Computers vs Humans

We computers finally beat you humans at Go.

Okay, I'll apply 10,000 years of CPU time to the initial —

What's next? Which quintessentially human thing should we learn to do better than you?

Sucks for you!

Being too cool to care about stuff.

Mm Hmm.

Sounds like you've already lost.

Damn. This is hard.

Is it? Never noticed.

https://xkcd.com/1875/
Fifty shades of winning

“It’s not just about winning and losing… It’s about the point spread.”

Payoff: One player wants it as big as possible, the other wants it as small as possible.

Zero-sum assumption: my gain is your loss, and vice versa.

Can formulate basic win-loss scenario in this setting.

What if one move guarantees me a payoff of 1.0 … but another move guarantees me a payoff of 10.0? I would go for the bigger payoff!

Principle: Suppose for each move available to me, I know the largest guaranteed payoff. Which move should I choose? The one that guarantees the largest payoff!
The Game-Tree Game

Two players, + and -. Game starts at root.
At a + node, the + player gets to choose which child.
At a - node, the - player gets to choose which child.
The game ends when a leaf is reached, at which point the - player pays the + player the amount labeling the leaf.
The Game-Tree Game

Two players, + and -. Game starts at root.
At a + node, the + player gets to choose which child.
At a - node, the - player gets to choose which child.
The game ends when a leaf is reached, at which point the - player pays the + player the amount labeling the leaf.
The Game-Tree Game

Strategy for playing the game-tree game:
First figure out the value of each node.
+ player goes to child with largest value,
- player goes to child with smallest value.

What is value of a node?
The payoff if both players play optimally.
How can you compute it?
Value of a leaf is just the label.
Value of a + node is maximum of its child values.
Value of a - node is minimum of its child values.
The Game-Tree Game

Strategy for playing the game-tree game:
First figure out the value of each node.
+ player goes to child with largest value,
- player goes to child with smallest value.

What is value of a node?
The payoff if both players play optimally.
How can you compute it?
Value of a leaf is just the label.
Value of a + node is maximum of its child values.
Value of a - node is minimum of its child values.

Quiz: Find the values
Finding value

Suppose tree is represented by procedure `successors: node -> node list`

Define `type node_label = Plus | Minus | Leaf of float`

Suppose you have procedure `get_label: node -> node_label`

Quiz: Write procedure `compute_value: node -> float`

**input:** a node in a game tree

**output:** value of that node
Every game is game-tree game
(Well, every deterministic two-player zero-sum game.)

As discussed in previous lecture, each node represents state of the game. Choose one player, say Player 1, to be the + player. For state that is a win for Player 1, use +1.0 as the payoff. For state that is a win for Player 2, use -1.0 as the payoff. For state that is a draw, use 0.0 as the payoff.

Use value to decide which move to make. For Player 1, choose whichever move brings game to state with largest value. For Player 2, choose whichever move brings game to state with smallest value.

Does this work? Not quite…. 
Static game-state evaluator

The algorithm (*minimax algorithm*) is right but …

It would take far too long to explore whole game tree. Instead, use a procedure to estimate payoff of a state. Terminate search after only three or four ply. (A *ply* is what we ordinarily call a *turn.*)

The payoff estimator is called `estimate_value` in our project. Usually it is called a *static evaluator*. *(Evaluator* because it computes a value. *Static* because does not consider moves or different states.

Static evaluator depends on game. Traditionally, designed by expert (e.g. chess grandmaster).
Game AIs

Static evaluator depends on game
Traditionally, designed by expert (e.g., chess grandmaster)

Latest work using deep learning to develop the static evaluator

ARTICLE

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

David Silver*, Aja Huang*, Chris J. Maddison¹, Arthur Guez¹, Laurent Sifre¹, George van den Driessche¹, Julian Schrittwieser¹, Ioannis Antonoglou¹, Veda Pannerevelam¹, Marc Lanctot¹, Sander Dieleman¹, Doinik Grew¹, John Nham⁶, Nal Kalchbrenner², Ilya Sutskever⁷, Timothy Lillicrap¹, Madeleine Leach⁷, Koray Kavukcuoglu⁰, Thore Graepel² & Demis Hassabis⁷
There’s more to game-tree search than minimax

In graph search, we used successP/existsP
This procedure used short-circuiting: stopped after finding first true result
In first attempt at game-tree search, we used existsP and allP
These procedures used short-circuiting

   existsP stopped after finding first true result
   allP stopped after finding first false result

What is the equivalent of short-circuiting for minimax algorithm?

How can we stop early when computing maximum or minimum?
There’s more to game-tree search than minimax

Quiz: Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

Quiz: Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

**Key idea:** You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

**Key idea:** You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

**Key idea:** You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.

We now know value $\geq 27$
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

**Key idea:** You don’t need to know values of all nodes to choose move.

We now know value \( \geq 27 \)
There’s more to game-tree search than minimax

Quiz: Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

**Key idea:** You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

Quiz: Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

Quiz: Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

Quiz: Choose best move for + player and compute value.

Key idea: You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

**Key idea:** You don’t need to know values of all nodes to choose move.
There’s more to game-tree search than minimax

**Quiz:** Choose best move for + player and compute value.

![Game tree diagram](image)

**Key idea:** You don’t need to know values of all nodes to choose move.
Alpha-beta pruning

Short-circuiting *minimum* procedure:

**input:** float $\alpha$, procedure $f$, list $[x_1; x_2; \ldots ; x_k]$

**output:** minimum among $[f \ x_1; f \ x_2; \ldots ; f \ x_k]$ if minimum is greater than $\alpha$, else $-\infty$

If you already know that overall value is at least $\alpha$ then any node with value less than $\alpha$ is irrelevant.

If $f \ x_1 \leq \alpha$, no need to explore $x_2$ or $x_3$.

If $f \ x_2 \leq \alpha$, no need to explore $x_3$.

**Strategy for alpha-pruning:** keep track of best overall value $\alpha$ achieved so far. Can use $\alpha$ to prune the search, not evaluate nodes if they are not relevant.

**Programming:** the search procedure takes an extra argument, $\alpha$.

We do short-circuiting in *minimum* procedure—why not also *maximum* procedure?
Alpha-beta pruning

Short-circuiting minimum procedure:
**input**: float $\alpha$, procedure $f$, list $[x_1, x_2, \ldots, x_k]$  
**output**: minimum among $[f(x_1); f(x_2); \ldots; f(x_k)]$ if minimum is greater than $\alpha$, else $-\infty$

If you already know that overall value is at least $\alpha$ then any node with value less than $\alpha$ is irrelevant.

Short-circuiting maximum procedure:
**input**: float $\beta$, procedure $f$, list $[x_1, x_2, \ldots, x_k]$  
**output**: maximum among $[f(x_1); f(x_2); \ldots; f(x_k)]$ if max is less than $\beta$, else $\infty$

If you already know that overall value is at most $\beta$ then any node with value greater than $\beta$ is irrelevant.

If $f(x_1) \geq \beta$, no need to explore $x_2$ or $x_3$.
If $f(x_2) \geq \beta$, no need to explore $x_3$.

Programming strategy: keep track of both $\alpha$ and $\beta$, short-circuit max and min.
One more game-tree search technique: iterative deepening

How many ply to search?

Often you have a time limit
but you don’t know how long a given ply depth would take.

**Iterative deepening:**
- try depth 1
- if you haven’t run out of time, try depth 2
- if you still haven’t run out of time, try depth 3
- and so on