Artificial Intelligence

12. General Game Playing

One AI to Play All Games and Win Them All

Jana Koehler Álvaro Torralba

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summer Term 2019

Thanks to Dr. Peter Kissmann for slide sources

Agenda

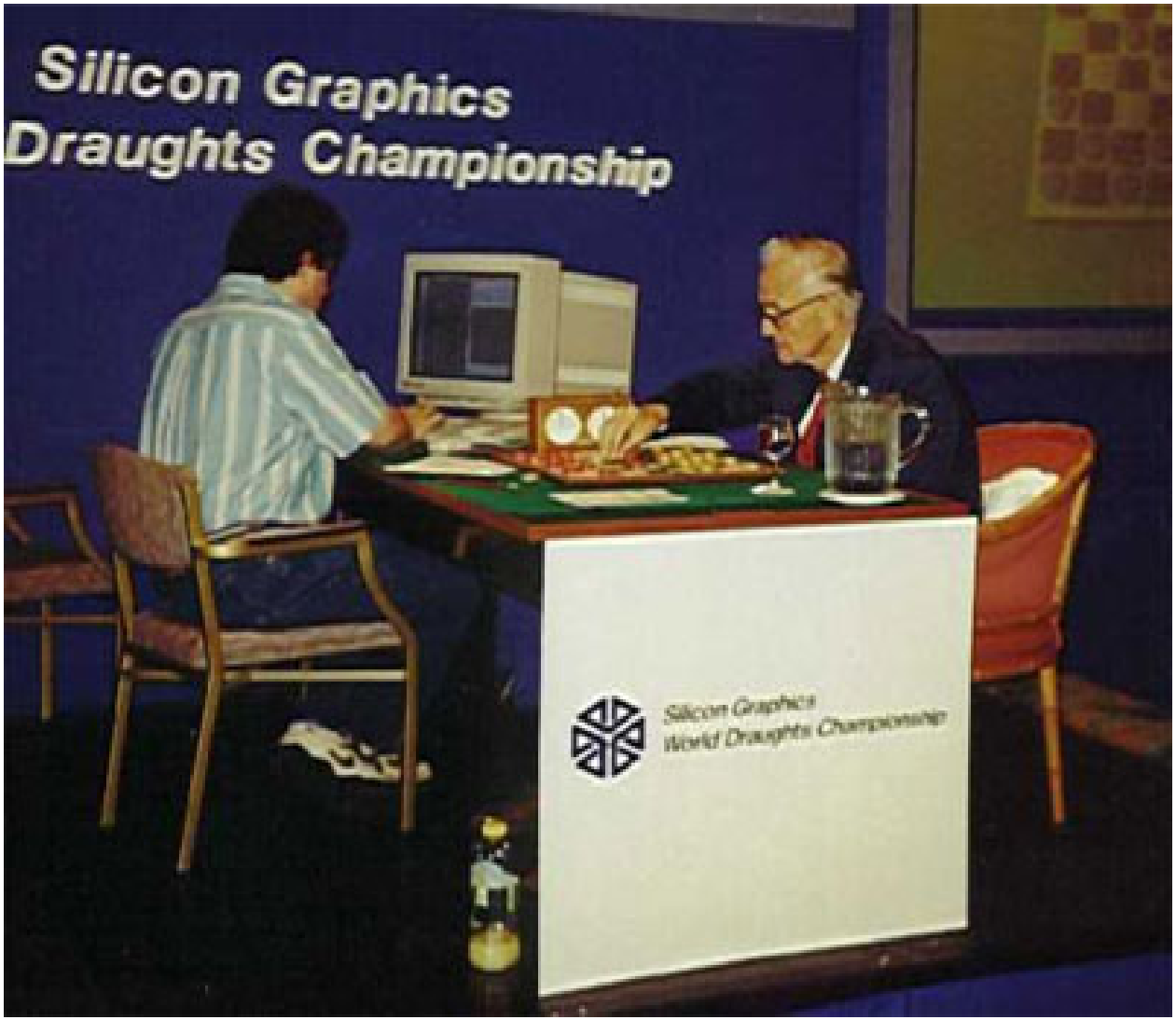
1. Introduction
2. The Game Description Language (GDL)
3. Playing General Games
4. Learning Evaluation Functions: Alpha Zero
5. Conclusion

Deep Blue Versus Garry Kasparov (1997)

Games That *Deep Blue* Can Play

1. Chess

Chinook Versus Marion Tinsley (1992)

