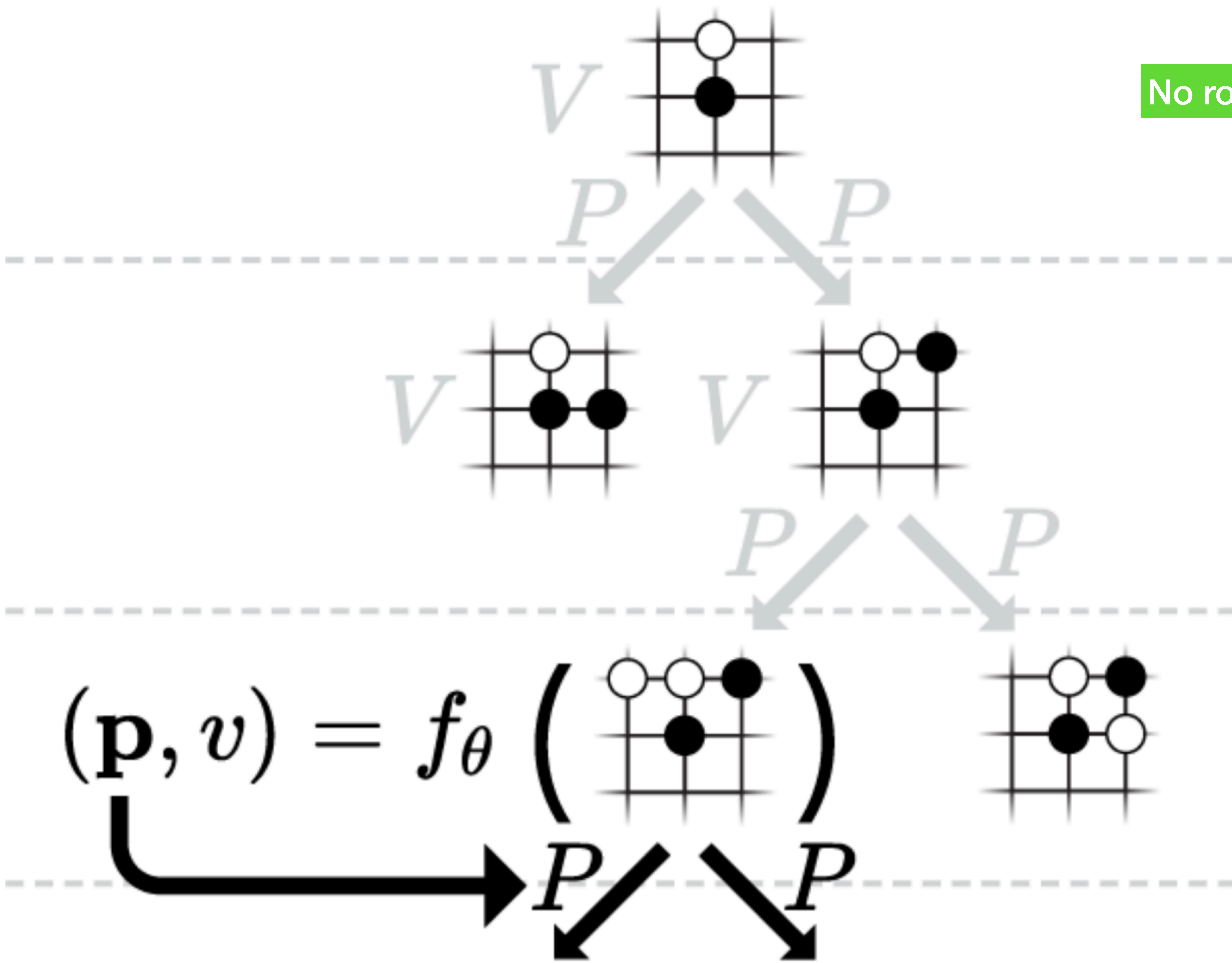
- Two sources to encourage exploration:
  - Dirichlet noise added to top-level P(s) of MCTS
    - Non-zero chance of any move happening
- Upper-confidence bound when evaluating move in MCTS
  - Encourage actions whose confidence is low

Dirichlet noise: sums to given value (so can be a probability), and, with parameter used in AlphaGoZero,
makes most of the probability focused in small area

