Announcements

- This is the last lecture for this course!
  - Hooray!, Fireworks !!!
- Next week you start the project, which runs until the end of term.

Monte Carlo Tree Search (MCTS) summary

Current position: a subtree in memory.

- Find the best node whose children to expand.

  Selection: Using UCB, find the best node to expand.
  Expansion: Expand the children of that node.

- simulate the game from that point (e.g. random roll-outs) to reach terminal nodes and estimate the values values from the current position.
- A method for choosing the best action.

More on selection

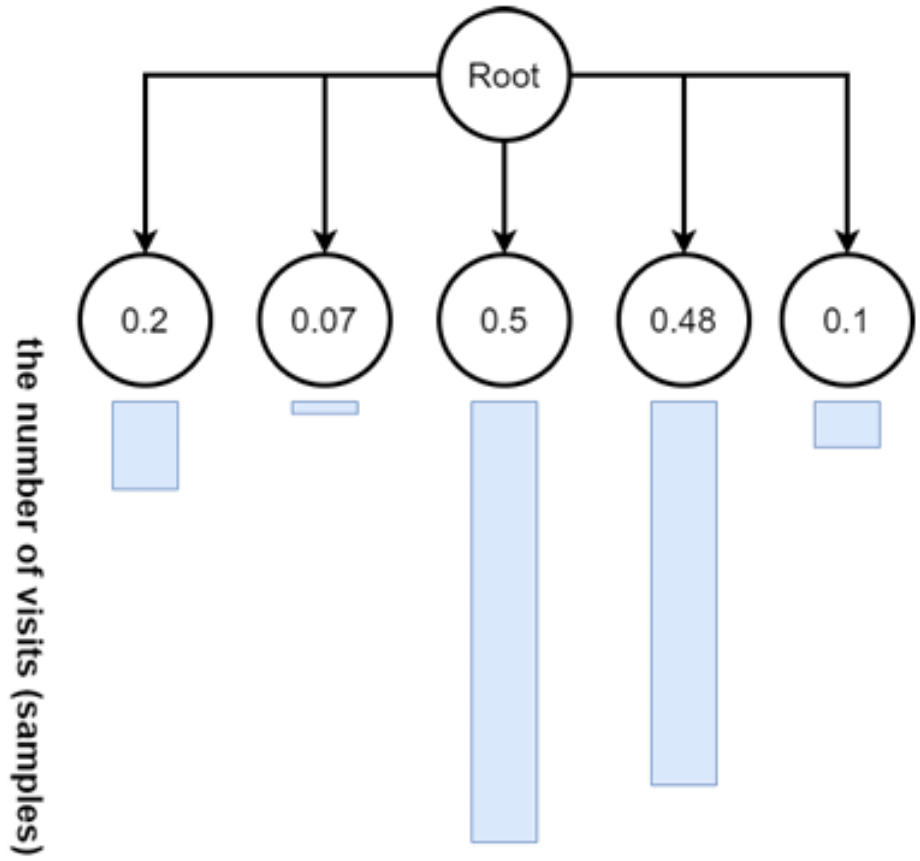


Figure 2: The root denotes the starting state. The average scores $(Q(s, a))$ of actions leading to next states are shown inside the circles. The bars denote how many simulations started from a particular action. This is an abstract example - not taken from any particular game.

Variations

Different methods can be used for the different phases.
You could use heuristics for selection

- Select for expansion the node with the highest value of the heuristic.

You could use heuristics for roll-outs

- During roll-outs, use the heuristic to select the action for each player.
- But the heuristic must be quick to calculate.

Games (besides GO) which use MCTS

Bridge: Ginsberg's GIB program competes with expert bridge players (1998).

Scrabble: MAVEN defeats the world scrabble champion (1998).

Hex: MoHex becomes world champion Hex player (2009). Not any longer.

Monte Carlo methods in general

A broad class of computational algorithms

- rely on repeated random sampling
- use randomness to find answers which might be deterministic in principle.

“Monte Carlo method”, Wikipedia,
`https://en.wikipedia.org/wiki/Monte_Carlo_method`

Monte Carlo methods in reinforcement learning in general

In games,

1. Generate a sequence through the game tree to a terminal position;
2. Recording all states, actions, and rewards received.

In order to build a strategy, improve a strategy, evaluate a strategy, etc.

