Foundations of Artificial Intelligence

41. Board Games: Minimax Search and Evaluation Functions

Thomas Keller and Florian Pommerening

University of Basel

May 17, 2023

Board Games: Overview

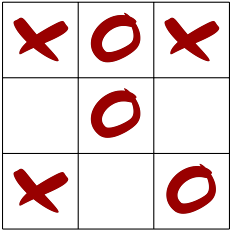
chapter overview:

- 40. Introduction and State of the Art
- 41. Minimax Search and Evaluation Functions
- 42. Alpha-Beta Search
- 43. Stochastic Games
- 44. Monte-Carlo Tree Search Framework
- 45. Monte-Carlo Tree Search Configurations

Minimax Search

Example: Tic-Tac-Toe

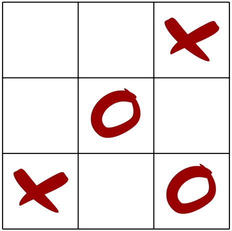
consider it's the turn of player ×:



If the utility for win/draw/lose for player × is +1/0/-1,
what is an appropriate utility value for the depicted position?

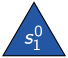
Example: Tic-Tac-Toe

consider it's the turn of player :



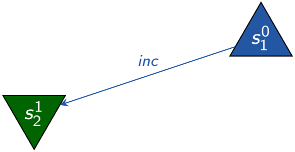
And what about this one?

Idea and Example



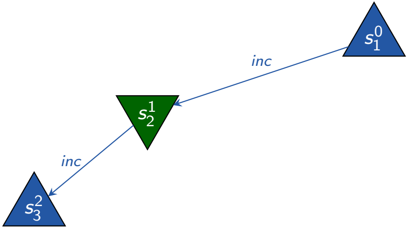
- depth-first search in game tree

Idea and Example



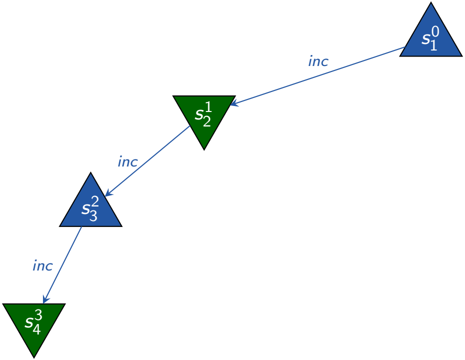
- depth-first search in game tree

Idea and Example



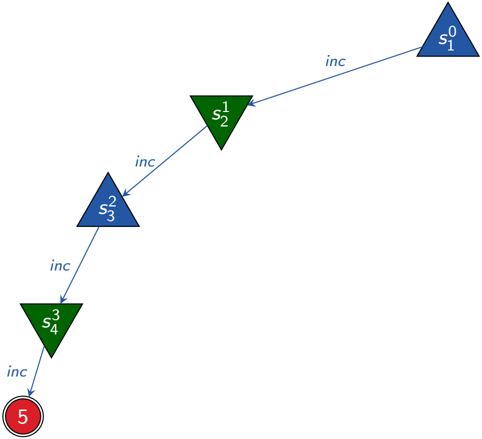
- depth-first search in game tree

Idea and Example



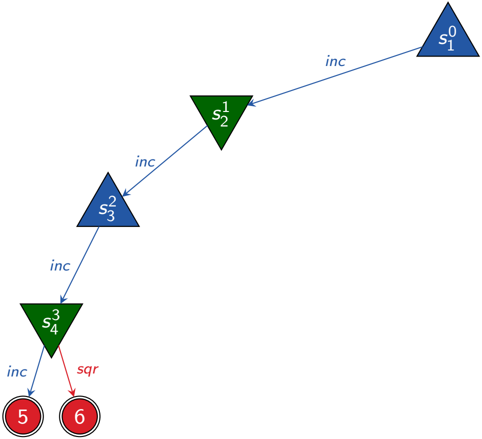
- depth-first search in game tree

Idea and Example



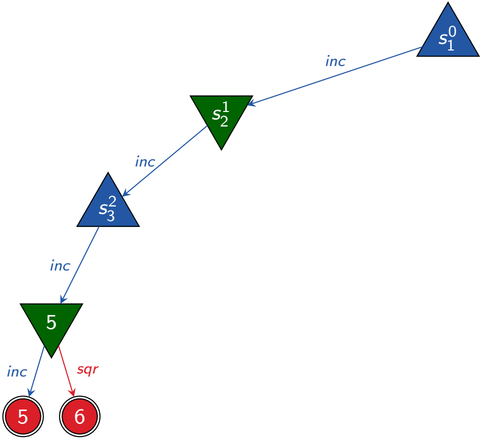
- depth-first search in game tree
- determine utility value of terminal position with utility function

Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function

Idea and Example

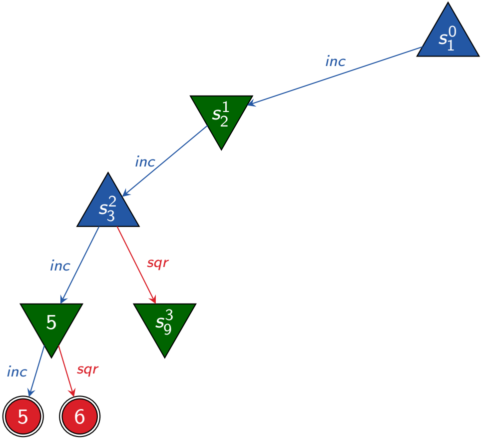


- depth-first search in game tree
- determine utility value of terminal position with utility function

- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

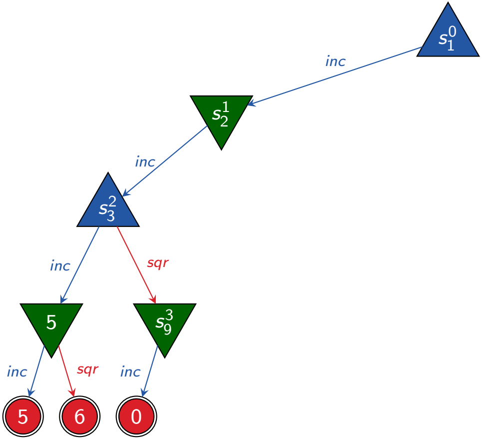
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

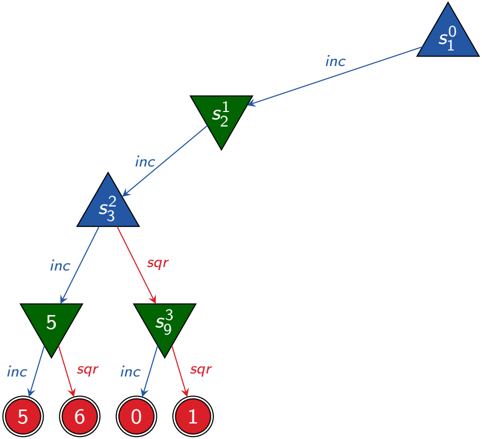
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

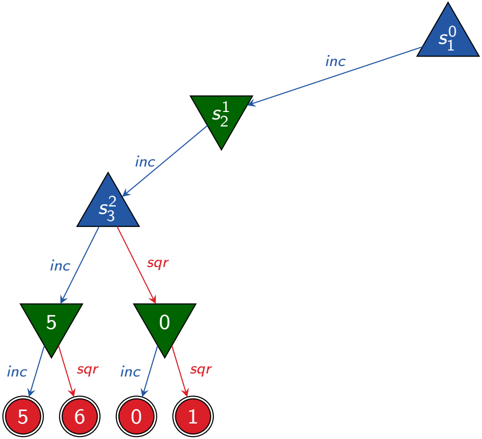
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

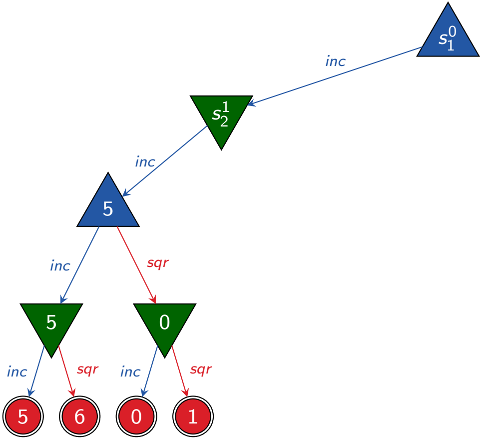
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

Idea and Example

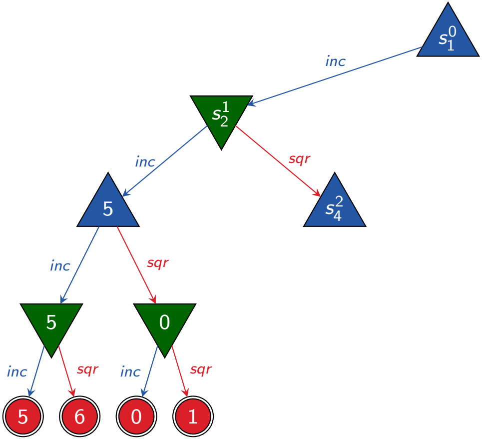


- depth-first search in game tree
- determine utility value of terminal position with utility function

- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

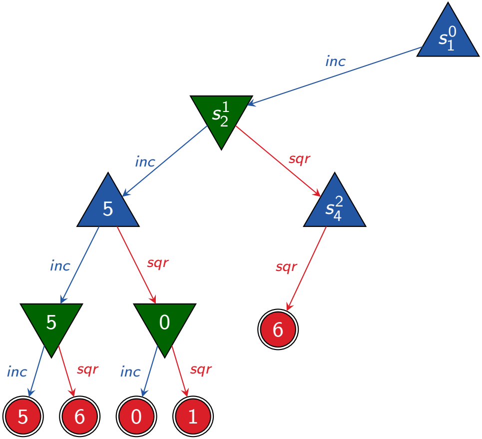
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

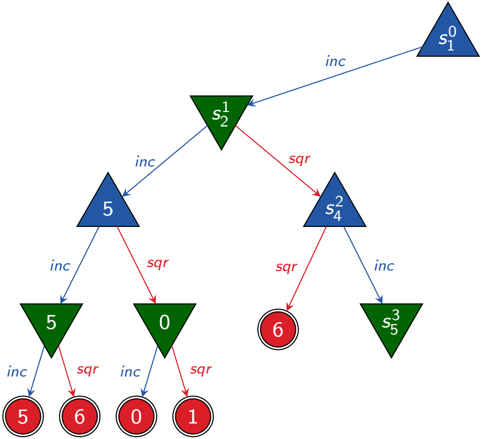
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

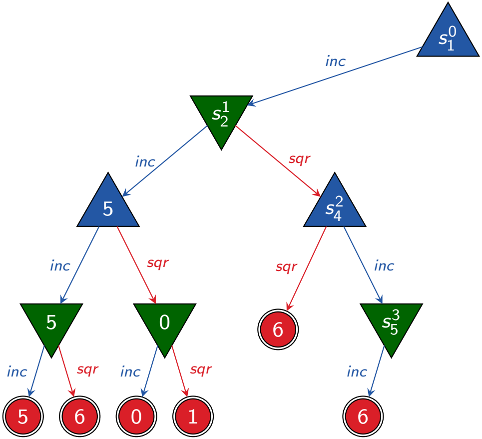
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

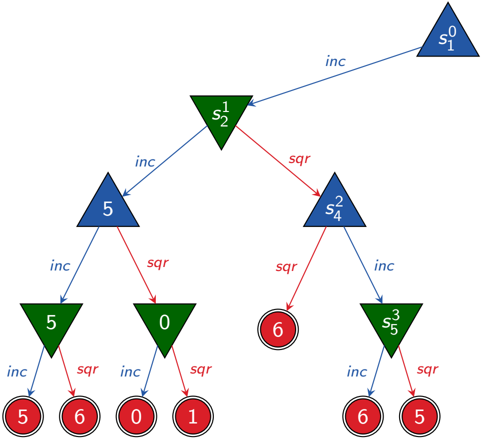
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

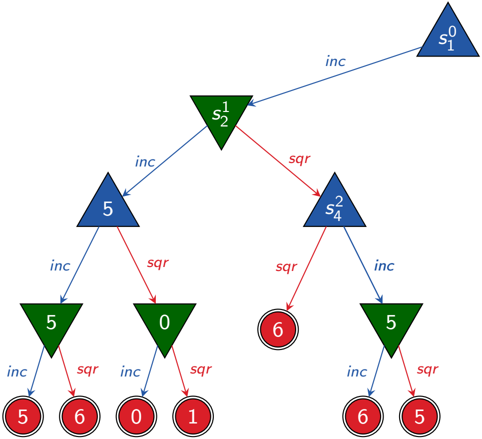
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

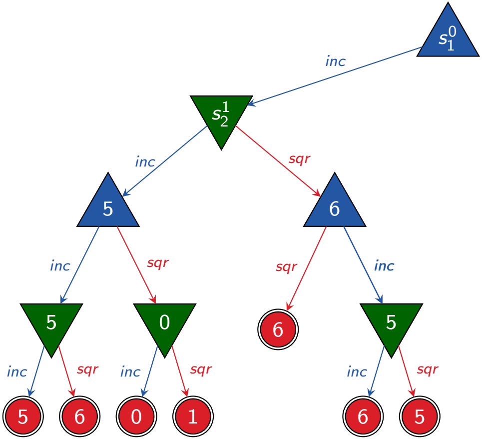
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

