AlphaGo

How it works

PPT provider: Shane (Seungwhan) Moon

PhD student, Carnegie Mellon University

rank

AlphaGo vs European Champion (Fan Hui 2-Dan*)

October 5 – 9, 2015

<Official match>

- Time limit: 1 hour
- AlphaGo Wins (5:0)

AlphaGo vs World Champion (Lee Sedol 9–Dan)

March 9 – 15, 2016

<Official match>

- Time limit: 2 hours
- reward: 1 million USD

Venue: **Seoul**, Four Seasons Hotel

alphaGo Won

4:1

Lee Sedol 9-dan vs AlphaGo

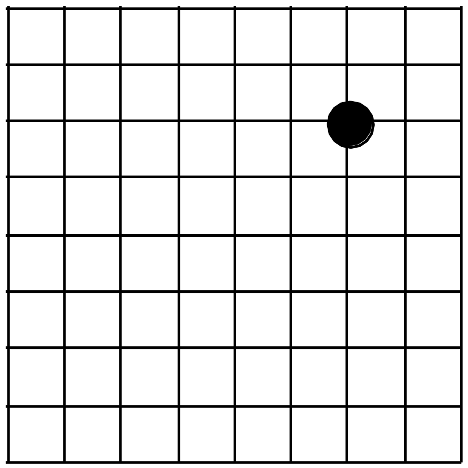
Born in south Korean
12, became a professional player
20, became the world champion

Computer Go AI?

DeepMind in London
2010, start
2014, Google
2015, alphaGO

Computer Go AI – Definition

d = 1



s (state)

$$= \left(\begin{matrix} 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&\mathbf{1}&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{matrix} \right) \quad 19$$

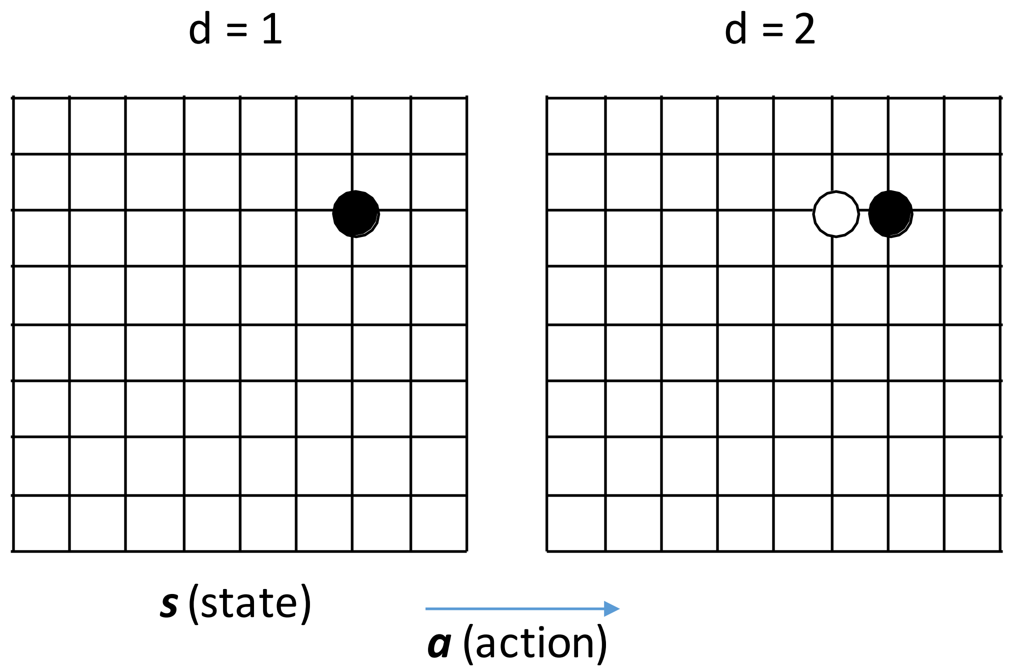
(e.g. we can represent the board into a matrix--like form)

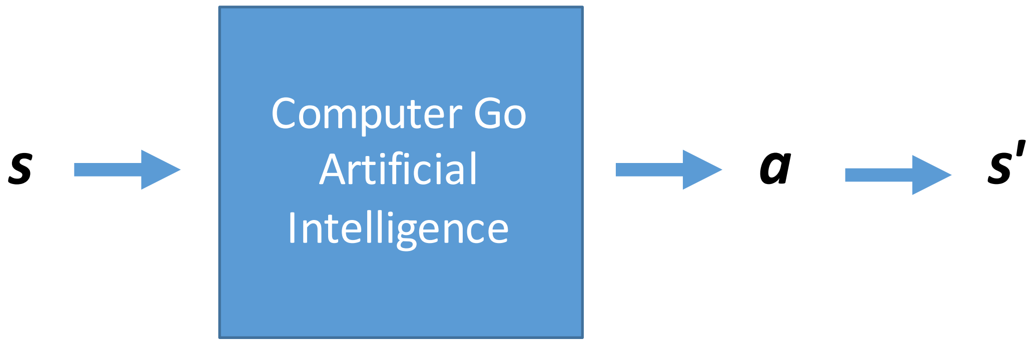
**** The actual model uses other features than board positions as well, (Extended table 4 in paper)***

Extended Data Table 4 | Input features for rollout and tree policy

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

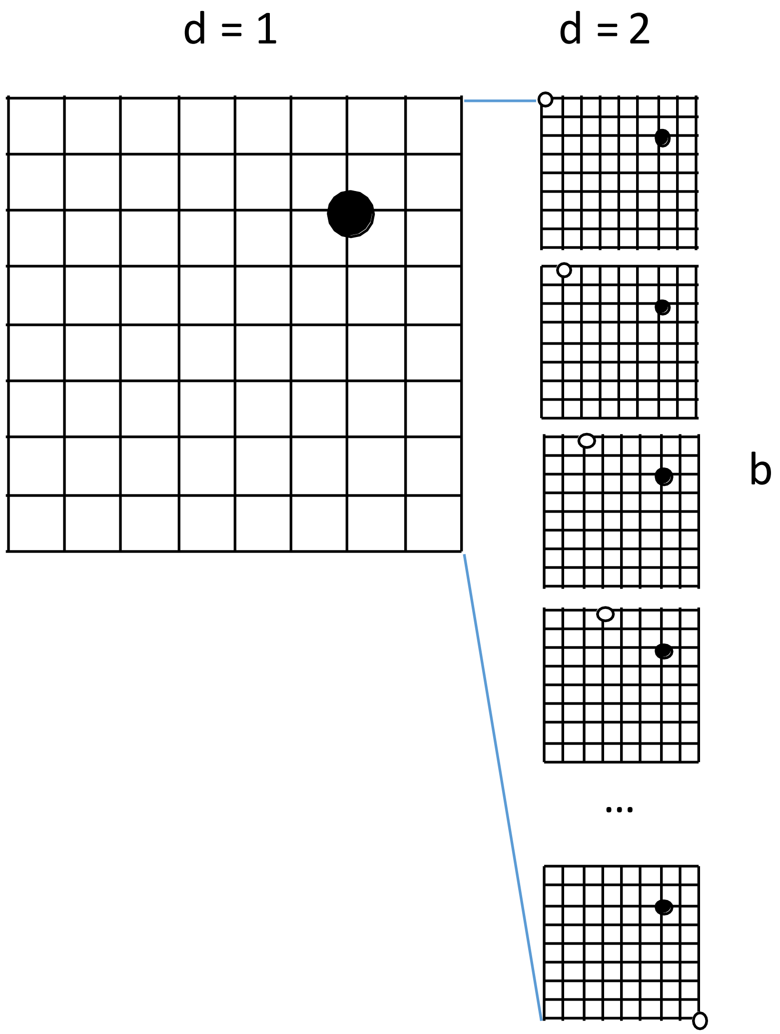
Computer Go AI – Definition





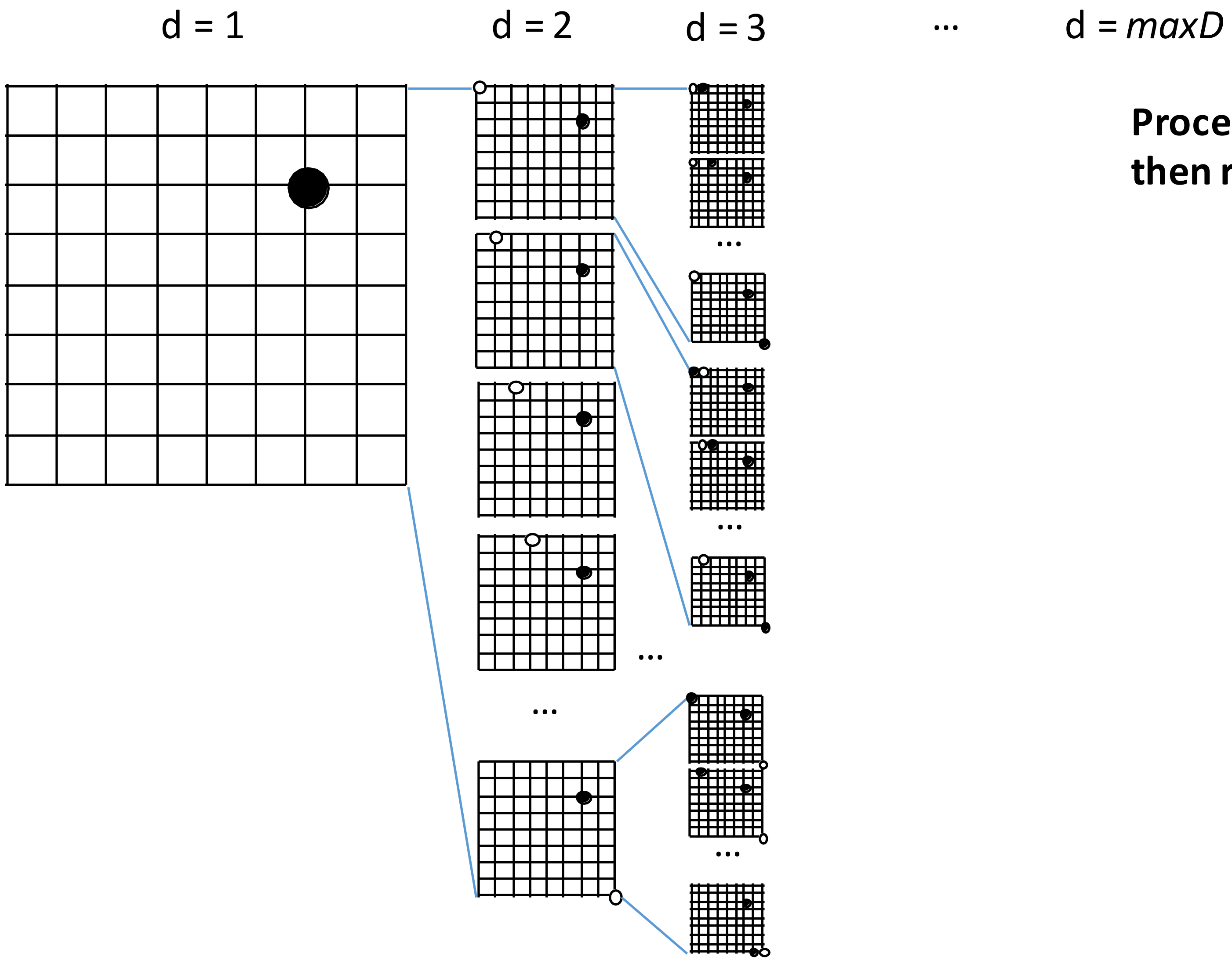
Given ***s***, pick the best ***a***

Computer Go AI – An Implementation Idea?



