Elements of DSAI

AlphaGo Part 2: Game Tree Search, Learning Architectures

Search & Learn: A Recipe for AI Action Decisions

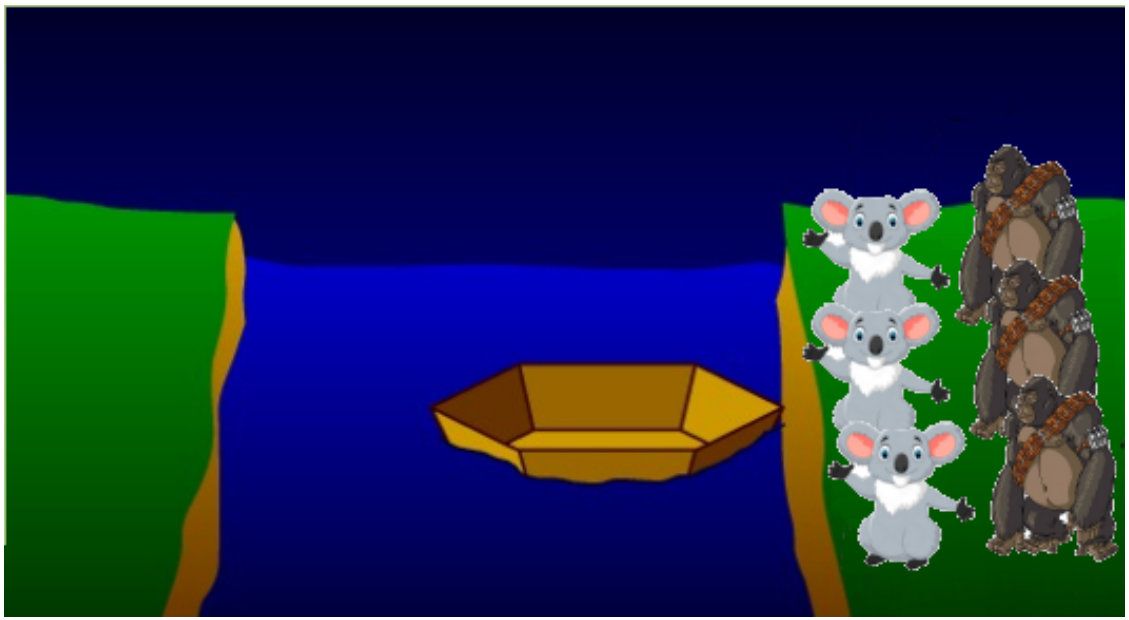
Jörg Hoffmann

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2019/20

Competitive Agents?

Quote AI Introduction: "Single agent vs. multi-agent: One agent or several? Competitive or collaborative?"



→ Single agent!

- Several koalas, several gorillas trying to beat these up.
- BUT there is only a single acting entity – one player decides which moves to take (who gets into the boat).

Competitive Agents!

Quote AI Introduction: "Single agent vs. multi-agent: One agent or several? Competitive or collaborative?"

→ Multi-agent competitive!

- TWO players deciding which moves to take.
- Conflicting interests.

Agenda: Game Search, AlphaGo Architecture

- **Games:** What is that?

  → Game categories, game solutions.

- **Game Search:** How to solve a game?

  → Searching the game tree.

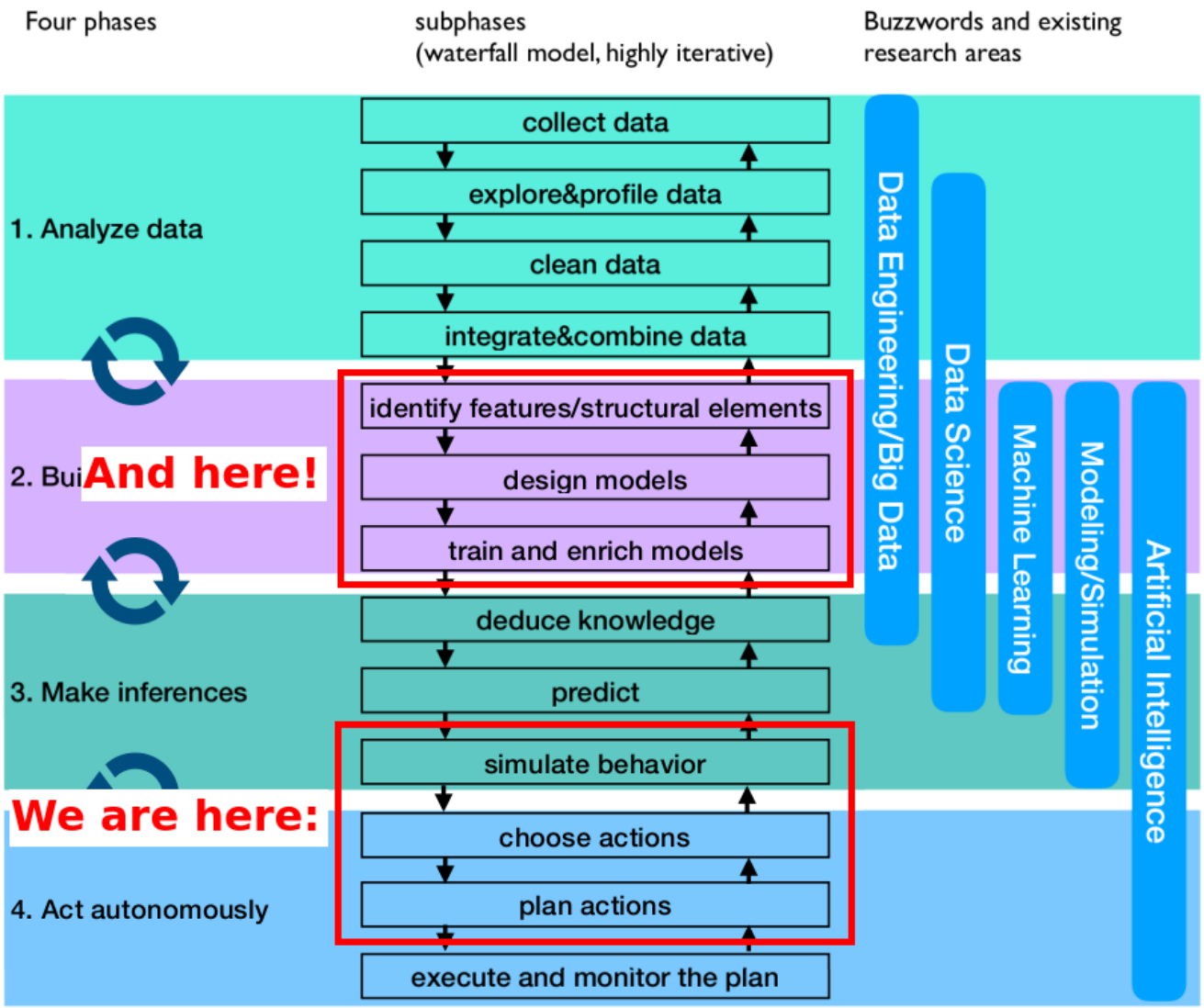
- **Evaluation Functions:** How to evaluate a game position?

  → Heuristic functions for games.

