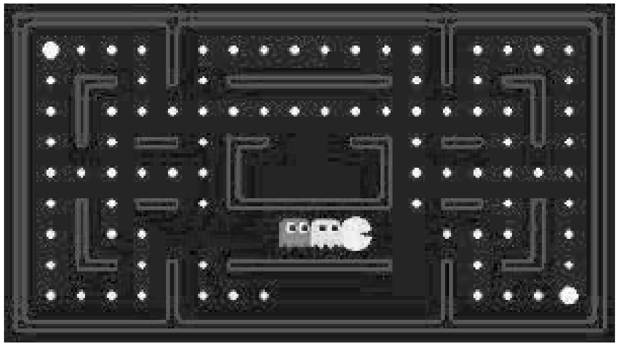
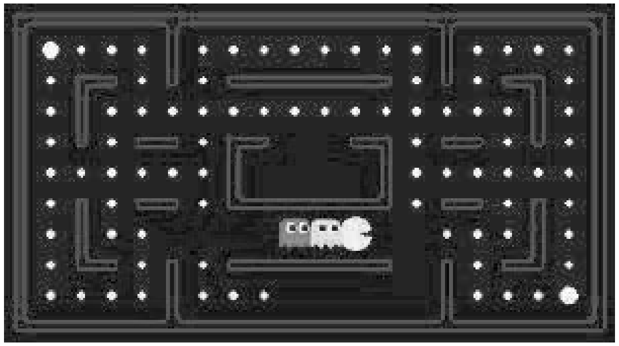
Games II



Announcement

- Midterm is next week (Wednesday, 5/8, 6pm-8pm)
- Topics: all material up to and including today's lecture
- Logistics: look for detailed post on Ed

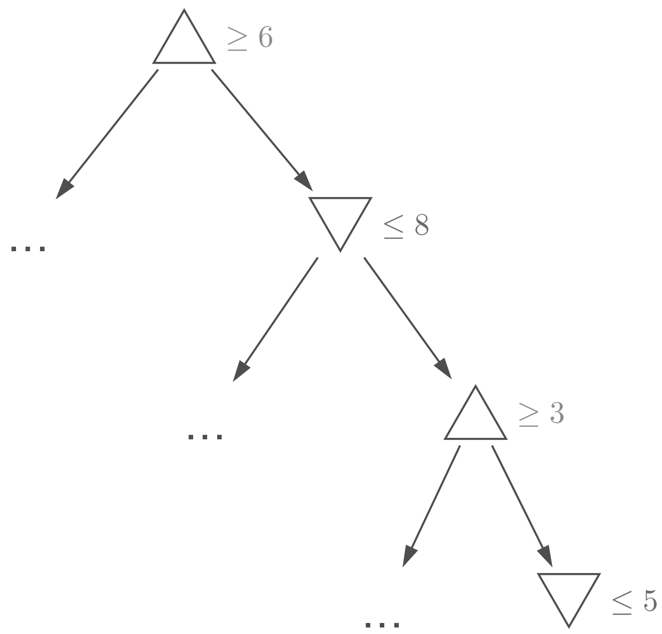
Games: alpha-beta pruning recap



Alpha-beta pruning

Key idea: optimal path

The optimal path is path that minimax policies take.
Values of all nodes on path are the same.

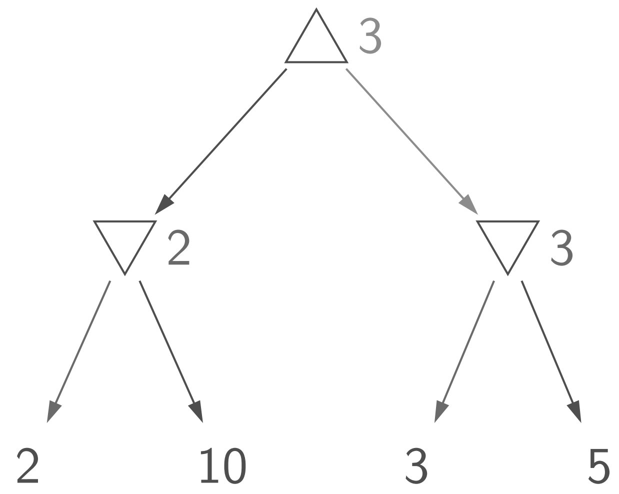


- a_s: lower bound on value of max node s
- b_s: upper bound on value of min node s
- Prune a node if its interval doesn't have non-trivial overlap with every ancestor (store $\alpha_s = \max_{s' \preceq s} a_{s'}$ and $\beta_s = \min_{s' \preceq s} b_{s'}$)

Move ordering

Pruning depends on order of actions.

