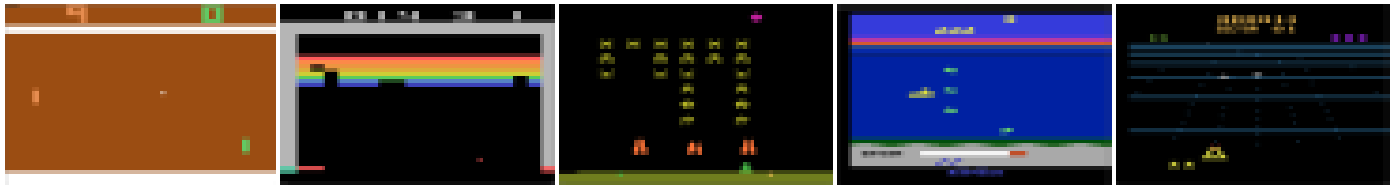
# Learning to Play Games with Deep Reinforcement Learning

Stefan Knerr

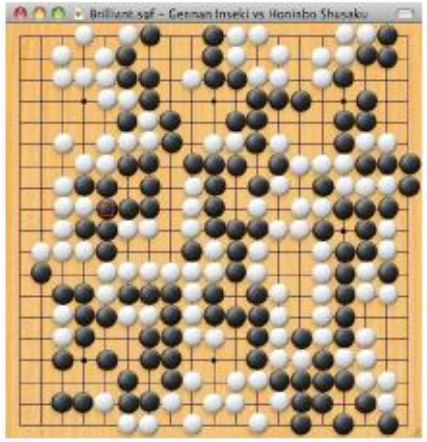
Nantes Machine Learning MeetUp

7 September 2020

# Games



Atari console games



Complicated long-term strategies.

Realistic Worlds

# Advantages of Games for AI Research

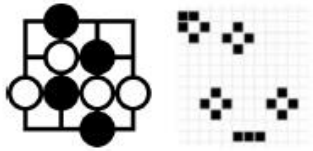
*Infinite* supply of *fully* labeled data

Controllable and replicable

Low cost per sample

Faster than real-time

Less safety and ethical concerns



Complicated dynamics with simple rules.

# Short history of Game AI

- Chess, Checkers, ...: efforts since the 50ies
- Backgammon: NeuroGammon (Gerald Tesauro, IBM), advanced level, 1989/1995
- Chess: DeepBlue (IBM) wins against Garry Kasparov (world champion), 1997
- Jeopardy: Watson (IBM) wins against human champions, 2011
- Atari:  (Deep Mind), 2014
- Go: AlphaGo (Deep Mind) wins against Lee Sedol (world champion), 2016
- ... Starcraft, Dota

- Game AI is one of the highlights of Deep Learning.

Rémi Coulom, creator of *Crazy Stone* Go program presented twice at our MeetUp.
Crazy Stone was one of the worldwide best Go programs prior to Deep Mind's AlphaGo.

# We will focus here on **Go,** which is:

- Competitive: you want to win
- Deterministic: no randomness, no dices
- 2-player
- Turn-taking
- Zero-sum: what one player wins, the other player looses
- Perfect information: both players have all information on status of game

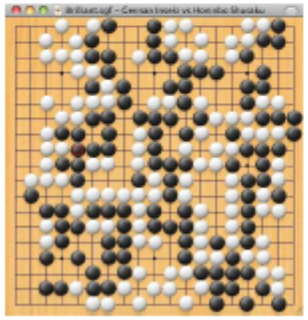
Other such games: chess, go, checkers, shogi, backgammon, ...

# Adversarial Search

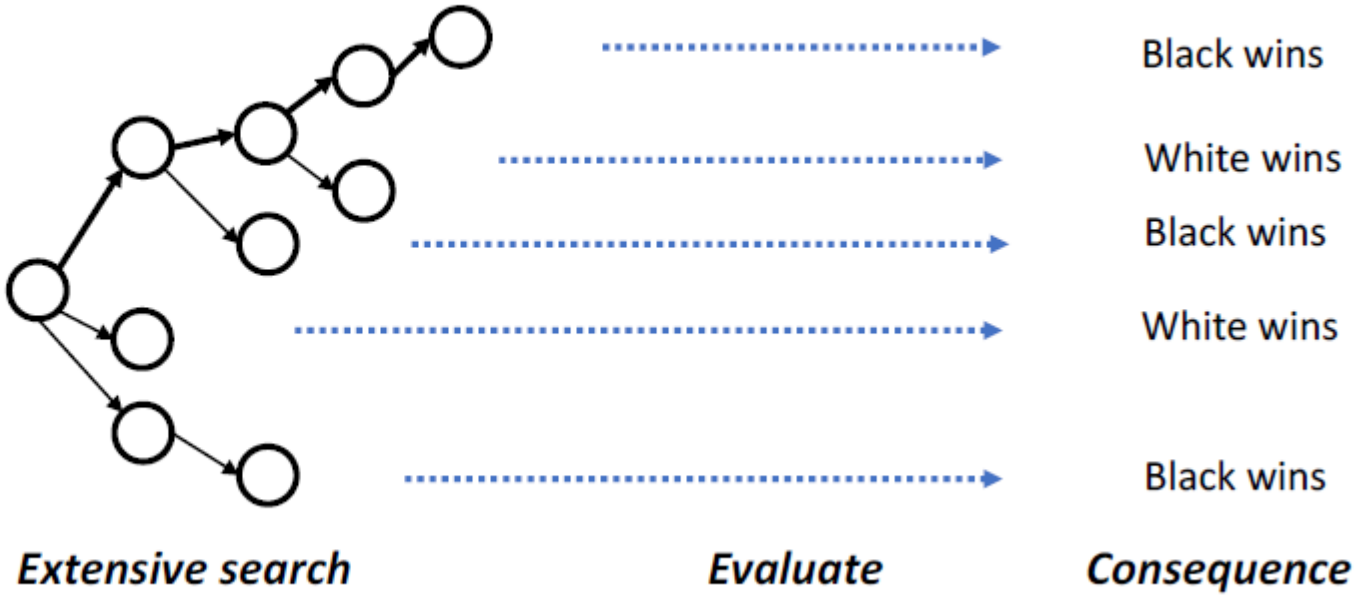
- Search in game tree
- Find most probable path to goal state starting from initial state
- Exhaustive search: Minimax
- alpha-beta pruning

Go search tree:
breadth$^{depth}$ ≈ $200^{150}$ possible sequences of moves!
Exhaustive search is infeasible.

