Man vs. Machine In Weiqi

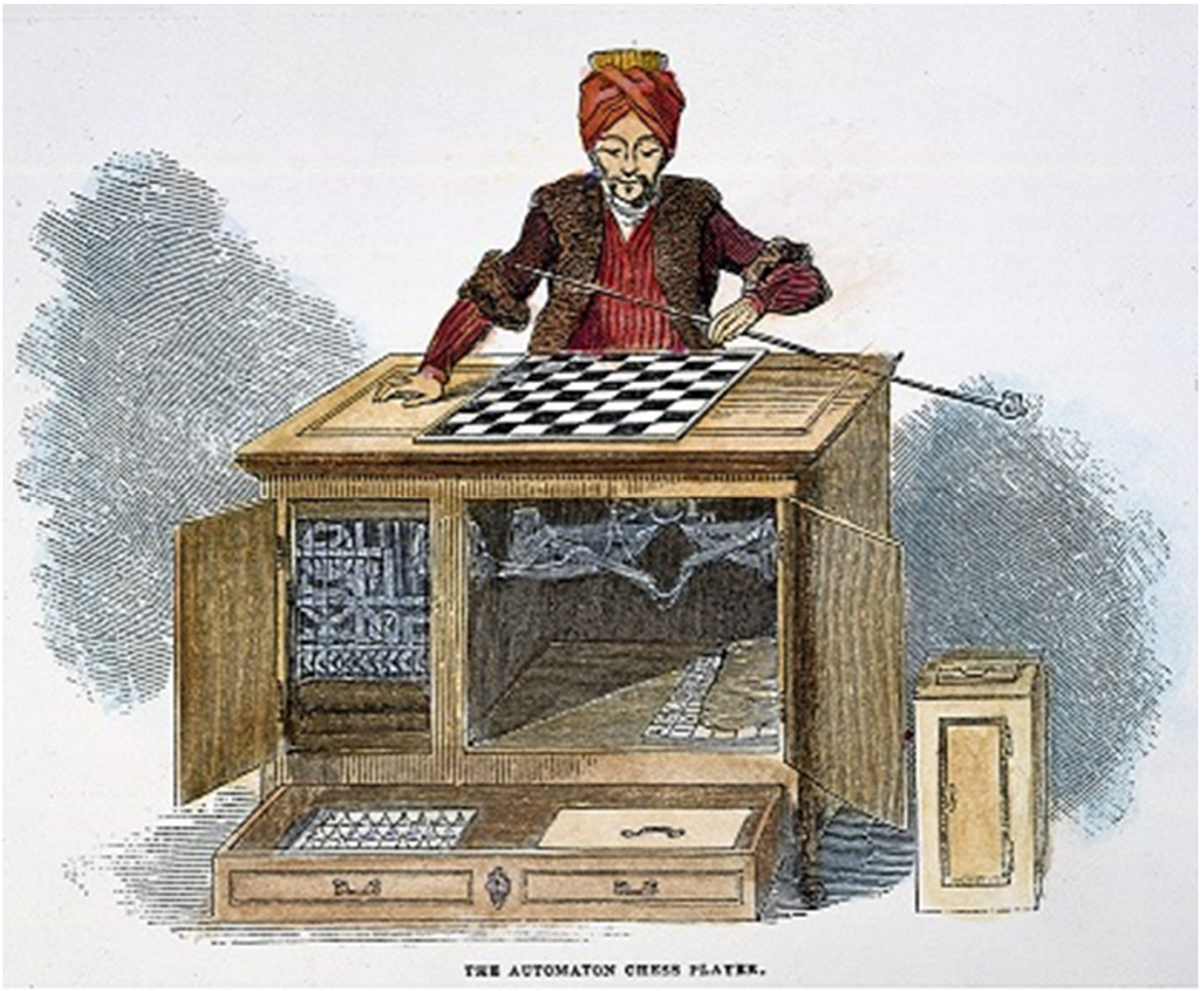
Recent development in Weiqi Machines (AlphaGo)