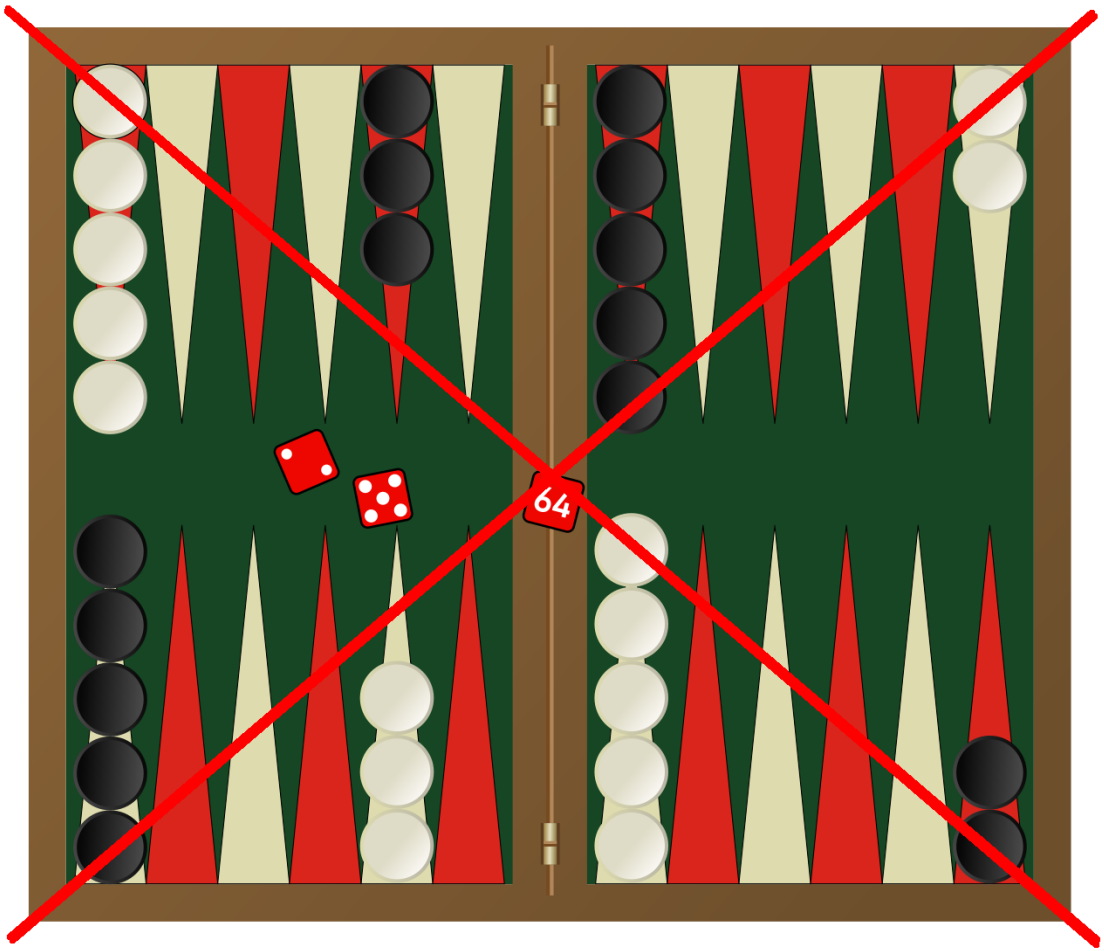
- **AlphaGo:** How does it work?

  → Overview of AlphaGo architecture, and changes in Alpha(Go) Zero.

Positioning in the DSAI Phase Model

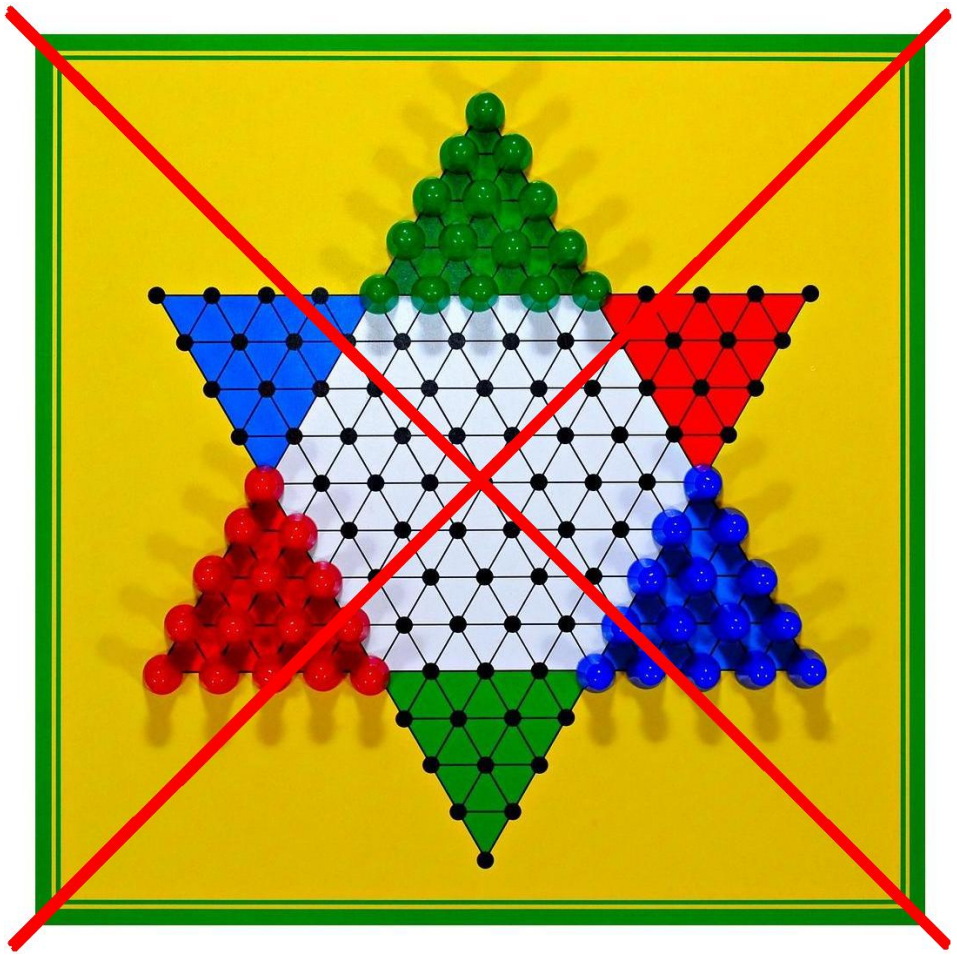


Which Games?



→ No chance element.

Which Games?



→ Exactly two players.

Which Games?

→ Game state fully observable.

Which Games?

→ Player utilities are diametrically opposed.
(Else: game theory, equilibria, auctions, . . .)

These Games!

Restrictions:

- The game state is fully observable.
- The outcome of each move is deterministic.
- Game states discrete, finite number of possible moves and game states.
- There are no infinite runs of the game: a terminal state is always reached after a finite number of steps.
- Two-player zero-sum game: two players, terminal states have utility with utility(player1) $= -$utility(player2).
- Our formulation (equivalent): single utility function u, players Max vs. Min trying to maximize vs. minimize u.
- Turn-taking: Players move alternatingly. Max begins.

Game State Space

Game State Space:

- States S consist of S^{Max} Max to move, S^{Min} Min to move, S^T terminal states.
- Actions A consist of A^{Max} and A^{Min}.
  $\rightarrow a \in A^{Max}$ applicable only in S^{Max}, outcome state in $S^{Min} \cup S^T$. Symmetrically for $a \in A^{Min}$.
- Utility function u: $S^T \mapsto \mathbb{R}$.

- Players Max vs. Min: White vs. Black.
- States: Board position + who's to move.
- Terminal states: Checkmate, Stalemate.
- Actions: Moves according to rules.
- Utility function: $+100$ if Black is checkmated, 0 if stalemate, -100 if White is checkmated.

Why is Game Playing Hard?

Reminder: Chess $|S| \approx 10^{43}$, Chess $|S| \approx 10^{171}$, number of atoms in the universe $\approx 10^{82}$.

$\rightarrow$ But that's not even the worst part! FreeCell $|S| \approx 10^{67}$ yet FreeCell is *much* easier to solve than Chess.