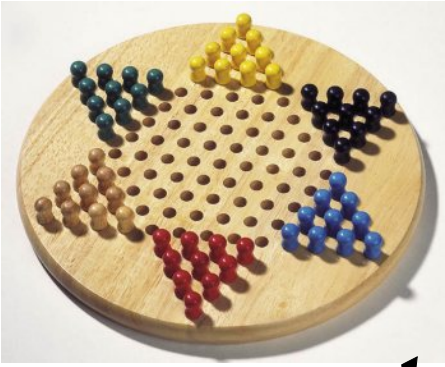


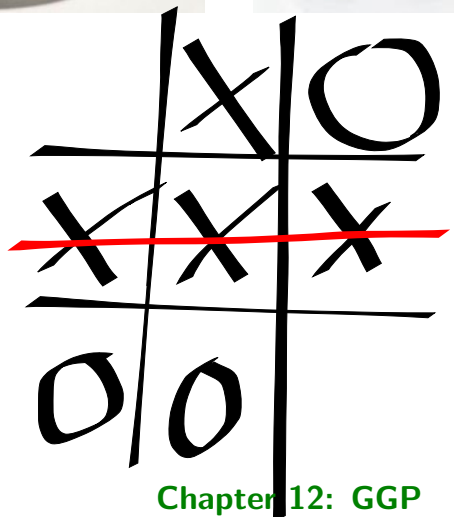
Games That *Chinook* Can Play

1. Checkers

Games That a General Game Player Can Play

1. Chess
2. Checkers
3. Chinese Checkers
4. Connect Four
5. Tic-Tac-Toe
6. ...





Games That a General Game Player Can Play (Ctd.)

5. ...
6. Othello
7. Nine Men's Morris
8. 15-Puzzle
9. Nim
10. Sudoku
11. Pentago
12. Blocker
13. Breakthrough
14. Lights Out
15. Amazons
16. Knightazons
17. Blocksworld
18. Zhadu
19. Pancakes
20. Quarto
21. Knight's Tour
22. n-Queens
23. Blob Wars
24. Bomberman (simplified)
25. Catch a Mouse
26. Chomp
27. Gomoku
28. Hex
29. Cubicup
30. ...

Games That a General Game Player Can Play (Ctd.)

94. ...
95. Bidding Tic-Tac-Toe
96. 9-Board Tic-Tac-Toe
97. Finding the way out of a labyrinth
98. Rock-Paper-Scissors
99. Iterated Prisoner's Dilemma

And many, many more...

General Game Playing

General game players can

- read and process arbitrary game descriptions
- play efficiently based on the game descriptions

Difference to specialized players (such as *Deep Blue*, *Chinook*, etc.):

General game players

- have no special knowledge about game at hand
- must establish any knowledge on their own

How *General* are General Games?

Nowadays: Game Description Language (GDL) [Love *et al.* (2008)]

Difference to Adversarial Search

- Any number of players (≥ 1)
- Simultaneous move games only (turn-taking games modeled by use of *noops*, i.e., moves that do not change the current state)
- Rewards integers in [0,100] for all players; not necessarily zero-sum

Some Limitations Remain

- finite
- discrete
- deterministic (i.e., no games of chance such as dice games)
- full information for all players (i.e., prohibits most card games)

Last two points covered by GDL-II (GDL for Incomplete Information) [Thielscher (2010)], though rarely used.

How to Describe a General Game?

Adversarial Search Game State Space

Recall: $\Theta = (S, A, T, I, S^T, u)$ (S: states/positions, A: actions/moves, T: transition relation, I: initial state, S^T: terminal states, u: utility function).

Minimal Requirements in GGP

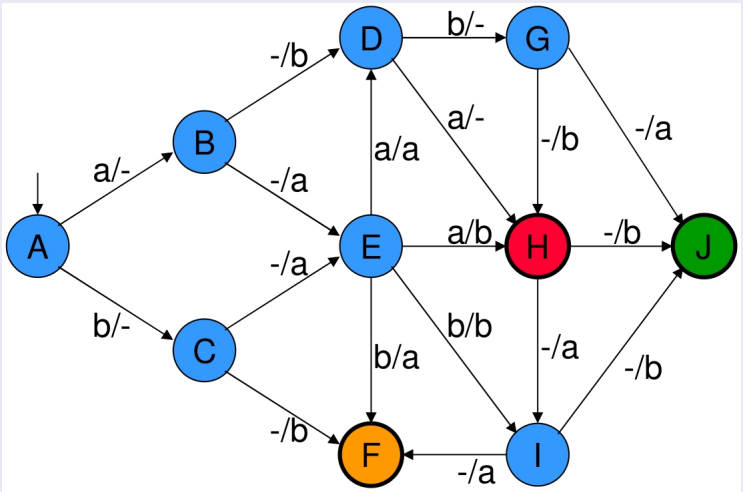
- State description
- Number of players
- Initial state
- Legal moves (for all players given current state) – the actions/moves
- Successor generation (given current state and all players' moves) – the transition relation
- Termination (given current state)
- Rewards (for all players given current state) – the utility

How to Describe a General Game? (Ctd.)

