Reinforcement Learning

Lecture 10

Monte Carlo Tree Search

AlphaGo (Zero)

Robert Peharz

Institute of Theoretical Computer Science
Graz University of Technology
Winter Term 2023/24

Policy Gradient

Policy Gradient: parametrize policy $\pi(a \mid s, \underline{\theta})$ with a vector $\underline{\theta}$ and maximize a suitable performance measure $J(\underline{\theta})$

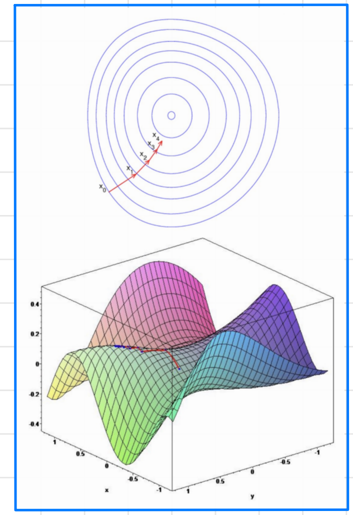


Image: D. Silver

$$J_1(\underline{\theta}) = v_\pi(s_1) = \mathbb{E}_\pi[G_1 \mid S_1 = s_1]$$

$$J_{EV}(\underline{\theta}) = \mathbb{E}_\pi[v_\pi(s)] = \sum_s \mu_\pi(s)\, v_\pi(s)$$

$$J_{ER}(\underline{\theta}) = \sum_s \mu_\pi(s)\, r(s)$$

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\underline{\theta})$$

finite differences: for small $h > 0$

$$\frac{\partial J}{\partial \theta_i} \approx \frac{J(\ldots, \theta_i + h, \ldots) - J(\ldots, \theta_i, \ldots)}{h}$$

REINFORCE

$$\nabla_{\underline{\theta}} J_1(\underline{\theta}) = \mathbb{E}_\tau\left[\left(\sum_{t=1}^{T-1} \nabla_{\underline{\theta}} \log \pi(a_t \mid s_t, \underline{\theta})\right) \bar{g}(\tau)\right]$$

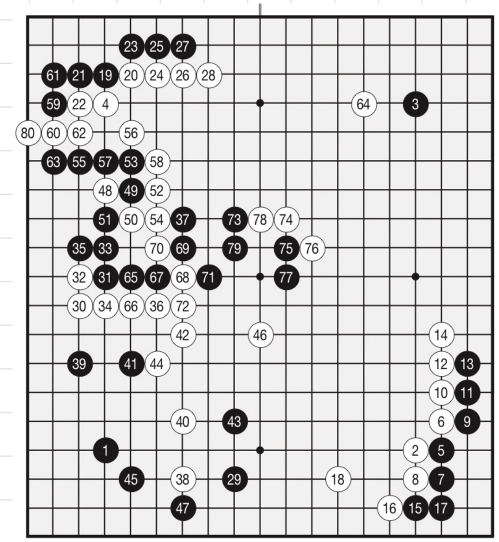
Actor-Critic learn both $\pi(a \mid s, \underline{\theta})$ and $\hat{q}(s, a; \underline{w})$

$$\underline{w} \leftarrow \underline{w} - \alpha^{\underline{w}}\, \delta\, \nabla_{\underline{w}} \hat{q}(S, A, w) \qquad \text{// update critic}$$

$$\underline{\theta} \leftarrow \underline{\theta} + \alpha^{\underline{\theta}}\, \hat{q}(S, A, \underline{w})\, \nabla_{\underline{\theta}} \log \pi(A \mid S, \underline{\theta}) \qquad \text{// update policy}$$

Monte Carlo Tree Search, AlphaGo

Computer Go: a Historical AI Challenge



- Go is an ancient board game, played by two players (black and white), taking turns
- a move consists of placing a stone on an intersection point of a 19 × 19 grid
- goal is to surround more territory than the opponent; additionally, surrounding groups of opponent's stones kills these, yielding also points
- while computer chess has outperformed humans in the late 90's, computer go programs were easily defeated
- Monte Carlo tree search (MCTS) proposed in 2007 was a key technique to advance computer go
- until 2014, computer go reached advanced amateur level
- 2015-2016: Alpha Go defeats leading professionals
- 2017: Alpha Go Zero, learned without human data, outperforms AlphaGo