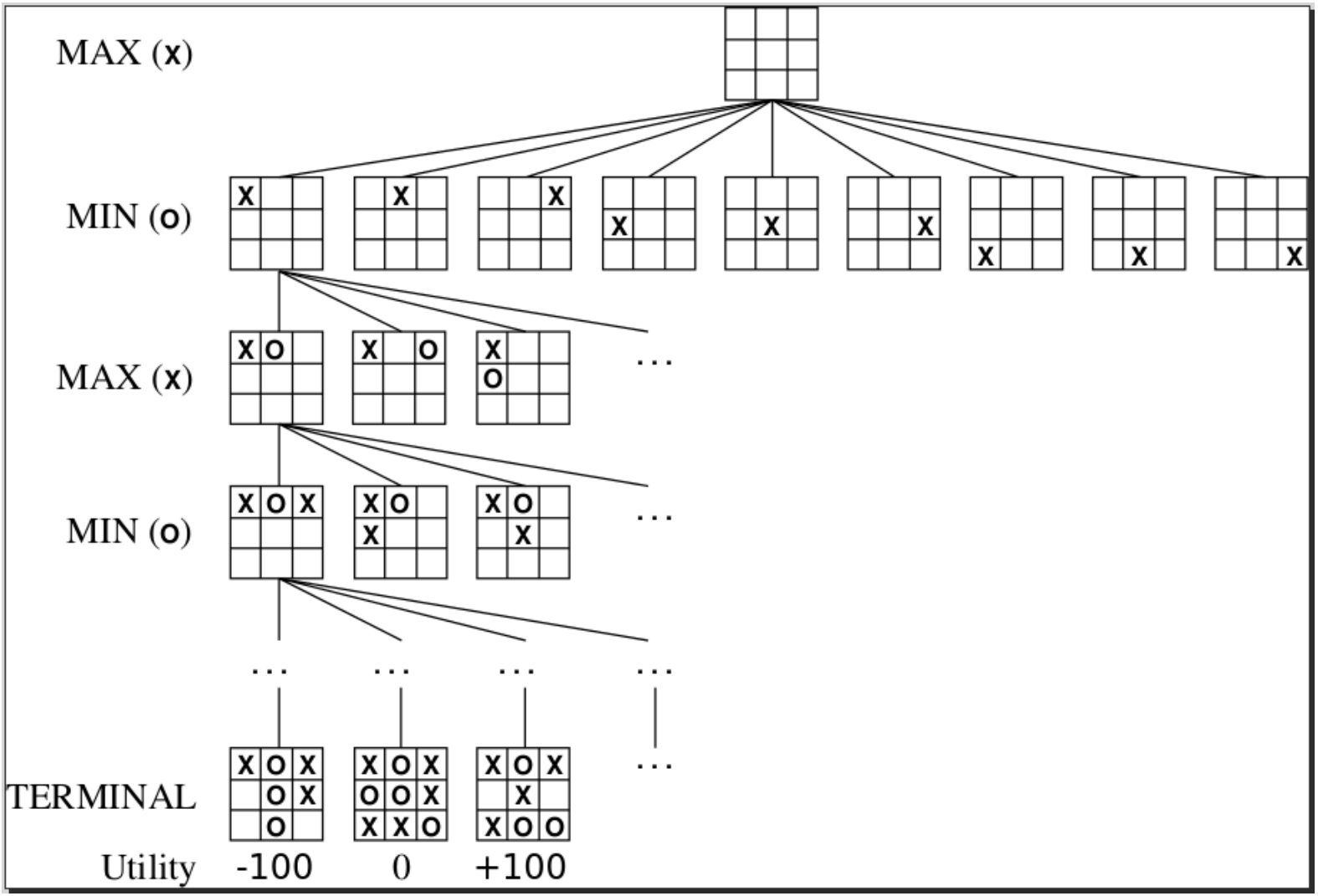
What is a "solution"?

- Single-agent search problem: Path from initial state to goal state.
- (Two-player zero-sum turn-taking) game: An action policy (= game strategy), reacting to all possible opponent moves!

  $\rightarrow$ For Max: function $\sigma^{Max} : S^{Max} \mapsto A^{Max}$. Max policy optimal if it maximizes u assuming perfect Min play.

$\rightarrow$ Computing an optimal policy is typically infeasible. Instead, compute the next move on demand, given the current game state.

Game Search Tree: Example Tic-Tac-Toe



The Minimax Algorithm

Input: State $s \in S^{Max}$, in which Max is to move.

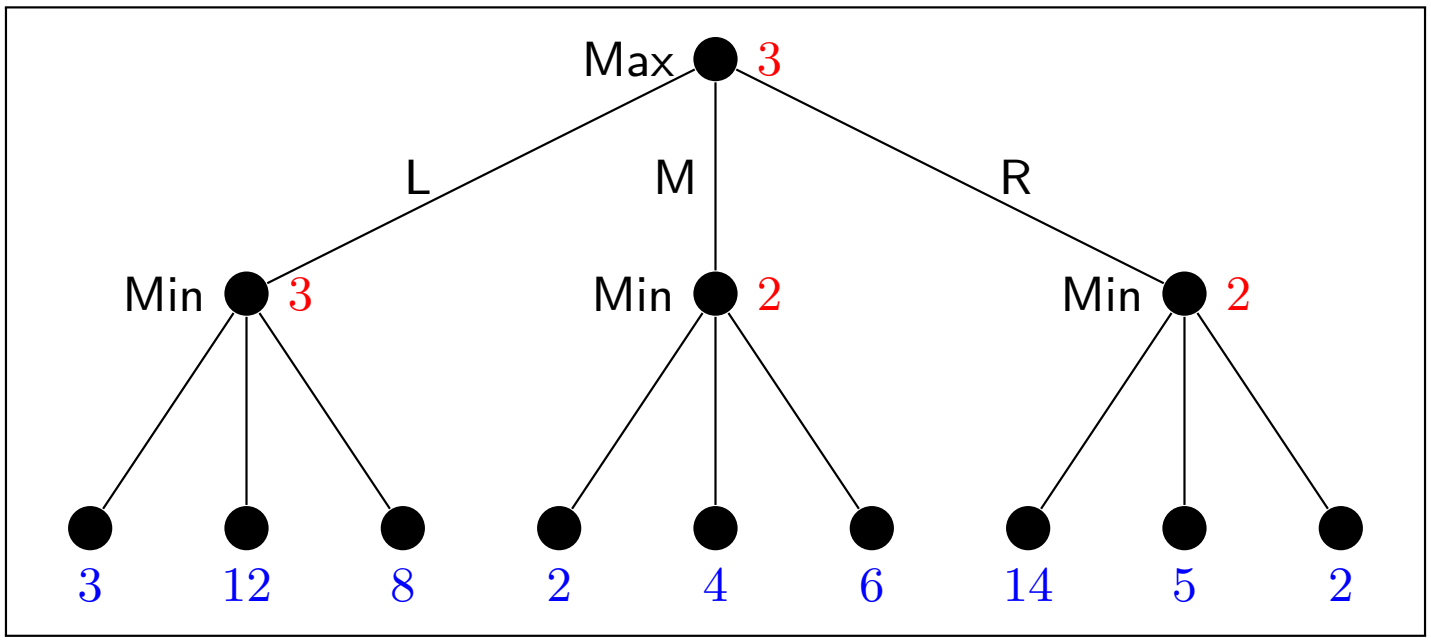
function Minimax-Decision(s) **returns** an action
  $v \leftarrow$ *Max-Value*(s)
  return an *action* $a \in$ *Actions(*s*)* yielding value v

function Max-Value(s) **returns** a utility value
  if *Terminal-Test*(s) **then return** $u(s)$
  $v \leftarrow -\infty$
  for each $a \in$ *Actions(*s*)* **do**
    $v \leftarrow \max(v,$*Min-Value*$($*ChildState*$(s,a)))$
  return v

function Min-Value(s) **returns** a utility value
  if *Terminal-Test*(s) **then return** $u(s)$
  $v \leftarrow +\infty$
  for each $a \in$ *Actions(*s*)* **do**
    $v \leftarrow \min(v,$*Max-Value*$($*ChildState*$(s,a)))$
  return v

Minimax: Example

Notation: blue: u value on terminal states; red: non-terminal state value as computed by Minimax.