Full State Space

Problems:

- State-space explosion problem
  - *Tic-Tac-Toe*: $5{,}478$
  - *Connect Four*: $4{,}531{,}985{,}219{,}092 \approx 4.5 \times 10^{12}$
  - *15-Puzzle*: $10{,}461{,}394{,}944{,}000 \approx 10^{13}$
- Thus, immense memory requirement $\rightarrow$ not practical



Alternative: Logical Input (Declarative Description)

Much more compact; easier to use for analysis.
Here, *Game Description Language* (GDL) used [Love *et al.* (2008)]

The Game Description Language (GDL) – States

Representation of States

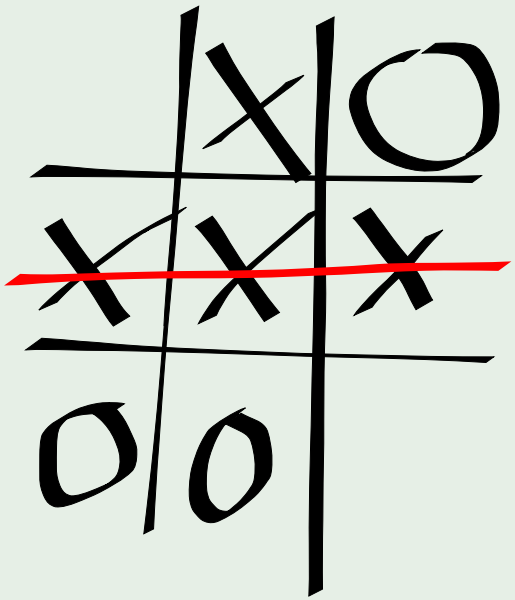
States represented by set of predicates of the form

```
(true <fact>)
```

meaning that `<fact>` is true in the current state.

Example (Tic-Tac-Toe)

```
{(true (cell 1 1 o)),
(true (cell 2 1 o)),
(true (cell 3 1 b)),
(true (cell 1 2 x)),
(true (cell 2 2 x)),
(true (cell 3 2 x)),
(true (cell 1 3 b)),
(true (cell 2 3 x)),
(true (cell 3 3 o))}
```



GDL – Rules

Rules in GDL

GDL based on *Datalog*. It uses (logical) *rules*. A rule is a kind of implication consisting of a head predicate and a body, which in turn consists of a number of predicates `<body_1>`, ..., `<body_n>`:

```
(<head> ⇐
    <body_1>
    ...
    <body_n>
)
```

- head is true if every part of body is true or body is empty.
- predicates can have arguments; may be constants or variables, latter denoted by prefix '?'
- in evaluation, a variable can be replaced by any constant, but all appearances of a variable in one rule must be replaced by the same constant.

GDL – Rules (Ctd.)

Example (Tic-Tac-Toe)

```
((row ?r ?p) ⇐                    Row ?r completely marked with ?p
    (true (cell 1 ?r ?p))
    (true (cell 2 ?r ?p))         if every cell in that row
    (true (cell 3 ?r ?p))         marked with ?p
)
```

Evaluation with our state for ?r = 1 and ?p = o:

Example (Tic-Tac-Toe)

```
((row 1 o) ⇐
    (true (cell 1 1 o))
    (true (cell 2 1 o))
    (true (cell 3 1 o))
)
```

```
{(true (cell 1 1 o)),
(true (cell 2 1 o)),
(true (cell 3 1 b)),
(true (cell 1 2 x)),
(true (cell 2 2 x)),
(true (cell 3 2 x)),
(true (cell 1 3 b)),
(true (cell 2 3 x)),
(true (cell 3 3 o))}
```

GDL – Rules (Ctd.)

Example (Tic-Tac-Toe)

```
((row ?r ?p) ⇐                    Row ?r completely marked with ?p
    (true (cell 1 ?r ?p))
    (true (cell 2 ?r ?p))         if every cell in that row
    (true (cell 3 ?r ?p))         marked with ?p
)
```

Evaluation with our state for ?r = 2 and ?p = x:

Example (Tic-Tac-Toe)

```
((row 2 x) ⇐
    (true (cell 1 2 x))
    (true (cell 2 2 x))
    (true (cell 3 2 x))
)
```

```
{(true (cell 1 1 o)),
(true (cell 2 1 o)),
(true (cell 3 1 b)),
(true (cell 1 2 x)),
(true (cell 2 2 x)),
(true (cell 3 2 x)),
(true (cell 1 3 b)),
(true (cell 2 3 x)),
(true (cell 3 3 o))}
```

GDL – Rules (Ctd.)

We can also iterate such rules.

Example (Tic-Tac-Toe)

`((line ?p) ⇐ (row ?r ?p))`	Line of three equal symbols `?p`, if some row `?r` has 3 `?p`, or
`((line ?p) ⇐ (column ?c ?p))`	some column `?c` has 3 `?p`, or
`((line ?p) ⇐ (diagonal ?p))`	some diagonal has 3 `?p`.

To evaluate `line`, we first must evaluate `row`. We already know that `(row 2 x)` holds, so `(line x)` holds as well.

GDL – Players

Players

For each player, a rule of the form

```
(role <player>)
```

must be specified.

Example (Tic-Tac-Toe)

```
(role xplayer)
(role oplayer)
```

Thus, we have two players, `xplayer` and `oplayer`

GDL – Initial State

Initial State

Set of all rules of the form

```
(init <fact>)
```

build the initial state (each such `<fact>` holds).