Combinatorial Games

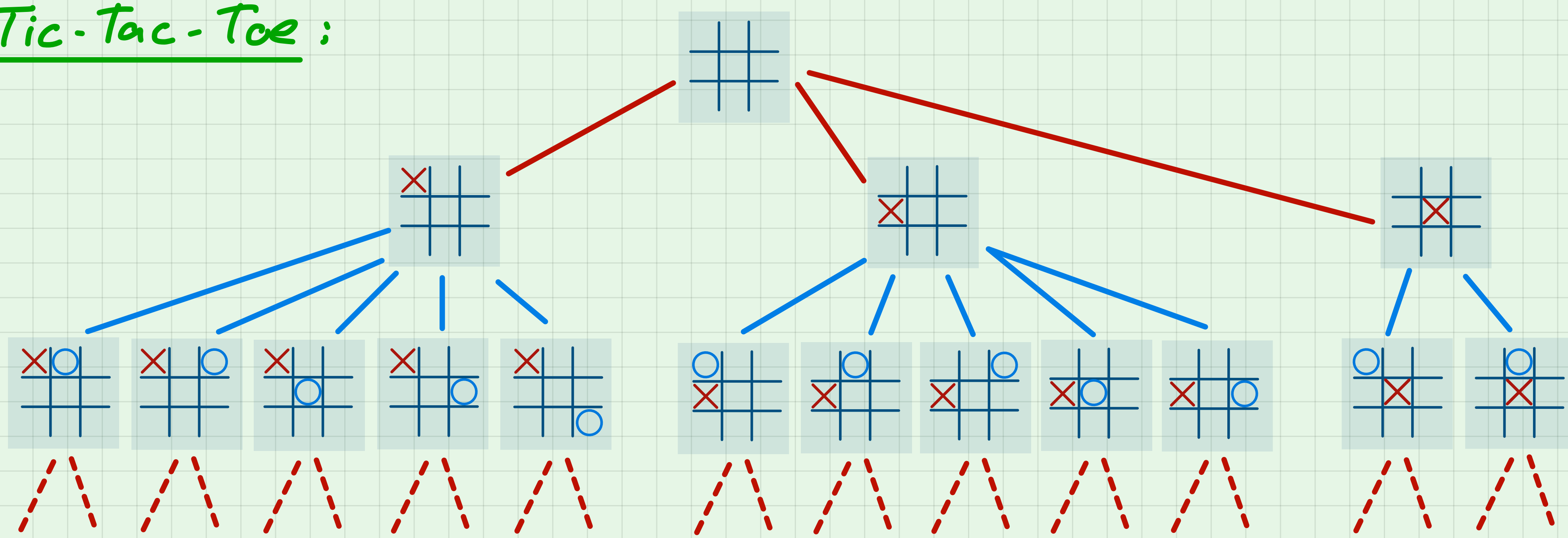
- MCTS is prominently used in games, as it
  1) requires a lot of compute
  2) requires a good model / simulator
- let us assume combinatorial games (2 players)
  - zero sum (one player wins, the other loses)
  - perfect information (completely observed state)
  - discrete actions
  - deterministic (actions determine next state, no chance element)
  - sequential (players take turns)
- chess, go, tic-tac-toe, shogi (Japanese chess)
- these games can (in principle) be solved via a game tree and backward induction

Game Tree

game tree (search tree) enumerates combinatorial game exhaustively:

- root node ≙ starting state
- outgoing edges of a state ≙ valid actions (moves)
- leaves ≙ terminal states;
  terminal states get a value +1 (win), -1 (lose), or 0 (draw)*