AlphaGo

Combines supervised learning, reinforcement learning, and Monte Carlo tree search.
Created by Deepmind team led by David Silver.
References:

- Chapter 16.6.1 of **Reinforcement Learning: An Introduction** Richard Sutton and Andrew Barto, 2022 available `incompleteideas.net/book/the-book.html` Best!
- **Mastering the Game of Go with Deep Neural Networks and Tree Search.** David Silver et al. in Nature, Vol. 529, pages 484–489; January 28, 2016. `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`
- See also **How the Computer Beat the Go Player**, Christof Koch, Scientific American Mind (July/August 2016), 27, 20–23. `http://www.nature.com/scientificamericanmind/journal/v27/n4/full/scientificamericanmind0716-20.html`

AlphaGo

Based on a variant of MCTS — “Asynchronous policy and value MCTS”

- Value of newly added node s is linear combination of output of a value network and result of roll-outs (layouts),

$$V(s) = (1 - \eta)\, V(s|\mathbf{W}) + \eta Q(s) \tag{1}$$

- Standard MCTS only uses the roll-outs.

Here η is an adjustable parameter; $\eta = 1/2$ found experimentally.

Aside — some terminology

Policy model: generates a policy

- probability of action given a state

$$\pi(a|s, \mathbf{W})$$

Value model: generates the value of a state or a state-action pair,

- as we have seen with Q-learning and state-value learning.

AlphaGo

Has four components:

Supervised learning policy network: A deep CNN which learns to replicate experts players using supervised learning.

Policy learning network: An identical reinforcement learning neural network, which is initialized to the above, but then learns using reinforcement learning and self-play using *policy gradient* RL.[1]

Value network: Learns the value of board positions for both players. Learns from the policy network.

Roll-out network: a very simple linear policy network which decides moves during roll-outs.

- Rather than random decisions
- To give fast roll-outs.

[1] Ch13 in Sutton and Barto

Phases of training

1. First train the SL-policy network using supervised learning.
2. Initialize RL-policy network to SL-policy network, then train that using policy-gradient reinforcement learning.
3. Train the value network using many Monte Carlo games using the RL-policy network to play the games.

Also train the roll-out policy using supervised learning.

Elements of AlphaGo - SL-Policy network

A deep, convolutional neural network (CNN) trained by supervised learning (SL)

- Trained on 30 million human expert moves.

Inputs: The board as a image. The state.

Outputs: The probability of every possible move. Using soft-max.

Elements of AlphaGo - RL-Policy network

- CNN with same structure as SL-Policy network above.
- Initialized as the SL-Policy network.
- Further trained using policy-gradient reinforcement learning[2].

[2] To be discussed later

Elements of AlphaGo - Value network

A CNN which predicts the value of states.

- Same structure as SL-policy network except output
- Input: Board as image (same as SL-Policy network).
- Output: Single continuous number representing the value of the board position input.
- Training: Trained using Experience Replay using 50 million mini-batches of 32 board positions.

Roll-out policy

- A simple linear network to learn a policy,
- trained on 8 million human moves,
- using supervised learning.