Example (Tic-Tac-Toe)

`(init (cell 1 1 b))`	Board
`...`	initially
`(init (cell 3 3 b))`	empty (`b`)
`(init (control xplayer))`	`xplayer` is starting player

Note: `control` is no GDL keyword; the player has to derive the connection between this fact and the player to move automatically.

GDL – Initial State (Ctd.)

In evaluation of rules we need `true` predicates:
To set the current state to the initial one, we must transform all `(init <fact>)` to `(true <fact>)` in our internal representation

Example (Tic-Tac-Toe)

```
(init (cell 1 1 b))                 {(true (cell 1 1 b)),
(init (cell 2 1 b))                 (true (cell 2 1 b)),
(init (cell 3 1 b))                 (true (cell 3 1 b)),
(init (cell 1 2 b))                 (true (cell 1 2 b)),
(init (cell 2 2 b))                 (true (cell 2 2 b)),
(init (cell 3 2 b))        ⇒        (true (cell 3 2 b)),
(init (cell 1 3 b))                 (true (cell 1 3 b)),
(init (cell 2 3 b))                 (true (cell 2 3 b)),
(init (cell 3 3 b))                 (true (cell 3 3 b)),
(init (control xplayer))            (true (control xplayer))}
```

GDL – Legal Moves

Legal Moves

A <player> can perform a certain <move> if the body of a rule

```
((legal <player> <move>) ⇐ <body>)
```

is evaluated to true.

Example (Tic-Tac-Toe)

```
((legal xplayer (mark ?x ?y)) ⇐
    (true (control xplayer))
    (true (cell ?x ?y b))
)
```

xplayer may mark cell (x, y), if it is xplayer's turn to move and cell (x, y) is currently empty (b).

Example (Tic-Tac-Toe)

```
((legal oplayer noop) ⇐
    (true (control xplayer))
)
```

oplayer may do nothing (noop), if it is xplayer's turn to move. Note: noop is no GDL keyword.

GDL – Chosen Moves

Representation of Chosen Moves

When we know that a <player> chose to perform some <move> we add a predicate of the form

```
(does <player> <move>)
```

to our internal representation of the current state.

Example (Tic-Tac-Toe)

```
{(true (cell 1 1 b)),
(true (cell 2 1 b)),
(true (cell 3 1 b)),
(true (cell 1 2 b)),
(true (cell 2 2 b)),
(true (cell 3 2 b)),
(true (cell 1 3 b)),
(true (cell 2 3 b)),
(true (cell 3 3 b)),
(true (control xplayer))}
```

```
xplayer: (mark 1 1)
   oplayer: noop
```

$\Rightarrow$

```
{(true (cell 1 1 b)),
(true (cell 2 1 b)),
(true (cell 3 1 b)),
(true (cell 1 2 b)),
(true (cell 2 2 b)),
(true (cell 3 2 b)),
(true (cell 1 3 b)),
(true (cell 2 3 b)),
(true (cell 3 3 b)),
(true (control xplayer)),
(does xplayer (mark 1 1)),
(does oplayer noop)}
```

GDL – Successor Generation

Successor Generation

Each <fact> that will be true in the successor state must appear in the head of a next rule

((next <fact>) ⇐ <body>)

whose body is evaluated to true in the current state given the chosen moves.

Example (Tic-Tac-Toe)

```
((next (cell ?m ?n x)) ⇐
    (does xplayer (mark ?m ?n))
)
```

(m, n) will be marked with x, if xplayer chose to mark it now.

Example (Tic-Tac-Toe)

```
((next (control oplayer)) ⇐
    (true (control xplayer))
)
```

It will be oplayer's turn next, if it was xplayer's turn now.

GDL – Successor Generation (Ctd.)

Important

Everything that can be derived from the `next` rules is true in the successor state.
Nothing else is true in the successor state.

Example (Tic-Tac-Toe)

If we have only one `next` rule:

```
((next (cell ?m ?n x)) ⇐
    (does xplayer (mark ?m ?n)))
```

If the `xplayer` chose to `(mark 1 1)`, we know that in the successor state `(true (cell 1 1 x))` holds.
However, nothing else holds – there is no memory involved. The old state is completely discarded and the successor state created based only on the outcome of the `next` rules.

GDL – Successor Generation (Ctd.)

Important

Thus, it is necessary that all things that do not change are explicitly modeled as well.

Example (Tic-Tac-Toe)

```
((next (cell ?m ?n x)) ⇐
    (true (cell ?m ?n x))
)
((next (cell ?m ?n o)) ⇐
    (true (cell ?m ?n o))
)
```

In the next state a cell will remain marked with x, if it is marked so in the current state. Same for o.

GDL – Successor Generation (Ctd.)

Generating Successor State

Having evaluated the `next` rules, we must store the successor state as the new current state.