Li CHENG

31Mar16

A Tale about the Chess Machine

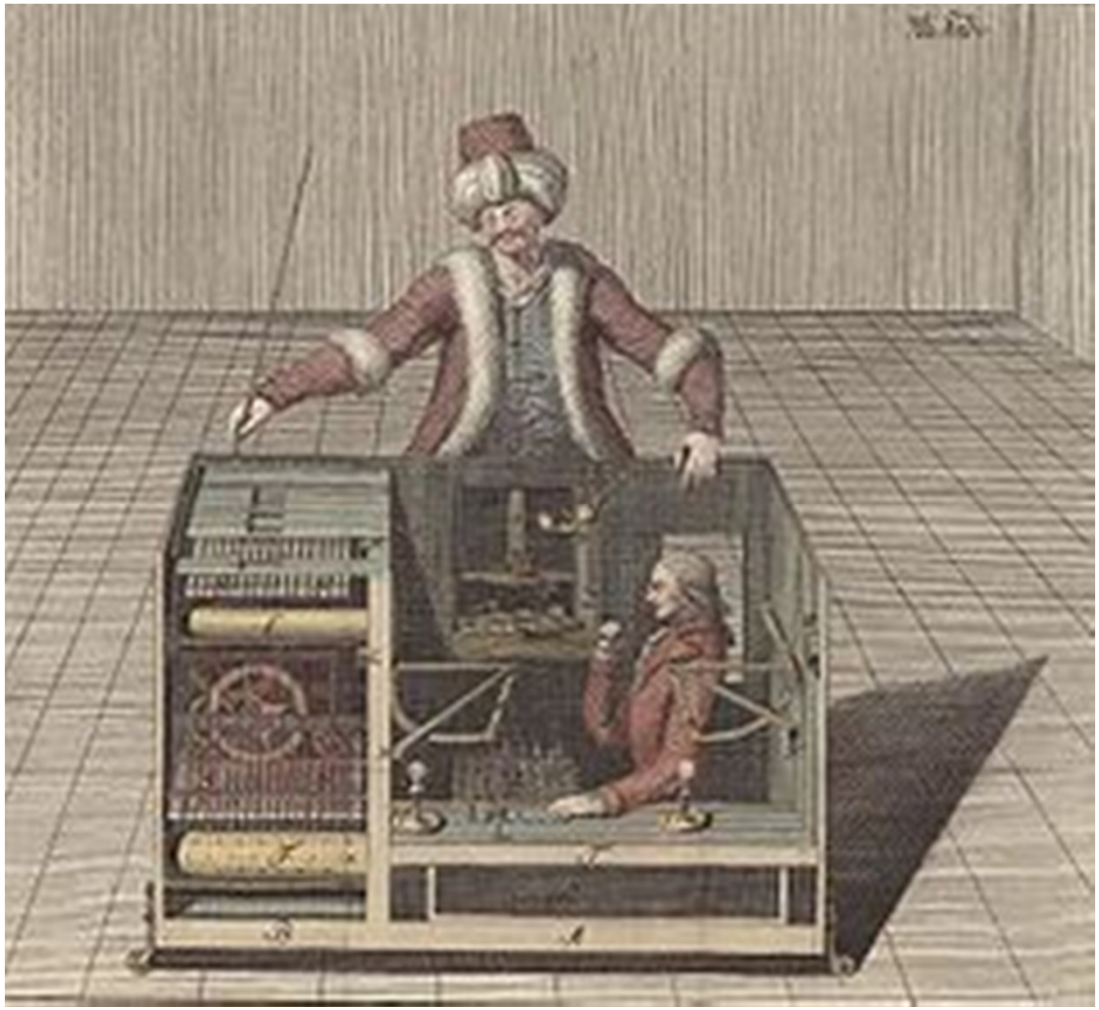
- Chess 1770



Wolfgang von Kempelen, “the Turk”

A Tale about the Chess Machine

- Chess 1770



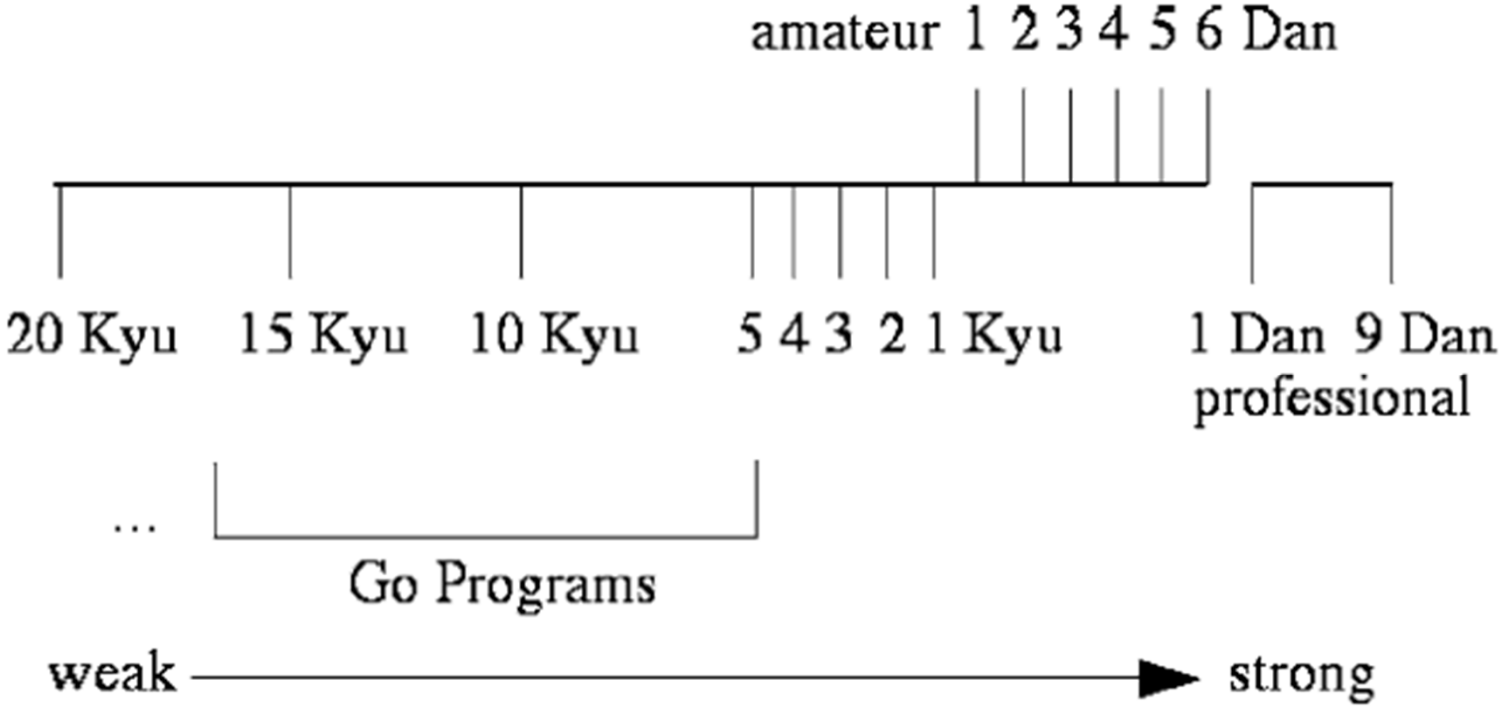
Wolfgang von Kempelen, “the Turk”

The REAL Chess Machine

- Deep Blue vs. Garry Kasparov 1997, 3.5:2.5

Weiqi

Weiqi


amateur 1 2 3 4 5 6 Dan
20 Kyu
15 Kyu
10 Kyu
5 4 3 2 1 Kyu
1 Dan 9 Dan
professional
...
Go Programs
weak
strong