*Lufei Ruan vs. Yifan Hou (2010)*

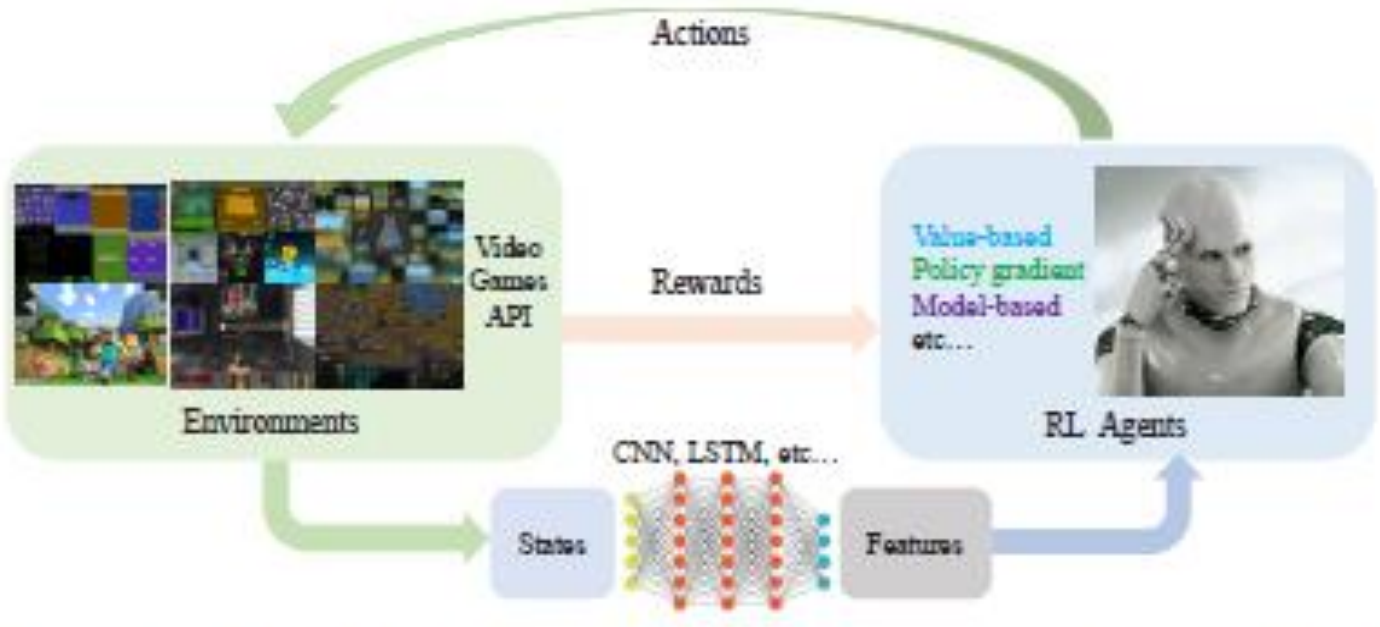


Current game situation



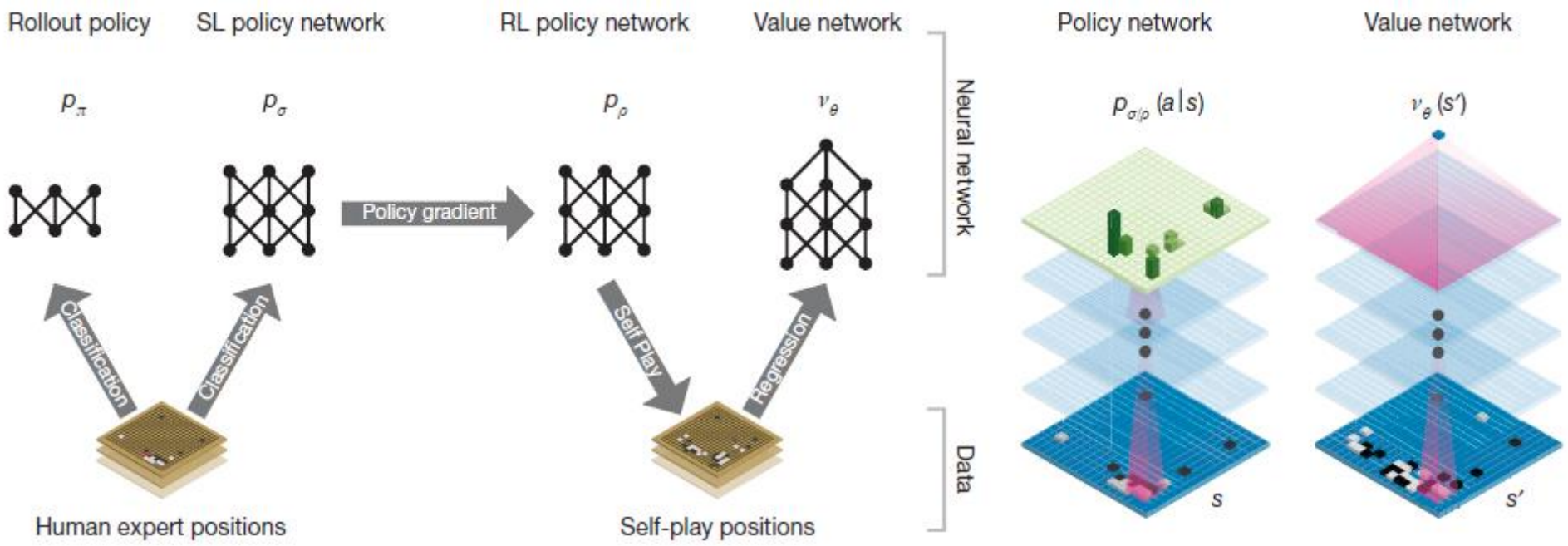
# Deep Reinforcement Learning and Games

- Idea: explore interesting branches in game tree → use future situations in game tree which are easier to evaluate (e.g. game won/lost).
- Model problem as Markov Decision Process
- Actor: policy for action a given state s: $\pi(a, s) = P(a \mid s)$
- Critic: value = discounted reward r expectation: $V_\pi(s) = E[\sum_{t=0}^{T} \gamma_t r_t \mid s_0 = s]$
- Often $r_t = 0$ except for $r_T = \pm 1$ depending on win or loss.