AlphaGo Zero

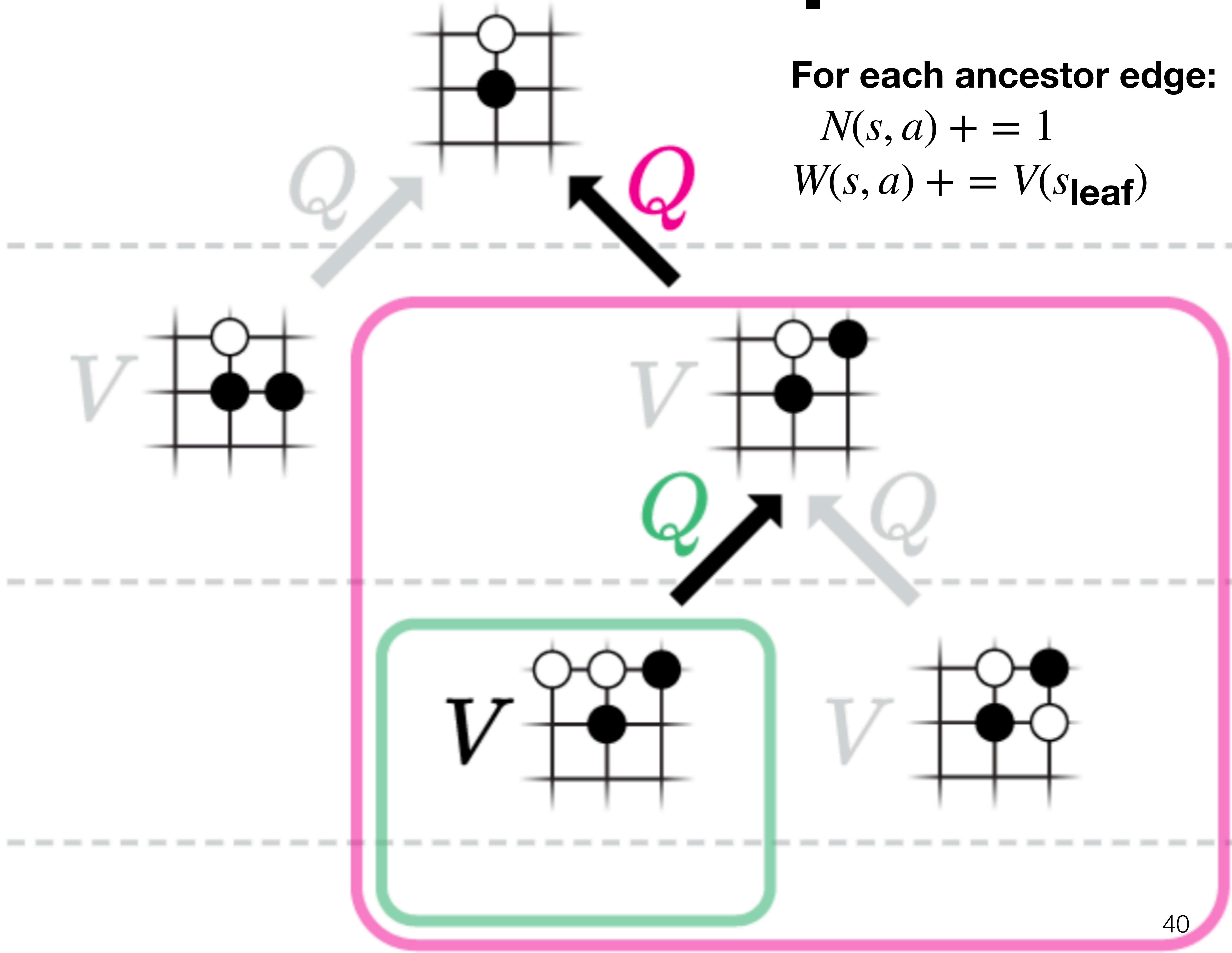
MCTS: Evaluate

No rollout!