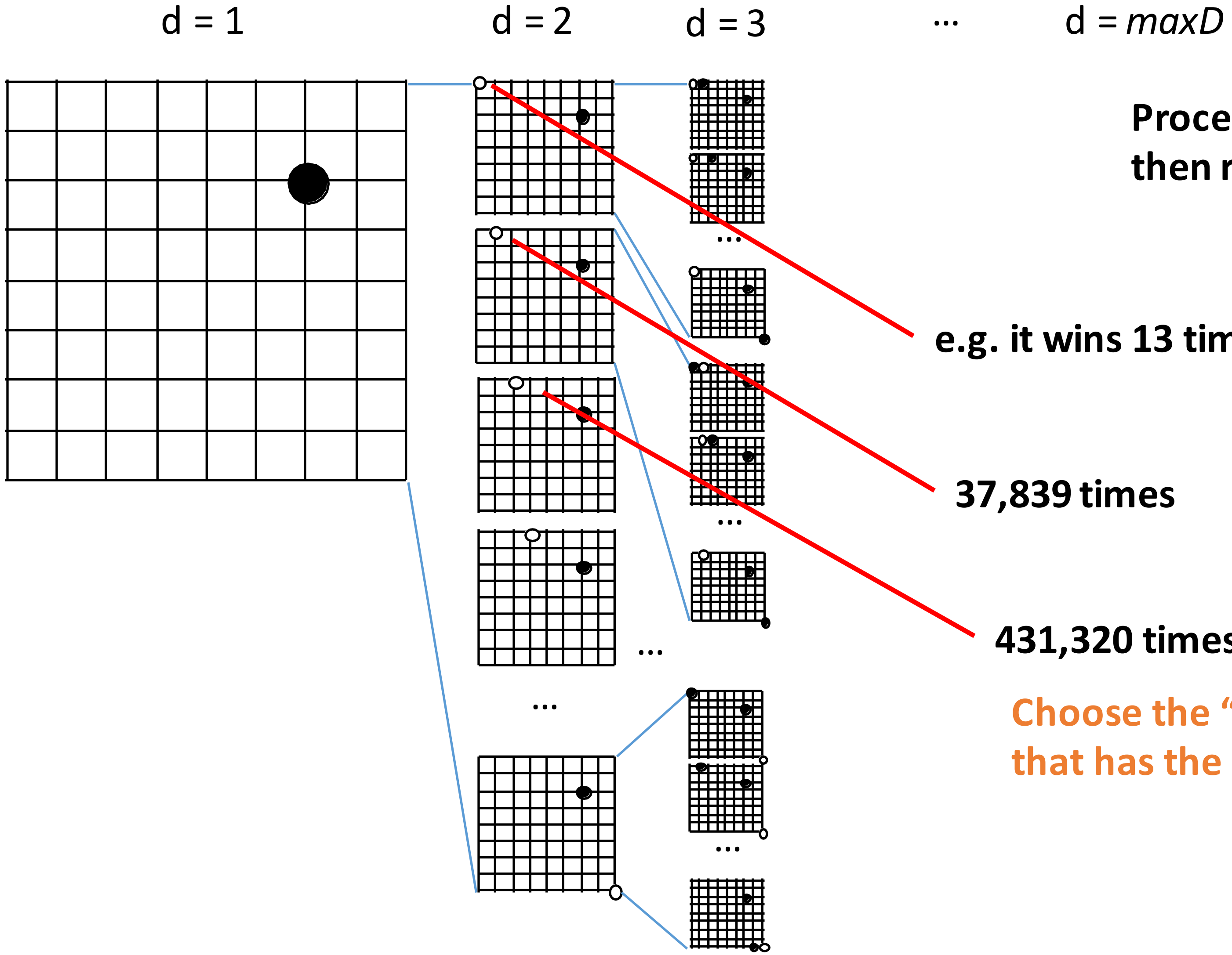
How about simulating all possible board positions?

Computer Go AI – An Implementation Idea?



Process the simulation until the game ends, then report win / lose results

Computer Go AI – An Implementation Idea?



Process the simulation until the game ends, then report win / lose results

e.g. it wins 13 times if the next stone gets placed here

37,839 times

431,320 times

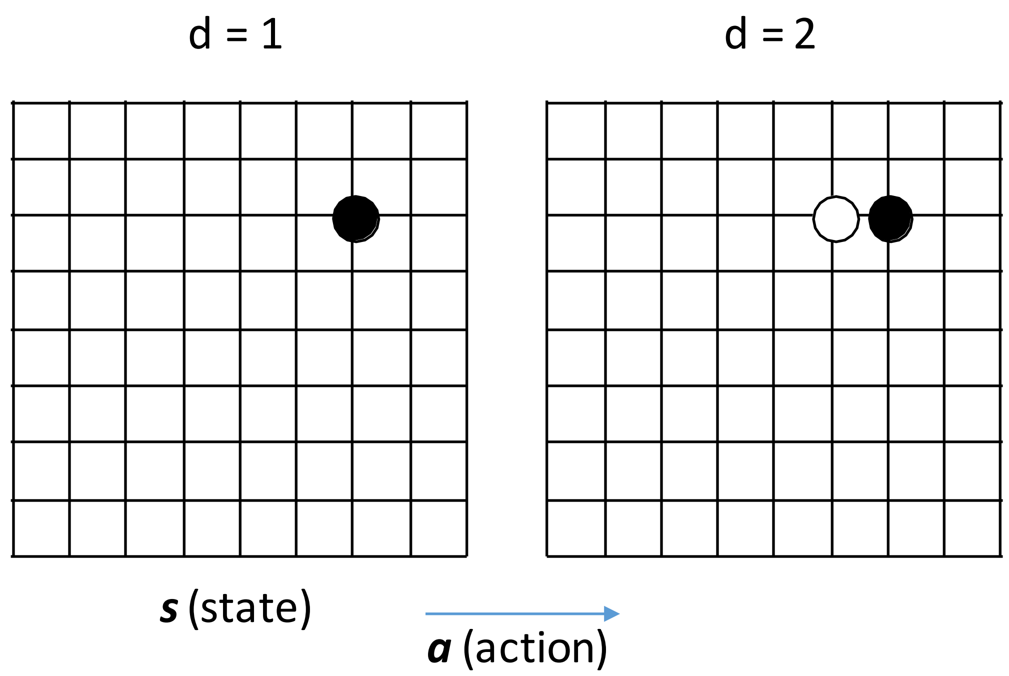
Choose the "nextaction / stone" that has the most win--counts in the full--scale simulation

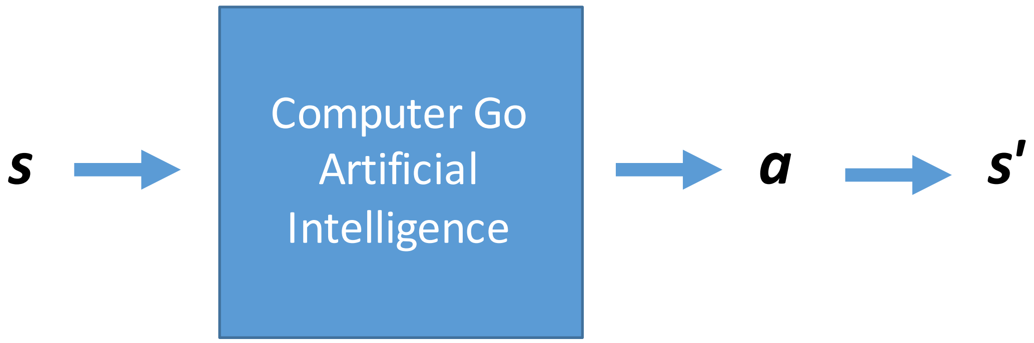
This is NOT possible; it is said the possible configurations of the board exceeds the number of atoms in the universe

b ~ 250

d ~ 150

5659799424266695229693199556804869862926581998836961
3684891343062096883241205281967122071574755988121587
2027456054864353492670804579578458515243255533277988
4338378906250000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000

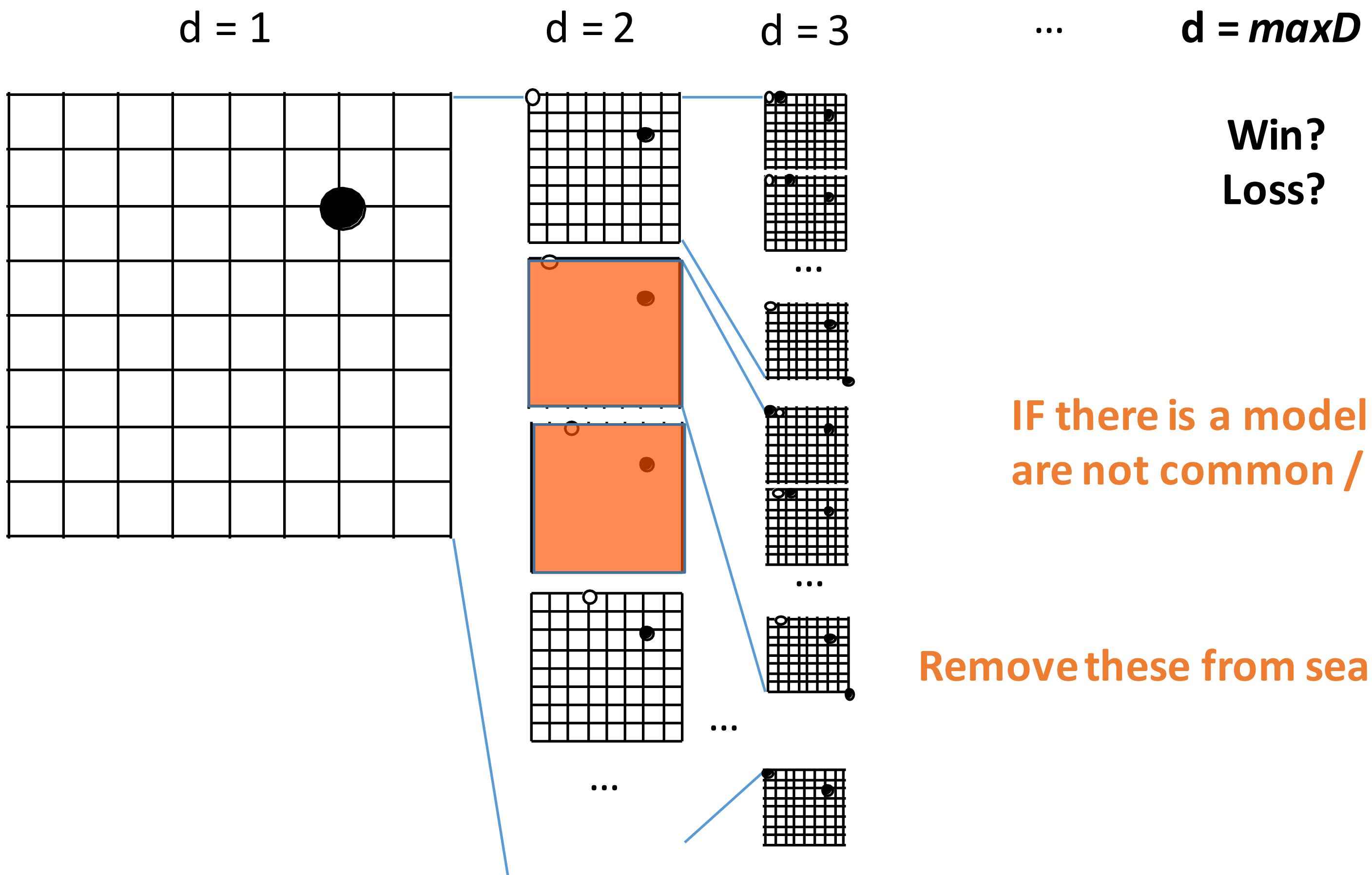




Key: To Reduce Search Space !!!