→ Which action is returned? L. Maximal utility is higher for R; but assuming perfect Min play, L is better. Choosing R would rely on the opponent to do something stupid.

Minimax, Pro and Contra

Pro:

- Returns an optimal action, assuming perfect opponent play.
- Extremely simple.

Contra:

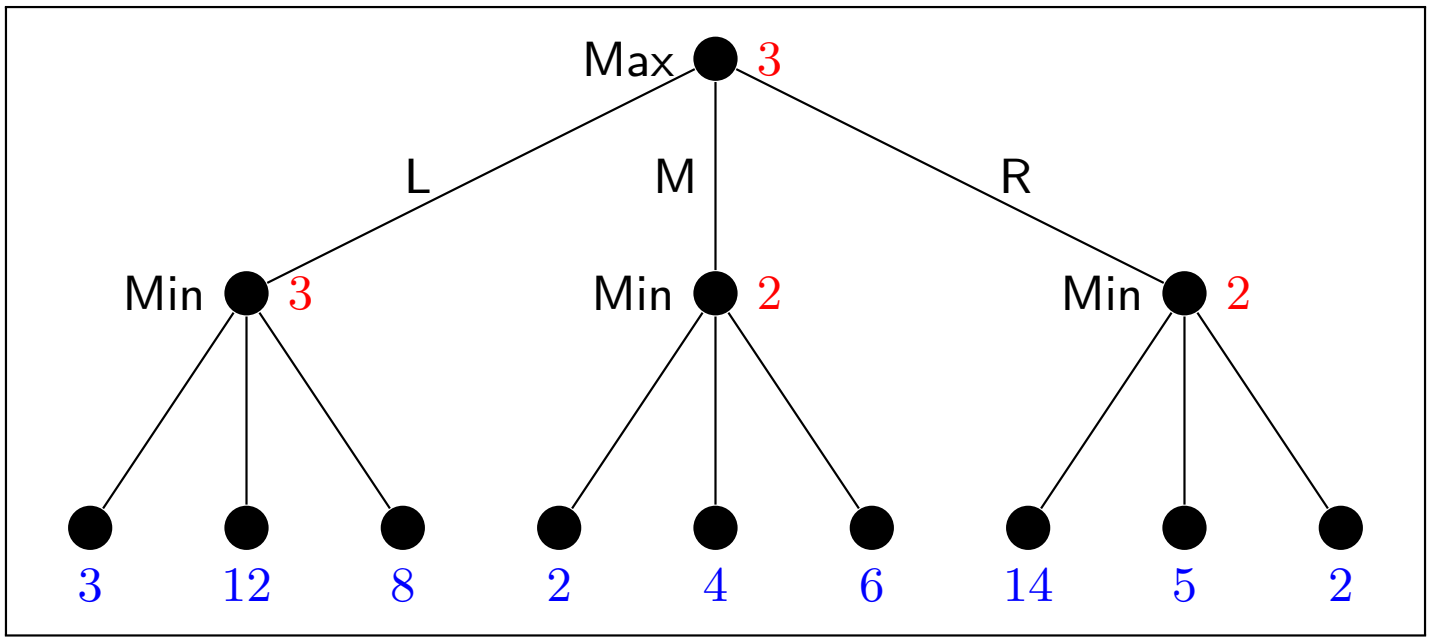
- Completely infeasible (search tree way too large).

Remedies:

- Alpha-beta pruning reduces search yet preserves optimality (not covered here).
- Limit search depth, apply evaluation function at cut-off states.
- Sparse search (MCTS) instead of exhaustive search.

Minimax With Depth Limit: Example

Notation: blue: evaluation function value on cut-off states; red: non-cut-off state value as computed by Minimax with depth limit 2.



$\rightarrow$ Search pretends that states at depth limit d (number of actions i.e. half-moves) are terminal; requires evaluation function to estimate their values (see next section).

MCTS, Action Choice for Max

Two-Player Zero-Sum MCTS: (change in red)

while time not up **do**
  select actions within T up to a state s' and $s' \xrightarrow{a'} s''$ s.t. $s'' \notin T$,
    with bias to maximize (minimize) reward in Max (Min) nodes
  rollout from s'' until terminal state t
  add s'' to T
  update, from a' up to root, #expansions and average rewards with $u(t)$
return an a for s with maximal average $reward(a)$
When executing a, keep the part of T below a

Notes:

- With suitable selection bias (e.g. UCT [Kocsis and Szepesvári (2006)]), action decisions in tree converge to optimal.
  $\Rightarrow$ Rewards converge to Minimax values.
- Sparse deep search = "focus on most relevant moves".
  $\Rightarrow$ Horizon problem not as critical. (May fall prey to "traps" though [Ramanujan *et al.* (2010)].)

Evaluation Functions

Definition: *Given a game with states S, a (heuristic) evaluation function is a function $h : S \mapsto \mathbb{R}$.*

- h estimates the expected utility of s. (In particular, we can use $h := u$ on terminal states)
- In Minimax: Impose depth limit, use h at (non-terminal) cut-off states.
- In MCTS: Use h as part of the state-value estimates. (e.g. AlphaGo: leaf state value estimate is linear combination of h and rollouts)

How to obtain h?

- Relaxed game? Possible in principle, too costly in practice.
- Encode human expert knowledge.
- Learn from data.

Position Evaluation in Chess

- Material: Pawn (Bauer) 1, Knight (Springer) 3, Bishop (Läufer) 3, Rook (Turm) 5, Queen (Dame) 9.
  $\rightarrow$ Rule of thumb:
  3 points advantage $\Longrightarrow$ safe win.
- Mobility: How many fields do you control?
- King safety, Pawn structure, ...

Linear Feature-Based Evaluation Functions

Functions taking the form:

$$h(s) := w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

f_i are features, w_i are weights.

How to obtain such functions?

- Features f_i designed by human experts.
- Weights w_i set by experts, or learned automatically (see later).

Discussion: Pro/Con

- Very fast. (Unless there are many features or computing their value is very expensive)
- Very simplistic. For example, assumes that features are independent. (But, e.g., value of Rook depends on Pawn structure)
- Human knowledge crucial in design of features.

Feature-Based Evaluation in Chess

- Material: Pawn (Bauer) 1, Knight (Springer) 3, Bishop (Läufer) 3, Rook (Turm) 5, Queen (Dame) 9.
  $\rightarrow$ Rule of thumb:
  3 points advantage $\Longrightarrow$ safe win.
- Mobility: How many fields do you control?
- King safety, Pawn structure, ...

$\rightarrow h(s) = \Delta pawn(s) + 3 * \Delta knight(s) + 3 * \Delta bishop(s) +$
$5 * \Delta rook(s) + 9 * \Delta queen(s)$ (Δ: #White$-$#Black)

$+w_k kingsafety(s) + w_p pawnstructure(s)$?

Supervised Learning of Evaluation Functions

Human expert annotates states with evaluation-function value
$\Rightarrow$ standard supervised learning problem.

- Set of annotated states, i.e., state/value pairs (s, v).
- Learn ML model that predicts output v from input s.

Possible ML methods: arbitrary ...