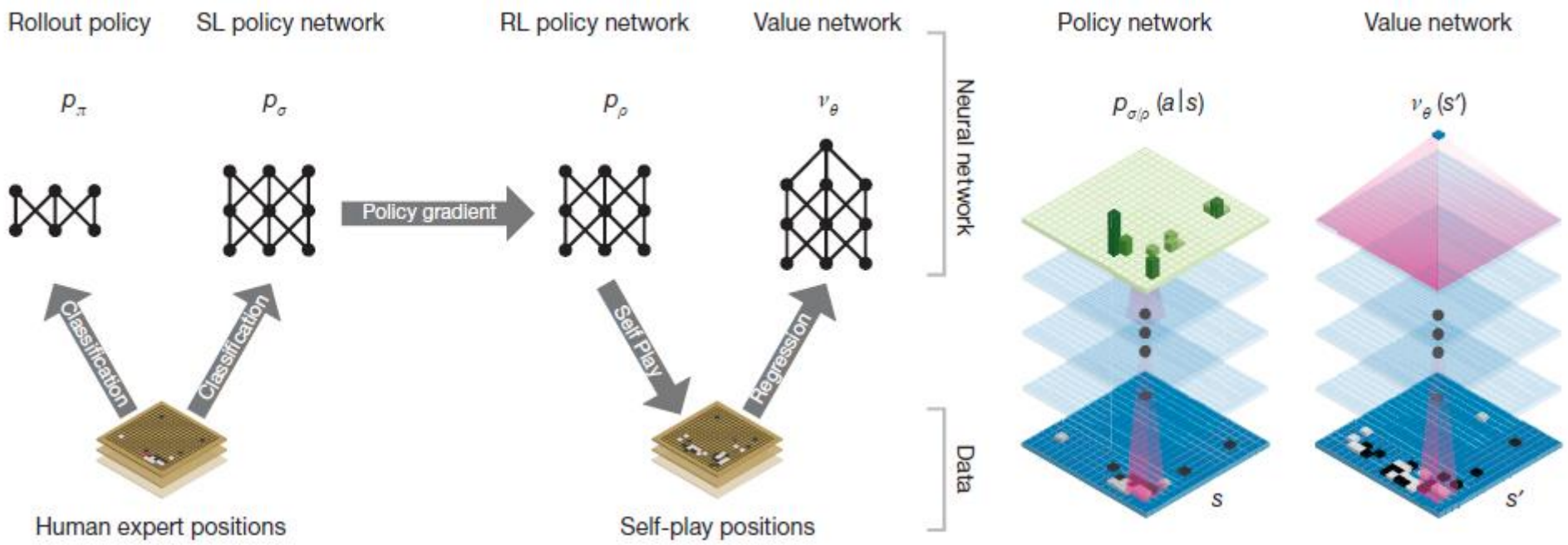


# AlphaGo (Deep Mind, 2016)

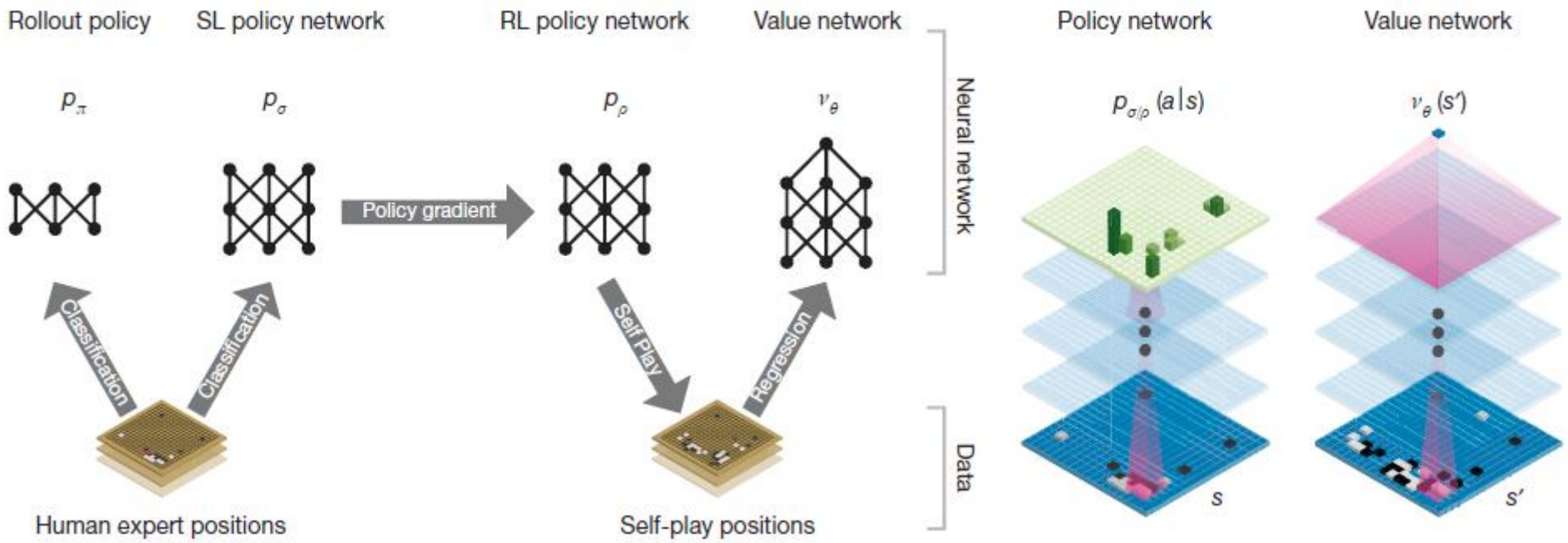
- Step 1: Supervised learning of 2 policy networks (SL policy, rollout)
  - Deep Convolutional Network (13 layers)
  - Data Set: 30 M positions of human games from KGS (6 Dan or better)
  - Input = actual board position (features), Output = next move → $P(a \mid s)$ distribution
  - SGD on Maximum Likelihood



- Step 2: Reinforcement Learning of policy network with self-play
  - Initialize network with SL policy weights