Weiqi

Rank	Name	♂♀	Flag	Elo
1	Ke Jie	♂		3615
2	Google AlphaGo			3584
3	Park Jungwhan	♂		3551
4	Iyama Yuta	♂		3534
5	Lee Sedol	♂		3518
6	Shi Yue	♂		3512
7	Kang Dongyun	♂		3506
8	Park Yeonghun	♂		3506
9	Mi Yuting	♂		3504
10	Kim Jiseok	♂		3498

Elo-like rating From (http://www.goratings.org/)

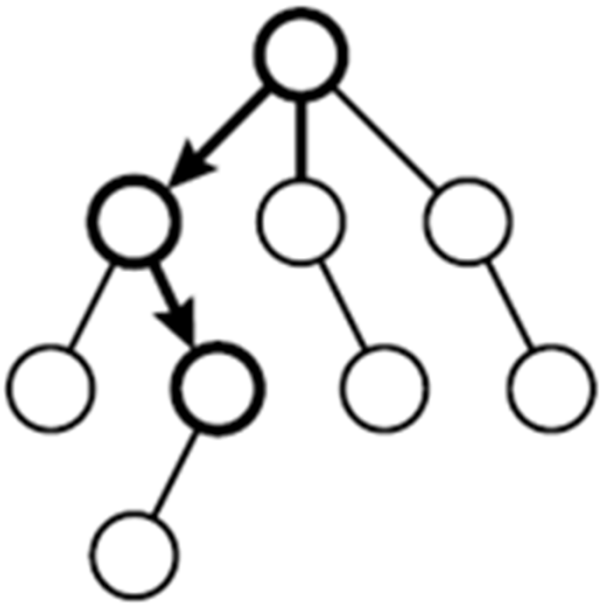
Challenges of Weiqi Machines

- Weiqi has 3^{361}(about 10^{170}) states vs. only about 10^{80} atoms in the entire universe
- Up to 361 legal moves
- Long-term effect of a move may only be revealed after hundreds of additional moves
- Last mind-sports game to be conquered

Game Theory of Weiqi Machines

- A two-player, perfect information, zero-sum game (weiqi, chess, checker, …)
- From MDP-> symmetric alternating Markov (SAM) game
- Two policies each for agent (black) and for opponent (white)
- Self-play policy: a policy used by both agent & opponent
- Minimax optimal policy for SAM game: a self play policy both maximizes agent's value function and minimized opponent's value function

Game Tree Search of Weiqi Machines



AlphaGo

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

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-

[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK. [2]Google, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA.
*These authors contributed equally to this work.

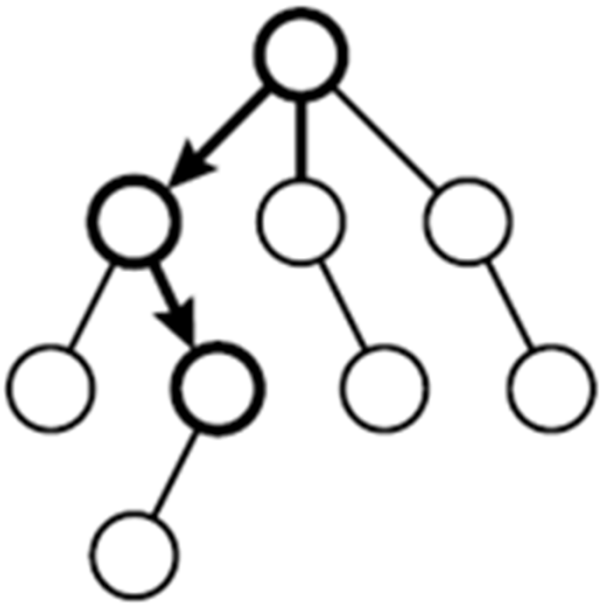
AlphaGo

- An engineering feat (about 2-year time)
- In fact, many underground efforts over many years
- AlphaGo vs. Lee Seedal (9-15 Mar 2016): 4-1, Huge success, a real earthquake in Weiqi community
- 17Mar16: South Korean govt. will invest 1 trillion won (US$0.86 Billion) in AI over next 5-year

Three Pillars of AlphaGo

- (Monte Carlo) Tree Search or MCTS
- Reinforcement Learning (learning by self-play)
- Deep Learning (aka Deep Convolutional Neural Network, or DCNN)