- Classic approach: learn weights in feature-based evaluation function.
- Recent breakthrough successes: neural networks!

Policies & Supervised Learning

Definition: *Given a game with Max states S^{Max} and actions A^{Max}, a Max policy is a function $p^{Max} : S^{Max} \mapsto A^{Max}$; symmetrically for Min. By p, we denote a (combined) policy for both players. A probabilistic policy returns a probability distribution over actions instead.*

- An optimal policy captures perfect play from both players.
- (Probabilistic) policies can be used as search guidance in MCTS: action selection in tree, action selection in rollouts.

Supervised learning of policies:

Human expert annotates states with preferred **moves**
$\Rightarrow$ standard supervised classification problem.

- Way more natural for humans; side effect of expert game play.
- e.g. KGS Go Server: 30 million positions with expert moves, used for training in AlphaGo.

Learning from Self-Play

Self-play for reinforcement learning:

Repeat: play a game using the current h and/or p; at each step (s_t, a_t) along the game trace, reinforce the game outcome in h and/or p.

- Evaluation function learning: update weights in h to reduce the error $h(s_t) - \mathit{game\text{-}outcome\text{-}value}$.
- Probabilistic policy learning: update weights in p to increase (game won)/decrease (game lost) the likelihood of choosing a_t in s_t.

Self-play to generate data for supervised learning:

Fix policy p. Repeat: play game using p; annotate the states in each game trace with the game outcome value.

- Use this data for supervised learning of evaluation function.
- Might sound strange, but actually successful: used in AlphaGo.

Reminder: 3 A.A. (Anno AlphaGo)

AlphaGo, March 2016:

- beats Lee Sedol (winner of 18 world titles).
- MCTS guided by neural networks (NN), trained by expert data and self-play.

AlphaGo Zero, early 2017:

- beats AlphaGo using NN trained without expert data.

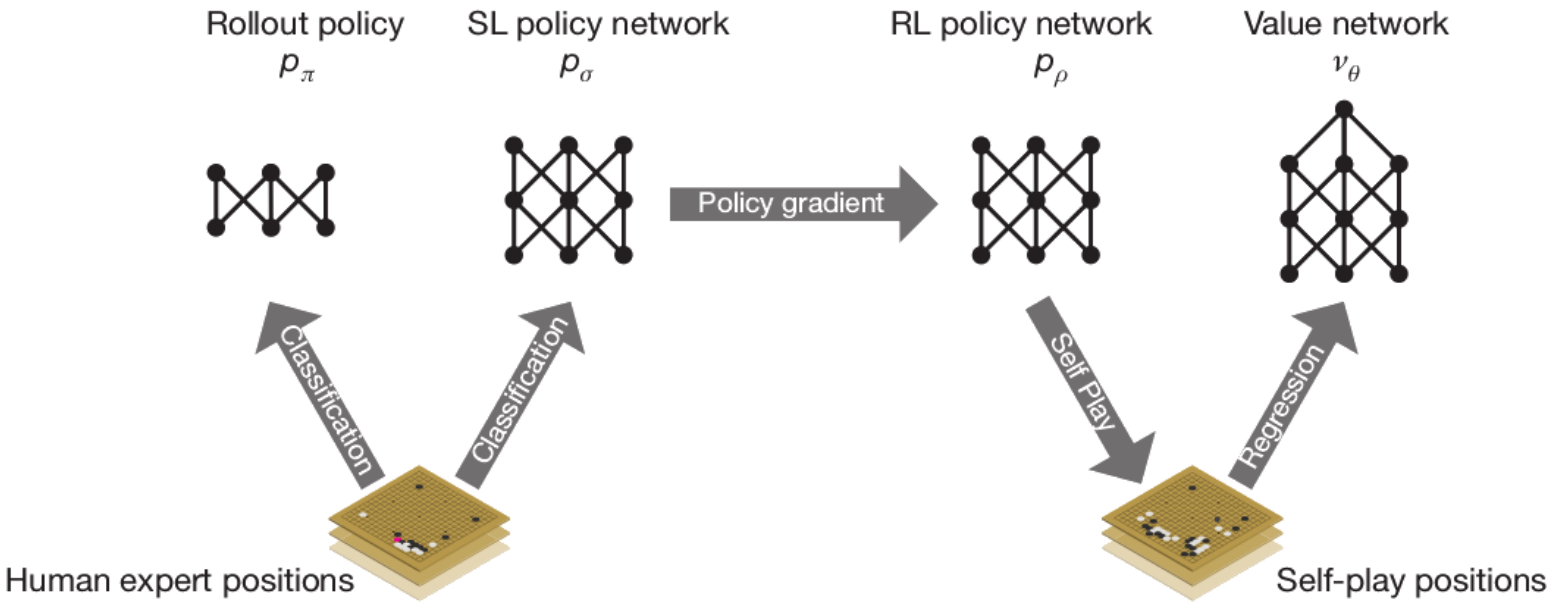
AlphaZero, late 2017:

- beats world-class computer players in Go, chess, and shogi.

. . . and now: the details! (some of them :-)

Learning in AlphaGo

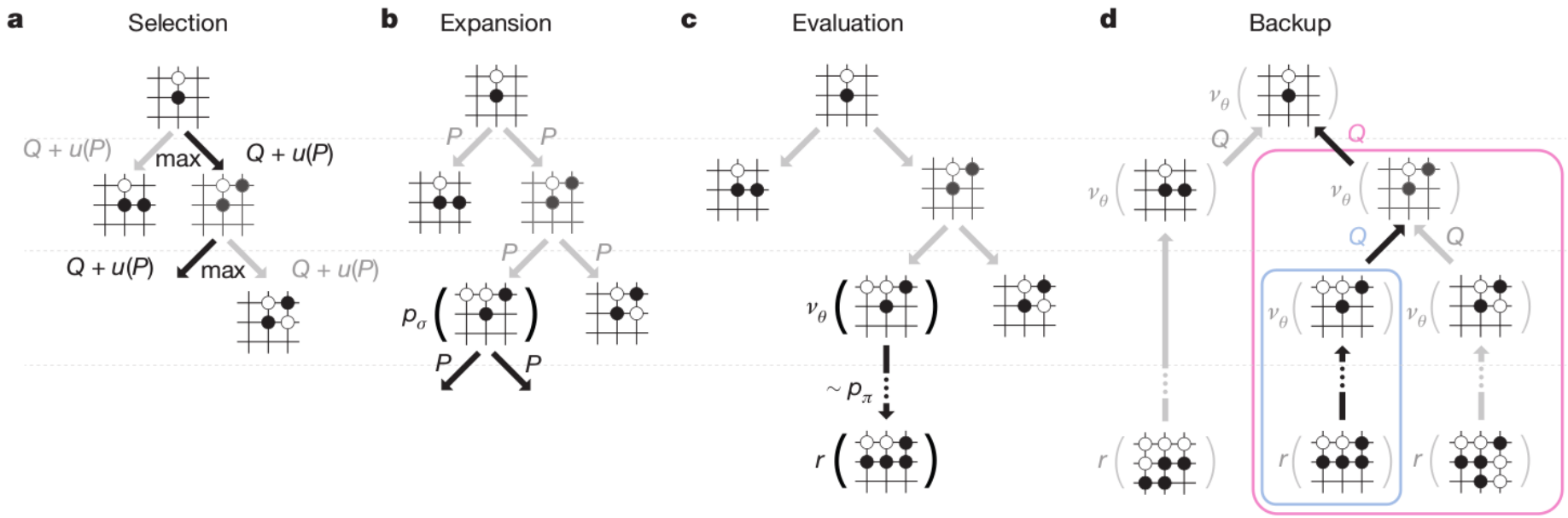
Illustration: (taken from [Silver *et al.* (2016)])



- SL policy network p_σ: Supervised learning from human expert data (cf. slide 25).
- Rollout policy p_π: Simple but fast version of p_σ (linear feature based function for each action, cf. slide 22; combined by softmax).
- RL policy network p_ρ: Start with p_σ, improve by reinforcement learning from self-play (cf. slide 26).
- Value network v_θ: Supervised learning, training data generated by self-play using p_σ (cf. slide 26).