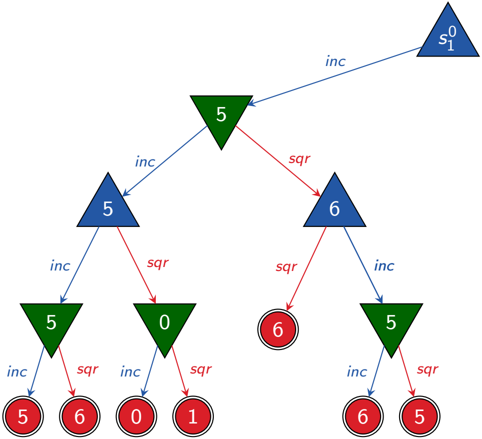
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

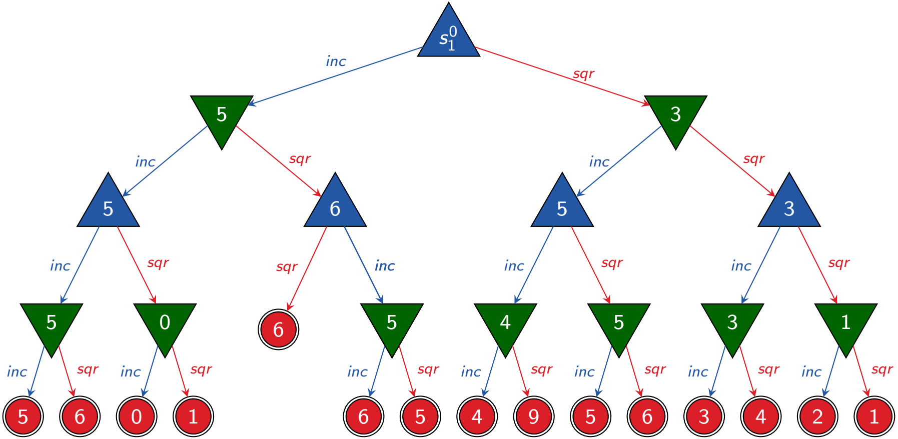
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

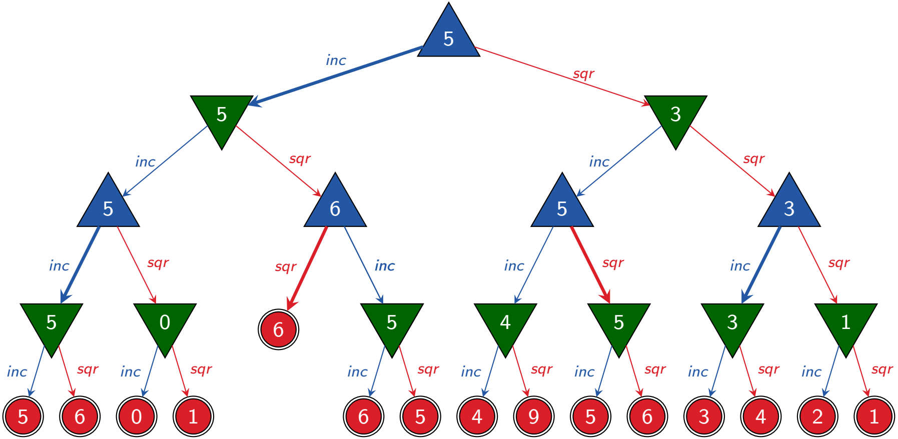
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- strategy: action that maximizes utility value (minimax decision)
- compute utility value of inner nodes

  from below to above through the tree:
  - *min*'s turn: utility value is minimum of utility values of children
  - *max*'s turn: utility value is maximum of utility values of children

Minimax: Pseudo-Code

function minimax(p)

```
if p is terminal position:
    return ⟨utility(p), none⟩
best_move := none
if player(p) = max:
    v := −∞
else:
    v := ∞
for each ⟨move, p′⟩ ∈ succ(p):
    ⟨v′, best_move′⟩ := minimax(p′)
    if (player(p) = max and v′ > v) or
       (player(p) = min and v′ < v):
        v := v′
        best_move := move
return ⟨v, best_move⟩
```

Discussion

- minimax is the simplest (decent) search algorithm for games
- yields optimal strategy* (in the game-theoretic sense, i.e., under the assumption that the opponent plays perfectly)
- *max* obtains at least the utility value computed for the root, no matter how *min* plays
- if *min* plays perfectly, *max* obtains exactly the computed value

(*) for finite trees; otherwise things get more complicated

Limitations of Minimax

What if the size of the game tree is too big for minimax?