  - Play games between network and randomly selected previous networks
  - Policy gradient. Reward $r_T = \pm 1$.

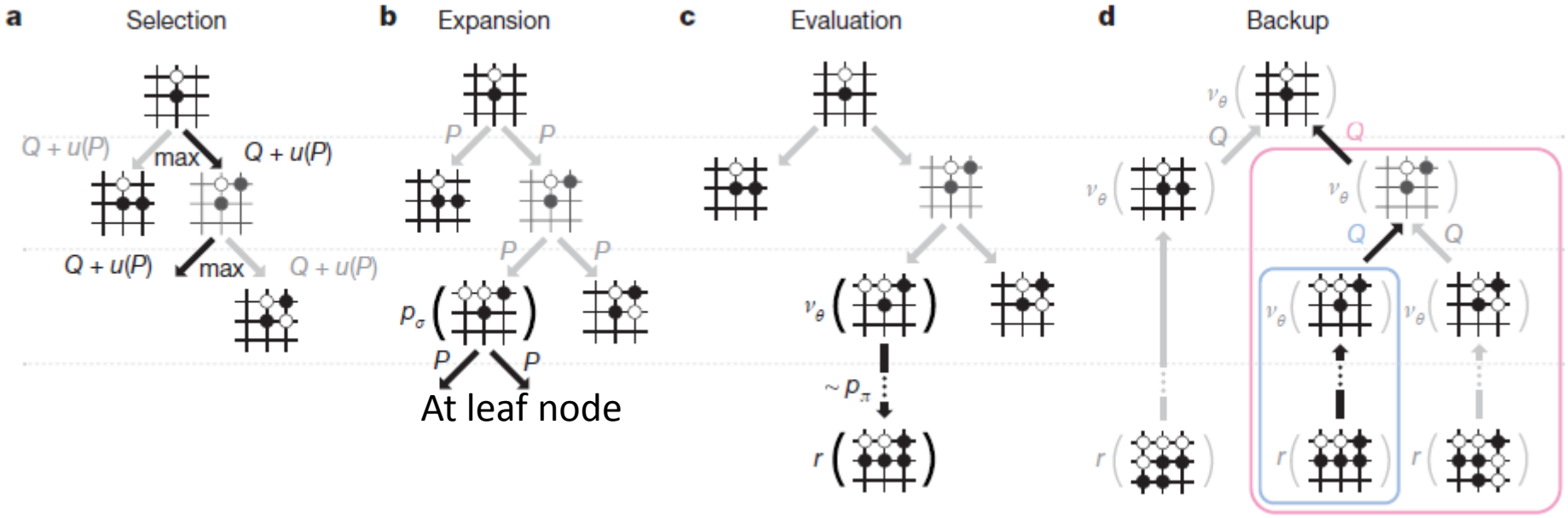


- Step 3: Reinforcement Learning of value network with self-play
  - Ideally we want $V_\pi(s)$ for the optimal policy π. In practice we estimate $V_\pi(s)$ for the best policy network we have.
  - Similar network structure as policy network, single output.
  - Regression with SGD on MSE.
  - Avoid overfitting by sampling positions from many different games.