Sketch the Main Ideas



How AlphaGo Works

overview

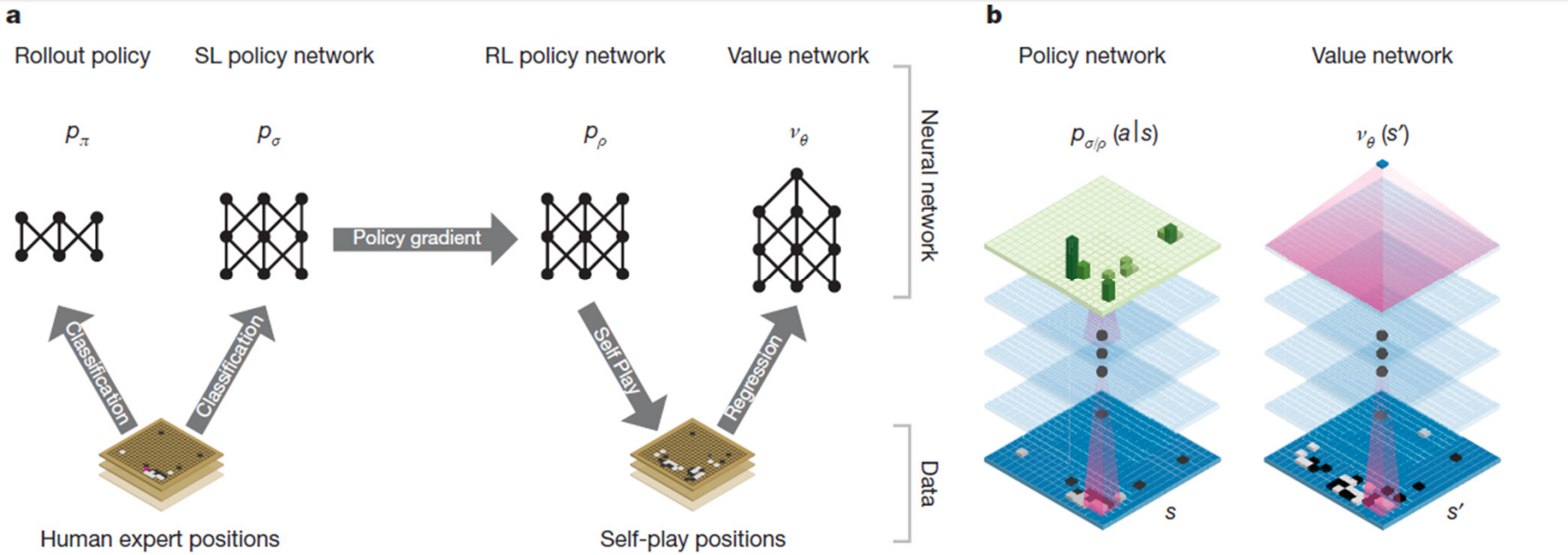


Figure 1 | Neural network training pipeline and architecture. **a**, A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set. **b**, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

From AlphaGo's 28Jan16 nature paper @ (http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html)

How AlphaGo Works

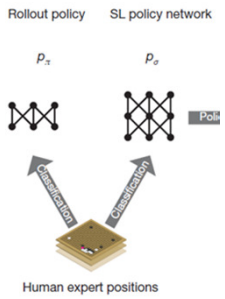
Supervised learning of policy networks

- The SL network is a 13-layer CNN trained from 30m positions from KGS server, to predict expert's move

- Final layer is a softmax layer outputs probability over all legal moves
- Trained on randomly sampled (s,a) pairs, using SG to max likelihood of selecting move a at state s:

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

- 57% prediction accuracy with all input features, and 55.7% with only board position and move history, vs. 44.4% from state-of-the-arts



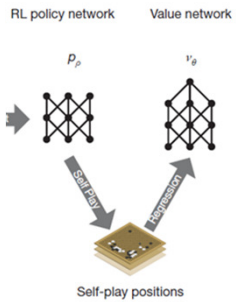
How AlphaGo Works

Reinforcement learning of policy networks

- The RL policy network p_ρ starts with SL policy network p_σ
- It learn by self-playing between itself and a randomly chosen previous version of itself
- The reward of curr state s r(s) is obtained by playing multiple games to end (roll-outs) and obtaining each time either +1 (black win) or -1
- Params of the policy at curr state is updated as

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

- Evaluation of RL policy network is by sampling each move as $a_t \sim p_\rho(\cdot|s_t)$ at curr state s_t .
- It won over SL policy network 80% of the games
- It won over Pachi (KGS 2D) 85% over 100,000 games



How AlphaGo Works

Reinforcement learning of value networks

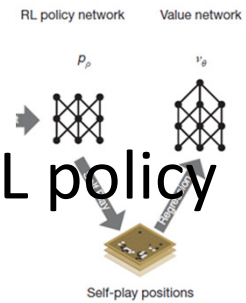
- Predict the outcome of curr state s played by policy p for both players $v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \ldots T} \sim p]$
- Ideally it is the optimal value func under perfect play, $v^*(s)$. In practice, it is v^{p_ρ} , the value function of our strongest play p_ρ . This is approximated by a θ-parametric CNN, as

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

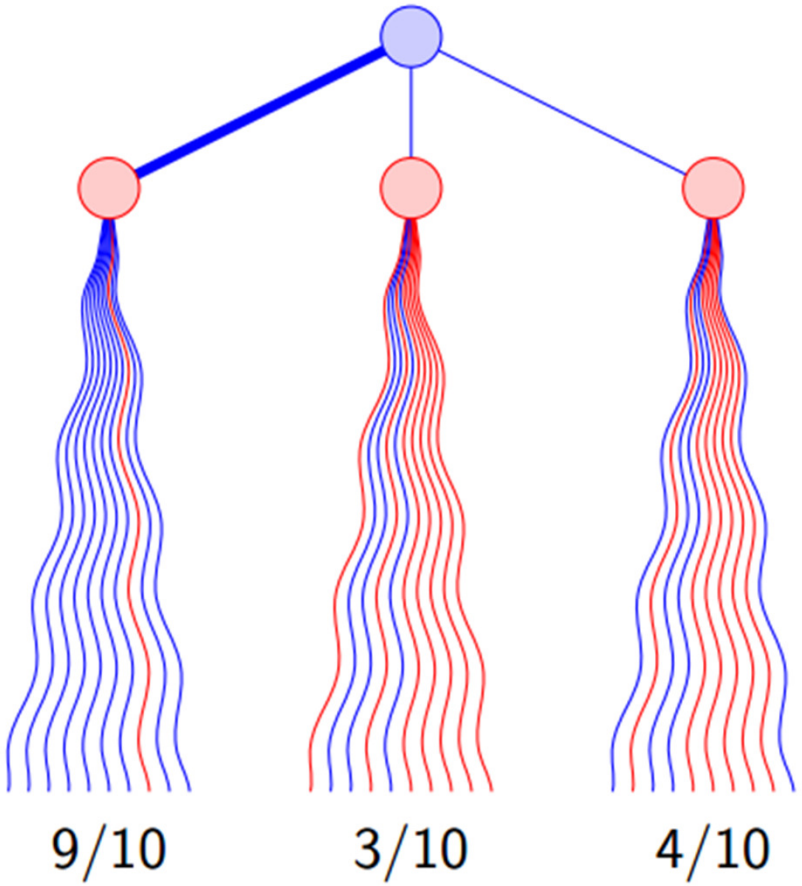
- Same architecture as RL policy network. Only it now predicts a single value instead of a distribution.
- It is trained by regression on (s,a), with SG to minimize the MSE between the predicted value $v_\theta(s)$ and ideal outcome z, as

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s))$$

- Comparable with Monte Carlo rollouts using the RL policy network, but with 15K times less computation.