AlphaGo pipeline

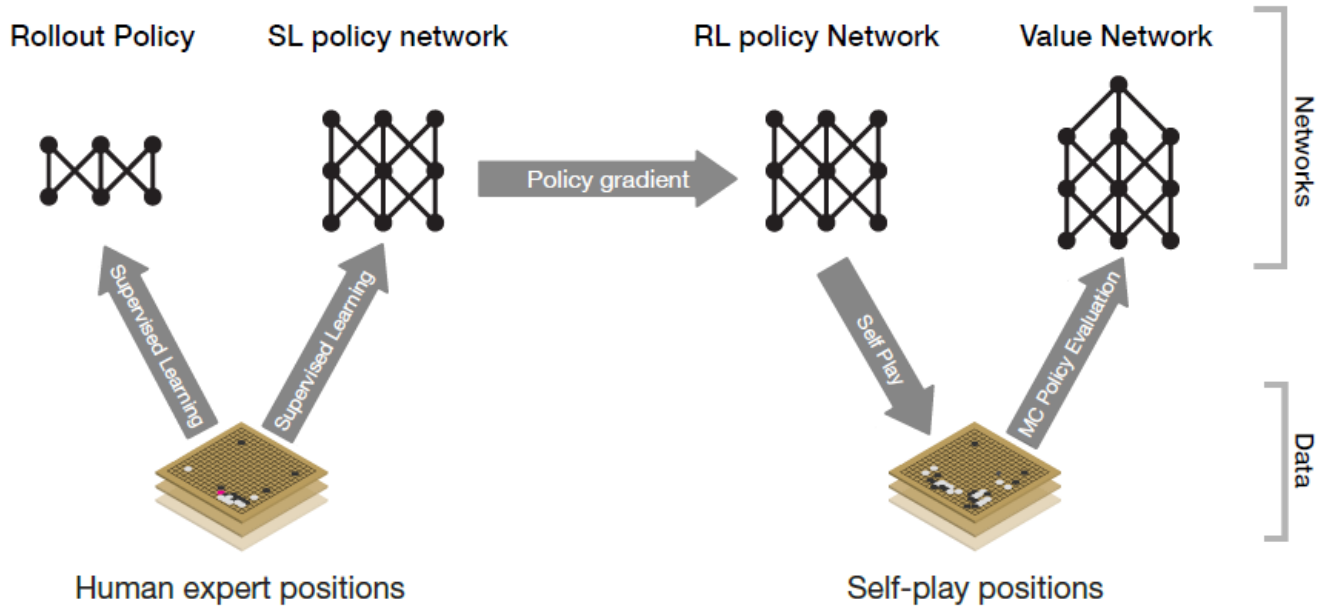


Figure 16.6: *AlphaGo* pipeline. Adapted with permission from Macmillan Publishers Ltd: *Nature*, vol. 529(7587), p. 485, ©2016.

But stolen by me from Sutton and Barto without permission

This produces top human performance

In March 2016, it beats Lee Sedol, 9 dan and the second most winning professional in the world. Summer 2017, beats Ke Jie 3 games to zero.

Policy learning versus value learning

- We have learned how to learn *value* functions.
  - Value of state-action pair — Q-learning
  - Value of a state — V-learning
  - From which we can derive the best action.
- Policy learning learns the policy directly.
  - Which action to take from each state.

Advantages of policy learning

- Deepmind team could observe the policy of expert players.

- Essential with an infinite state and/or action space.
- Can learn a probabilistic strategy in principle.

The parametrized policy function

Reference: Chapter 13 of Sutton and Barto.

- Approximate the policy function with a soft-max function,

$$\pi(a|s,\mathbf{W}) \approx \frac{e^{h(a|s,\mathbf{W})}}{\sum_b e^{h(b|s,\mathbf{W})}} \tag{2}$$

- s is the state, a is the action, the sum is over all actions from state s.
- Puts a probability over all actions.

Called *Soft-max in action preferences*

The parametrized policy function

- Approximate the policy function with a soft-max function,

$$\pi(a|s,\mathbf{W}) \approx \frac{e^{h(a|s,\mathbf{W})}}{\sum_b e^{h(b|s,\mathbf{W})}} \tag{3}$$

- h could be a deep neural network, with $\mathbf{W}$ representing all the weights for all layers.
- h could a simple linear model $\sum_i w_i \phi(a,s)$ with some features ϕ. (The Roll-out strategy in AlphaGo.)

We need to learn the values of $\mathbf{W}$ to make the best actions from each state highly probable.

REINFORCE update (Williams, 1992)

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \alpha J(t) \frac{\nabla \pi(A_t|S_t, \mathbf{W})}{\pi(A_t|S_t, \mathbf{W})}. \tag{4}$$

- Here A_t, S_t are the actual actions at time t, and $J(t)$ is the actual accumulated reward from time t to the end of the game,

$$J(t) = \sum_{t'=t}^{T} r_t \gamma^{t'-t}$$

  Approximate gradient ascent —increases the probability the A_t is taken from S_t in the future.

Cont

REINFORCE algorithm

A Monte Carlo algorithm.

1. Play to the end of the game, following the current $\pi(\cdot|\cdot, \mathbf{W})$.
2. For each step, apply equation (4).

REINFORCE pseudo-code

- **Input:** a differentiable parametrized policy model $\pi(a|s, \mathbf{W})$
- A value for the learning rate $\alpha > 0$
- Initialize $\mathbf{W}$.
- Loop forever (for each game)
  - Generate a game: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(a|s, \mathbf{W})$
  - For each step in the game, $t = 0, 1, \ldots, T-1$
    - $J(t) \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    - Update $\mathbf{W}$ following equation (4).

Notes

1. $J(t)$ is the actual future (discounted) reward. Not predicted reward.
   - We play to the end of that game, so the reward is known.
2. The sign of the rewards are important.
   - Positive accumulated rewards *increase* the probability of the actions taken in that game.
   - Negative rewards *decrease* the probability of the actions taken in that game.

Policy Gradient Eq

Conclusions

AlphaGo uses a (very) modified versus of Monte Carlo Tree Search (MCTS)

- Start with a strong (not ignorant) player.
- Uses a learned selection strategy.
- Uses a combined Monte Carlo - learned rollout strategy.
- Requires tons of data and tons of computational resources.

- Warning!!!