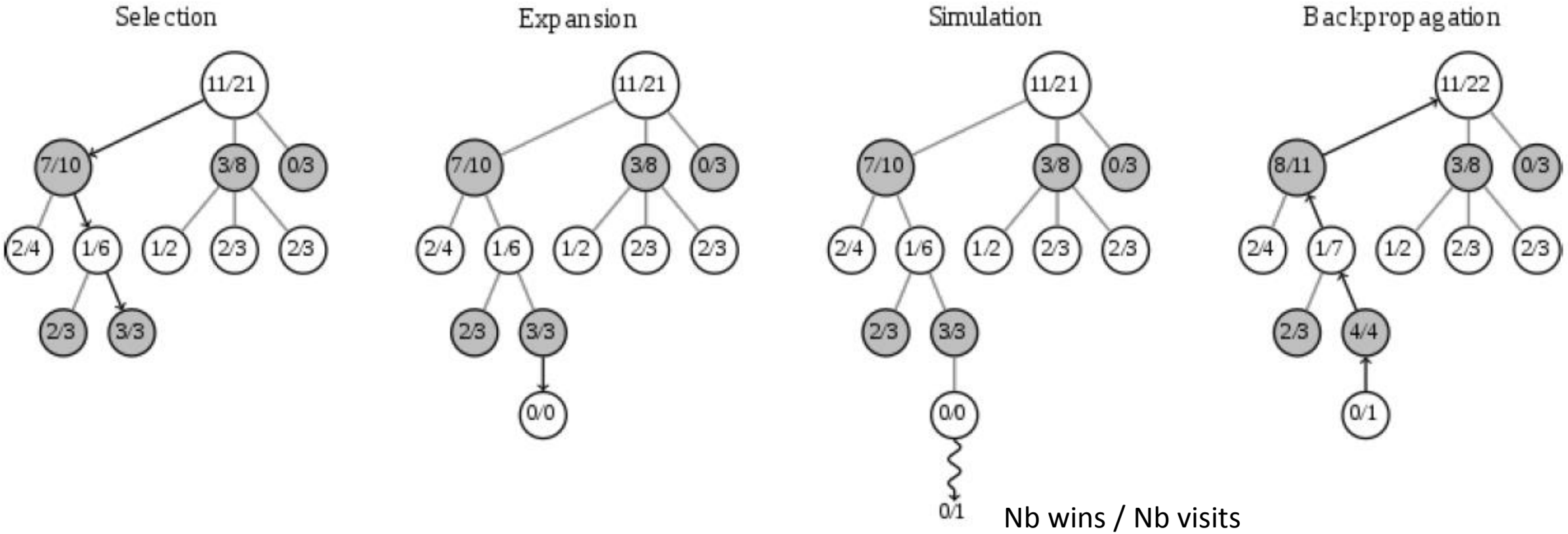


# Monte Carlo Tree Search (MCTS)

- Simulate games using policy net $P(a \mid s)$. Explore mostly high P branches (≠ random).
- Each node in tree stores $Q(s,a)$ value and visit count $N(s,a)$.
- Start at root node. Each node selects action $a_t = argmax(Q(s_t,a) + \frac{P(s,a)}{1+N(s,a)})$, balancing exploitation versus exploration during learning.
- At leaf node expand and evaluate: $Q(s_t,a)$ = value net + random MC with rollout net.
- Update all visited nodes.



# Monte Carlo Tree Search (simplified)

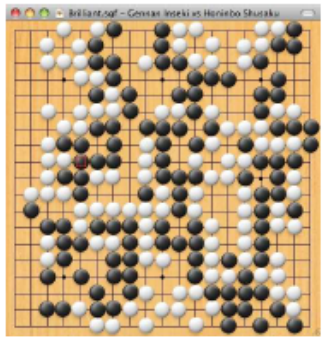



Nb wins / Nb visits

Watch out! When white looses, black wins.

# AlphaZero

- Like AlphaGo, MCTS intelligently guided by NN (policy + value)
- No initial supervised learning (“zero”), only self-play
- No handcrafted features, only stone positions for last 8 half-moves of both players
- Single 2-headed NN for policy and value (Multi-task Learning)
- ~ 40 layer ResNet with Conv blocks + 2/3 layers for policy/value head