Example (Tic-Tac-Toe)

```
{(true (cell 1 1 b)),
(true (cell 2 1 b)),
(true (cell 3 1 b)),
(true (cell 1 2 b)),
(true (cell 2 2 b)),
(true (cell 3 2 b)),
(true (cell 1 3 b)),
(true (cell 2 3 b)),
(true (cell 3 3 b)),
(true (control xplayer)),
(does xplayer (mark 1 1)),
(does oplayer noop)}
```

evaluate next rules ⇒

```
{(next (cell 1 1 x)),
(next (cell 2 1 b)),
(next (cell 3 1 b)),
(next (cell 1 2 b)),
(next (cell 2 2 b)),
(next (cell 3 2 b)),
(next (cell 1 3 b)),
(next (cell 2 3 b)),
(next (cell 3 3 b)),
(next (control oplayer))}
```

set succ. state ⇒

```
{(true (cell 1 1 x)),
(true (cell 2 1 b)),
(true (cell 3 1 b)),
(true (cell 1 2 b)),
(true (cell 2 2 b)),
(true (cell 3 2 b)),
(true (cell 1 3 b)),
(true (cell 2 3 b)),
(true (cell 3 3 b)),
(true (control oplayer))}
```

GDL – Termination

Termination

The game ends when the body of any rule

(terminal ⇐ <body>)

is evaluated to true.

Example (Tic-Tac-Toe)

	The game ends, if
(terminal ⇐ (line x))	we have a line of 3 xs.
(terminal ⇐ (line o))	Or 3 os.
(terminal ⇐ (boardfilled))	Or no cell is empty anymore.

GDL – Rewards

Rewards

A <player> will get a specified <reward> if the body of a rule

((goal <player> <reward>) ⇐ <body>)

is evaluated to true.

In GDL, rewards are integers in the range [0,100], with 100 being best.

Example (Tic-Tac-Toe)

((goal xplayer 100) ⇐ (line x))

xplayer gets 100 points if we have a line of 3 xs on the board.

Example (Tic-Tac-Toe)

((goal oplayer 0) ⇐ (line x))

In the same situation, oplayer gets 0 points.

GDL – Rewards (Ctd.)

General games allow much more general outcomes than in our adversarial search setting:

- (partly) cooperative games (i.e., several players get same reward → should work together to achieve highest possible outcome)
- different rewards depending on actual outcome – not necessarily zero-sum
- classical zero-sum games – here in the form of constant-sum games (the sums of the players' rewards always sum up to 100).

To transform zero-sum game from GGP setting to our adversarial search setting: Set utility to reward(first player) – reward(second player)

Example – Chomp

Reminder: Minimal Requirements

- State description
- Number of players
- Initial state
- Legal moves
- Successor generation
- Termination
- Rewards

```
(role eater)
(role muncher)

(init (chocolate 1 1)) [...]  (init (chocolate 8 7))
(init (control eater))

((legal ?player (bite ?x ?y)) ⇐
    (true (control ?player))
    (true (chocolate ?x ?y)))

((legal eater noop) ⇐ (true (control muncher)))
((legal muncher noop) ⇐ (true (control eater)))

((next (chocolate ?x ?y)) ⇐
    (true (chocolate ?x ?y))
    (does ?player (bite ?i ?j))
    (less_than ?x ?i))

((next (chocolate ?x ?y)) ⇐
    (true (chocolate ?x ?y))
    (does ?player (bite ?i ?j))
    (less_than ?y ?j))

((next (control eater)) ⇐ (true (control muncher)))
((next (control muncher)) ⇐ (true (control eater)))

((next (poisoned ?p)) ⇐ (does ?p (bite 1 1)))

(terminal ⇐ (true (poisoned ?p)))

((goal eater 100) ⇐ (true (poisoned muncher)))
((goal muncher 100) ⇐ (true (poisoned eater)))

((goal ?player 0) ⇐ (true (poisoned ?player)))

((less_than ?n ?m) ⇐ (succ ?n ?m))
((less_than ?n ?m) ⇐
    (succ ?k ?m)
    (less_than ?n ?k))

(succ 1 2) [...]  (succ 7 8)
```

Example – Chomp (Ctd.)

Side-Remark

Chomp is won by the starting player – no matter the size of the rectangular board.
Proof by strategy-stealing argument:

- Assume, the second player had a winning strategy
- Then that one must be able to find a winning move for every possible first move
- Assume that the first player bites only one piece (i.e., the top-right position)
- With this we know that the second player must have a winning move
- However, any move that the second player can play here could have been played by the first player in the initial state
- Thus, it must be the first player who has such a winning strategy

How Does GGP Work?

1. GGP server sends game description + `starttime` + `movetime` to players
2. players have `starttime` seconds for initial evaluation
3. players have `movetime` seconds for evaluation, then send `move` to server
4. server updates state
5. if `terminal`, goto step 9
6. server informs all players about all performed moves
7. players update their current state
8. continue with step 3
9. server sends `STOP` along with the chosen moves to all players; game ends

Let's Play a Game

http://ggpserver.general-game-playing.de

- Current form of GGP introduced around 2005 by Games-group of Stanford University
- Since 2005 annual GGP competitions at top-level AI conferences

Question!