Reducing Search Space

1. Reducing “action candidates” (Breadth Reduction)

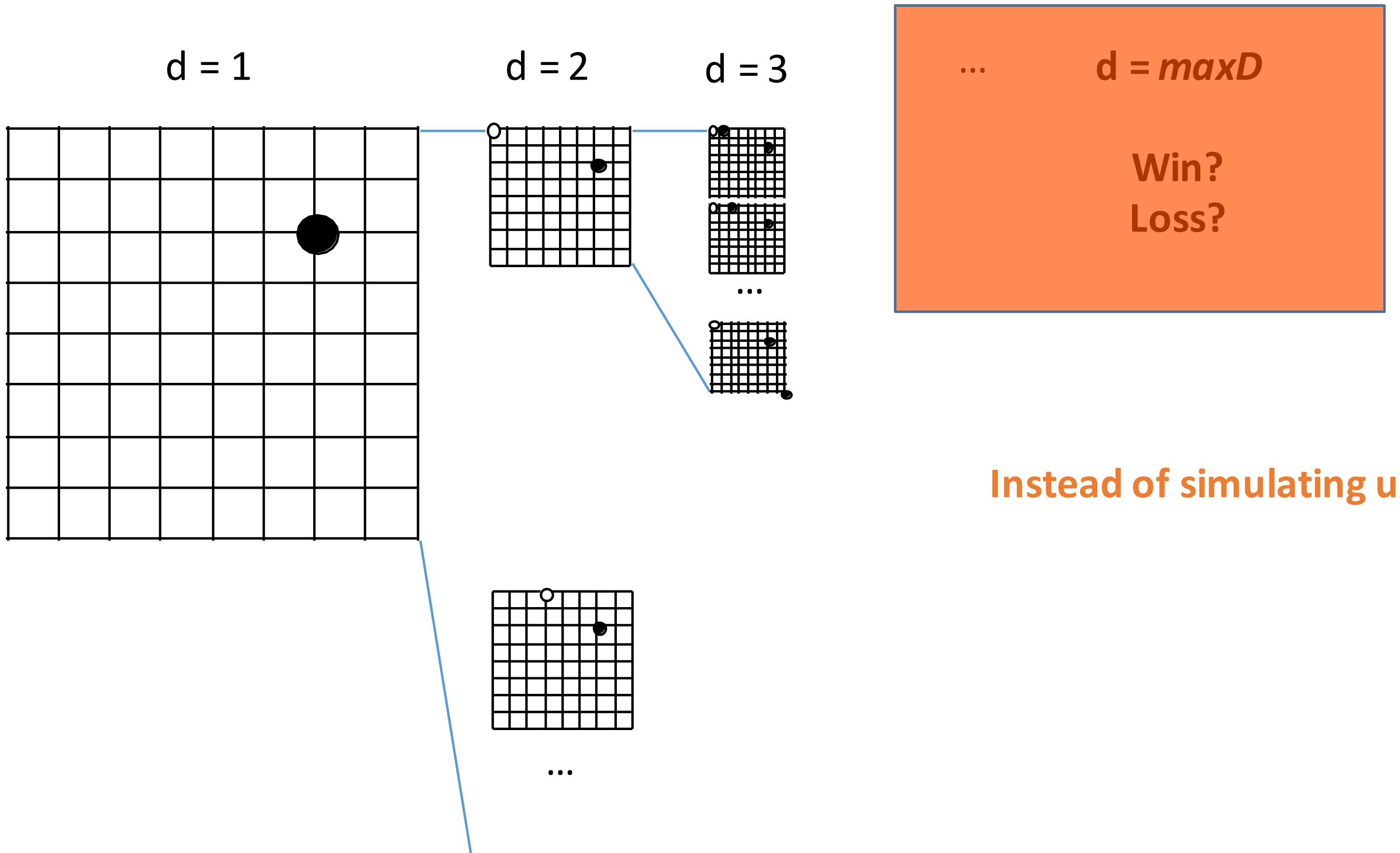


IF there is a model that can tell you that these moves are not common / probable (e.g. by experts, etc.) …

Remove these from search candidates in advance (breadth reduction)

Reducing Search Space

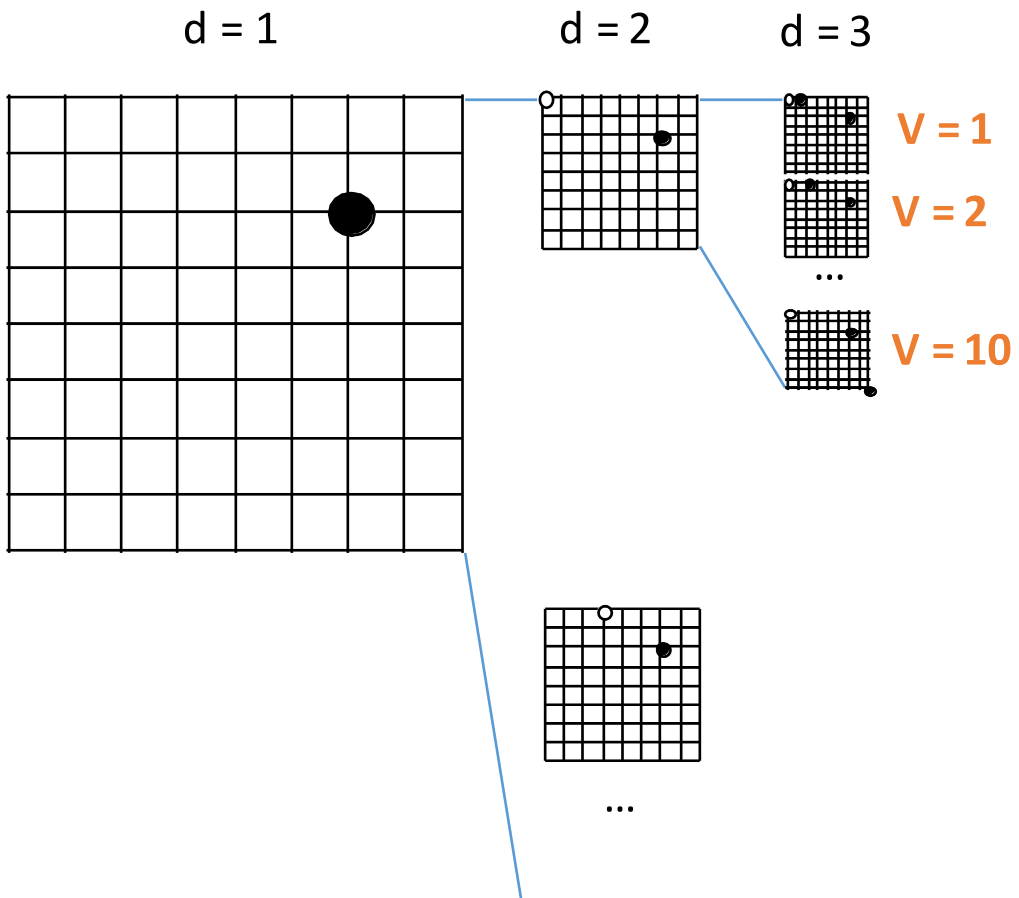
2. Position evaluation ahead of time (Depth Reduction)



Instead of simulating until the maximum depth..

Reducing Search Space

2. Position evaluation ahead of time (Depth Reduction)



IF there is a function that can measure:

V(*s*): "board evaluation of state *s*"

Reducing Search Space

1. **Reducing “action candidates” (Breadth Reduction)**

2. **Position evaluation ahead of time (Depth Reduction)**

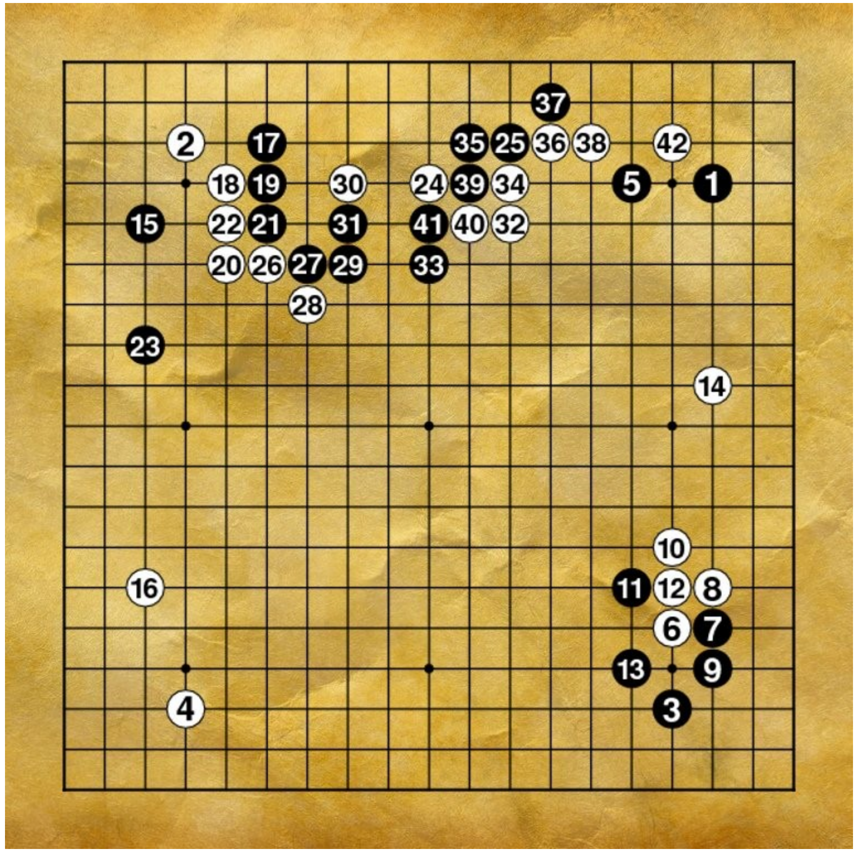
1. Reducing “action candidates”

Learning: **P (next action | current state)**

= P (a | s)

1. Reducing “action candidates”

(1) Imitating expert moves (supervised learning)