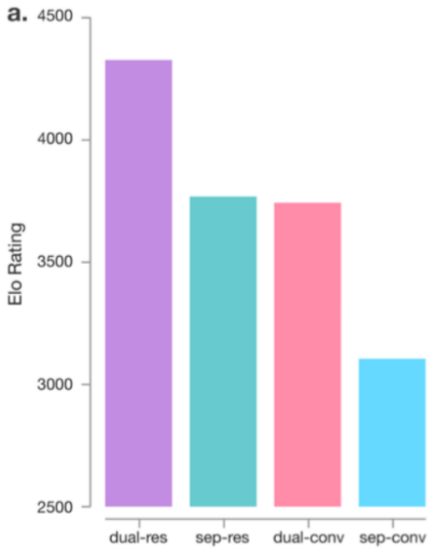


$s$

ResNet

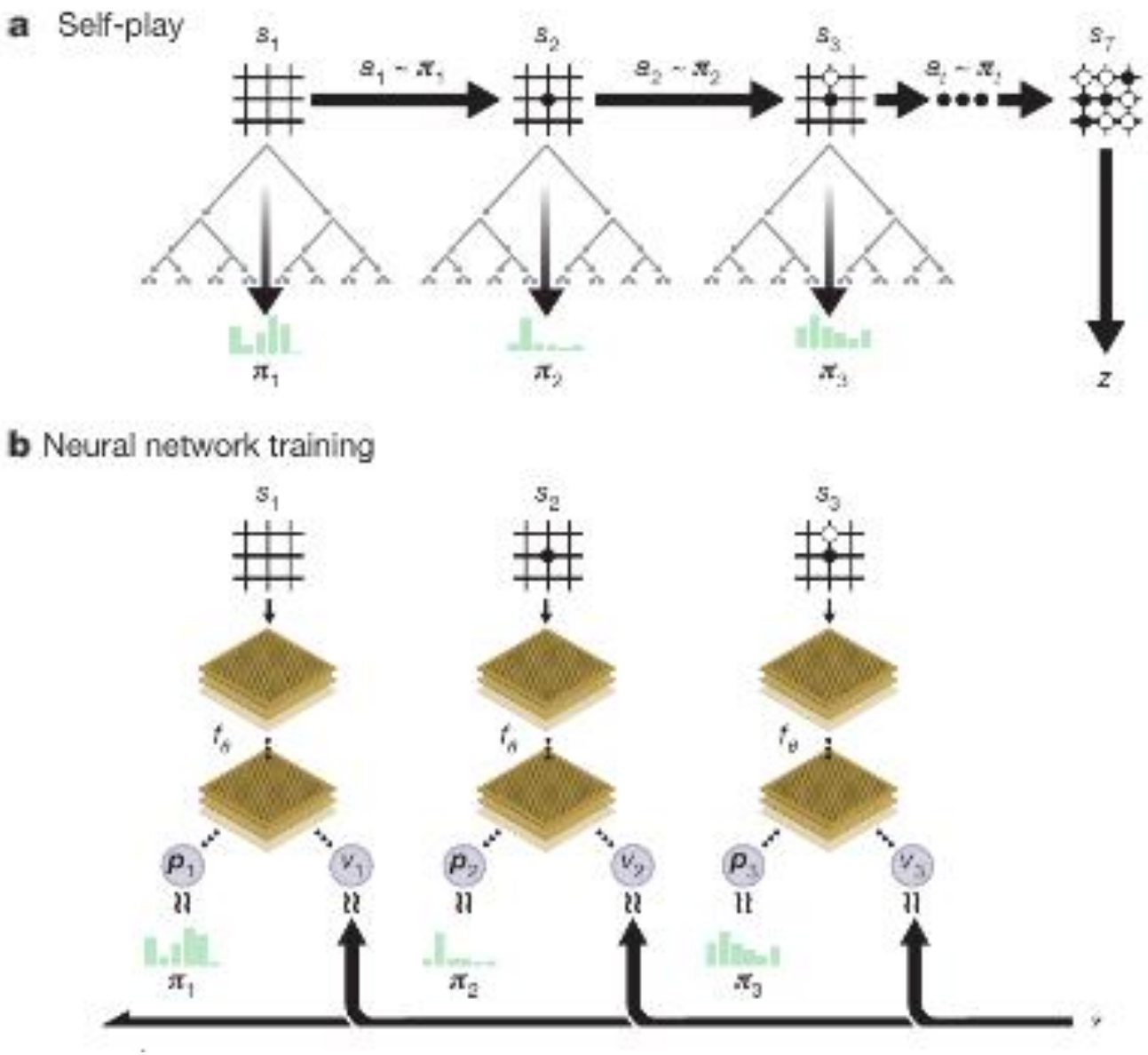
$\mathbf{p}_\theta(s)$

$V_\theta(s)$

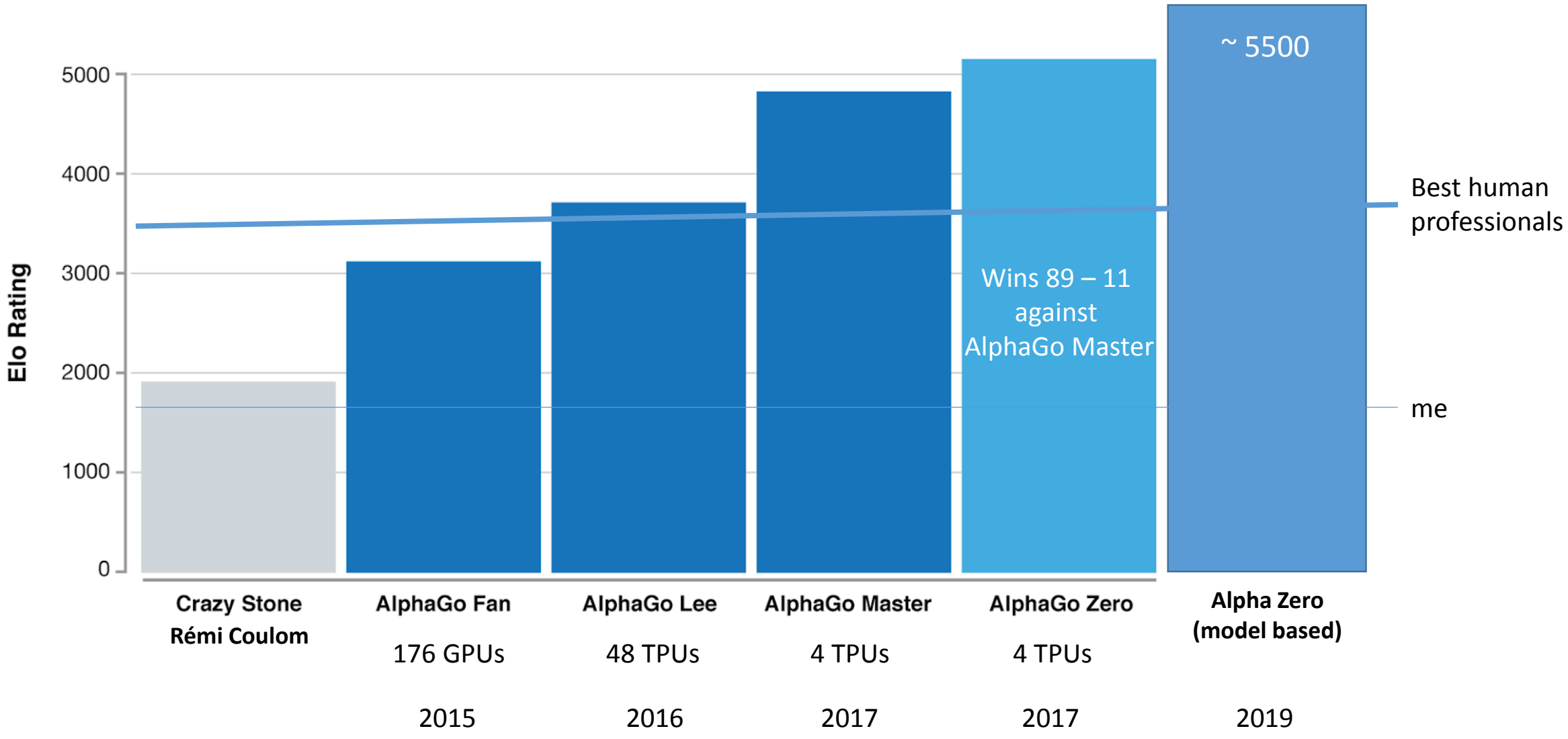


# MCTS in AlphaZero

- Very clever!
- MC during learning, not play
- Teach NN to approximate policy and value of MC
- Moves are selected by max Q value as in AlphaGo