What algorithm would you use for GGP:

1. Minimax
2. AlphaBeta
3. MCTS

Playing General Games

In earliest competitions (2005 and 2006) most successful players performed Alpha-Beta search. Alpha-Beta quite general but:...

- only defined for two-player turn-taking zero-sum games
- difficult to find reasonable automatically generated evaluation function (cannot use expert knowledge)

Rules of Chess $\rightarrow$ Pawn (Bauer) 1, Knight (Springer) 3, Bishop (Läufer) 3, Rook (Turm) 5, Queen (Dame) 9.

Since 2007 all winners of competition used UCT: though there is still a lot of margin of improvement ...

Lack of Evaluation Functions: Learning to the Rescue

(description of a game in modeling language)

training phase

(current position)

(knowledge about the game) $\Rightarrow$ general game player

(play)

Alpha Zero

Chess	Go	Shogi
1997	2016	2017
DeepBlue vs. Kasparov	AlphaGo vs. Lee Sedol	Bonanza vs. 9-dan

2017: AlphaZero trains itself in a few hours to play chess, go, and shogi.

Then, it defeats current best programs:

- Chess: Stockfish[1] (+28, =72, -0)
- Go: AlphaGoZero (+60, -40)
- Shogi: Elmo (+90, =2, -8)

[1] Not 100% clear if the setting was fair for Stockfish.

From Alpha Go to Leela Zero

Common aspects: combine MCTS and Neural Networks

- AlphaGo: specialized program to play Go
  - Used games from Go profesionals as training data
  - Several neural networks with a complicated architecture
- AlphaZero: common architecture for Chess, Go, and Shogi
  - Simpler than AlphaGo: a single neural network per game
  - Reinforcement learning from scratch without additional training data
  - But: not open source and only published partial data, used TPU's
- Leela Zero: open-source version of AlphaZero
  - Standard hardware (CPU and GPU)
  - LeelaChessZero won the TCEC Season 15 in May'19 (after training for more than a year, more than 200 million games.)

AlphaGo: Overview

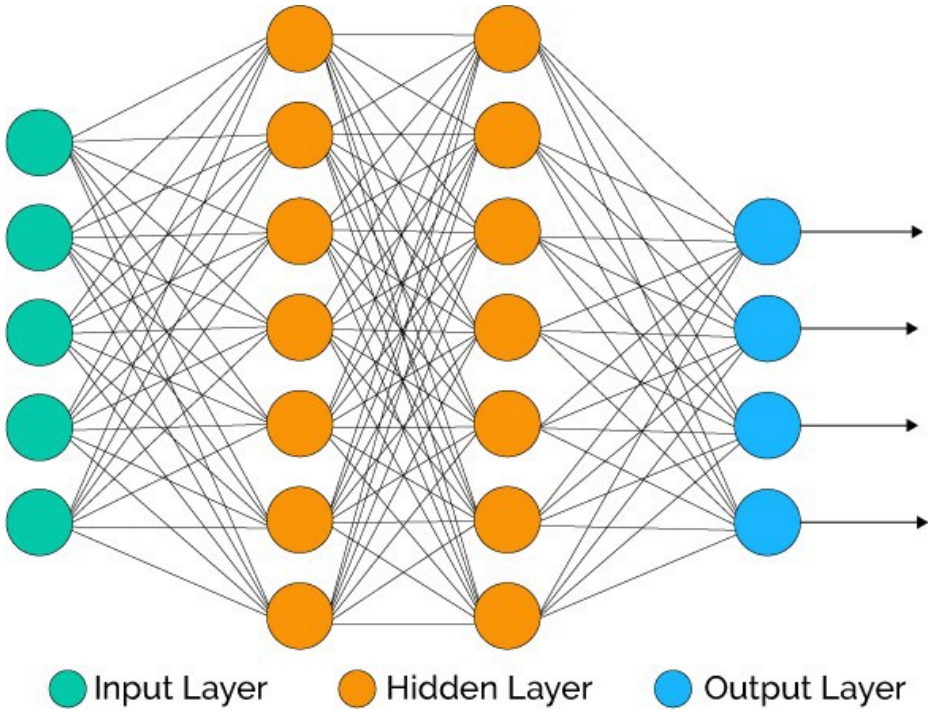
Neural Networks:

- Policy networks: Given a state s, output a probability distibution over the actions applicable in s.
- Value networks: Given a state s, outpout a number estimating the game value of s.

Combination with MCTS:

- Policy networks bias the action choices within the MCTS tree (and hence the leaf-state selection), and bias the random samples.
- Value networks are an additional source of state values in the MCTS tree, along with the random samples.

