Planning and Optimal Control

7. Reinforcement Learning for Playing Games

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue. 24.10.	(1)	1. Markov Models
Tue. 31.10.	—	— *Luther Day* —
Tue. 7.11	(2)	2. Hidden Markov Models
Tue. 14.11.	(3)	2b. (ctd.)
Tue. 21.11.	(4)	3. State Space Models
Tue. 28.11.	(5)	3b. (ctd.)
Tue. 5.12.	(6)	4. Markov Random Fields
Tue. 12.12.	(7)	4b. (ctd.)
Tue. 19.12.	(8)	4c. (ctd.)
Tue. 26.12.	—	— *Christmas Break* —
Tue. 9.1.	(9)	5. Markov Decision Processes
Tue. 16.1.	(10)	5b. (ctd.)
Tue. 23.1.	(11)	6. Reinforcement Learning
Tue. 30.1.	(12)	6b. (ctd.)
Tue. 6.2.	(13)	7. Reinforcement Learning for Games
next year		8. Partially Observable Markov Decision Processes

Outline

1. Introduction

2. Improving a Policy via Markov Chain Tree Search

3. A Policy Model That Learns to Improve

4. Evaluation

Outline

Why Games?

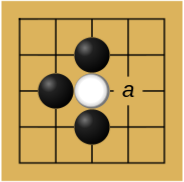
- ground truth mechanics
  - known and
  - simple (rules).
- consequences of actions taken in games can be assessed purely computationally
  - no costly/slow interactions required
    - with humans or
    - with the physical world.
- unlimited data (self play)
- difficult to win, requires intelligence.
- transition and reward model of the MDP are known,
  but its state space is too large to compute the optimal policy with the methods seen so far.

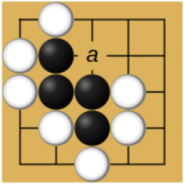
Go / Rules

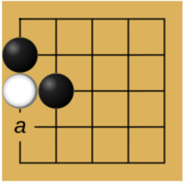
- start: board with 19×19 empty places.
- moves: first black, then white play alternatively:
  - set an own stone on an empty place and
    - remove all sets of connected enemy stones
      that are not connected to an empty place (capture).
    - remove all sets of connected own stones
      that are not connected to an empty place (self capture).
    - moves leading back to an earlier position are disallowed.
- end: once both players consecutively passed.
- win: the player owning more places wins, counting
  - places occupied with own stones plus
  - connected empty places surrounded by own stones (territory).

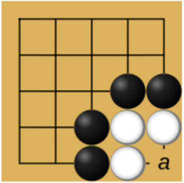
Go / Rules / Capture

- set an own stone on an empty place and
  - remove all sets of connected enemy stones
    that are not connected to an empty place (capture).





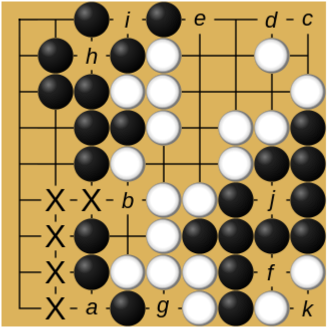


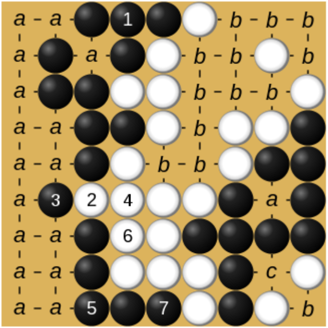


[source: Wikipedia, Rules of Go, https://en.wikipedia.org/wiki/Rules_of_Go.]

Go / Rules / Capture

- win: the player owning more places wins, counting
  - places occupied with own stones plus
  - connected empty places surrounded by own stones (territory).





[source: Wikipedia, Rules of Go, https://en.wikipedia.org/wiki/Rules_of_Go.]

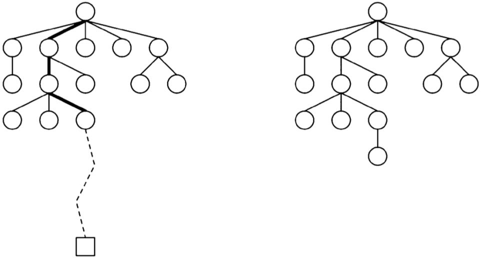
Outline

AI Search Problem

- problem: find the optimal policy given an MDP model
- search approach: find the optimal action sequence by
  1. looking ahead into the state/action space
     - usually represented by a search tree
  2. computing values at the terminal nodes
  3. propagating the values back to the root node