Monte Carlo Tree Search



Algorithm

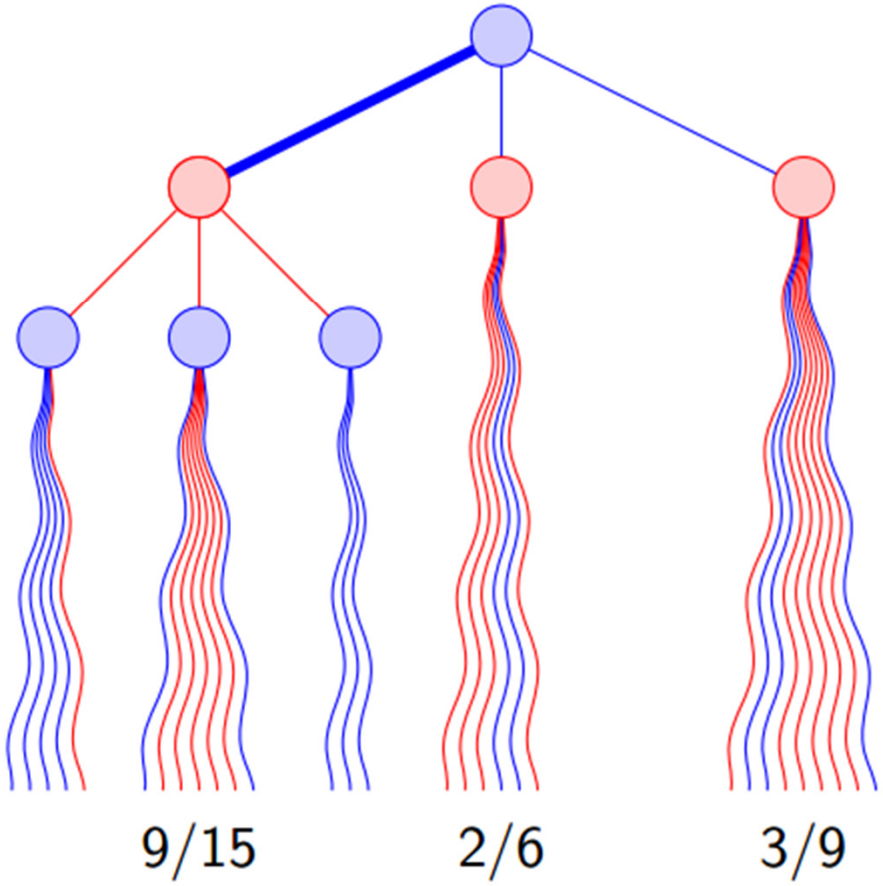
- N playouts for every move
- Pick the best winning rate
- 5,000 playouts/s on 19x19

Problems

- Evaluation may be wrong
- For instance, if all moves lose immediately, except one that wins immediately.

From Remi Coulom, the author of Crazy Stone (http://www.remi-coulom.fr/JFFoS/JFFoS.pdf)

Monte Carlo Tree Search



Principle

- More playouts to best moves
- Apply recursively
- Under some simple conditions: proven convergence to optimal move when #playouts$\rightarrow \infty$

From Remi Coulom, the author of Crazy Stone (http://www.remi-coulom.fr/JFFoS/JFFoS.pdf)