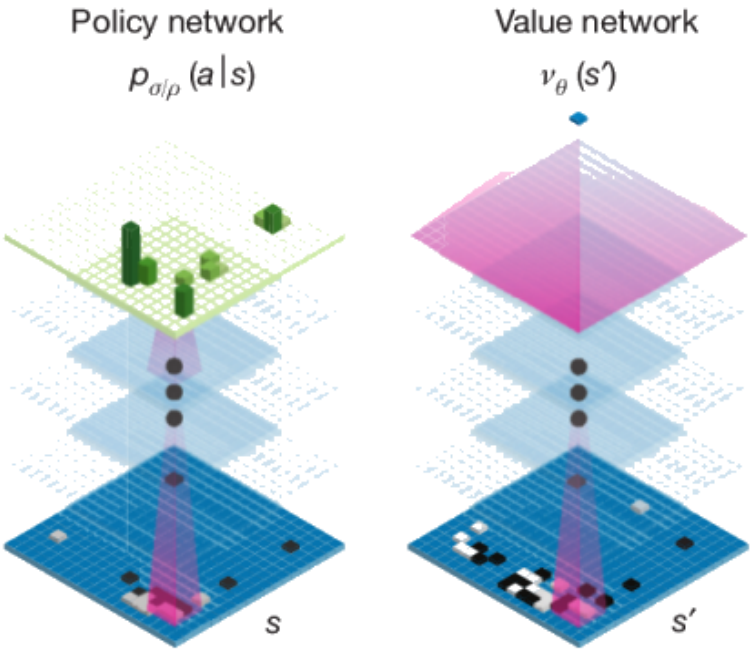
MCTS in AlphaGo

Illustration: (taken from [Silver *et al.* (2016)])



- SL policy network p_σ: Action choice bias (along with average value Q) within the tree ("P", gets smaller to "$u(P)$" with number of visits).
- Rollout policy p_π: Action choice in rollouts.
- RL policy network p_ρ: Not used here (used only to learn v_θ).
- Value network v_θ: Used to evaluate leaf states s, in weighted linear sum with the value returned by a random sample on s.

Neural Networks in AlphaGo



Neural network architecture. The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters;

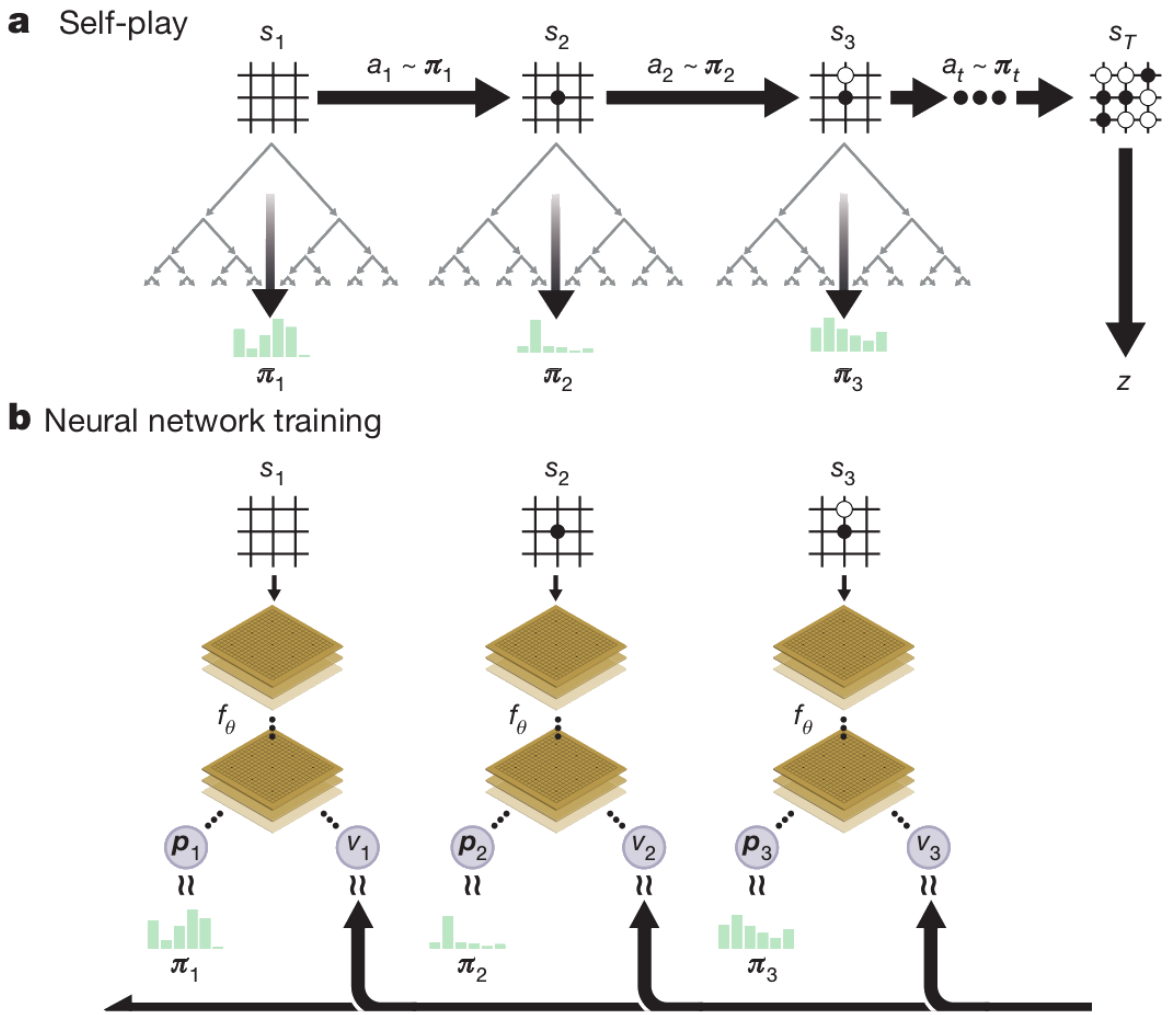
The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

(Illustration and text taken from [Silver *et al.* (2016)])

- Architecture from image classification: convolutional NN, softmax at end.
- "Image" = game board, multiple feature planes encoding game rules (stone liberties etc.) visually.
- Size "small" compared to recent results in image classification (cf. Bernt Schiele's lectures): Work done in 2014–2016, leveraging NN architecture of that time. This changes next ...