Can't prune the 5 node:



[live solution: alpha-beta pruning]

Move ordering

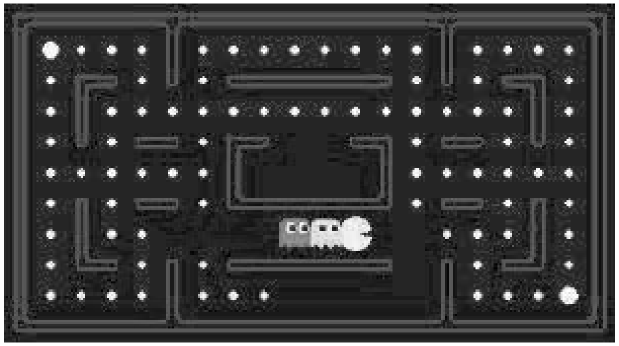
Which ordering to choose?

- Worst ordering: $O(b^{2 \cdot d})$ time
- Best ordering: $O((\sqrt{b - \frac{3}{4}} + \frac{1}{2})^{2 \cdot d}) \simeq O(b^{2 \cdot 0.5d})$ time
- Random ordering: $O(b^{2 \cdot 0.75d})$ time when $b = 2$
- Random ordering: $O((\frac{b-1+\sqrt{b^2+14b+1}}{4})^{2 \cdot d})$ for general b

In practice, can use evaluation function Eval(s):

- Max nodes: order successors by decreasing Eval(s)
- Min nodes: order successors by increasing Eval(s)

Games: TD-learning



Evaluation function

Old: hand-crafted

Example: chess

$$\begin{aligned}
&\text{Eval}(s) = \text{material} + \text{mobility} + \text{king-safety} + \text{center-control} \\
&\text{material} = 10^{100}(K - K') + 9(Q - Q') + 5(R - R') + \\
&\qquad\qquad 3(B - B' + N - N') + 1(P - P') \\
&\text{mobility} = 0.1(\text{num-legal-moves} - \text{num-legal-moves}') \\
&\ldots
\end{aligned}$$

New: learn from data

$$\text{Eval}(s) = V(s; \mathbf{w})$$

Model for evaluation functions

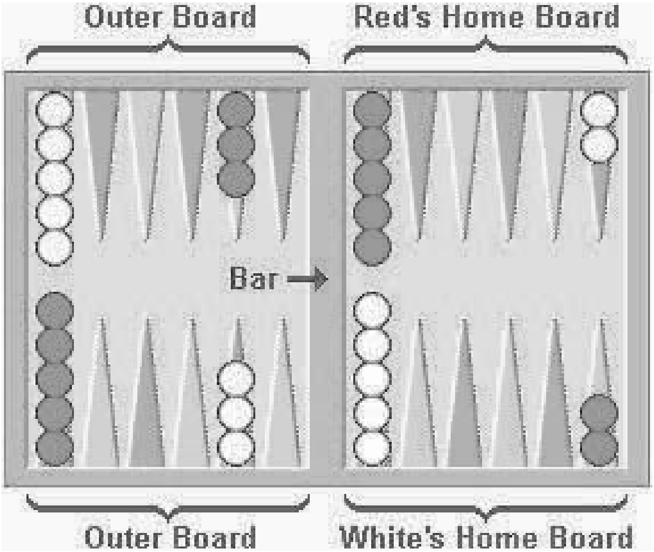
Linear:

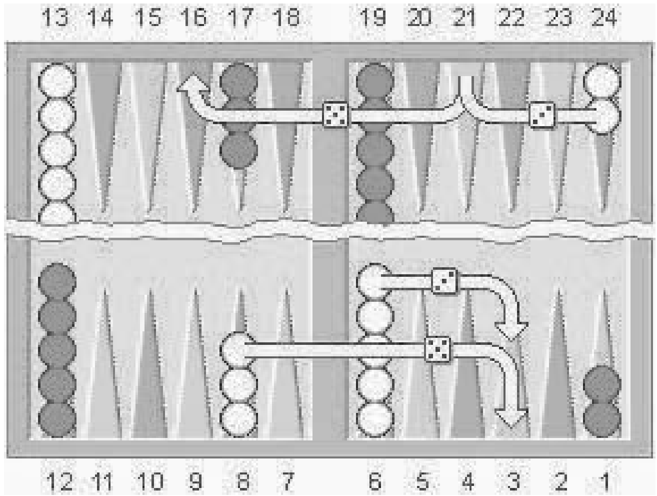
$$V(s; \mathbf{w}) = \mathbf{w} \cdot \phi(s)$$

Neural network:

$$V(s; \mathbf{w}, \mathbf{v}_{1:k}) = \sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(s))$$

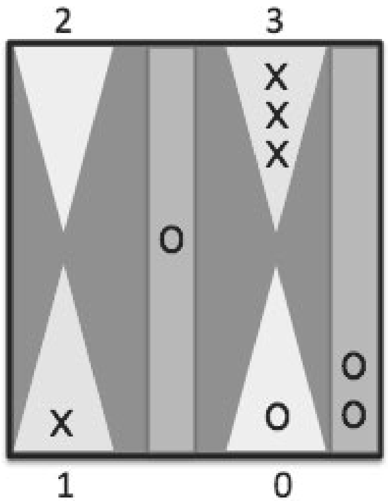
Example: Backgammon





Features for Backgammon

state s



Features $\phi(s)$:

$[(\# \text{ o in column 0}) = 1]$: 1

$[(\# \text{ o on bar})]$: 1

$[(\text{fraction o removed})]$: $\frac{1}{2}$

$[(\# \text{ x in column 1}) = 1]$: 1

$[(\# \text{ x in column 3}) = 3]$: 1

$[(\text{is it o's turn})]$: 1

Generating data

Generate using policies based on current $V(s; \mathbf{w})$:

$$\pi_{\text{agent}}(s; \mathbf{w}) = \arg \max_{a \in \text{Actions}(s)} V(\text{Succ}(s, a); \mathbf{w})$$

$$\pi_{\text{opp}}(s; \mathbf{w}) = \arg \min_{a \in \text{Actions}(s)} V(\text{Succ}(s, a); \mathbf{w})$$

Note: don't need to randomize (ϵ-greedy) because game is already stochastic (backgammon has dice) and there's function approximation

Generate episode:

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$$

Learning algorithm

Episode:

$s_0; a_1, r_1, s_1; a_2, r_2, s_2, a_3, r_3, s_3; \ldots, a_n, r_n, s_n$

A small piece of experience:

$$(s, a, r, s')$$

Prediction:

$$V(s; \mathbf{w})$$

Target:

$$r + \gamma V(s'; \mathbf{w})$$

General framework

Objective function:

$$\frac{1}{2}(\text{prediction}(\mathbf{w}) - \text{target})^2$$

Gradient:

$$(\text{prediction}(\mathbf{w}) - \text{target})\nabla_{\mathbf{w}}\text{prediction}(\mathbf{w})$$

Update:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \underbrace{(\text{prediction}(\mathbf{w}) - \text{target})\nabla_{\mathbf{w}}\text{prediction}(\mathbf{w})}_{\text{gradient}}$$

Temporal difference (TD) learning

Algorithm: TD learning

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{V(s;\mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma V(s';\mathbf{w}))}_{\text{target}}]\nabla_{\mathbf{w}} V(s;\mathbf{w})$$

For linear functions:

$$V(s;\mathbf{w}) = \mathbf{w} \cdot \phi(s)$$

$$\nabla_{\mathbf{w}} V(s;\mathbf{w}) = \phi(s)$$

Example of TD learning

Step size $\eta = 0.5$, discount $\gamma = 1$, reward is end utility

Example: TD learning

S1	**r:0**	**S4**	**r:0**	**S8**	**r:1**	**S9**
$\phi : \binom{0}{1}$		$\phi : \binom{1}{0}$		$\phi : \binom{1}{2}$		$\phi : \binom{1}{0}$
$\mathbf{w} : \binom{0}{0}$	p:0	$\mathbf{w} : \binom{0}{0}$	p:0	$\mathbf{w} : \binom{0}{0}$	p:0	$\mathbf{w} : \binom{0.5}{1}$
	t:0		t:0		t:1	
	p-t:0		p-t:0		p-t:-1	

S1	**r:0**	**S2**	**r:0**	**S6**	**r:0**	**S10**
$\phi : \binom{0}{1}$		$\phi : \binom{1}{0}$		$\phi : \binom{0}{0}$		$\phi : \binom{1}{0}$
$\mathbf{w} : \binom{0.5}{1}$	p:1	$\mathbf{w} : \binom{0.5}{0.75}$	p:0.5	$\mathbf{w} : \binom{0.25}{0.75}$	p:0	$\mathbf{w} : \binom{0.25}{0.75}$
	t:0.5		t:0		t:0.25	
	p-t:0.5		p-t:0.5		p-t:-0.25	

Comparison

Algorithm: TD learning

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{V}_\pi(s; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_\pi(s'; \mathbf{w}))}_{\text{target}}] \nabla_\mathbf{w} \hat{V}_\pi(s; \mathbf{w})$$

Algorithm: Q-learning

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \max_{a' \in \text{Actions}(s)} \hat{Q}_{\text{opt}}(s', a'; \mathbf{w}))}_{\text{target}}] \nabla_\mathbf{w} \hat{Q}_{\text{opt}}(s, a; \mathbf{w})$$

Comparison

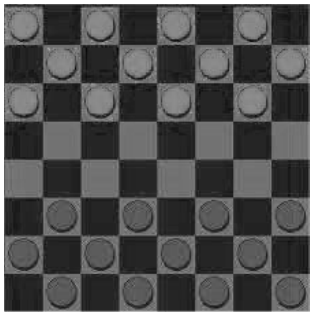
Q-learning:

- Operate on $\hat{Q}_{\text{opt}}(s, a; \mathbf{w})$
- Off-policy: value is based on estimate of optimal policy
- To use, don't need to know MDP transitions $T(s, a, s')$

TD learning:

- Operate on $\hat{V}_{\pi}(s; \mathbf{w})$
- On-policy: value is based on exploration policy (usually based on $\hat{V}_{\pi}$)
- To use, need to know rules of the game $\text{Succ}(s, a)$

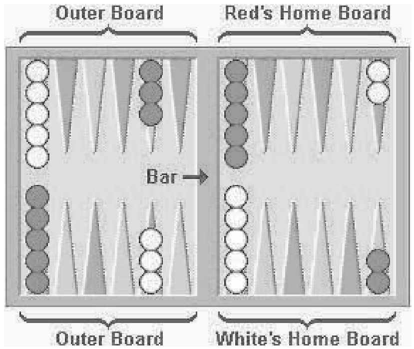
Learning to play checkers



Arthur Samuel's checkers program [1959]:

- Learned by playing itself repeatedly (self-play)
- Smart features, linear evaluation function, use intermediate rewards
- Used alpha-beta pruning + search heuristics
- Reach human amateur level of play
- IBM 701: 9K of memory!

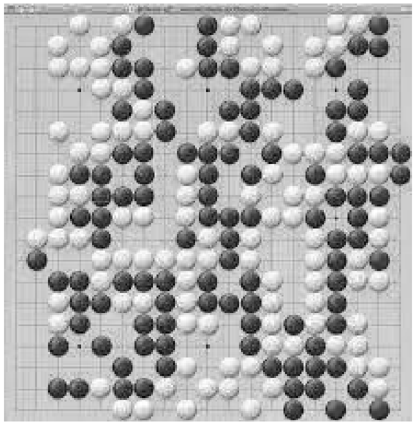
Learning to play Backgammon



Gerald Tesauro's TD-Gammon [1992]:

- Learned weights by playing itself repeatedly (1 million times)
- Dumb features, neural network, no intermediate rewards
- Reached human expert level of play, provided new insights into opening

Learning to play Go



AlphaGo Zero [2017]:

- Learned by self play (4.9 million games)
- Dumb features (stone positions), neural network, no intermediate rewards, Monte Carlo Tree Search
- Beat AlphaGo, which beat Le Sedol in 2016
- Provided new insights into the game

Summary so far

- Parametrize evaluation functions using features
- TD learning: learn an evaluation function

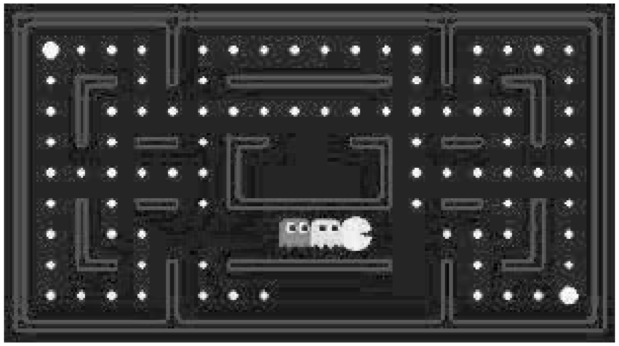
$$(\text{prediction}(\mathbf{w}) - \text{target})^2$$

Up next:

Turn-based $\longrightarrow$ Simultaneous

Zero-sum $\longrightarrow$ Non-zero-sum

Games: simultaneous games



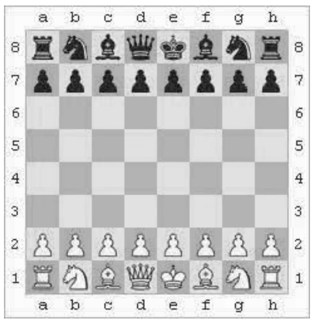
think and share

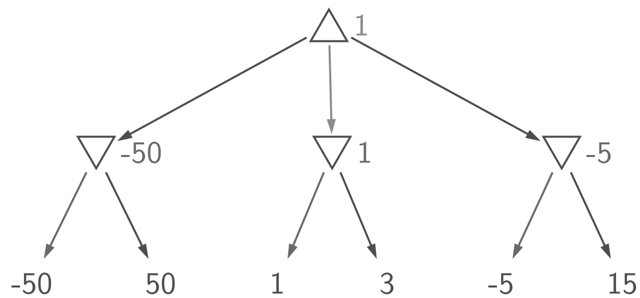
Question

For a simultaneous two-player zero-sum game (like rock-paper-scissors), can you still be optimal if you reveal your strategy?

yes
no

Turn-based games:





Simultaneous games:

?

Two-finger Morra

Example: two-finger Morra

Players A and B each show 1 or 2 fingers.
If both show 1, B gives A 2 dollars.
If both show 2, B gives A 4 dollars.
Otherwise, A gives B 3 dollars.