How AlphaGo Works

MCTS

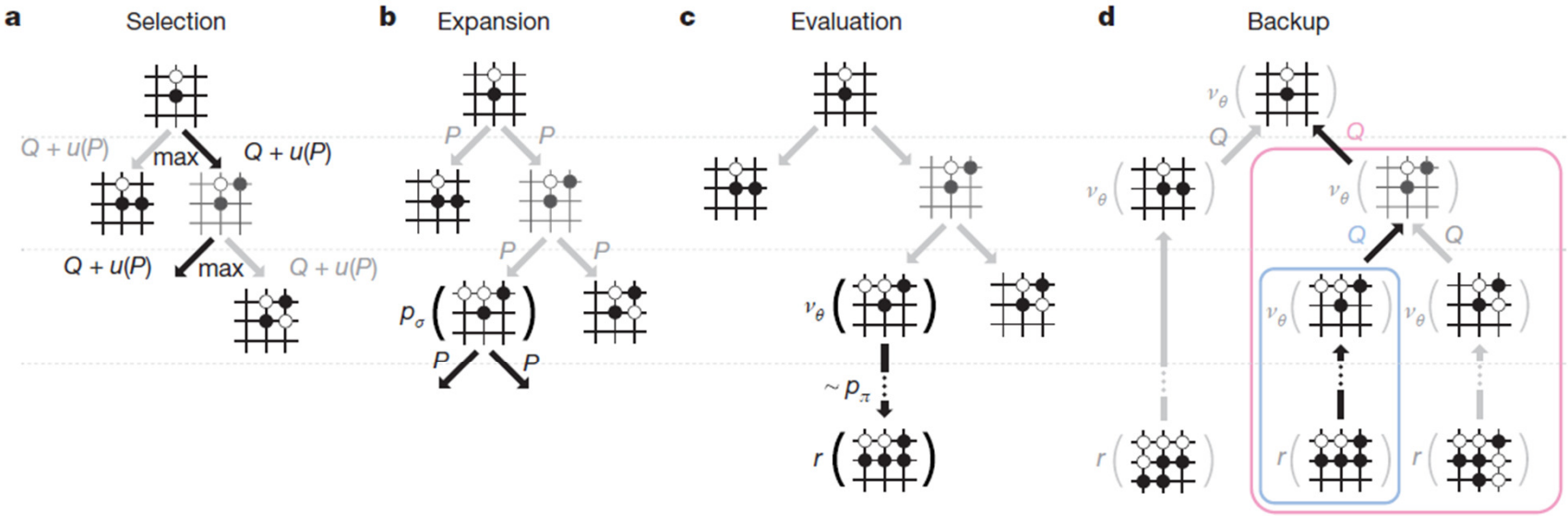
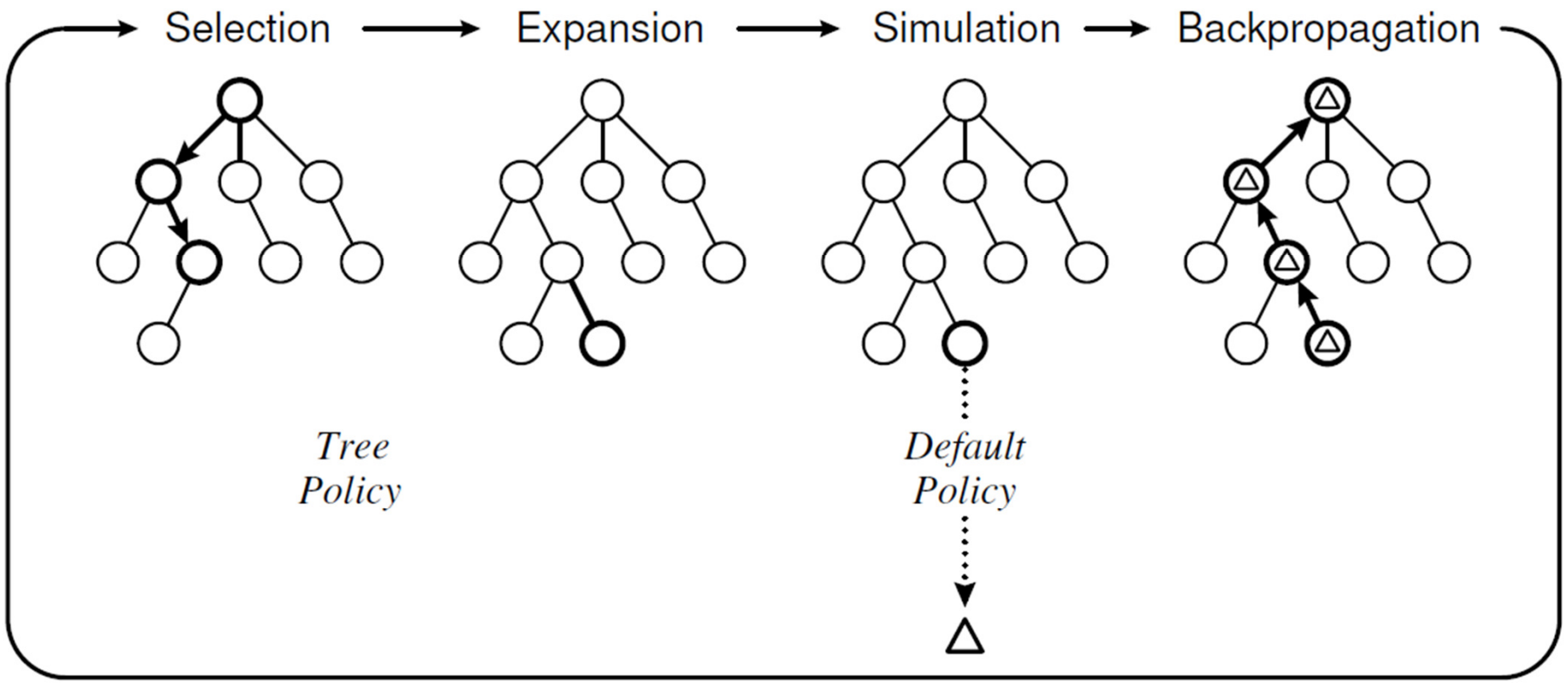


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

From AlphaGo's 28Jan16 nature paper @ (http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html)

How AlphaGo Works

Nothing new in MCTS, see e.g.



One iteration of the general MCTS approach.