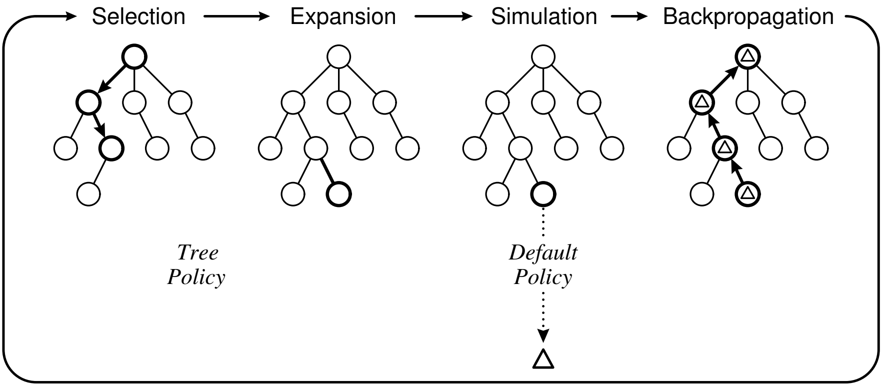
  - finally: make decision based on expected values at the root node.
- adversarial search:
  - player chooses best action for himself. vs. opponent will choose worst action for the player.
  - minimax algorithm: in every node, alternatively
    - choose action with highest value (for the player; player's move)
    - choose action with the lowest value (for the player; opponent's move)
- most state spaces are too large for such a complete enumeration.

Markov Chain Tree Search (MCTS) / Idea



[source: Browne et al. 2012]

Markov Chain Tree Search (MCTS) / Steps



[source: Browne et al. 2012]

Markov Chain Tree Search (MCTS)

- apply MCTS when the MDP is not know,
  but we have just estimates
  - for the value function, for the optimal policy
  - does not find the optimal policy,
    but hopefully will **improve** the actual estimates.

- given an initial policy π and
  estimated value function v
  find an improved action value function v' by
  - averaging over the values of follow-up states of an action a in a state s (any time step ahead)

  and an improved policy π'
  - by using the argmax policy of v' ("max child")
  - here: by relative frequencies of the actions ("robust child").

Markov Chain Tree Search (MCTS)

- given an initial policy π and
  estimated value function v

  simulate sequences from a generating policy
  - that initially is π,
  - later on shifts towards actions leading to high (estimated) value.
- to accomplish this, measure:
  - $N(s, a)$: how often an action a has been taken in state s during simulation so far.
  - $V(s, a)$: the total value seen in follow-up states of action a taken in state s (any time step ahead)
- a useful such generating policy (PUCT algorithm):

$$\tilde{\pi}(s, a; N, V, \pi) := \frac{V(s, a)}{N(s, a)} + c \frac{\sqrt{\sum_{a' \in A} N(s, a')}}{1 + N(s, a)} \pi(s, a)$$

  - where c is an exploitation/exploration trade-off weight.

Markov Chain Tree Search (MCTS)

policy-mcts-improved(s_0, π, v, K, c):
  create tree with root s_0, child indices A and edge attributes p, N, V:
    $p(s_0, a) := \pi(s_0, a), N(s_0, a) := 0, V(s_0, a) := 0$ for all $a \in A$
  for $k := 1, \ldots, K$:
    $t := 0$
    do
      $a_t := \arg\max_{a \in A} \tilde{\pi}(s_t, a) := \frac{V(s_t,a)}{N(s_t,a)} + c \frac{\sqrt{\sum_{a' \in A} N(s_t,a')}}{1+N(s_t,a)} p(s_t, a)$
      $s_{t+1} :=$ execute-action(s_t, a_t)
      $t := t + 1$
    while a child node s_t of s_{t-1} for action a_{t-1} exists already in the tree
    create child node s_t of s_{t-1} for action a_{t-1}
      $p(s_t, a) := \pi(s_t, a), N(s_t, a) := 0, V(s_t, a) := 0$ for all $a \in A$
    $v_t := v(s_t)$
    for $t' := 0, \ldots, t-1$:
      $N(s_{t'}, a_{t'}) := N(s_{t'}, a_{t'}) + 1$
      $V(s_{t'}, a_{t'}) := V(s_{t'}, a_{t'}) + v_t$
  $\pi'(a) := N(s_0, a)^{1/\tau} / \sum_{a' \in A} N(s_0, a')^{1/\tau}$
  return π'

where

- $s_0 \in S$ state to improve policy for.
- π original policy.
- $v \in \mathbb{R}^S$ original value function.
- $K \in \mathbb{N}$ number of simulation samples.
- τ annealing rate.
- c exploitation/exploration trade-off.
- returns π' an improved policy for state s_0.