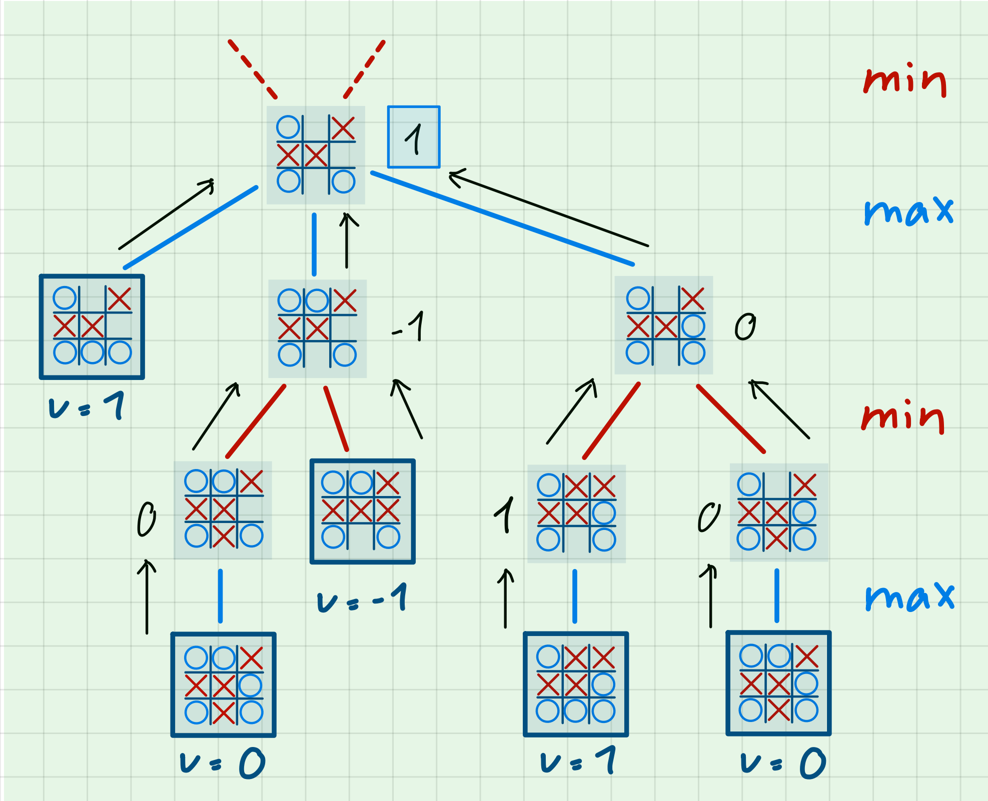
Tic-Tac-Toe:



* in RL terms, this value is formally the reward for (s,a) leading to the terminal state; all other (s,a) get reward 0.

Backward Induction

- values at leaves are known, due to rules of the game
- if the whole game tree can be constructed, values can be propagated up via backward induction
  - minimize over opponent's actions
  - maximize over own actions
- performs perfect play on both sides



- however, game trees are generally too large:

  chess: $\approx 10^{40}$

  go: $\approx 10^{170}$

- MCTS basically constructs a sub-tree and approximates the nodes' values

Decision Time Planning vs. Background Planning

- in previous lectures, we learned the whole
  - value function v_π
  - q-function q_π
  - policy π
  - environment p
- this can be called "background planning/learning"
- idea of decision time planning is to focus computation on the most important state: the current state S_t

  Advantage: better decisions, tailored to current state

  Disadvantage: requires additional computation; usually, there is a fixed compute budget for each decision

- MCTS is a decision time algorithm, building up the decision tree from the current state S_t