Project starts next week

- No more lectures or examples classes. Office hours continues. Labs take place in the lecture timeslots:
  - Lab 1, Tuesdays 4–5pm, Collab 1&2 and Kilburn G41.
  - Lab 2, Thursdays 3–4pm, Collab 1& 2 and Kilburn G41.

  There will be a lecturer and/or GTA in each.

Lab attendance

Generally optional. Exceptions:

1. Week 9 (Week commencing Nov 20) — Discuss and submit proforma.
2. Presentations — You must attend your presentation slot.

Otherwise, a place to work and get help.

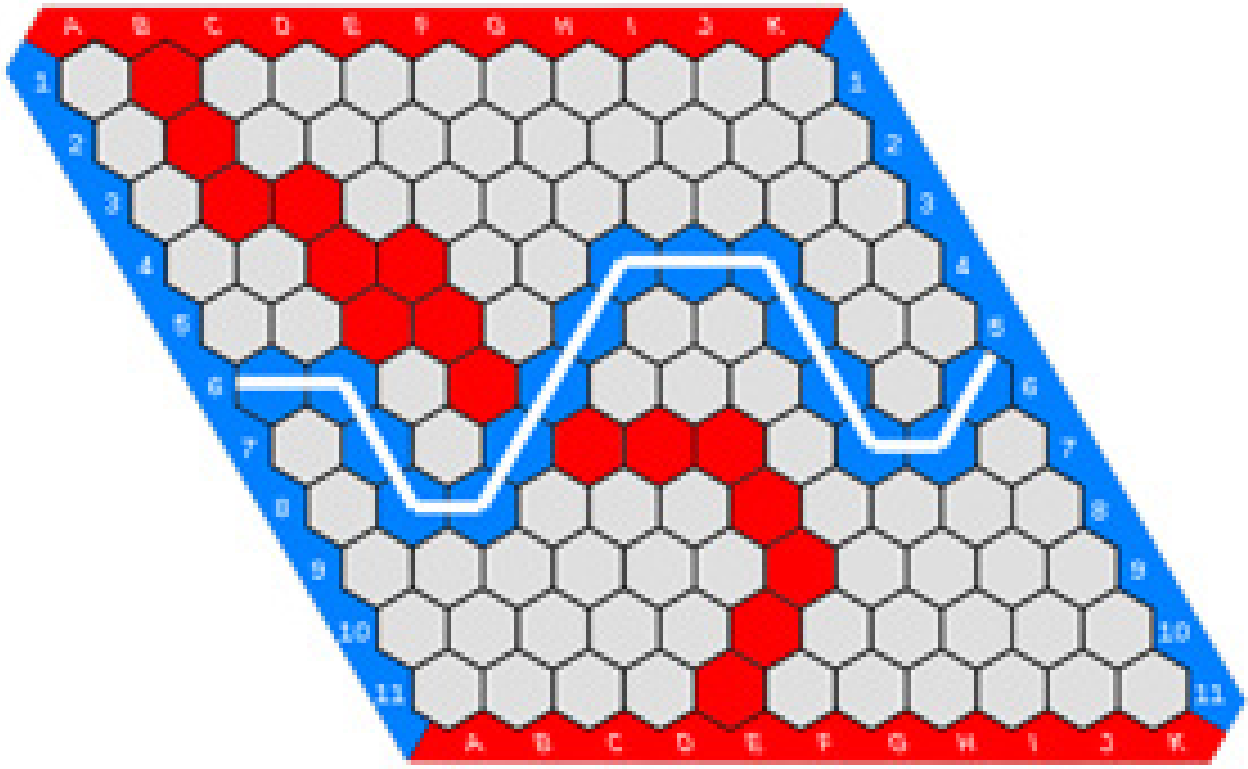
The Project

The Project: Produce a bot which plays the game HEX.

- Any language which can run on linux-based tournament machine.
- Any method.
  - MiniMax with alpha-beta pruning,
  - Q-learning with function approximation,
  - TD(λ),
  - AlphaZero,
  - Other

The game Hex

- A turn-based, two player, zero-sum game.
- Played on an 11 $\times$ 11 board.
- Winner is first to connect their sides of the board.



Pie rule

The pie rule, also called the swap rule, is used.
Suppose Alice is Player 1, the red player. Bob is Player 2 the blue player.
The game starts

1. Alice makes the first move.
2. On Bob's first move (**only then!**), he can
   - 2.1 Make a move as blue and is blue for the rest of the game.
   - 2.2 Swap.
     - 2.2.1 Then Alice becomes the blue player, and Bob because the red player for the rest of the game.
     - 2.2.2 Alice must move in response to her opening move.
     - 2.2.3 Bob then responds.
     - 2.2.4 The game continues.

Time limit

Each side has 5 minutes to play the game.