How AlphaGo Works

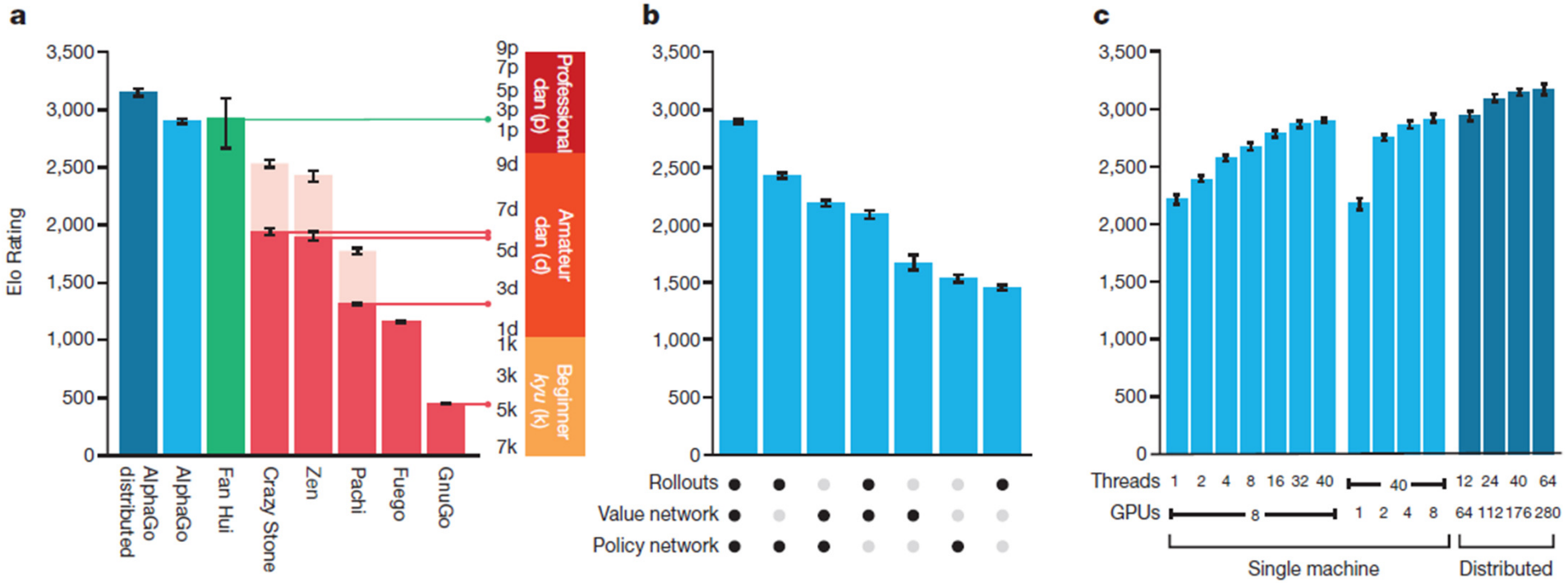
Search with policy and value networks

- The RL policy network and value network are combined in a MCST to select actions by look-ahead search.
- The tree is traversed by simulation. An edge of the MCST contains an state-action value $Q(s,a)$. At each time step t of each simulation, an action is selected by $$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$$
- Here u is a UCB value, usually in the form of $\sqrt{\frac{2\ln n}{n_j}}$, where n_j is the number of time arm j is played and n is the total number of plays so far
- Value of the leaf node, $V(s_L)$, is a combination of two sources, $$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

Where $v_\theta(s_L)$ is from the value network, and z_L is based on the fast roll-outs of policy p_π

AlphaGo

evaluations



AlphaGo

hardware

Configuration and performance

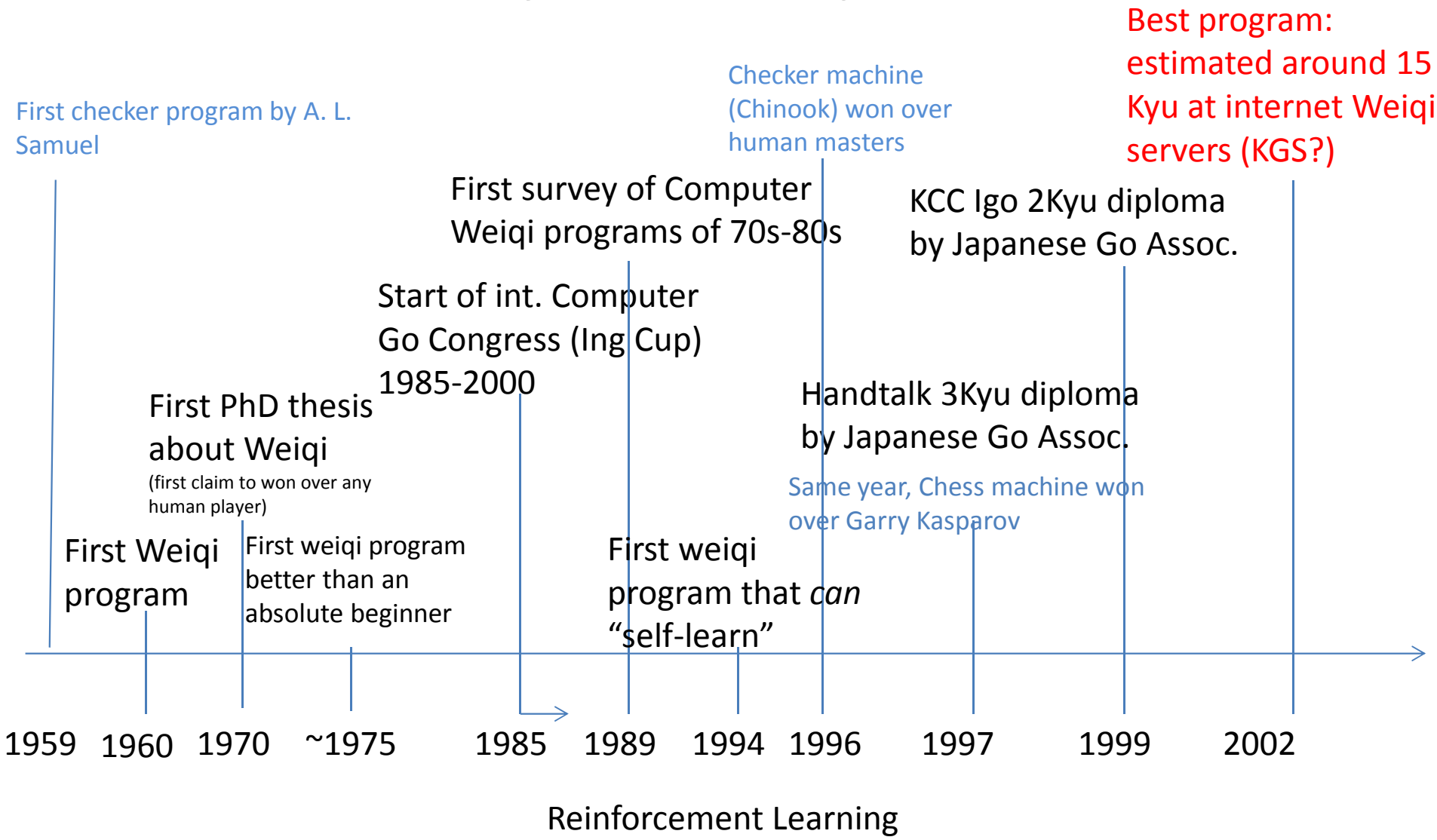
Configuration	Search threads	No. of CPU	No. of GPU	Elo rating
Single[6] p.10-11	40	48	1	2,151
Single	40	48	2	2,738
Single	40	48	4	2,850
Single	40	48	8	2,890
Distributed	12	428	64	2,937
Distributed	24	764	112	3,079
Distributed	40	1,202	176	3,140
Distributed	64	1,920	280	3,168

?Against H. FAN Oct15

?Against S. Lee Mar16

From wikipedia, (https://en.wikipedia.org/wiki/AlphaGo)