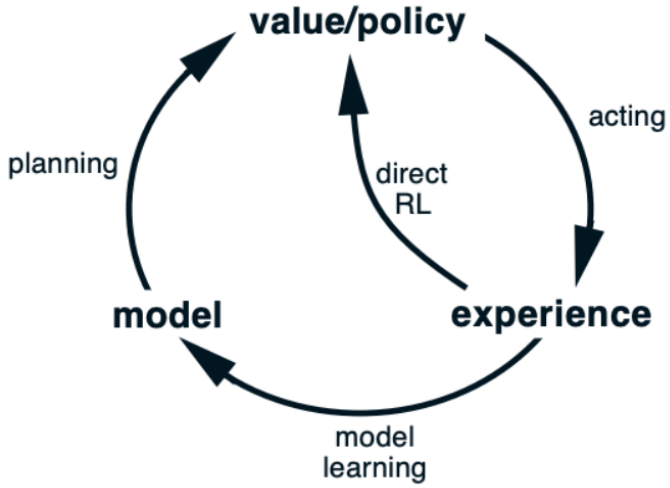
# Elo Rating of AlphaGo




200 point gap = 75% probability of winning

# MuZero

- Real world: dynamics of environment is often complex and unknown. Perfect simulator not available.
- Games: rules not supplied
- Only self-play with model based planning and RL / MC
- Model only learns what is necessary for planning: $a_t$, $v_t$, $r_t$




- 3 components:
  - Representation function h: board → latent state representation
  - Dynamics function g: state $s_t$ and candidate action $a_t$ → state $s_{t+1}$ and reward $r_t$
  - Prediction function f: policy $\pi_t$ (next move) and value $v_t$ (who wins)

- → SOTA on 57 Atari games
- → matched AlphaZero for Go, Chess, Shogi

# Dota 2


- OpenAI Five

- multiplayer online battle arena (MOBA) video game (Valve)
- Proximal Policy Learning (RL); self-play
- continuous observation & action spaces; partially observable state space; long run (20K)
- separate LSTM networks to learn each hero
- 256 GPUs and 128,000 CPU cores trained for months, accumulating 180 years of game experience each day
- beats 2018 *Dota 2* champion team in a 2019 series of games

# Conclusions

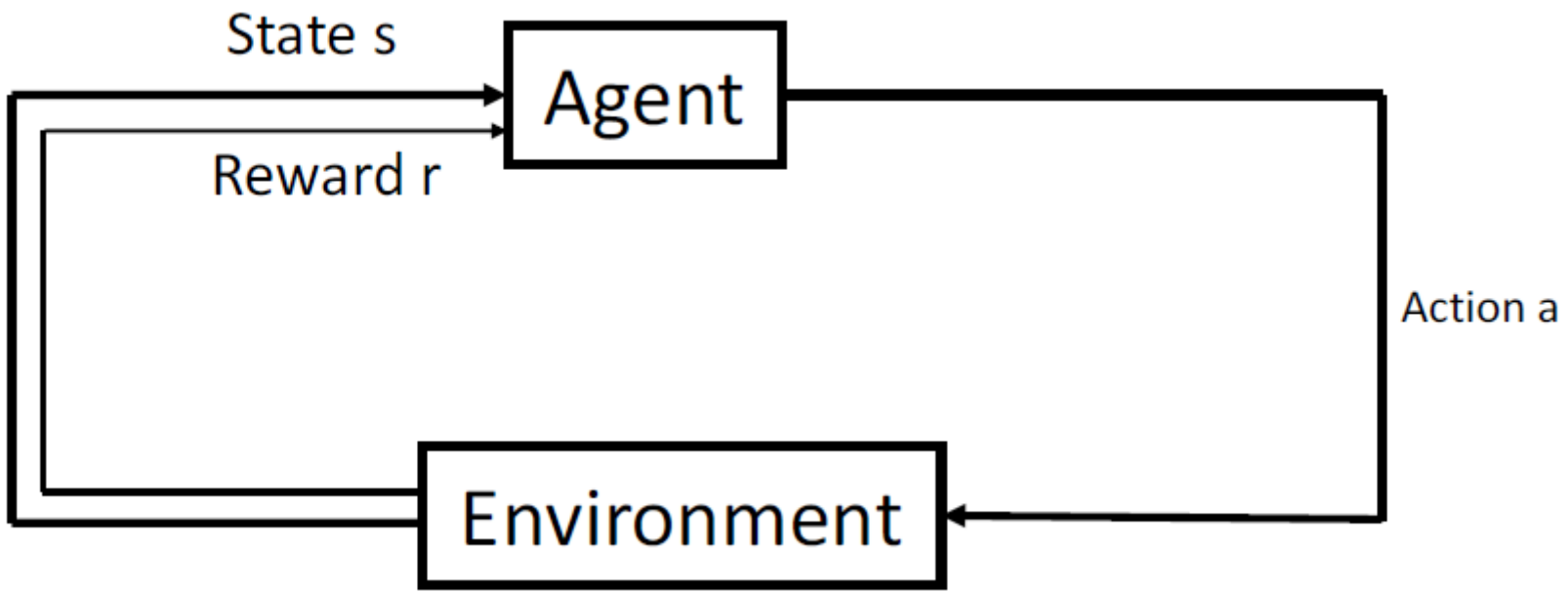
- Very impressive super-human performance for games like Chess, Go, checkers, Atari, ...
- Ideal application for Deep Learning as training data can often be easily generated in self-play
- Deep Learning is a data inefficient approach. Humans learn very differently (concept learning)
- Black boxes, difficult to interpret. Different from divide & conquer problem solving.
- Community moves to more complex games:  Poker, Minecraft, Starcraft, car racing, Dota 2, ...