- When the opponent plays, your clock stops.
- When you play, the opponent clock stops.

If a player runs out of time, that player loses.

Marking

Marking scheme is on Blackboard in the AI and Games Project folder.

Final mark: 50% project; 50% exam.

Project mark: 50% approach; 50% tournament performance.

Tournament performance: 75% wins; 25% speed.

Approach mark

Two components:

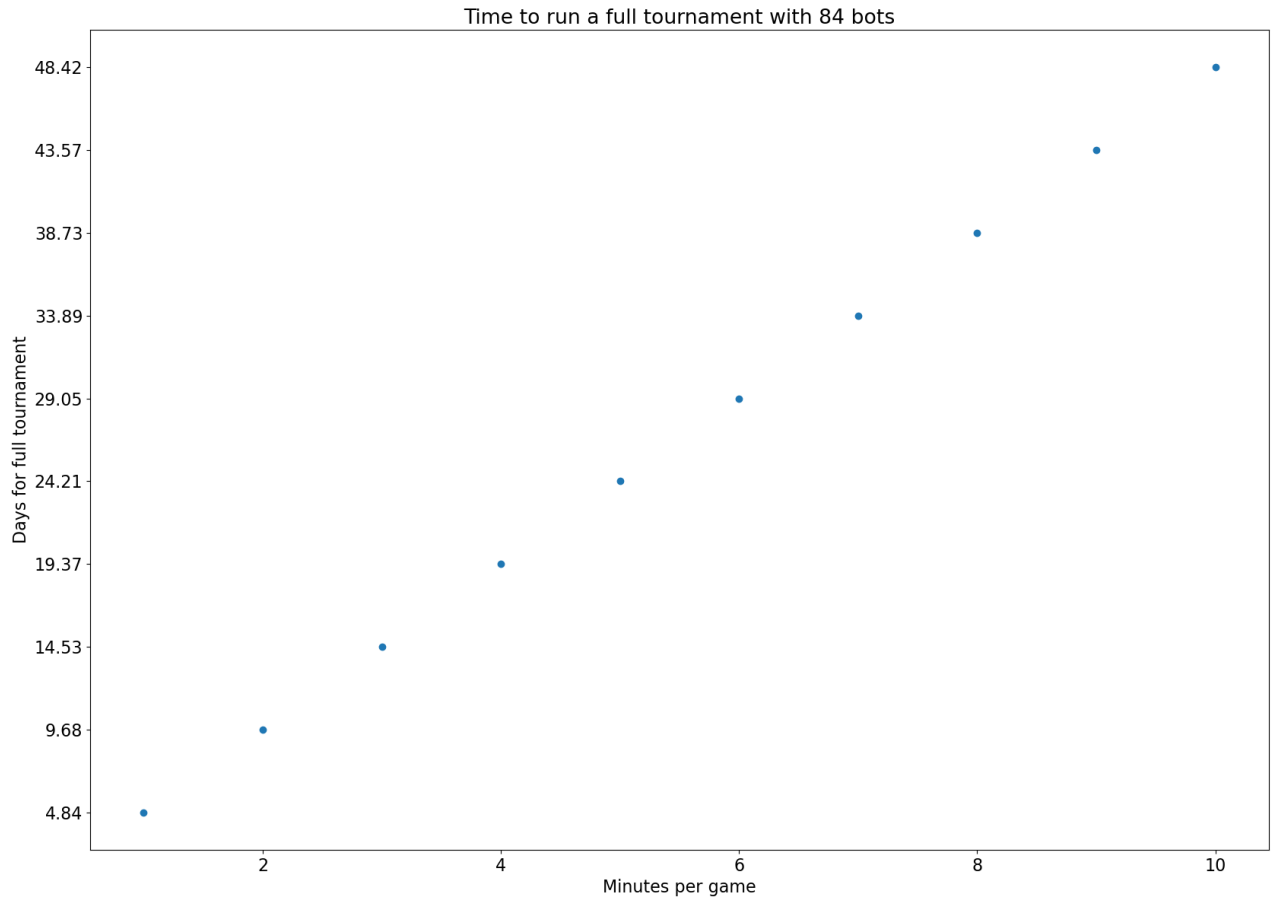
Presentation: Each group will give a 10min presentation describing the approach.

Journal: Each group is given a journal on Blackboard, which no other group can see. Each student should make entries of their contributions, and there needs to be a description of what was ultimately submitted.

Performance mark

New approach this year. Each submitted bot will be played against a set of marking bots to determine the win performance. Speed performance marked relative to speed of all submitted bots.

Why no tournament this year?



Thus endth the lectures
for this course, anyway