Outline

1. Introduction

2. Improving a Policy via Markov Chain Tree Search

3. A Policy Model That Learns to Improve

4. Evaluation

Policy Model $\hat{\pi}$

- input: suitable representation of state s_t
  - richer: last t_0 states $s_{t-t_0+1}, \ldots, s_{t-1}, s_t$
- output:
  - **value function**: estimated value $\hat{v}$ of the input state.

$$v := \begin{cases} +1, & \text{if player wins} \\ -1, & \text{else} \end{cases}$$

  - **improved policy**: estimated selection probabilities $\hat{\pi} \in \mathbb{R}^A$ of an look-ahead improved policy of $\hat{\pi}$ for this state, e.g.,

$$\pi := \text{policy-MCTS-improved}(\hat{\pi})$$

Policy Model $\hat{\pi}$ / Loss

multitask loss/objective function:

$$\ell(\hat{\pi}, \hat{v}; \pi, v) := (v - \hat{v})^2 + \pi^T \log^\circ \hat{\pi} + \lambda ||\theta||^2$$

$$v := \begin{cases} +1, & \text{if player wins} \\ -1, & \text{else} \end{cases}$$

$$\pi := \text{policy-MCTS-improved}(\hat{\pi}), \in \mathbb{R}^A$$

$$\hat{\pi}, \hat{v} := \text{current estimations of optimal policy and value}$$
$$(=\text{output of DNN})$$

$$\theta := \text{parameters of DNN}$$

Policy Model $\hat{\pi}$ / AlphaGo Zero for Go (1/2)

- input:
  - board as 19×19 image with 17 binary channels
    - 8 for locations of white stones in last 8 positions
    - 8 for locations of black stones in last 8 positions
    - 1 for who's turn (same value for all pixels)
- 1 initial convolutional block:
  - convolution (3×3, stride 1, 256 filters), batch normalization, rectifier.
- 19 residual blocks:
  - convolution (3×3, stride 1, 256 filters), batch normalization, rectifier, convolution (3×3, stride 1, 256 filters), batch normalization, skip connection adding block input, rectifier.

Policy Model $\hat{\pi}$ / AlphaGo Zero for Go (2/2)

- two heads for two outputs:
  - scalar $\hat{v}$:
    - convolution (1×1, stride 1, 1 filter), batch normalization, rectifier, fully connected layer (size 256), rectifier, fully connected layer (size 1), tanh.
  - vectorized $(\hat{p}(a))_{a \in A}$, where A are all locations and action "pass".
    - convolution (1×1, stride 1, 2 filters), batch normalization, rectifier, fully connected layer (size $19^2 + 1 = 362$), logistic.
- 22.8M parameters
- with 19 residual blocks,
  the receptive field of the last block is the whole board.

learn-policy(N):
  $\theta :=$ randomly initialization
  do until convergence:
    $\mathcal{D} := \{\text{sample-policy}(\hat{\pi}(\theta)) \mid n = 1, \ldots, N\}$
    $\theta :=$ update-model$(\theta, \mathcal{D})$
  return θ

where

- $N \in \mathbb{N}$ sample size per model update
- returns θ parameters of the policy model $\hat{\pi}$ (and value model $\hat{v}$)

Required Computational Resources

data foundation:

- none
  - model learns only from generated data,
    not from human games.

computational resources:

- 3 days version:
  - 192 GPU days (= 3 days on 64 GPUs)
  - 4.9M games, 1,600 MCTS simulations/move
  - 700,000 minibatches a 2,048 positions each (possibly overlapping).
  - network with 20 residual blocks (40+ convolutional layers)
- 40 days version:
  - 2560 GPU days
  - network with 40 residual blocks (80+ convolutional layers)

Outline

1. Introduction

2. Improving a Policy via Markov Chain Tree Search

3. A Policy Model That Learns to Improve

4. Evaluation

Evaluation Criteria

a. Elo rating:
  - a score that predicts how likely one player wins over another one

b. accuracy of predicting the next move of a human expert player.

c. accuracy of predicting the outcome of a match between professional players given a position.