[play with a partner]

think and share

Question

What was the outcome?

player A chose 1, player B chose 1
player A chose 1, player B chose 2
player A chose 2, player B chose 1
player A chose 2, player B chose 2

Payoff matrix

Definition: single-move simultaneous game

Players $=\{\text{A},\text{B}\}$

Actions: possible actions

$V(a,b)$: **A's utility** if A chooses action a, B chooses b

(let V be **payoff matrix**)

Example: two-finger Morra payoff matrix

A \ B	1 finger	2 fingers
1 finger	2	-3
2 fingers	-3	4

Strategies (policies)

Definition: pure strategy

A pure strategy is a single action:
$a \in$ Actions

Definition: mixed strategy

A mixed strategy is a probability distribution
$0 \leq \pi(a) \leq 1$ for $a \in$ Actions

Example: two-finger Morra strategies

Always 1: $\pi = [1, 0]$
Always 2: $\pi = [0, 1]$
Uniformly random: $\pi = [\frac{1}{2}, \frac{1}{2}]$

Game evaluation

Definition: game evaluation

The **value** of the game if player A follows π_{A} and player B follows π_{B} is

$$V(\pi_{\mathsf{A}}, \pi_{\mathsf{B}}) = \sum_{a,b} \pi_{\mathsf{A}}(a)\pi_B(b)V(a,b)$$

Example: two-finger Morra

Player A always chooses 1: $\pi_{\mathsf{A}} = [1, 0]$

Player B picks randomly: $\pi_{\mathsf{B}} = [\frac{1}{2}, \frac{1}{2}]$

Value: $\boxed{-\frac{1}{2}}$

[whiteboard: matrix]

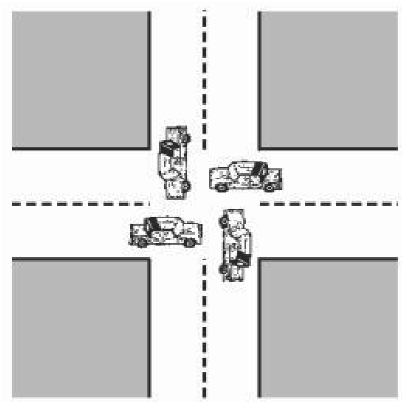
How to optimize?

Game value:

$$V(\pi_A, \pi_B)$$

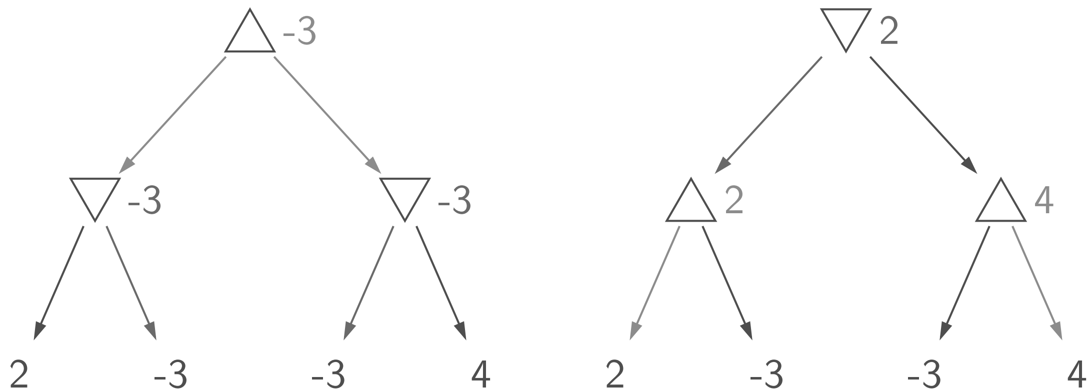
Challenge: player A wants to maximize, player B wants to minimize...

simultaneously



Pure strategies: who goes first?

Player A goes first: Player B goes first:



Proposition: going second is no worse

$$\max_a \min_b V(a,b) \leq \min_b \max_a V(a,b)$$

Mixed strategies

Example: two-finger Morra

Player A reveals: $\pi_{\mathsf{A}} = [\frac{1}{2}, \frac{1}{2}]$

Value $V(\pi_{\mathsf{A}}, \pi_B) = \pi_B(1)(-\frac{1}{2}) + \pi_B(2)(+\frac{1}{2})$

Optimal strategy for player B is $\pi_B = [1, 0]$ (**pure!**)