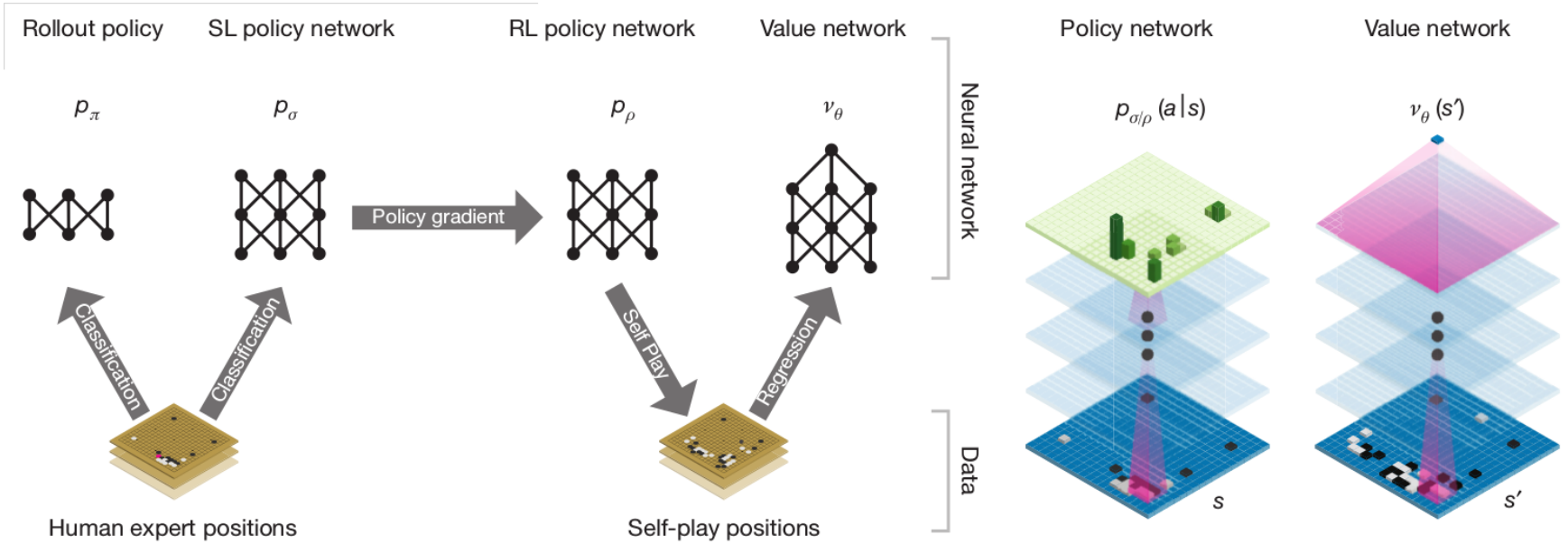
Neural Networks



- Input layer: Description of the game state
- Output layer: What we want to predict (e.g. utility of the state in value networks, probability of a in policy networks)
- Supervised Learning: Given a set of training data (positions for which we know their utility), configure the net so that the error is minimized for those positions.

Neural Networks in AlphaGo

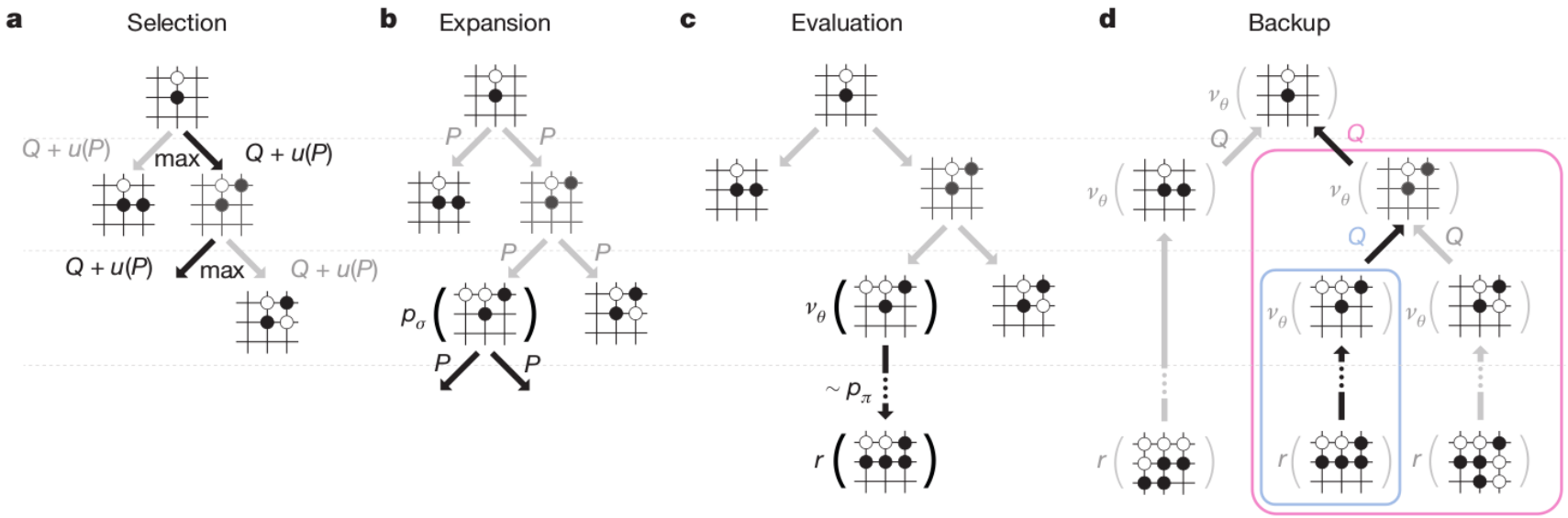
Illustration: (taken from [Silver *et al.* (2016)])