$\rightsquigarrow$ Heuristic Alpha-Beta Search

Evaluation Functions

Evaluation Functions

Definition (evaluation function)

Let $\mathcal{S}$ be a game with set of positions S.
An evaluation function for $\mathcal{S}$ is a function

$$h : S \to \mathbb{R}$$

which assigns a real-valued number to each position $s \in S$.

Looks familiar? Commonalities? Differences?

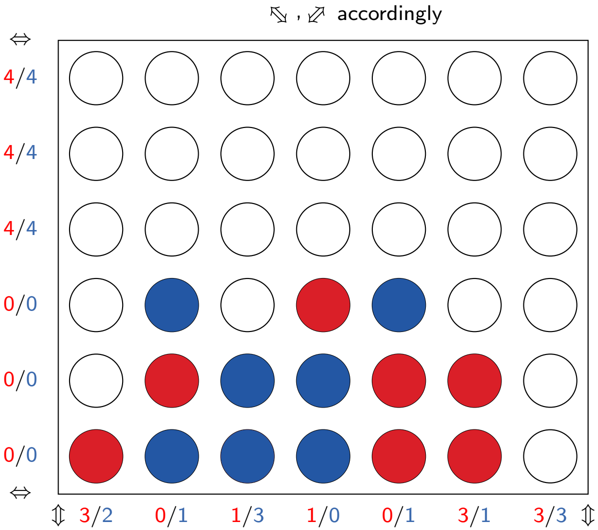
Intuition

- problem: game tree too big
- idea: search only up to predefined depth
- depth reached: estimate the utility value according to heuristic criteria (as if terminal position had been reached)

accuracy of evaluation function is crucial

- high values should relate to high "winning chances"
- at the same time, the evaluation should be efficiently computable in order to be able to search deeply

Example: Connect Four



evalution function: difference of number of possible lines of four

General Method: Linear Evaluation Functions

expert knowledge often represented with weighted linear functions:

$$h(s) = w_0 + w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s),$$

where w_i are weights and f_i are features.

General Method: Linear Evaluation Functions

expert knowledge often represented with weighted linear functions:

$$h(s) = w_0 + w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s),$$

where w_i are weights and f_i are features.

- assumes that feature contributions are mutually independent (usually wrong but acceptable assumption)
- features are (usually) provided by human experts
- weights provided by human experts or learned automatically

General Method: Linear Evaluation Functions

expert knowledge often represented with weighted linear functions:

$$h(s) = w_0 + w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s),$$

where w_i are weights and f_i are features.

example: evaluation function in chess

feature	f_p^{player}	f_k^{player}	f_b^{player}	f_r^{player}	f_q^{player}
no. of pieces	pawn	knight	bishop	rook	queen
weight for *max*	1	3	3	5	9
weight for *min*	-1	-3	-3	-5	-9

often additional features based on pawn structure, mobility, ...

$$\rightsquigarrow h(s) = f_p^{max}(s) + 3f_k^{max}(s) + 3f_b^{max}(s) + 5f_r^{max}(s) + 9f_q^{max}(s) \\ -f_p^{min}(s) - 3f_k^{min}(s) - 3f_b^{min}(s) - 5f_r^{min}(s) - 9f_q^{min}(s)$$

General Method: State Value Networks

alternative: evaluation functions based on neural networks

- value network takes position features as input (usually provided by human experts)
- and outputs utility value prediction
- weights of network learned automatically

General Method: State Value Networks

alternative: evaluation functions based on neural networks

- value network takes position features as input (usually provided by human experts)
- and outputs utility value prediction
- weights of network learned automatically

example: value network of Alpha Go

- start with policy network trained on human expert games
- train sequence of policy networks by self-play against earlier version
- final step: convert to utility value network (slightly worse informed but much faster)

David Silver et al., Mastering the game of Go with deep neural networks and tree search (Nature, 2016)

How Deep Shall We Search?

- objective: search as deeply as possible within a given time
- problem: search time difficult to predict
- solution: iterative deepening
  - sequence of searches of increasing depth
  - time expires: return result of previously finished search
  - overhead acceptable (see ai12 lecture)
- refinement: search deeper in “turbulent” states (i.e., with strong fluctuations of the evaluation function) $\rightsquigarrow$ quiescence search
  - example chess: deepen the search after capturing moves

Summary

Summary

- Minimax is a tree search algorithm that plays perfectly (in the game-theoretic sense), but its complexity is $O(b^d)$ (branching factor b, search depth d).
- In practice, the search depth must be bounded $\rightsquigarrow$ apply evaluation functions.