Proposition: second player can play pure strategy

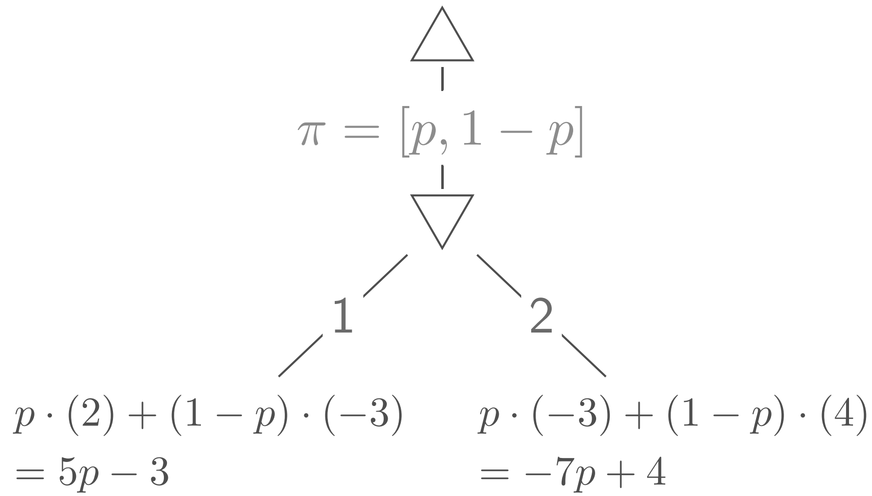
For any fixed mixed strategy π_{A}:

$$\min_{\pi_{\mathsf{B}}} V(\pi_{\mathsf{A}}, \pi_B)$$

can be attained by a pure strategy.

Mixed strategies

Player A first reveals his/her mixed strategy

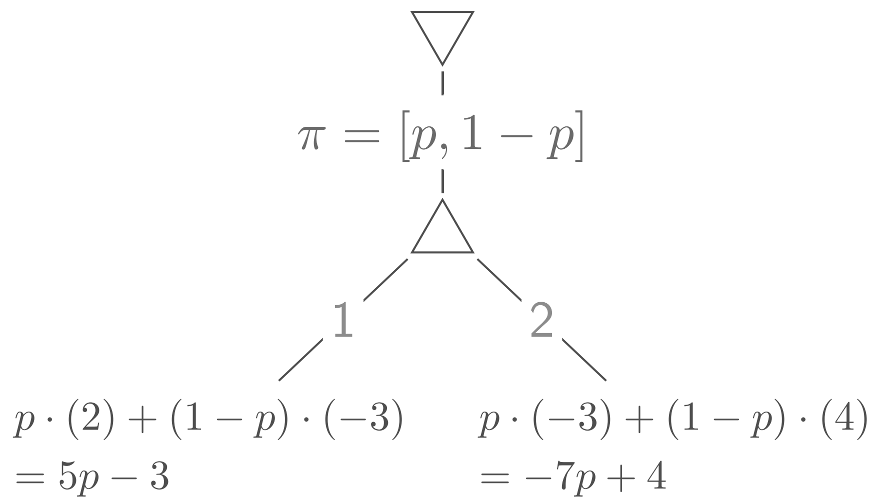


Minimax value of game:

$$\max_{0 \leq p \leq 1} \min\{5p-3, -7p+4\} = \boxed{-\frac{1}{12}} \text{ (with } p = \tfrac{7}{12}\text{)}$$

Mixed strategies

Player B first reveals his/her mixed strategy



Minimax value of game:

$$\min_{p \in [0,1]} \max\{5p-3, -7p+4\} = \boxed{-\frac{1}{12}} \text{ (with } p = \tfrac{7}{12}\text{)}$$

General theorem

Theorem: minimax theorem [von Neumann, 1928]

For every simultaneous two-player zero-sum game with a finite number of actions:

$$\max_{\pi_{\mathsf{A}}} \min_{\pi_{\mathsf{B}}} V(\pi_{\mathsf{A}}, \pi_{\mathsf{B}}) = \min_{\pi_{\mathsf{B}}} \max_{\pi_{\mathsf{A}}} V(\pi_{\mathsf{A}}, \pi_{\mathsf{B}}),$$

where $\pi_{\mathsf{A}}, \pi_{\mathsf{B}}$ range over **mixed strategies**.

Upshot: revealing your optimal mixed strategy doesn't hurt you!

Proof: linear programming duality

Algorithm: compute policies using linear programming

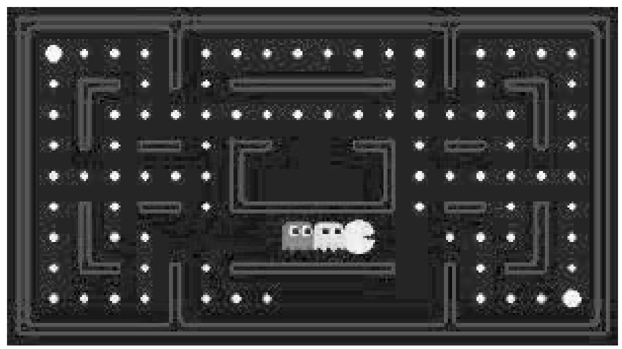
Summary

- Challenge: deal with simultaneous min/max moves

- Pure strategies: going second is better

- Mixed strategies: doesn't matter (von Neumann's minimax theorem)

Games: non-zero-sum games



Utility functions

Competitive games: minimax (linear programming)

Collaborative games: pure maximization (plain search)

Real life: ?