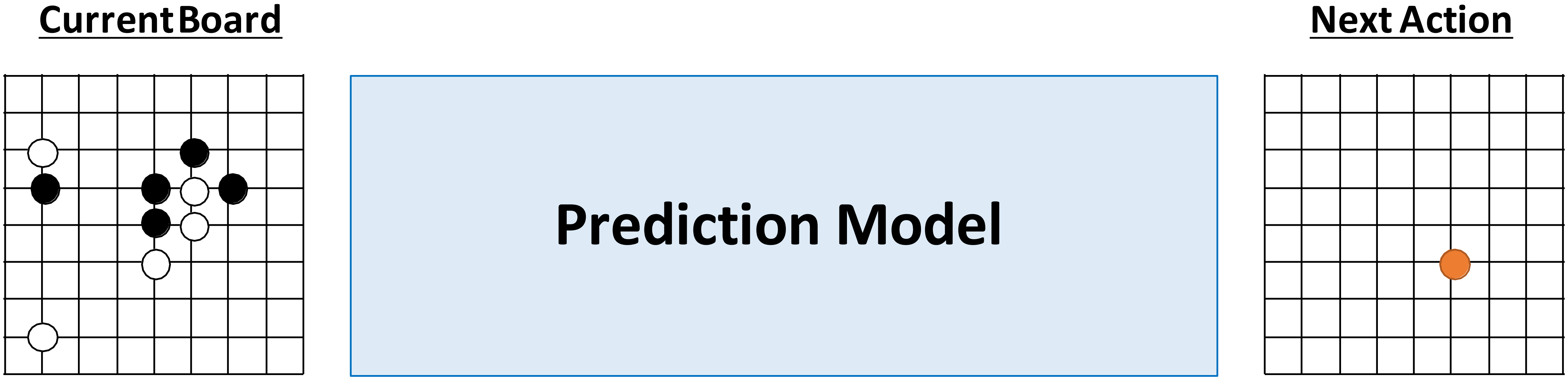
Current State		**Next Action**
s1	**Prediction Model**	a1
s2		a2
s3		a3

Data: Online Go experts (5~9 dan) (KGS)

160K games, 30M board positions

1. Reducing “action candidates”

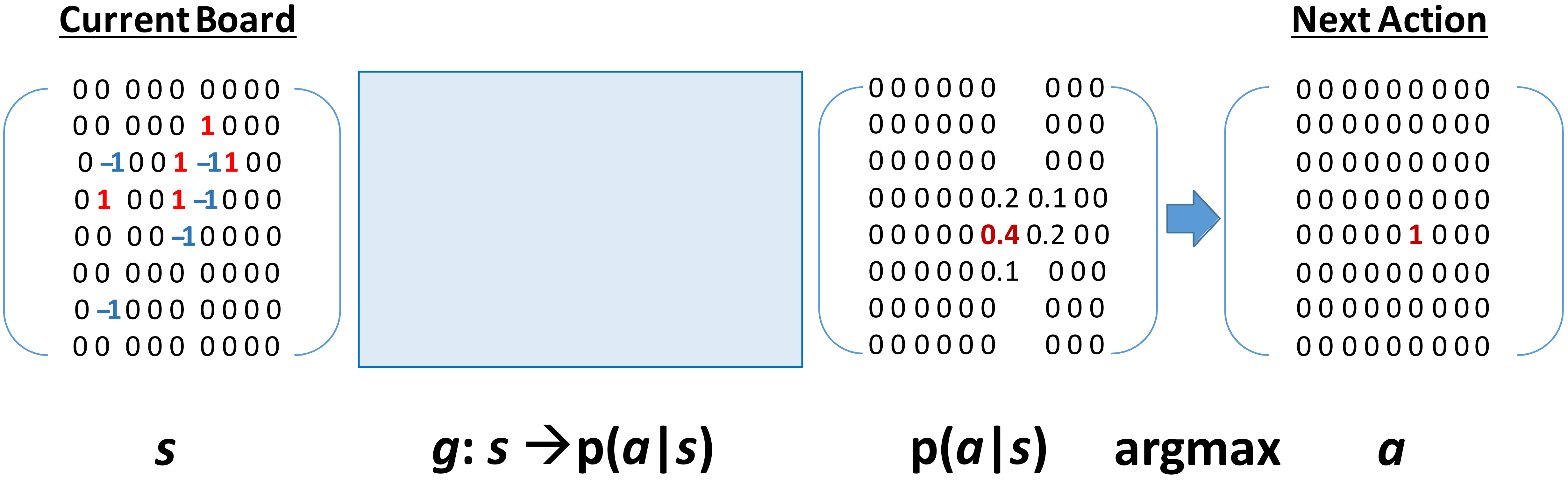
(1) Imitating expert moves (supervised learning)



There are 19 X 19 = 361
possible actions
(with different probabilities)

1. Reducing “action candidates”

(1) Imitating expert moves (supervised learning)



1. Reducing "action candidates"

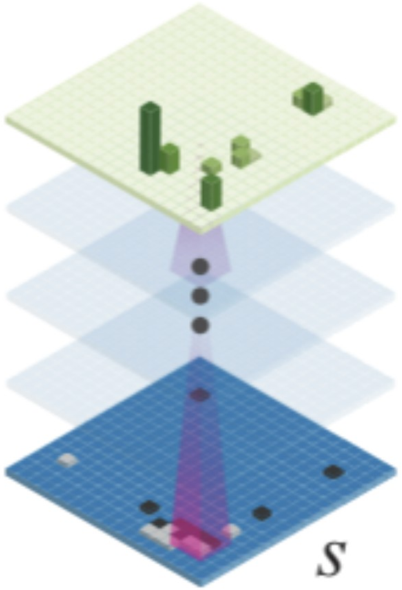
(1) Imitating expert moves (supervised learning)

Current Board

$$
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & -1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