Rollout Policy

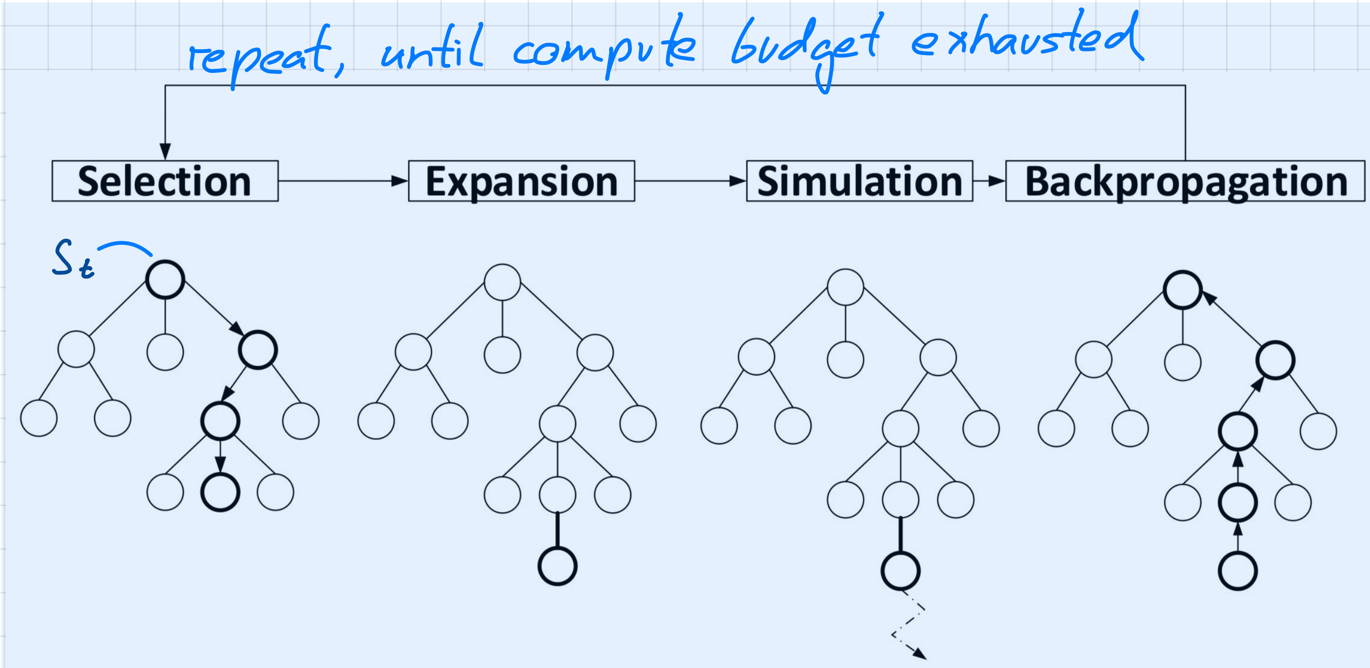
- assume that the game is in state S_t (= board configuration) and it is our turn (need to pick an action A_t)
- a rollout policy simulates the game until the end
- simulates both our policy $\hat{\pi}_{ro}(a|s)$ and the opponent $\hat{\pi}_{ro}^{(opp)}(a|s)$
- if $\hat{\pi}_{ro}^{(opp)}(a|s)$ is fixed, we might interpret it as part of environment, leading to a fixed MDP
- often, the rollout policy might be very simple, eg. random play
- basic rollout algorithm
  - simulate M rollouts (episodes)
  - $\hat{q}(S_t, a) = \frac{1}{|I_a|} \sum_{m \in I_a} G_m$
    - where G_m is the m^{th} return
    - I_a indices of rollout starting with a

    for games: 0, -1, 1
  - pick action $A_t = \arg\max_a \hat{q}(S_t, a)$
- essentially, greedy policy of $\hat{\pi}_{ro}(a|s)$ using Monte Carlo estimation

Monte Carlo Tree Search

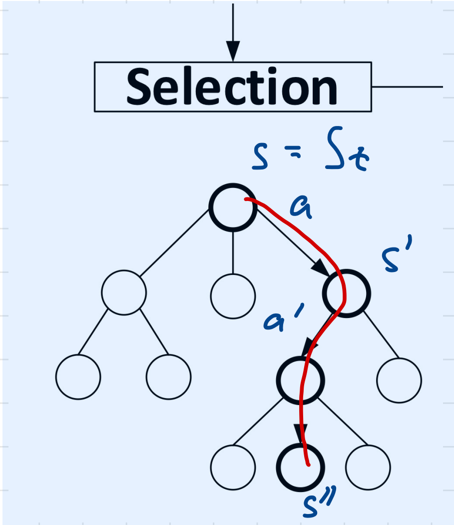
Image: Monte Carlo Tree Search: A Review of Recent Modifications and Applications