- *Rollout policy* p_π*:* Simple but fast, $\approx$ prior work on Go.
- *SL policy network* p_σ*:* Supervised learning, human-expert data ("learn to choose an expert action").
- *RL policy network* p_ρ*:* Reinforcement learning, self-play ("learn to win").
- *Value network* v_θ*:* Use self-play games with p_ρ as training data for game-position evaluation v_θ ("predict which player will win in this state").

Neural Networks + MCTS in AlphaGo

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



- *Rollout policy* p_π*:* Action choice in random samples.
- *SL policy network* p_σ*:* Action choice bias within the UCTS tree (stored as "P", gets smaller to "$u(P)$" with number of visits); along with quality Q.
- *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 linear sum with the value returned by a random sample on s.

Alpha Zero: MCTS

- Neural Network: function $f(s)$ that returns $p_a = P(a|s)$ for all $a \in A$, plus expected utility from position s.
- Run Monte-Carlo Tree Search with the following modifications:
  - In leaf nodes, the neural network is evaluated (no random play until end of the game).
  - When traversing the tree, the probability of choosing a depends on p_a, apart from the number of times that a was previously selected and the estimated utility of its outcome.
- Why not use Alpha Beta with such an evaluation function?

  $\rightarrow$Taking average of evaluation function reduces the impact of the "noise" in the learnt evaluation function

Alpha Zero: Training Phase

- Goal: Learn function $f(s)$ that returns $p_a = P(a|s)$ for all $a \in A$, plus expected utility from position s.
- f is learnt by playing games against itself. After each game, f is updated to:
  1. Minimize the error between expected and real outcome.
  2. Minimize the difference between p_a and probability of a to be selected by MCTS in s.
- Requires very little domain knowledge:
  - Rules of the game (BlackBox formulation)
  - The neural network architecture is matched to the grid-structure of the board (in a non trivial way)
  - The typical number of legal moves is used to scale exploration noise
  - Long games are considered draws

AlphaZero, Conclusion?

Opinions?

- **My 5 cents:** Definitely a great achievement.
- Lots of questions to explore. OpenSource project for Go (https://github.com/leela-zero/leela-zero) and chess(`http://lczero.org/`)
- "Search + neural networks" looks like a great formula for general problem solving.

  → I expect lots of research on this in the coming decade(s). (In FAI, amongst others.)
- AlphaZero for GGP?
  AlphaZero is simpler than AlphaGo but its design is still quite intricate (architecture, neural network architectures, ...).

  → Still some human expertise in here.

  → How much of this is reusable in/generalizes to other problems?

Summary

- GGP is an extension of classical search problems and adversarial search problems: Support for any number of players, simultaneous moves and non-zero-sum utilities.
- Programmer does not know which game will be played, thus domain-independent algorithms must be used (no expert knowledge).
- Game Description Language GDL used for concise description of general games.

Discussion

- Alpha-Beta used early on. Later, change of paradigm to simulation-based search (UCT) – very fast and no expert knowledge needed.
- However, UCT bad if many *traps* in search space
- Research in general evaluation functions not far enough to beat UCT based players consistently
- Several ways overcoming this in current research:
  - Use pure Alpha-Beta search, find efficient general evaluation functions (e.g. learning as in AlphaZero).
  - Use UCT and enhance it with Minimax-based approaches (e.g., propagate Minimax values instead of averages within search tree, and/or use evaluation function instead of playouts with some probability) to bridge the gap between these approaches.

Web Sources

- Lots of additional information (e.g., GGP lecture slides, literature, basic players) collected at `http://www.general-game-playing.de`
- GGP Server online (originally developed at TU Dresden):
  - `http://ggpserver.general-game-playing.de`
  - Large number of game descriptions (currently more than 290)
  - Everyone can register
  - Possibility of self-play
  - Possibility to register player (and let it play)
  - Round-robin all the time
  - Often used in competitions

References I

Nathaniel C. Love, Timothy L. Hinrichs, and Michael R. Genesereth. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford Logic Group, 2008.

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.

Michael Thielscher. A general game description language for incomplete information games. In Maria Fox and David Poole, editors, *Proceedings of the 24th National Conference of the American Association for Artificial Intelligence (AAAI'10)*, pages 994–999, Atlanta, GA, USA, July 2010. AAAI Press.