Supervised Learning Policy

Prediction Model

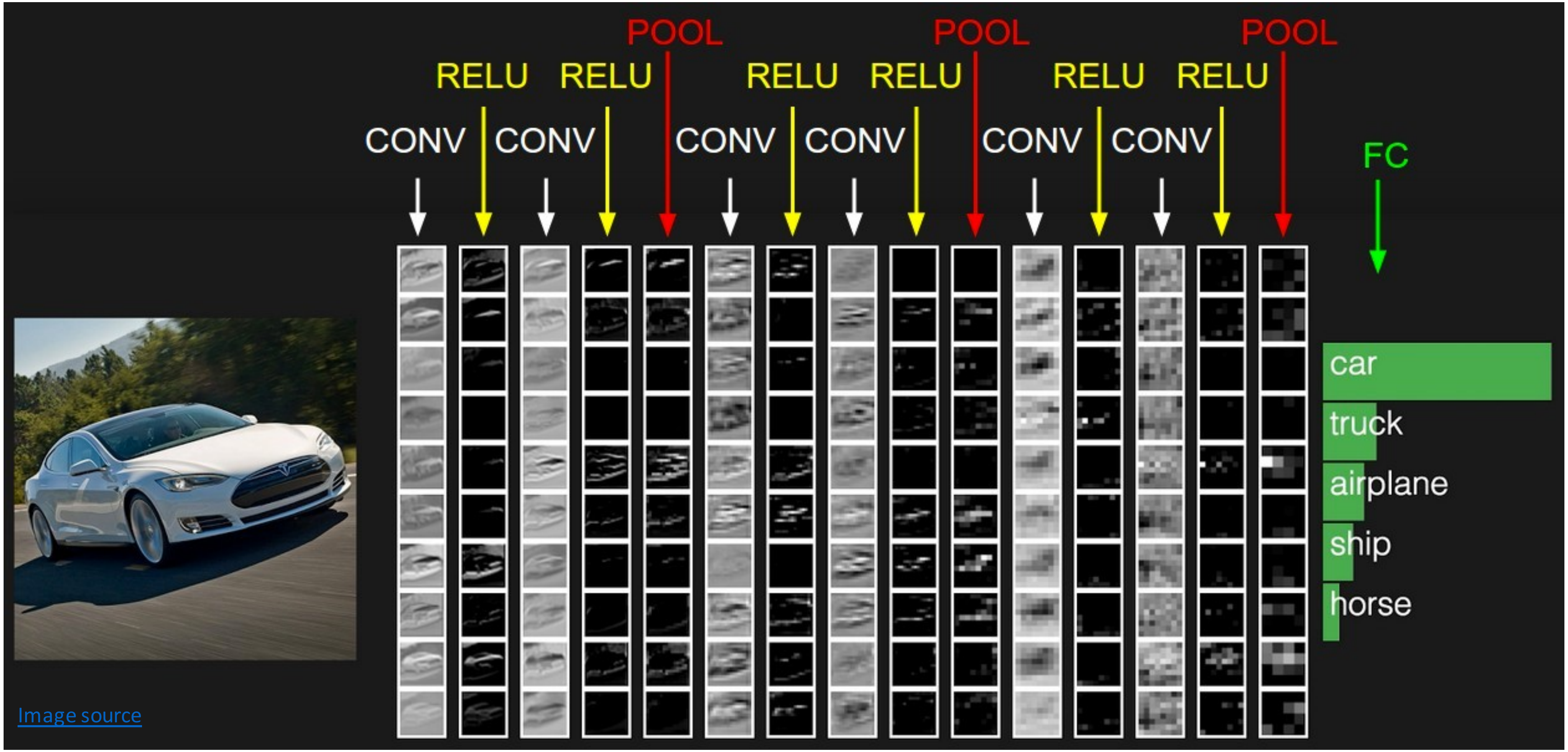


Next Action

$$
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

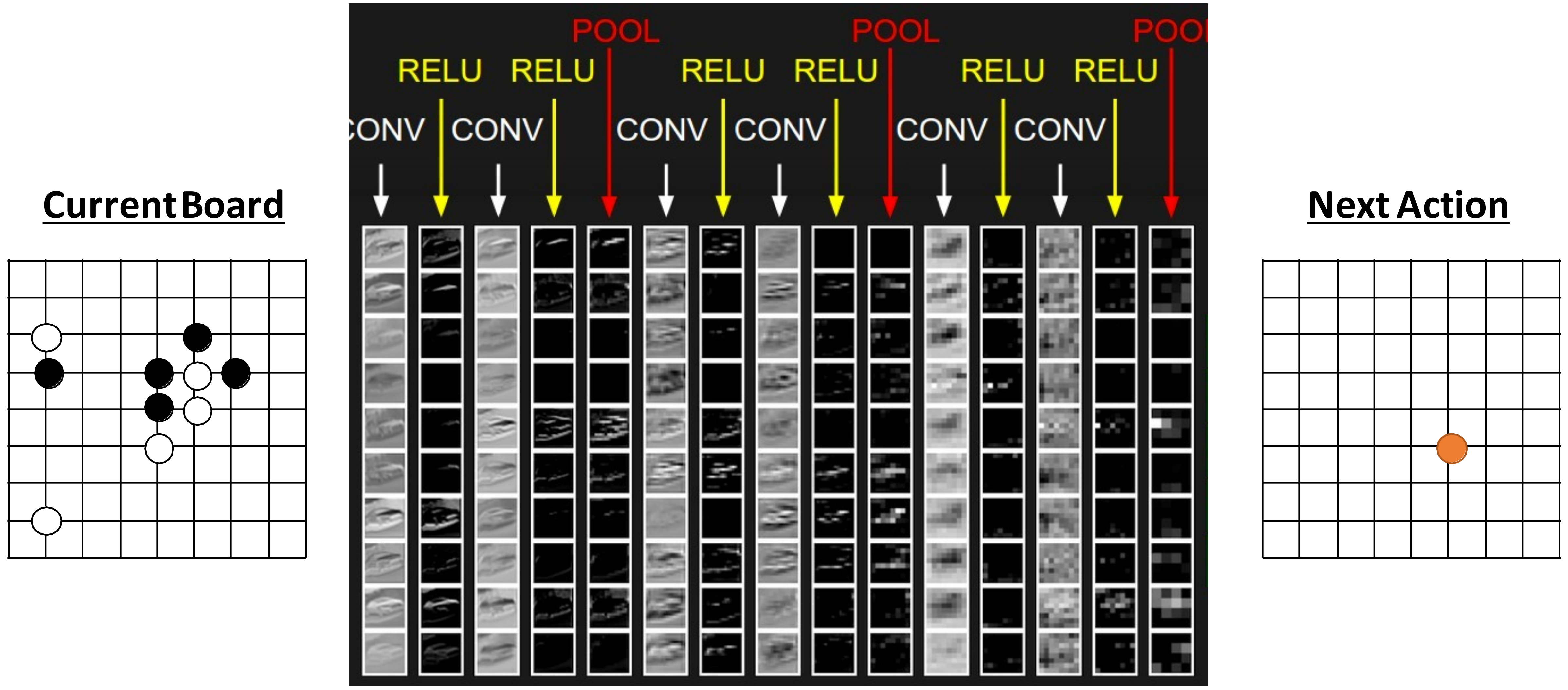
s $\quad$ $g: s \rightarrow \mathrm{p}(a|s)$ $\quad$ $\mathrm{p}(a|s)$ $\quad$ **argmax** $\quad$ a

Convolutional Neural Network (CNN)



CNN is a powerful model for image recognition tasks; it abstracts out the input image through convolution layers

Convolutional Neural Network (CNN)

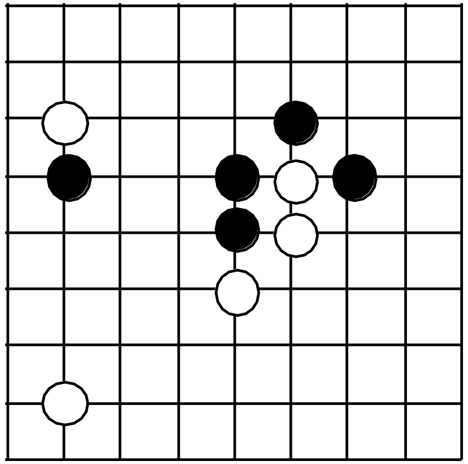


And they use this CNN model (similar architecture) to evaluate the board position; *which learns "some" spatial invariance*

1. Reducing "action candidates"

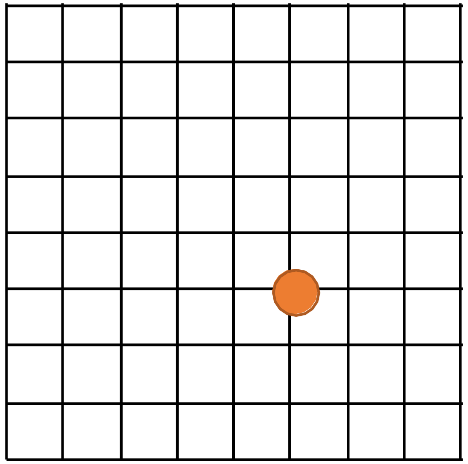
(1) Imitating expert moves (supervised learning)

Current Board



Expert Moves Imitator Model (w/ CNN)

Next Action

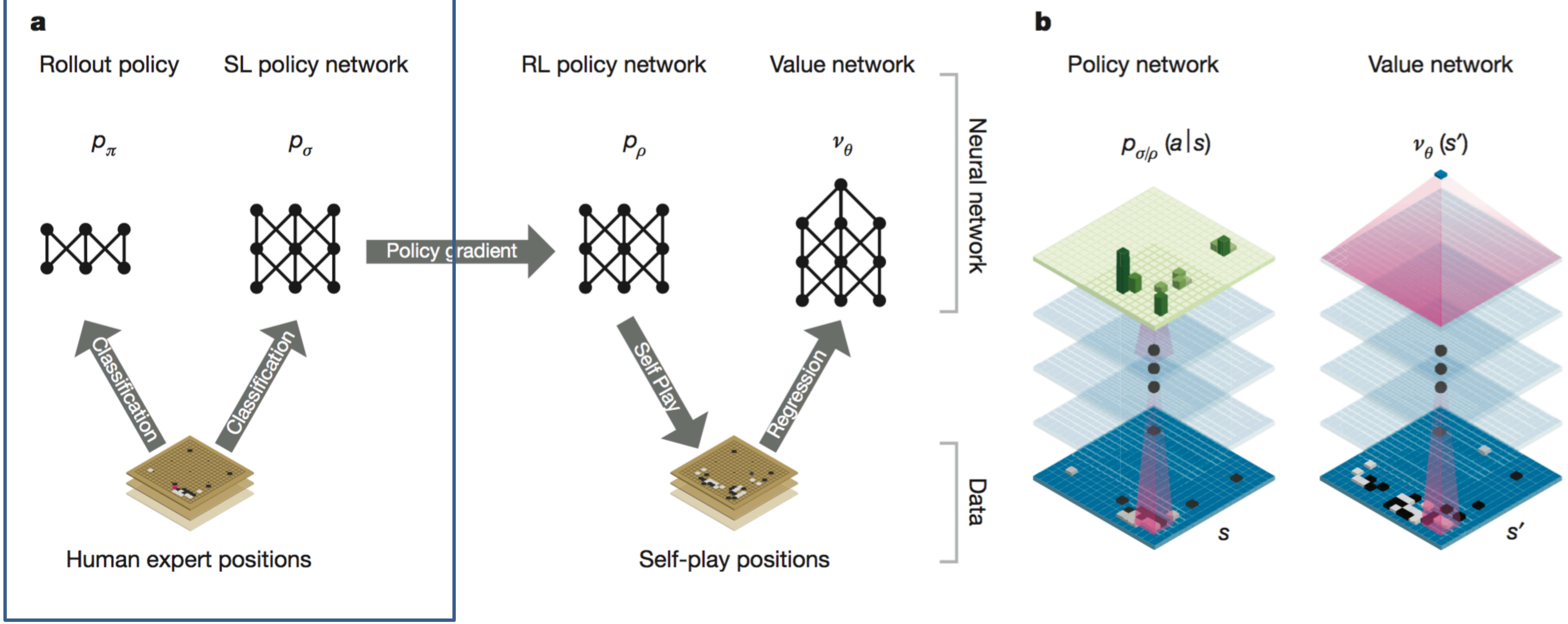


50 GPUs, 3 week

Training: $\Delta\sigma \propto \dfrac{\partial \log p_\sigma(a|s)}{\partial \sigma}$

ACC: 57%

AlphaGo


a
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
Neural network
Data
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s'

1. Reducing "action candidates"

(2) Improving through self--plays (reinforcement learning)

Improving by playing against itself

Expert Moves Imitator Model (w/ CNN)	vs	Expert Moves Imitator Model (w/ CNN)

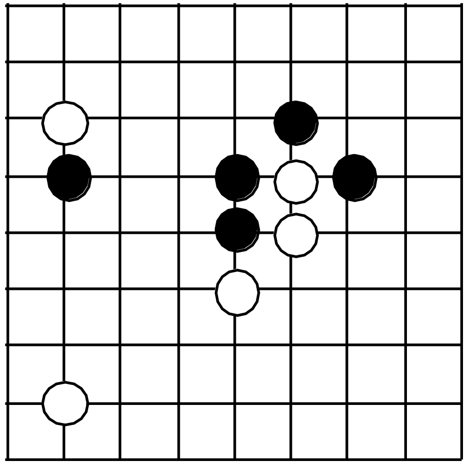
Goal shift:

Imitation -> winning !!

1. Reducing “action candidates”

(2) Improving through self--plays (reinforcement learning)

<u>Board position</u>



Expert Moves Imitator Model (w/ CNN)

<u>win/loss</u>

Loss

z = –1

Win

z = +1

t

Training: $\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$.

1. Reducing "action candidates"

(2) Improving through self--plays (reinforcement learning)

Older models vs. newer models

Model ver 1.1	**VS**	**Updated Model ver 1.3**

It uses the same topology as the expert moves imitator model, and just uses the updated parameters

→ **Return**: board positions, win/lose info

1. Reducing “action candidates”

(2) Improving through self--plays (reinforcement learning)

Model ver 1.5	**VS**	**Updated Model ver 2.0**

Return: board positions, win/lose info

1. Reducing "action candidates"

(2) Improving through self-plays (reinforcement learning)