# References


- Yuandong Tian, Facebook, ACMMM17.tutorial
- Bruno Bouzy et al., Computer Go: an AI Oriented Survey, 2001
- Volodymyr Mnih et al., Playing Atari with Deep Reinforcement Learning, NIPS, 2014
- David Silver et al., Mastering the Game of Go with deep neural networks and tree search, Nature, 2016
- David Silver et al., Mastering the Game of Go without Human Knowledge, 2017
- Julian Schrittwieser et al., Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model, arXiv, Nov 2019

# Back-up slides

# Reinforcement Learning



# Reinforcement Learning

- Difficult or impossible to evaluate present situations → use future situations in game tree which are easier to evaluate (e.g. game won/lost).
- Model-free RL methods:
  - sample actions from policy + policy gradient,
  - state-value function (Q-Learning),
  - combination of both: Actor-Critic
  - Games ok, ~~real world~~ (needs massive amounts of data)
- Model-based RL: MDP with 2 components:
  - state transition model, predicting the next state,
  - reward model, predicting the expected reward during that transition.
- On-policy, off-policy
- Multi-agent
- Experience Replay (paper with Bengio 2020)

# AlphaGo

- Deep Mind (Google, London) develops Go software based on NN, RL, and MC.
- 2015 AlphaGo trained by Fan Hui (European Go champion)
- 2016: AlphaGo beats Go legend Lee Sedol 4 – 1
- 2017: AlphaGo Zero, only self-play, beats AlphaGo Master 89 - 11
- One of the highlights of modern AI.

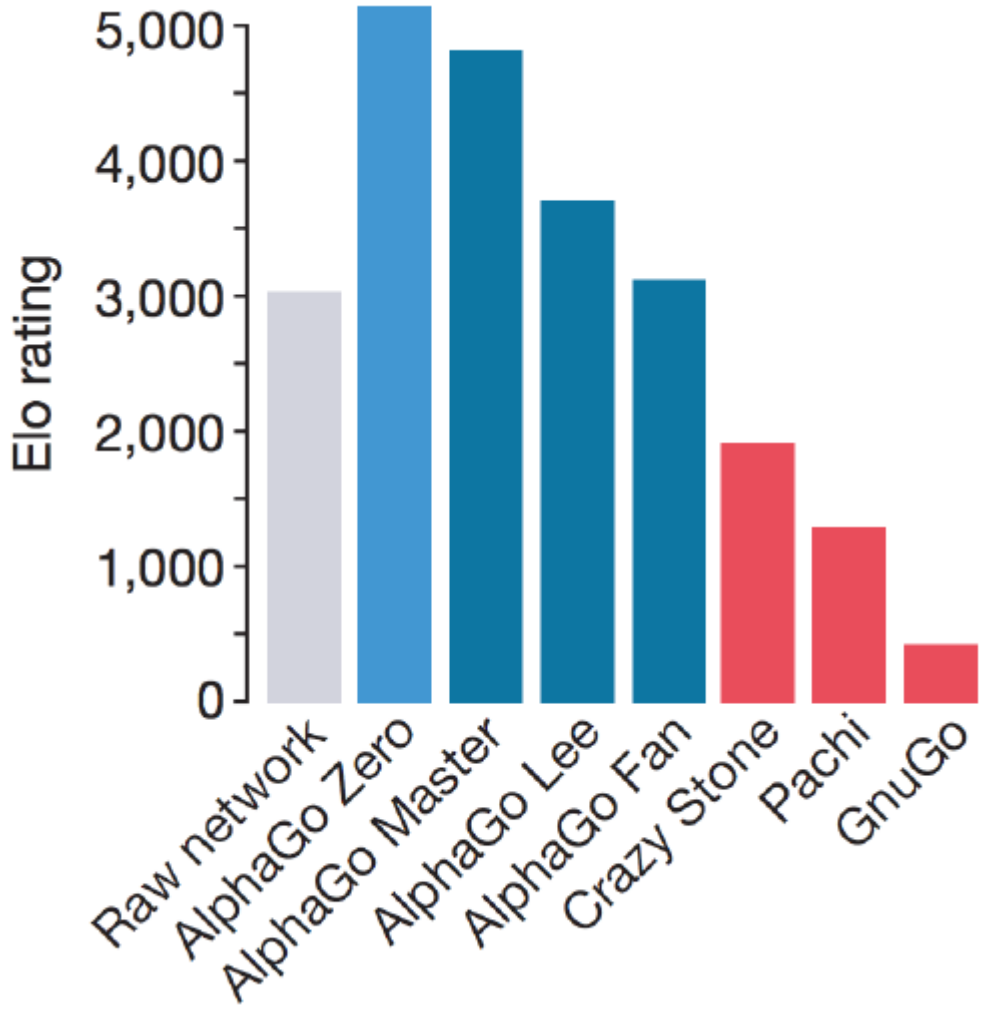
# AlphaGo Feature Description of Board

**Extended Data Table 2 | Input features for neural networks**

| 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 |

Feature planes used by the policy network (all but last feature) and value network (all features).

# How good is AlphaGo?



# MuZero