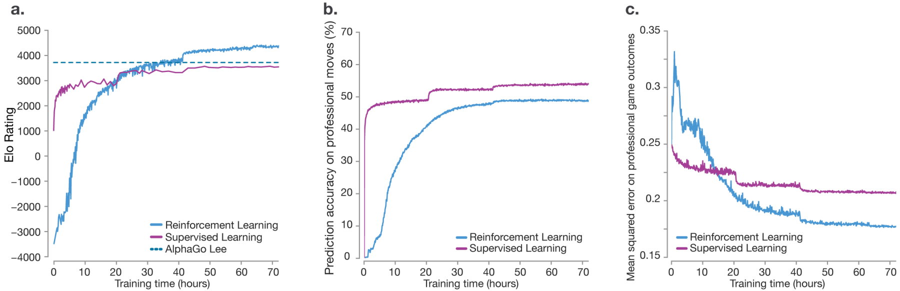
Elo Ratings

$$
\begin{aligned}
p(a \text{ wins against } b) &= \text{logistic}(\frac{1}{400}(r_a - r_b)) = \frac{1}{1 + e^{-\frac{1}{400}(r_a - r_b)}} \\
&= \text{logistic}(\frac{1}{400}(r^T(e_a - e_b)))
\end{aligned}
$$

- Elo ratings are the weights of a logistic regression model for the player ID predictor.
- To evaluate alphaGoZero, the Elo ratings of alphaGo are used for reference.
  - and the alphaGo Elo ratings had been computed from its games against human Go masters.

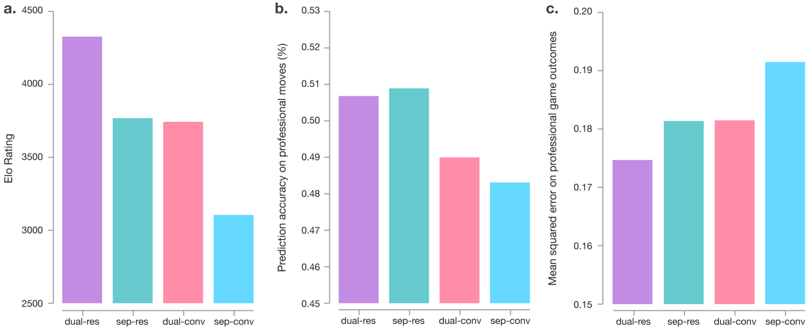
Note: e_a denotes the a-th unit vector in $\mathbb{R}^N$ with N the number of players, i.e., $(e_a)_b = \mathbb{I}(a = b)$.

Self-Play RL vs. Supervised Learning



[source: Silver et al. 2017]

Value/Policy Model Architectures

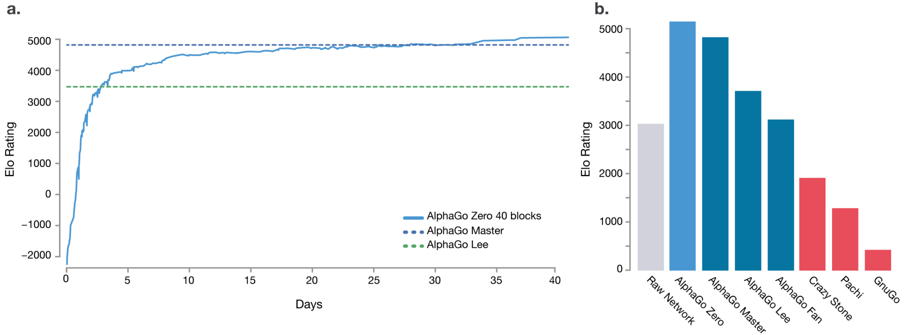


[source: Silver et al. 2017]

different architectures:

- multitask value/policy model ("dual")
  vs. two separate value and policy models ("sep")
- residual deep neural network ("res")
  vs. purely convolutional deep neural network ("conv")

Self-Play RL (AlphaGo Zero) vs. Other Systems



[source: Silver et al. 2017]

Go as Testbed for Reinforcement Learning

- simple representation of the state
- simple representation of moves
- easily scalable problem sizes: vary board sizes
- hard task for humans
- easily scalable policy/value model
  - deep neural network, complexity scaled by number of layers

Further Readings

- About the policy model and the Go application:
  - Silver et al. [2017]
  - Some details, esp. about the MCTS used, are described more detailed in a precursor paper:
    - Silver et al. [2016]
- Monte Carlo Tree Search (MCTS):
  - a brief summary: `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`
  - a survey: Browne et al. [2012]
  - polynomial upper confidence tree algorithm (PUCT):
    - Auger et al. [2013]
    - beware, the reference in Silver et al. [2017] is wrong.
- Generally about adversarial search:
  - Russell and Norvig 2009, ch. 5

References

David Auger, Adrien Couetoux, and Olivier Teytaud. Continuous upper confidence trees with polynomial exploration–consistency. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 194–209. Springer, 2013.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice hall Upper Saddle River, 3rd edition, 2009.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian Bolton. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.