Expert Moves Imitator Model

VS

Most Updated Model

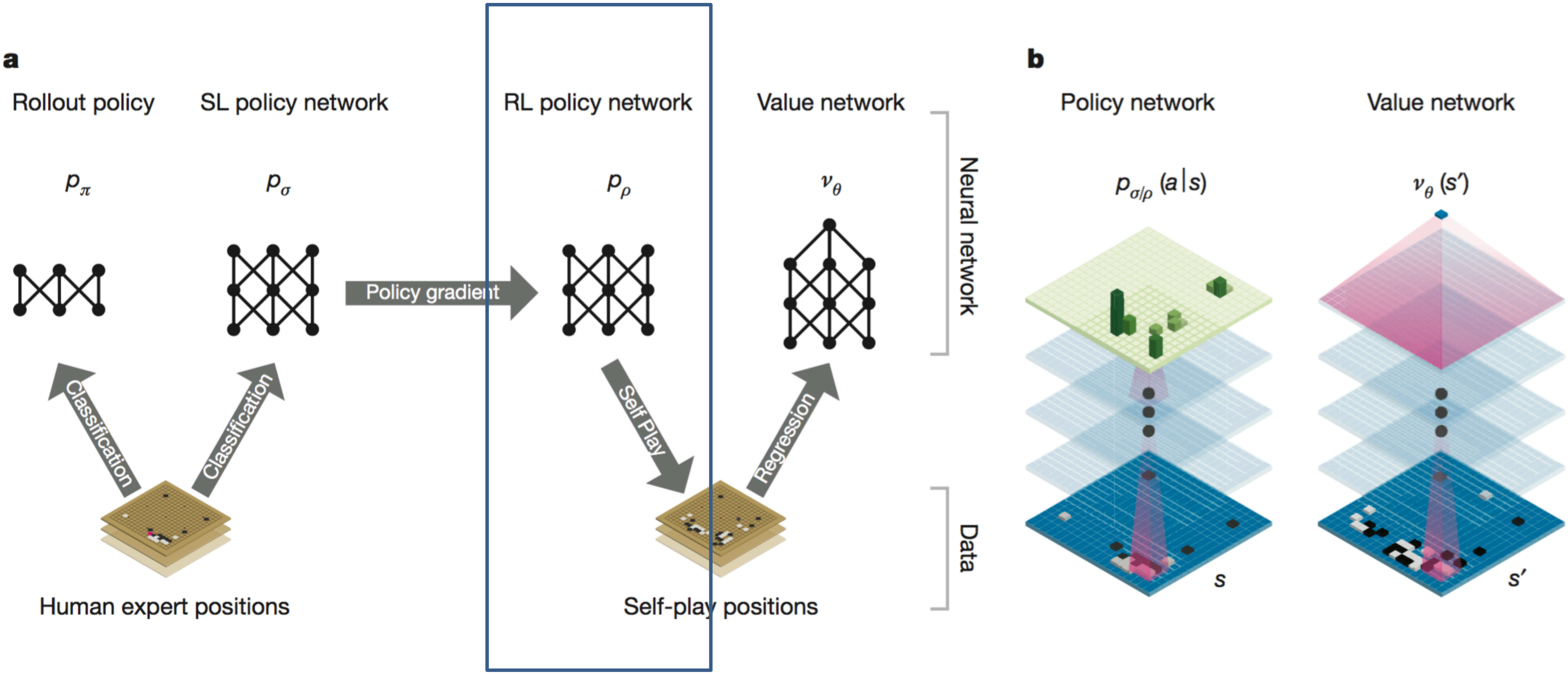
Supervised Learning Policy

Reinforcement Learning Policy

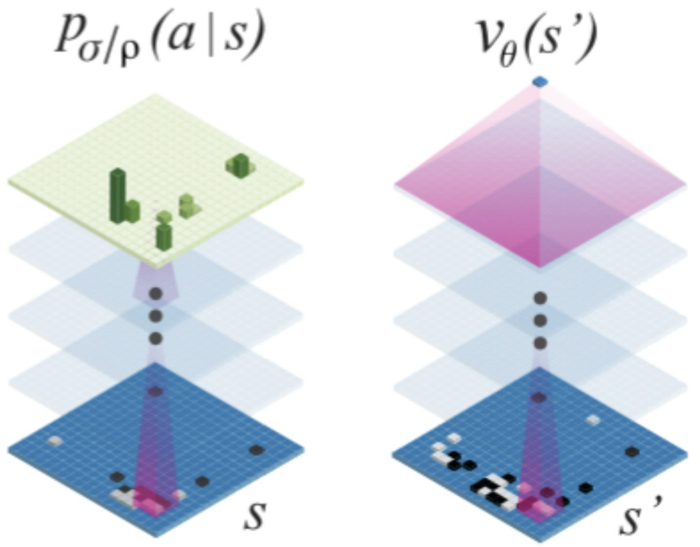
The final model wins 80% of the time when playing against the first model

50 GPUs, 1 day

AlphaGo


a
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
Neural network
Data
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s'

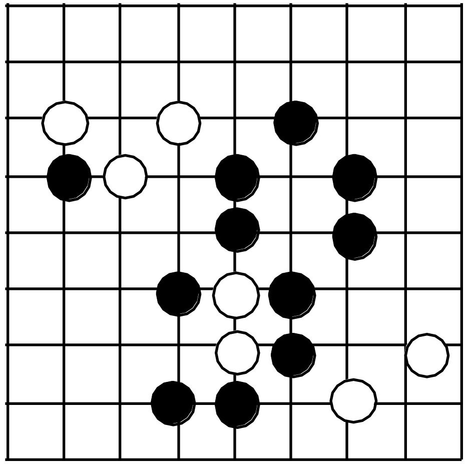
2. Board Evaluation



Adds a regression layer to the model
Predicts values between 0~1
Close to 1: a good board position
Close to 0: a bad board position

<u>Board Position</u>

<u>Win / Loss</u>



Updated Model ver 1,000,000

Value Prediction Model (Regression)

Win (0~1)

2. Board Evaluation

30 Million Positions

Uniq Position for each game

Random Once

SL Policy Sampling

RL Policy Sampling

Win/loss

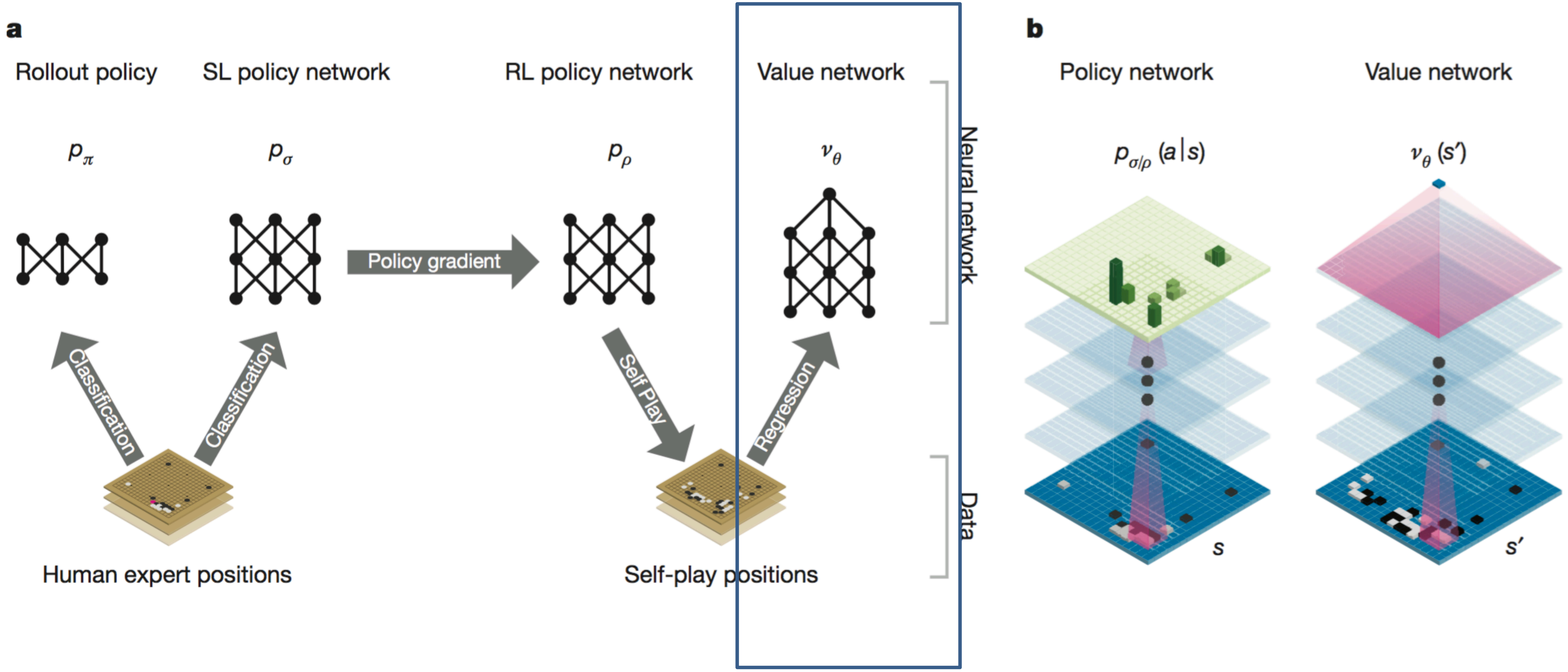
Start

U -1

(U+1) State: s

50 GPUs, one week **Training:** $$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

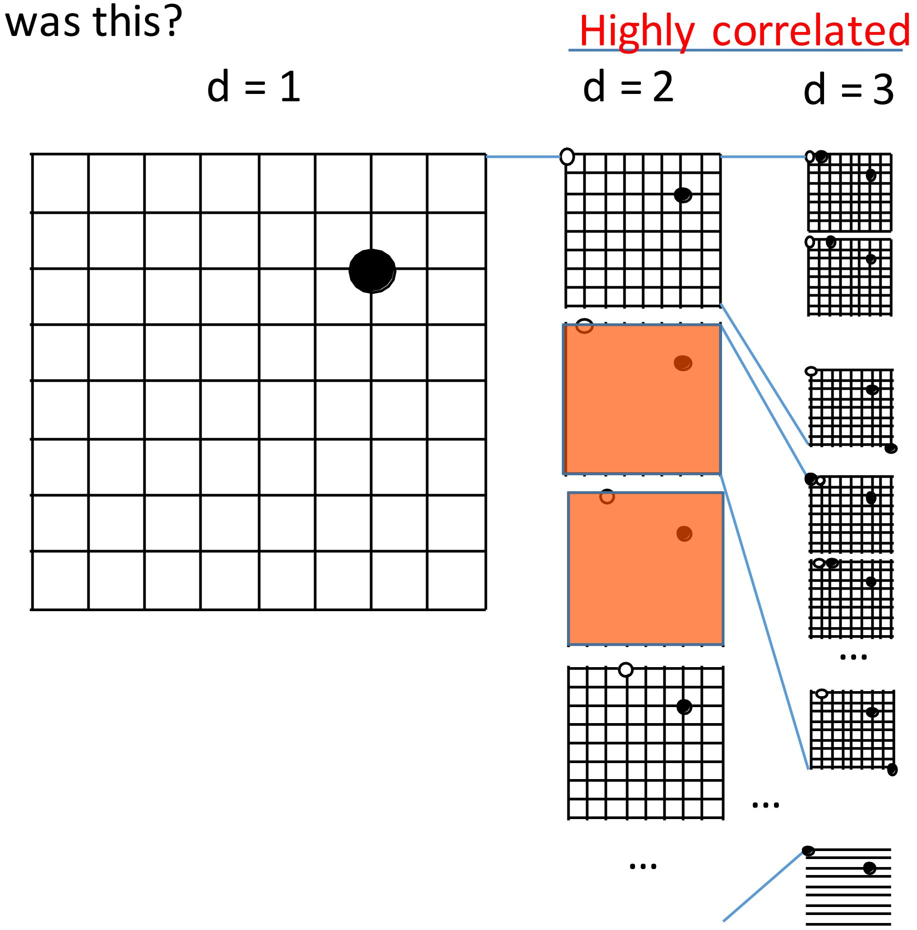
AlphaGo



a
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
Neural network
Data
Human expert positions
Self-play positions
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s'

In-class Question

Q(2). To train the value network, a set of distinct positions, each from a different game, was constructed. The policy network is then played against itself from that position to get an estimate of the value of that state. The learning update is then only applied to the initial state from which the play started, as no updates are made to states in the rest of the game. Why was this?