AlphaGo Zero

MCTS: Backup



For each ancestor edge:

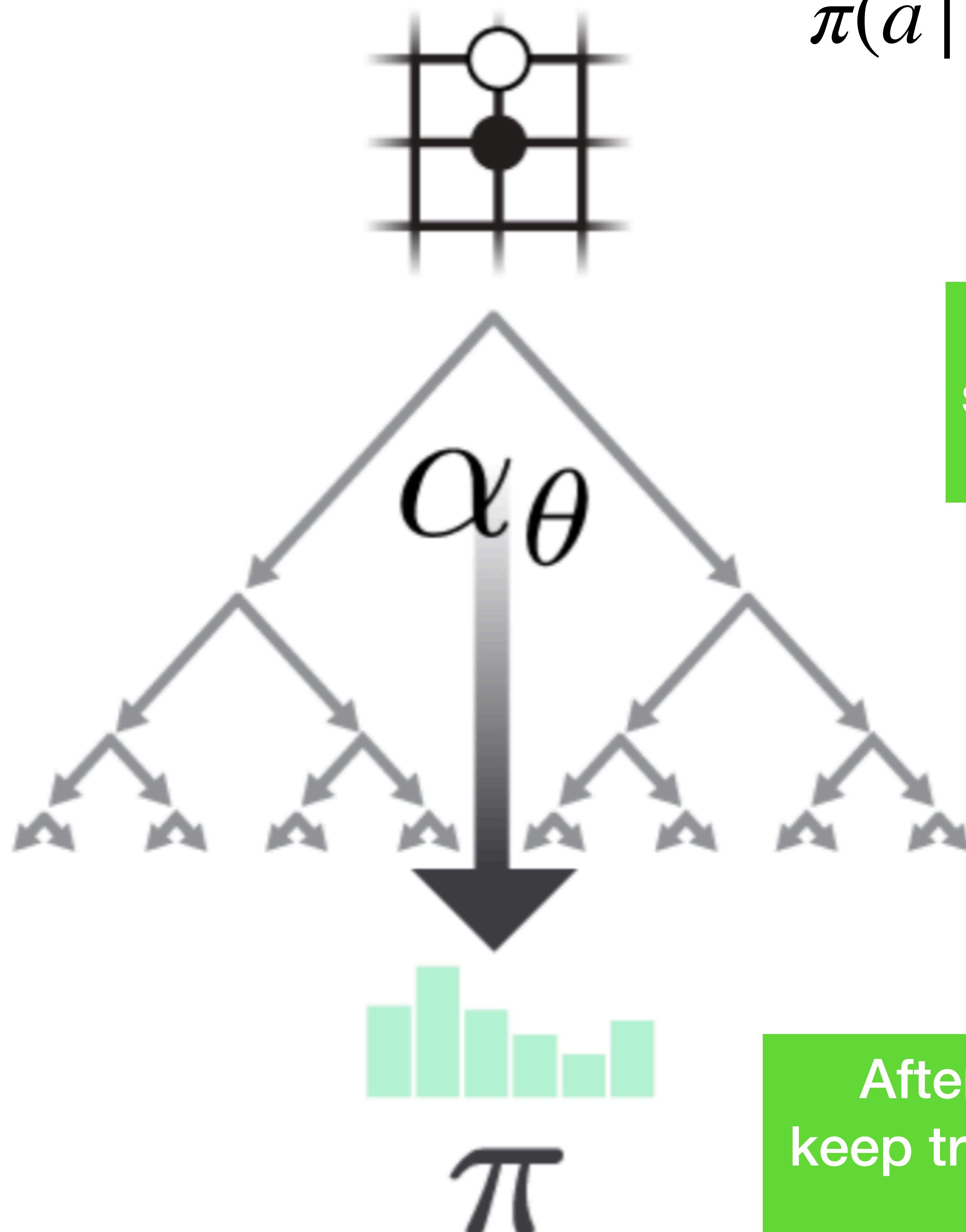
$N(s,a) \mathrel{+}= 1$

$W(s,a) \mathrel{+}= V(s_{\textbf{leaf}})$

AlphaGo Zero

MCTS: Play

$$\pi(a \mid s_0) = \frac{N(s_0, a)^{1/\tau}}{\sum_b N(s_0, b)^{1/\tau}}$$



τ is a temperature constant: starts near 1 and ends close to zero. For play: close to zero

After an action is chosen from π, keep tree starting from resulting state, clear rest of tree

AlphaGo Zero

What Go-specific knowledge is used?

- Network knows input format (19x19 board, etc.)
- Network knows how many possible actions (19x19+1)
- MCTS knows which actions lead to which states
- MCTS knows whether a state is terminal
- MCTS knows how to score a terminal state
- MCTS does dihedral reflection or rotation (takes advantage of symmetry of Go board)