Maciej Świechowski,* Konrad Godlewski,† Bartosz Sawicki,‡ Jacek Mańdziuk§



- MCTS iteratively grows a sub-tree of the overall search tree, rooted at the current state S_t
- in the selection stage, it traverses the current sub-tree using a tree policy, until a terminal state or an untried action
- an untried action leads to a new leaf (expansion), for which simulations (rollouts) are performed
- simulation results are then backpropagated to estimate values

Upper Confidence Bound, Tree Policy



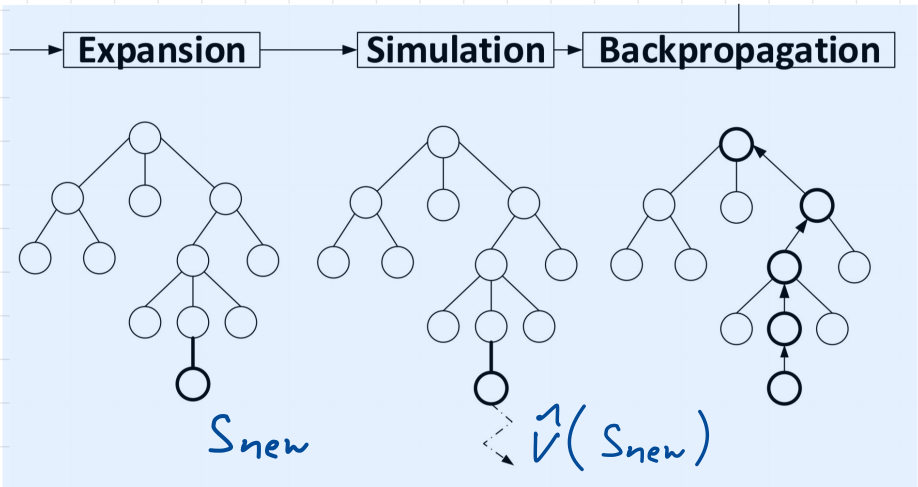
- tree policy needs to balance exploration and exploitation (focus on promising actions)
- common choice: upper confidence bound (UCB)

$$\text{select} \quad \arg\max_a \; \hat{q}_t(s,a) + C\sqrt{\frac{\log N(s)}{N(s,a)}}$$

UCB estimate of $q^*(s,a)$

- $\hat{q}_t(s,a)$ is the current value estimate for any s, a in the sub-tree
- $N(s,a)$ is the number of times a has been selected for s
- $N(s) = \sum_a N(s,a)$ is the number of times s has been visited
- C is a hyper-parameter, often $\sqrt{2}$ for $-1 \leq q(s,a) \leq 1$
- infrequent actions are preferred
- UCB is considered ∞ when $N(s,a) = 0$

Expansion, Simulation, Backpropagation



Expansion

when sub-tree traversal reaches a state with untried actions, it tries a new action, adding new leaf S_{new}

Simulation

starting at the (state corresponding to) the new leaf S_{new}, run one or more rollouts using some rollout policy, yielding a value estimate $\hat{v}(S_{new})$ (if S_{new} is terminal, this is exact)

Backpropagation

update $\hat{q}$ for all (s,a) pairs in the sub-tree along selected path:

$$N(s,a) \leftarrow N(s,a) + 1$$

$$\hat{q}(s,a) \leftarrow \hat{q}(s,a) + \frac{1}{N(s,a)}\left(\hat{v}(S_{new}) - \hat{q}(s,a)\right)$$

(running average)

Making a Decision

- when computational budget is exhausted, MCTS returns a decision
  - max child: $A_t = \arg\max_a \hat{q}(S_t, a)$
  - robust child: $A_t = \arg\max_a N(S_t, a)$
- after that, MCTS continues with S_{t+1}
- often, the search tree is re-used (using sub-tree rooted at A_{t+1})

many modifications of MCTS have been proposed, specifically AlphaGo(Zero), combining it with neural nets

Mastering the game of Go with deep neural networks and tree search

David Silver[1*], Aja Huang[1*], Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

Mastering the game of Go without human knowledge

David Silver[1*], Julian Schrittwieser[1*], Karen Simonyan[1*], Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]