Over-represents

Reducing Search Space

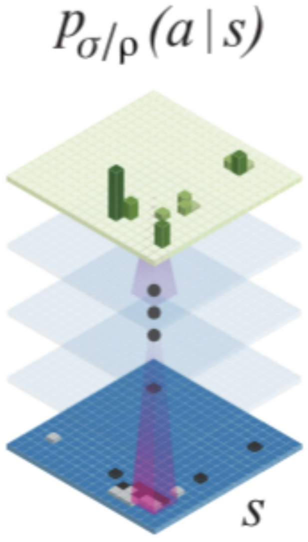
1. **Reducing "action candidates" (Breadth Reduction)**

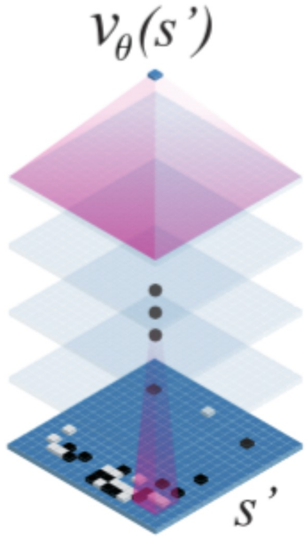
   Policy Network

   Reinforcement learning policy + (rollout policy, simple model to speed up the sampling)

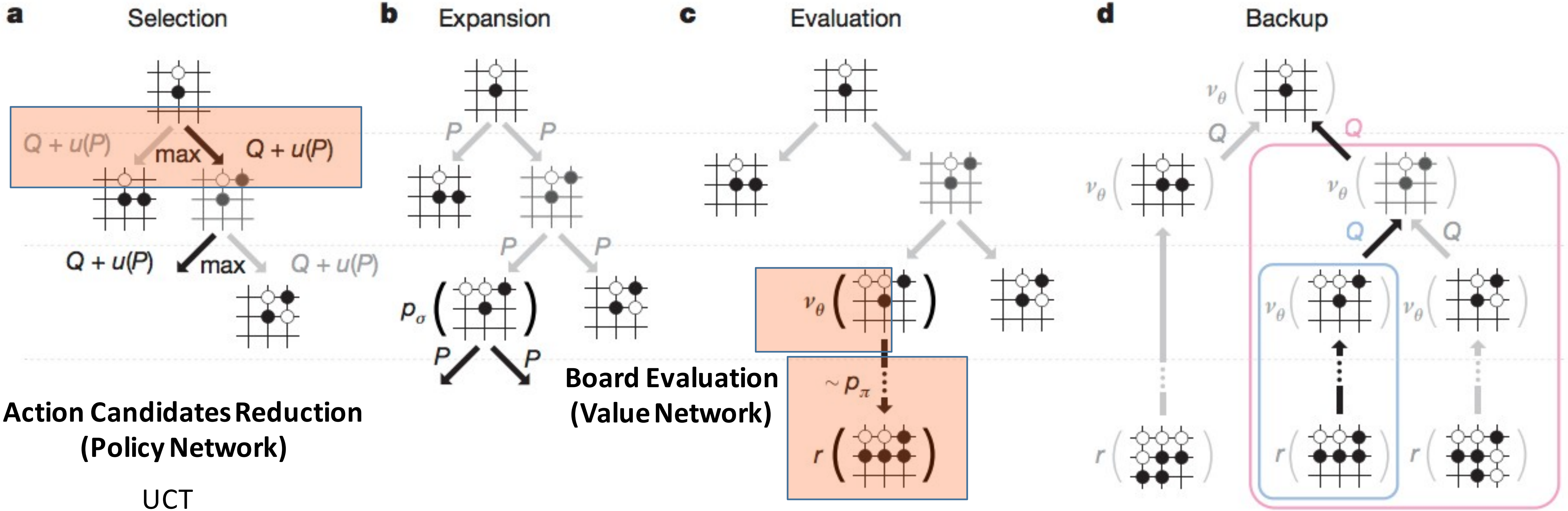
2. **Board Evaluation (Depth Reduction)**

   Value Network





Looking ahead (w/ Monte Carlo Search Tree) + UCT



Action Candidates Reduction (Policy Network)

UCT

Board Evaluation (Value Network)

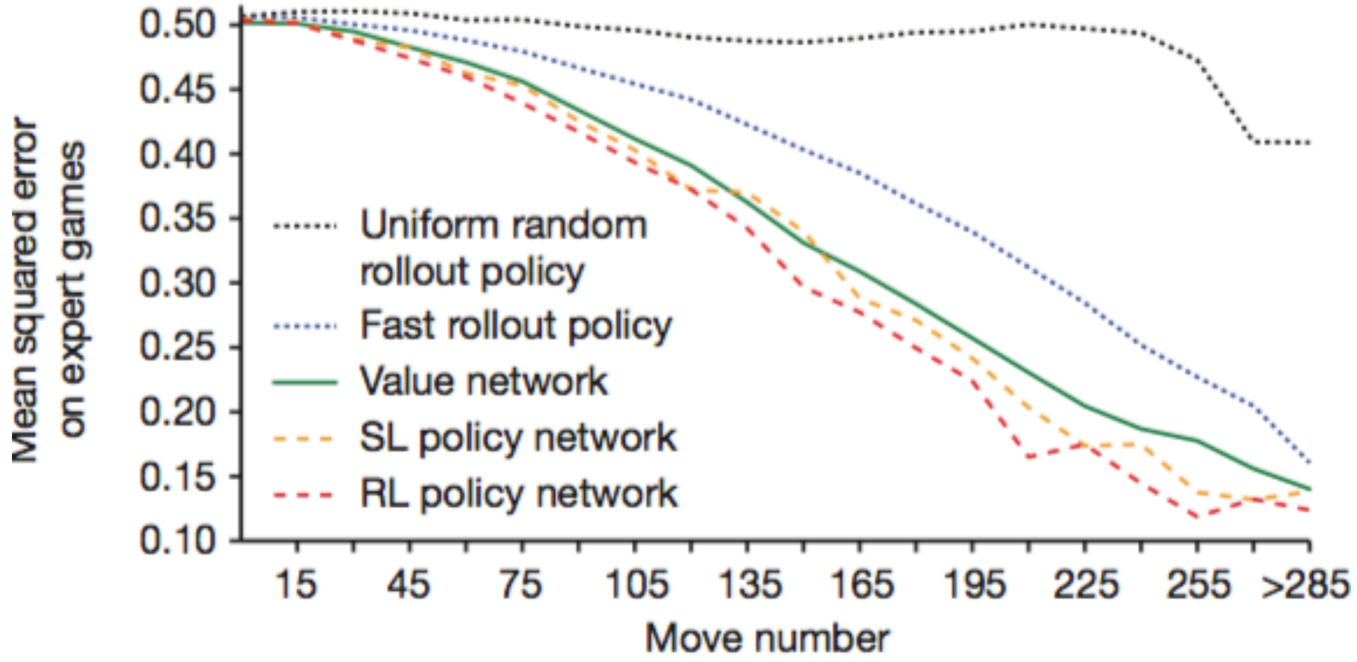
(Rollout): Faster version of estimating p(a|s)

→uses shallow networks (3 ms →2µs)

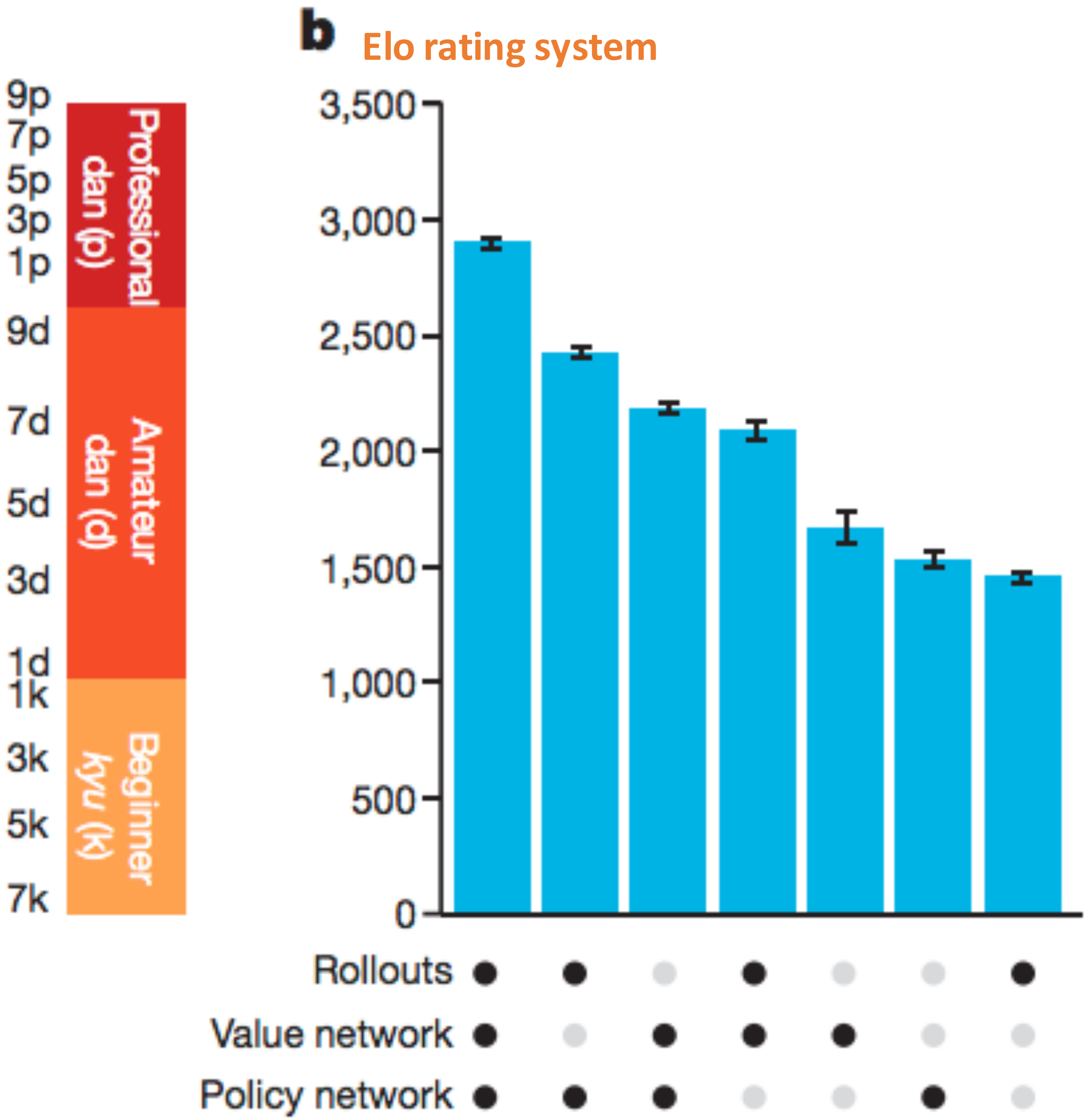
In-class Question

Q(1). For AlphaGo, two policies are learned directly from expert games, one using a deep network, and the other a linear approximator. The linear approximator, which was used to generate moves during the playouts, is much faster to compute but is less accurate at predicting good moves. Explain why they might have made the decision to use the less accurate policy? In addition, note that they could have uniformly selected moves from all possible moves, which would have been even faster to computer. Discuss what this suggests about how we should develop policies used to generate playouts in Monte Carlo Tree Search.