A bit History of Weiqi Machines

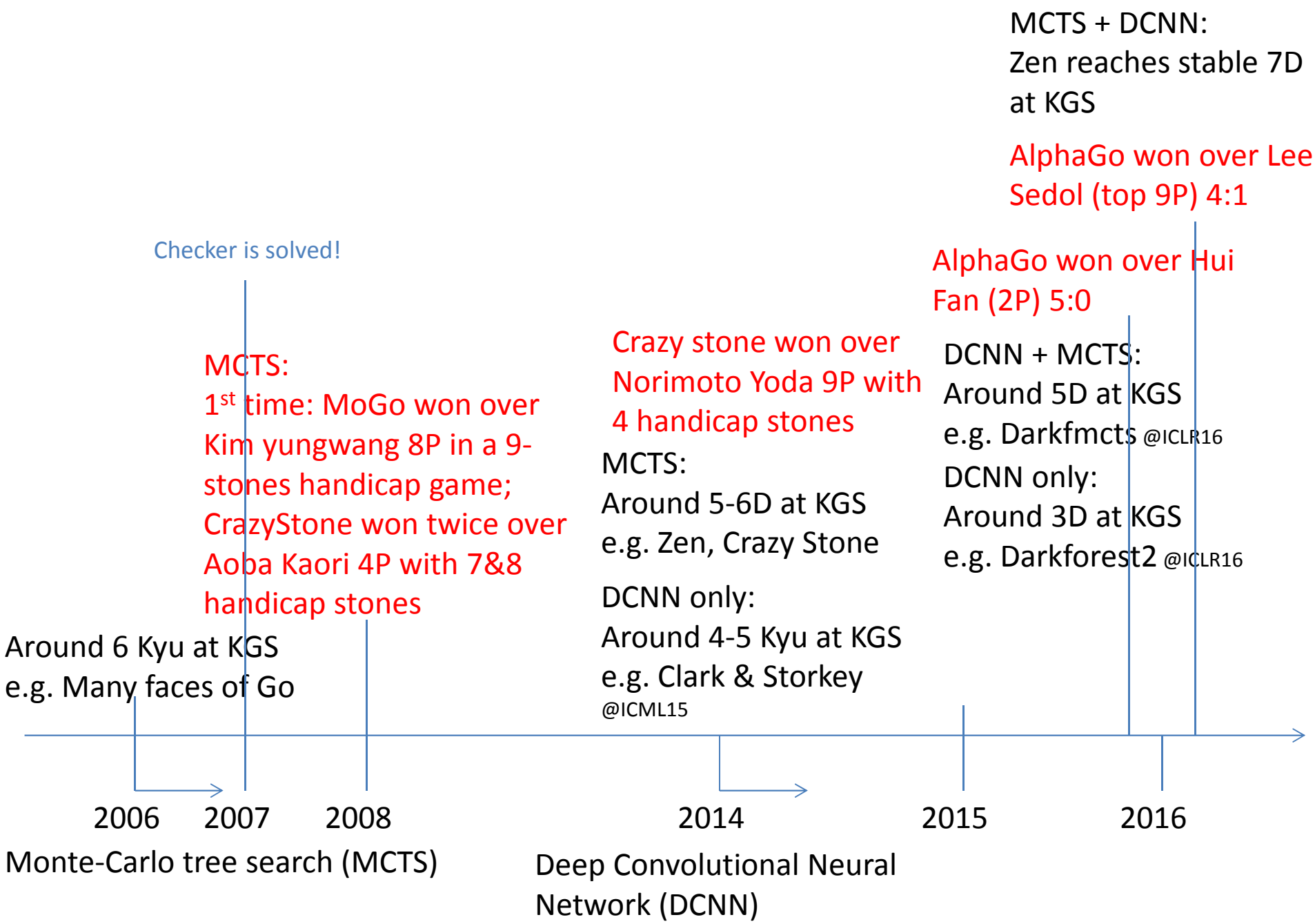


Reinforcement Learning

A bit History of Weiqi Machines

The research of Go programs is still in its infancy, but we shall see that to bring Go programs to a level comparable with current Chess programs, investigations of a totally different kind than used in computer chess are needed.

-- John McCarthy 1990


MCTS + DCNN:
Zen reaches stable 7D at KGS
AlphaGo won over Lee Sedol (top 9P) 4:1
Checker is solved!
AlphaGo won over Hui Fan (2P) 5:0
DCNN + MCTS:
Around 5D at KGS
e.g. Darkfmcts @ICLR16
DCNN only:
Around 3D at KGS
e.g. Darkforest2 @ICLR16
Crazy stone won over Norimoto Yoda 9P with 4 handicap stones
MCTS:
1st time: MoGo won over Kim yungwang 8P in a 9-stones handicap game;
CrazyStone won twice over Aoba Kaori 4P with 7&8 handicap stones
MCTS:
Around 5-6D at KGS
e.g. Zen, Crazy Stone
DCNN only:
Around 4-5 Kyu at KGS
e.g. Clark & Storkey @ICML15
Around 6 Kyu at KGS
e.g. Many faces of Go
2006
2007
2008
2014
2015
2016
Monte-Carlo tree search (MCTS)
Deep Convolutional Neural Network (DCNN)

Future Look of Weiqi Machines

three versions of “Solved” Problems

- Ultra-weakly solved: From start of game, know the optimal outcome (of black)
- Weakly solved: From start of game, best play (of black) to end regardless of the opponent’s play
- Strongly solved: From any state of a game, best play (of black) to end

How AlphaGo Works

Here search with policy and value networks

- The RL policy network and value network are combined in a MCST to select actions by look-ahead search.
- The tree is traversed by simulation. An edge of the MCST contains an state-action value $Q(s,a)$. At each time step t of each simulation, an action is selected by $a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$
- Here u is a bonus value $u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$, $N(s,a)$ is visit count,

and $P(s,a) = p_\sigma(a|s)$ is the prior prob for an action a. we also have

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i) \qquad Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

- Value of the leaf node, $V(s_L)$, is a combination of the two sources,

$$V(s_L) = (1-\lambda) v_\theta(s_L) + \lambda z_L$$

Where $v_\theta(s_L)$ is from the value network, and z_L is based on the fast roll-outs of policy p_π