Learning in AlphaGo Zero

Illustration: (taken from [Silver *et al.* (2017)])



Self-play:

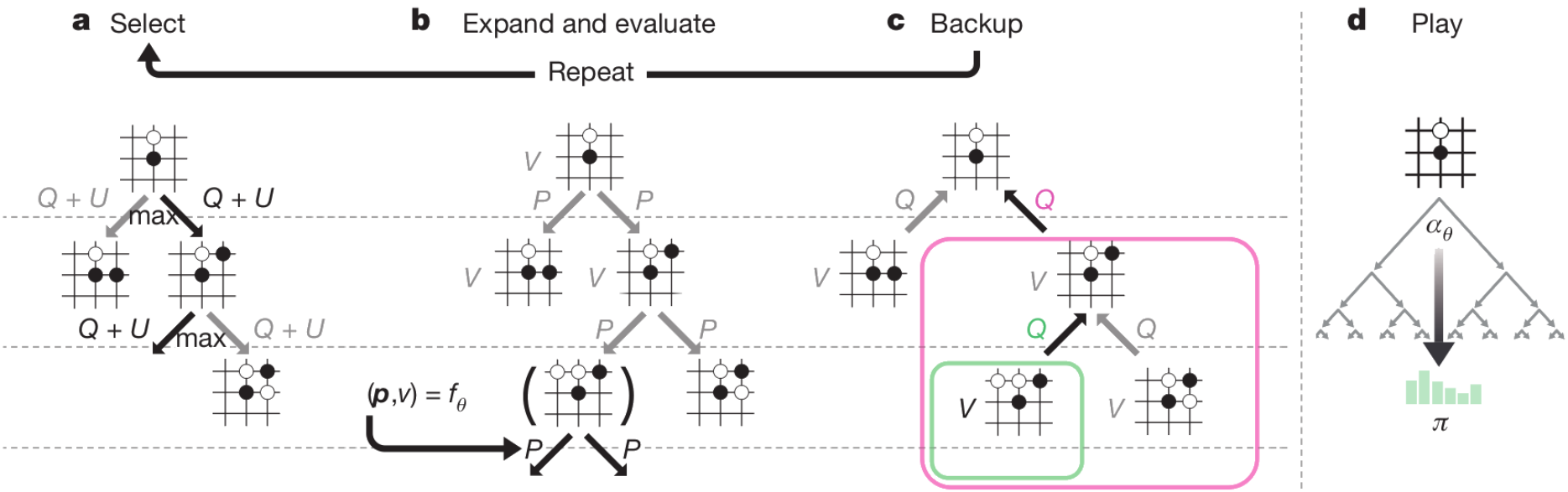
- Game: MCTS guided by current neural network.
- Learning: update weights to reduce error of v, and to move p closer to action π_i chosen by MCTS.
- MCTS controls exploitation vs. exploration trade-off for reinforcement learning.

Single neural network f_θ:

- Output (p, v): move probabilities p, value v. $\rightarrow$ Probabilistic policy *and* evaluation function.
- Residual blocks [He *et al.* (2016)], much improved performance.

MCTS in AlphaGo Zero

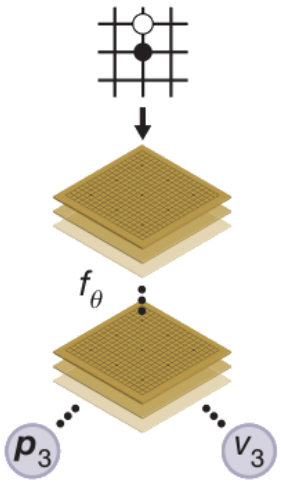
Illustration: (taken from [Silver *et al.* (2017)])



- Basically as in AlphaGo.
- Except: No rollouts! Leaf-state evaluation = NN output v.

  $\rightarrow$ Monte-Carlo tree search without "Monte-Carlo" :-) . . . like a heuristic search with MCTS-style node-expansion strategy.

Neural Network (Singular!) in AlphaGo Zero



The input features s_t are processed by a residual tower that consists of a single convolutional block followed by either 19 or 39 residual blocks[4].

The convolutional block applies the following modules:
(1) A convolution of 256 filters of kernel size 3×3 with stride 1
(2) Batch normalization[18]
(3) A rectifier nonlinearity

Each residual block applies the following modules sequentially to its input:
(1) A convolution of 256 filters of kernel size 3×3 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A convolution of 256 filters of kernel size 3×3 with stride 1
(5) Batch normalization
(6) A skip connection that adds the input to the block
(7) A rectifier nonlinearity

The output of the residual tower is passed into two separate 'heads' for computing the policy and value. The policy head applies the following modules:
(1) A convolution of 2 filters of kernel size 1×1 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A fully connected linear layer that outputs a vector of size $19^2 + 1 = 362$, corresponding to logit probabilities for all intersections and the pass move

The value head applies the following modules:
(1) A convolution of 1 filter of kernel size 1×1 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A fully connected linear layer to a hidden layer of size 256
(5) A rectifier nonlinearity
(6) A fully connected linear layer to a scalar
(7) A tanh nonlinearity outputting a scalar in the range $[-1, 1]$

The overall network depth, in the 20- or 40-block network, is 39 or 79 parameterized layers, respectively, for the residual tower, plus an additional 2 layers for the policy head and 3 layers for the value head.

(Illustration and text taken from [Silver *et al.* (2016)])

- Architecture from more recent image classification works: now including residual blocks! $\rightarrow$ Enables much deeper network.
- Evaluation function and policy are just different "heads" to the same network.
- 19 vs. 39 residual blocks: 19 in an initial system version; 39 in the final version.

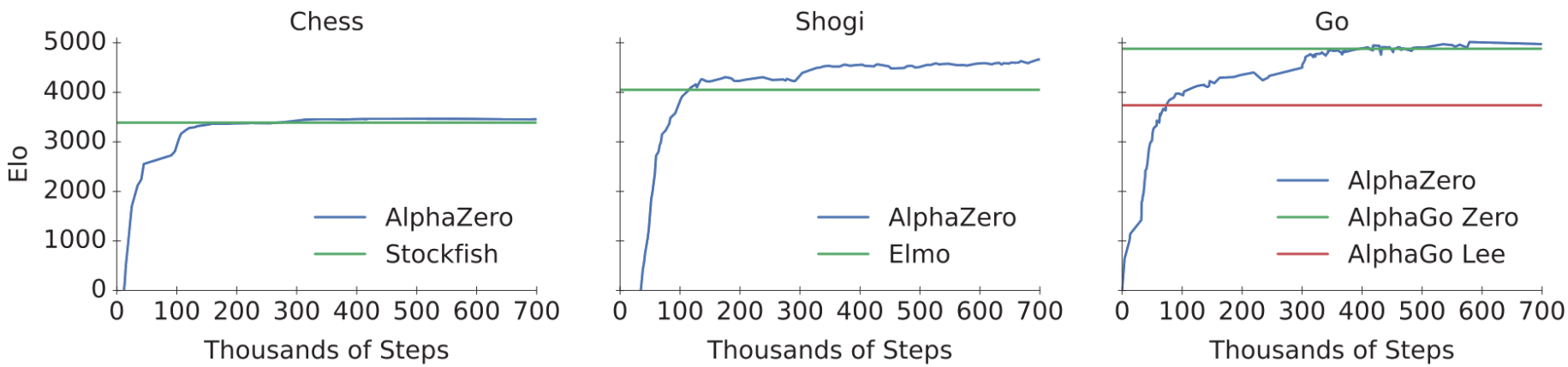
$\rightarrow$ Keys to success: 1. Integration of reinforcement learning with MCTS. 2. Leveraging recent NN architectures, in particular residual blocks.

Changes in AlphaZero

AlphaGo Zero with relatively small changes . . .

- No symmetry handling (applies only to Go).
- Various smallish details/parameters in configurations.

. . . generalizes very well to chess and shogi:



Across AlphaGo/Zero: NN Input/Output Representation

AlphaZero:

- Input: "$N \times N \times (MT + L)$ image stack ... each set of planes represents the board position at a time-step ...".
- Output: "move in chess ... $8 \times 8 \times 73$ stack of planes ... first 56 planes represent possible queen moves for any piece ...".

$\rightarrow$ Image-like representation of both, game state and moves. Crucial for success of NN methods originating in image classification.

AlphaGo Zero and AlphaGo:

- Similar.
- Just easier for Go than for chess and shogi.

AlphaGo/Zero: Conclusion

Amazing progress!

→ "Search & Learn NN" seems a great recipe for AI action decisions.

I expect lots of research on this in the coming years – in my own research group amongst many others.

Limitations: Beyond (board) games?