Prisoner's dilemma

Example: Prisoner's dilemma

Prosecutor asks A and B individually if each will testify against the other.

If both testify, then both are sentenced to 5 years in jail.
If both refuse, then both are sentenced to 1 year in jail.

If only one testifies, then he/she gets out for free; the other gets a 10-year sentence.

[play with a partner]

think and share

Question

What was the outcome?

player A testified, player B testified
player A refused, player B testified
player A testified, player B refused
player A refused, player B refused

Prisoner's dilemma

Example: payoff matrix

B \ A	testify	refuse
testify	$A = -5, B = -5$	$A = -10, B = 0$
refuse	$A = 0, B = -10$	$A = -1, B = -1$

Definition: payoff matrix

Let $V_p(\pi_{\mathsf{A}}, \pi_{\mathsf{B}})$ be the utility for player p.

Nash equilibrium

Can't apply von Neumann's minimax theorem (not zero-sum), but get something weaker:

Definition: Nash equilibrium

A **Nash equilibrium** is $(\pi^*_{\mathsf{A}}, \pi^*_{\mathsf{B}})$ such that no player has an incentive to change his/her strategy:

$$V_{\mathsf{A}}(\pi^*_{\mathsf{A}}, \pi^*_B) \geq V_{\mathsf{A}}(\pi_{\mathsf{A}}, \pi^*_{\mathsf{B}}) \text{ for all } \pi_A$$
$$V_{\mathsf{B}}(\pi^*_{\mathsf{A}}, \pi^*_B) \geq V_{\mathsf{B}}(\pi^*_{\mathsf{A}}, \pi_{\mathsf{B}}) \text{ for all } \pi_B$$

Theorem: Nash's existence theorem [1950]

In any finite-player game with finite number of actions, there exists **at least one** Nash equilibrium.

Examples of Nash equilibria

Example: Two-finger Morra

Nash equilibrium: A and B both play $\pi = [\frac{7}{12}, \frac{5}{12}]$.

Example: Collaborative two-finger Morra

Two Nash equilibria:

- A and B both play 1 (value is 2).
- A and B both play 2 (value is 4).

Example: Prisoner's dilemma

Nash equilibrium: A and B both testify.

Summary so far

Simultaneous zero-sum games:

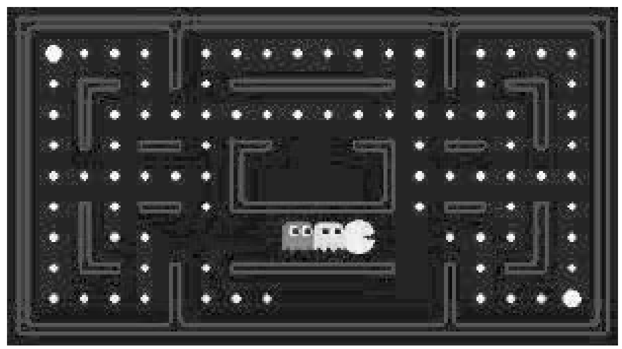
- von Neumann's minimax theorem
- Multiple minimax strategies, single game value

Simultaneous non-zero-sum games:

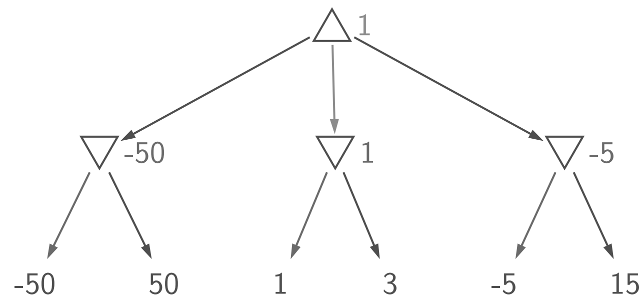
- Nash's existence theorem
- Multiple Nash equilibria, multiple game values