2 us VS 3 ms



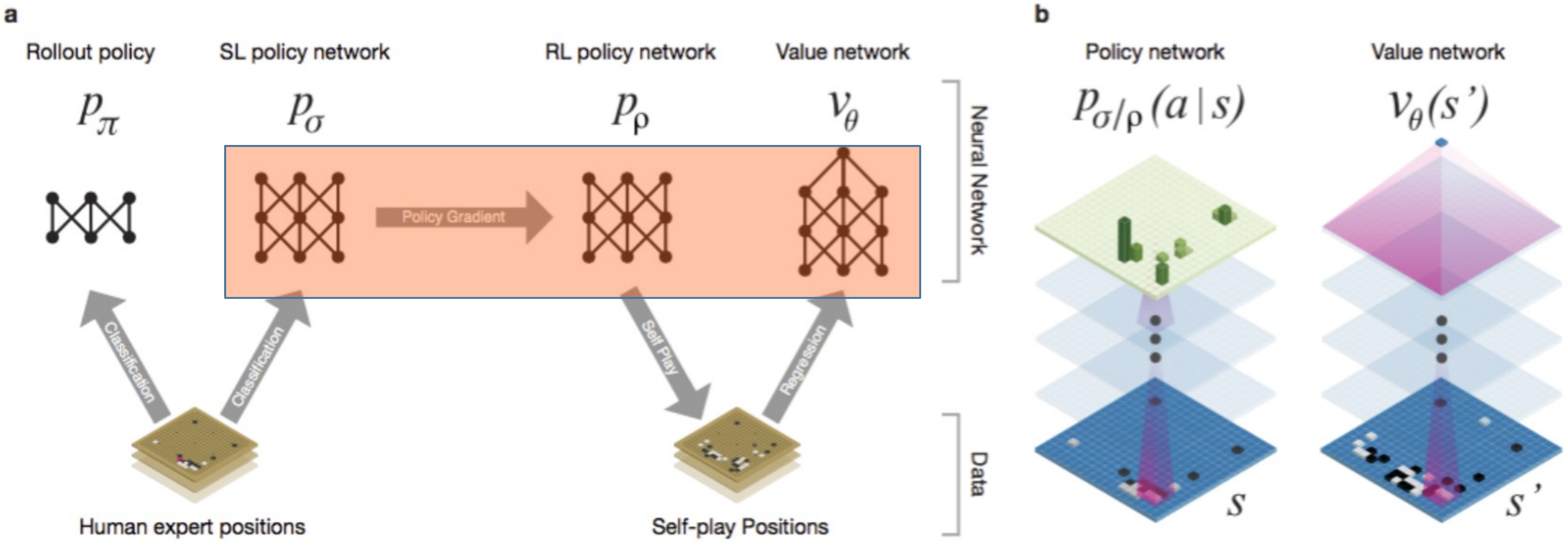
Results



Performance with different combinations of AlphaGo components

Takeaways

Use the networks trained for a certain task (with different loss objectives) for several other tasks

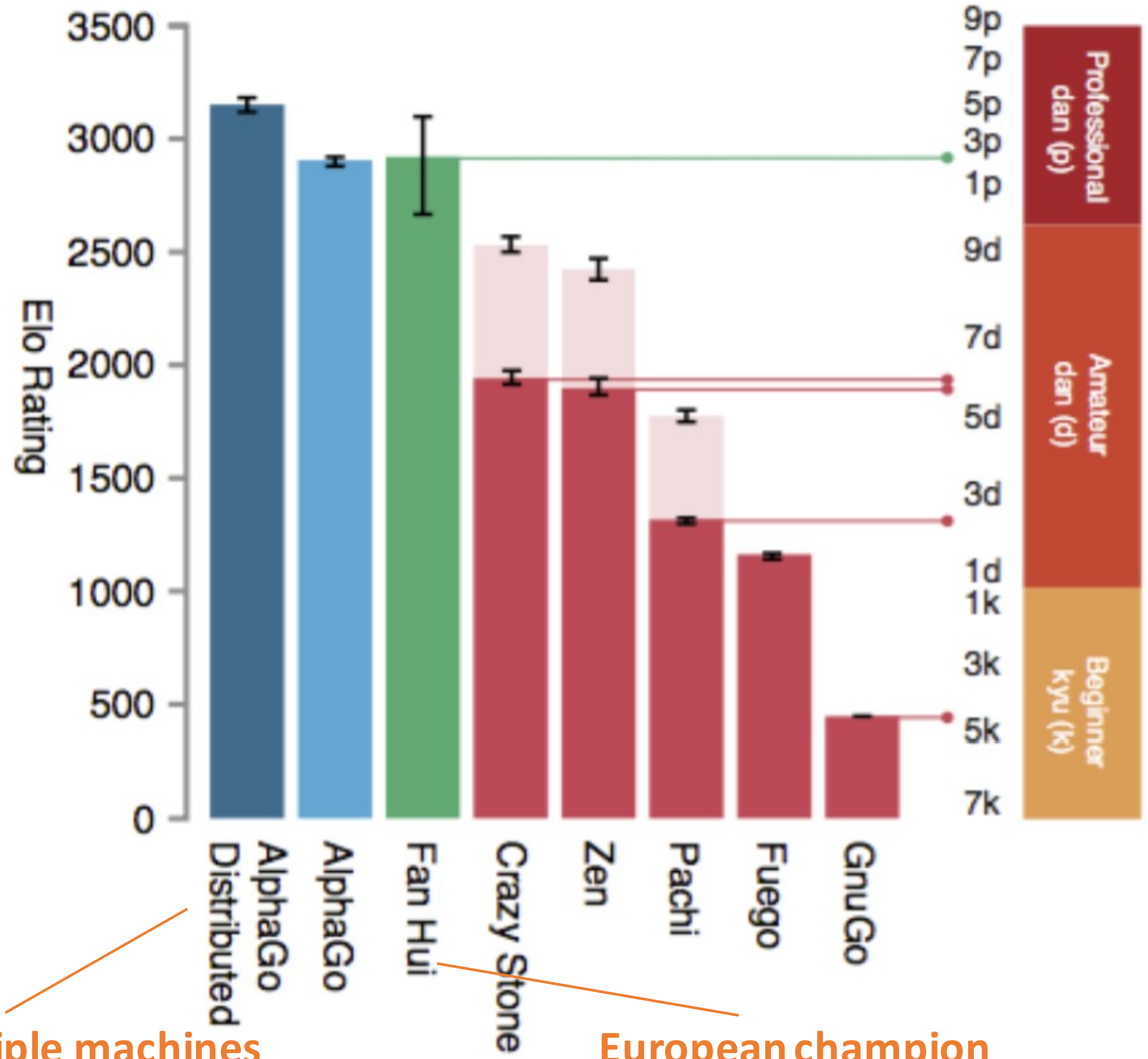


Lee Sedol vs AlphaGo Energy Consumption

Lee Sedol	AlphaGo
– Recommended calories for a man per day : ~2,500 cal – Assumption: Lee consumes the entire amount of per--day calories in this one game 2,500 cal * 4.184 J/cal ~= 10k [J]	– Assumption: CPU: ~100 W, GPU: ~300 W – **1,202** CPUs, **176** GPUs 170,000 J/sec * 5 hr * 3,600 sec/hr ~= 3,000M [J] = 300 k Lee

A very, very rough calculation ;)

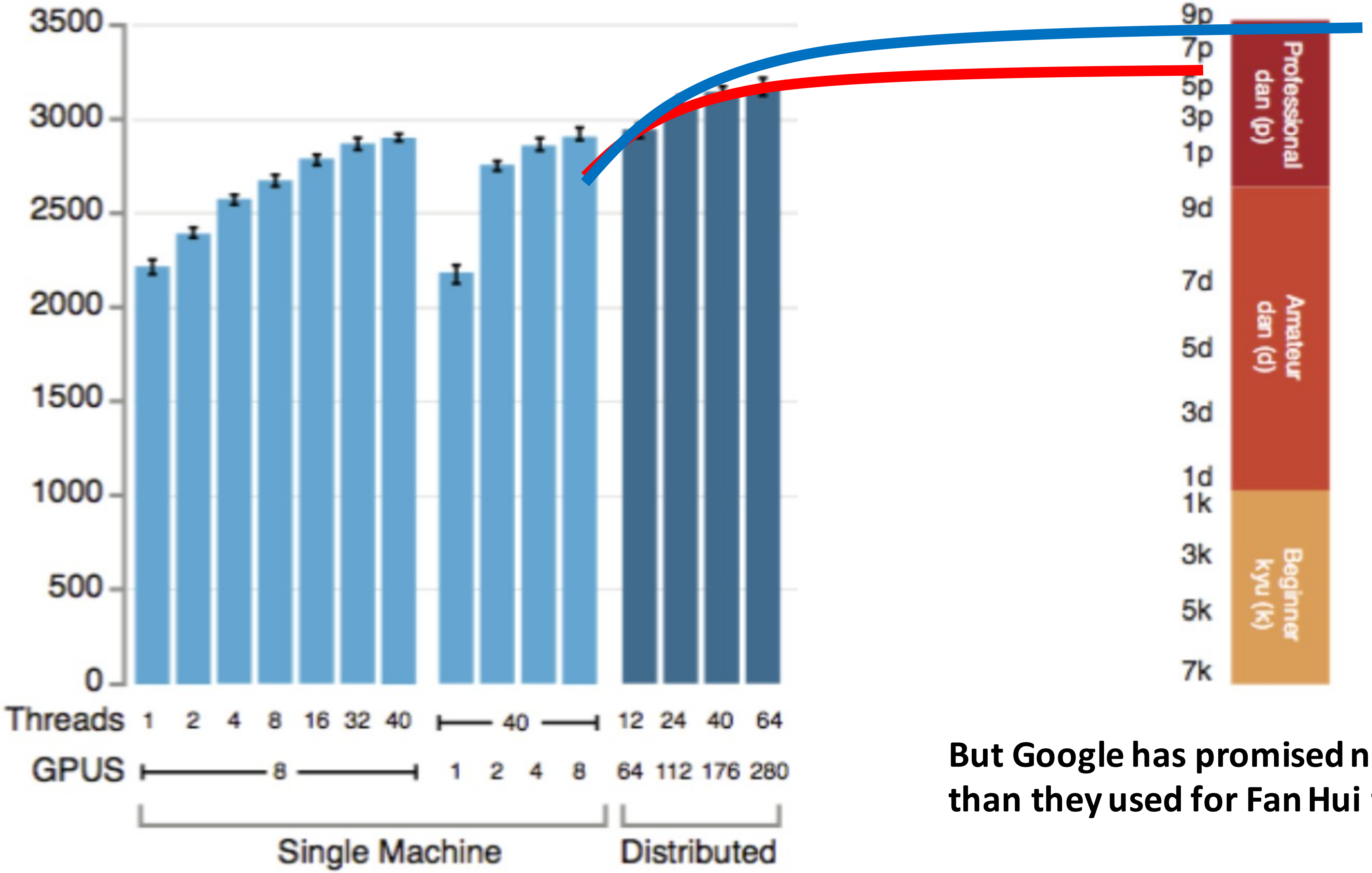
AlphaGo is estimated to be around ~5–dan



= multiple machines

European champion

Taking CPU / GPU resources to virtually infinity?



No one knows
how it will converge

But Google has promised not to use more CPU/GPUs than they used for Fan Hui for the game with Lee

AlphaGo learns millions of Go games everyday

AlphaGo will presumably converge to some point eventually.

However, in the Nature paper they don’t report how AlphaGo’s performance improves as a function of times AlphaGo plays against itself (self--plays).

Conclusion

First Time

Computer can defeat professional player in Go

Reference

- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016): 484--489.