1. How well does this generalize to problems with no image-like structure? With incomplete information? Multiple agents? Where random play does not produce interesting data?
2. How to find "the right" hyperparameters (NN architecture etc)? Especially without > 20 full-time researchers and "5000 first-generation tensor processing units (TPUs)"?
3. In many problems, generating training data is not easy ("mild" example: autonomous driving; extreme example: NASA).

Beyond Board Games: Progress on Limitation 1

StarCraft: [Vinyals *et al.* (2019)]

- Game map ($\approx$ board) *plus* other lists/sets data (unit attributes etc).
  $\rightarrow$ "Scatter connections" in NN architecture.
- Incomplete information, need move history to judge game state.
  $\rightarrow$ LSTM NN architecture (as in natural language processing).
- Multiple collaborative and competitive agents.
  $\rightarrow$ Multi-agent self-play learning (each agent separate NN).
- Random play does not produce interesting strategies (hence self-play reinforcement learning insufficient on its own).
  $\rightarrow$ Human knowledge (supervised learning and more).
- Generalization beyond StarCraft unclear at this point.

OpenAI Dota: Related techniques; details not (yet?) available.

Summary

- Games involve several agents with (partially) conflicting utilities. Chess and Go are immensely complex, but belong to the simplest class of games: finite states, finite runs, fully observable, two-player, turn-taking, zero-sum.
- The Minimax algorithm solves such games exactly, but is infeasible in practice. MCTS can be applied by maximizing (minimizing) reward in Max (Min) nodes.
- Evaluation functions h estimate expected game-state utility. Traditional designs are feature-based, but h can be learned through supervised learning or reinforcement learning in self-play. Probabilistic policies approximate action choices, and can be learned through these same methods.
- The AlphaGo/Zero system family combines MCTS with evaluation functions and probabilistic policies represented as neural networks.

  AlphaGo involved many engineering tricks. AlphaGo simplified this, and that simplification generalizes to chess and shogi in AlphaZero.

→ "Search & Learn NN" seems a great recipe for AI action decisions. Potentially way beyond board games!

Reading

- *Chapter 5: Adversarial Search*, Sections 5.1 – 5.4 [Russell and Norvig (2010)].

  Content: Easy-to-read treatment of classical game-playing algorithms. Nothing about MCTS though, nor about learning evaluation functions/probabilistic policies.

Reading

- *Mastering the game of Go with deep neural networks and tree search* [Silver *et al.* (2016)].

  Content: The AlphaGo paper. I would not recommend reading the "METHODS" part (which compactly describes many technical details) in your first term. But the main/front part is reasonably easy to read and gives a nice overview of the system and results.

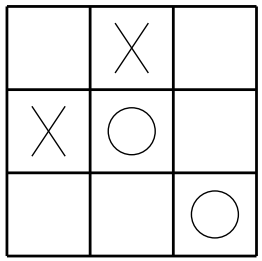
- *Mastering the game of Go without human knowledge* [Silver *et al.* (2017)].

  Content: The AlphaGo Zero paper. Remarks similar to previous.

- *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play* [Silver *et al.* (2018)].

  Content: The AlphaZero paper. Remarks similar to previous (the technical details here are in the "Supplementary Materials").

Quiz: Winners and Losers in Tic-Tac-Toe



- Tic-Tac-Toe.
- Max = x, Min = o.
- Max wins: $u = 100$; Min wins: $u = -100$; stalemate: $u = 0$.

Quiz

What's the Minimax value for the state shown above? (Note: Max to move)

(A): 100 (B): -100

$\rightarrow 100$: Max moves; choosing the top left corner, it's a certain win for Max.

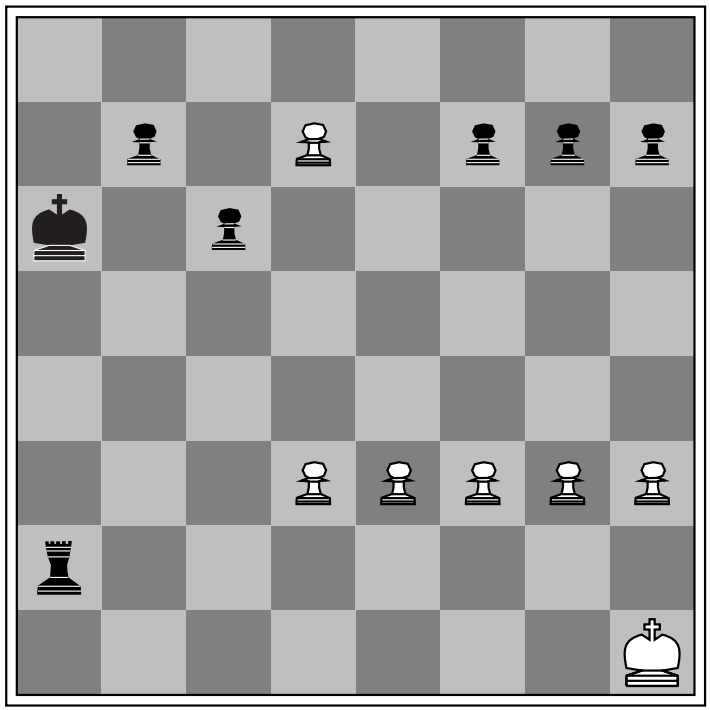
Quiz

What's the Minimax value for the initial game state?

(A): 100 (B): -100

$\rightarrow 0$: Given perfect play of both opponents, Tic-Tac-Toe always results in a stalemate. (Seen "War Games", anybody?)

Quiz: The Horizon Problem



Black to move

Quiz

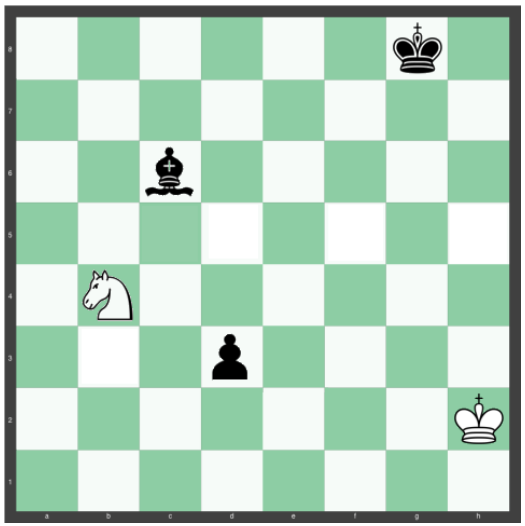
Who's gonna win here?

(A): White (B): Black

- White wins (Pawn cannot be prevented from becoming a queen.)
- Black has a $+4$ advantage in material, so if we cut-off here then our evaluation function will say "-100, black wins".
- The loss for black is beyond our horizon unless we search extremely deeply: Black can hold off the end by repeatedly giving check to White's king.

$\rightarrow$ In other words: Minimax is not robust to inaccurate cut-off evaluations.

Quiz: Feature-Based Evaluation in Chess???



- White to move.
- $h(s) = \Delta pawn(s) + 3 * \Delta knight(s) + 3 * \Delta bishop(s) + 5 * \Delta rook(s) + 9 * \Delta queen(s)$.
  (Δ: #White−#Black)

Quiz

Say Minimax with depth limit d uses h at cut-off states. Which move does it choose in this state with $d = 1$ i.e. considering only the first action? For which values of d does it choose the best action?

$\rightarrow$ With $d = 1$, Minimax chooses to capture the black bishop due to the superior material advantage.

$\rightarrow$ The best action is to capture the black pawn, as this is the only way to prevent it from turning into a queen. To see this, we need $d \geq 4$.

References I

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293, 2006.

Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On adversarial search spaces and sampling-based planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 242–245, 2010.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. 2010.

References II

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

References III

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michal Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Remi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wnsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019.