Huge literature in game theory / economics

Games: recap



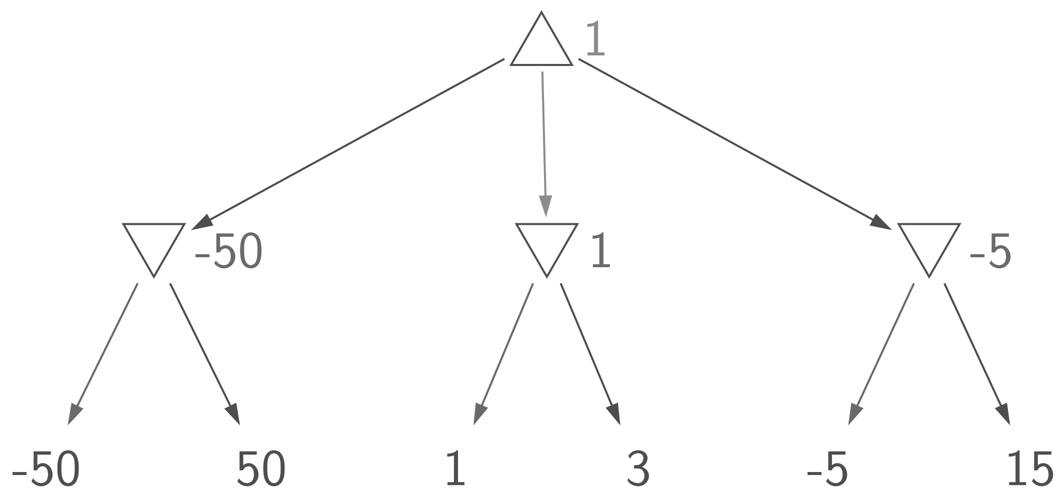
Summary



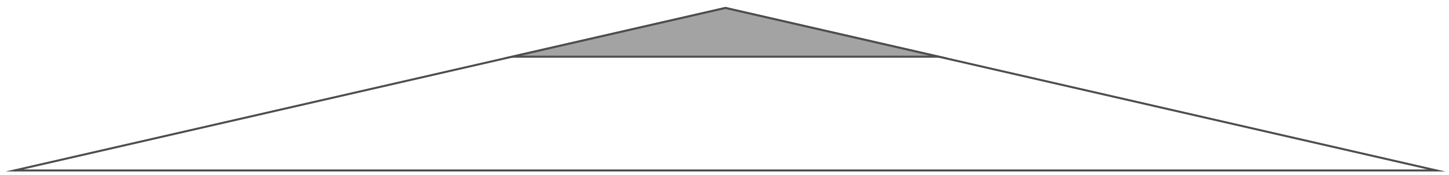
- Game trees: model opponents, randomness
- Minimax: find optimal policy against an adversary
- Evaluation functions: domain-specific, approximate
- Alpha-beta pruning: domain-general, exact

Review: minimax

agent (max) versus opponent (min)



Review: depth-limited search



$$V_{\text{minmax}}(s, d) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \text{Eval}(s) & d = 0 \\ \max_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s, a), d) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s, a), d - 1) & \text{Player}(s) = \text{opp} \end{cases}$$

Use: at state s, choose action resulting in $V_{\text{minmax}}(s, d_{\text{max}})$

Summary

- Main challenge: not just one objective
- Minimax principle: guard against adversary in turn-based games
- Simultaneous non-zero-sum games: mixed strategies, Nash equilibria
- Strategy: **search game tree + learned evaluation function**

Chess

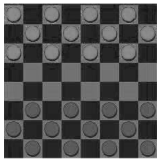
1997: IBM's Deep Blue defeated world champion Gary Kasparov

Fast computers:

- Alpha-beta search over 30 billion positions, depth 14
- Singular extensions up to depth 20

Domain knowledge:

- Evaluation function: 8000 features
- 4000 "opening book" moves, all endgames with 5 pieces
- 700,000 grandmaster games
- Null move heuristic: opponent gets to move twice



Checkers

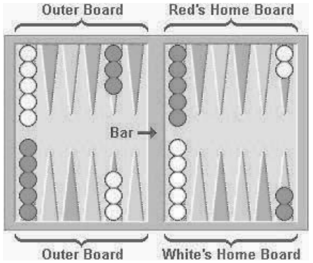
1990: Jonathan Schaeffer's **Chinook** defeated human champion; ran on standard PC

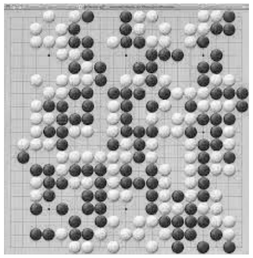
Closure:

- 2007: Checkers solved in the minimax sense (outcome is draw), but doesn't mean you can't win
- Alpha-beta search + 39 trillion endgame positions

Backgammon and Go

Alpha-beta search isn't enough...



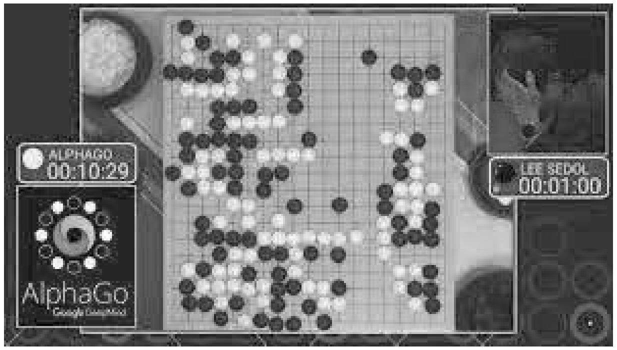


Challenge: large branching factor

- Backgammon: randomness from dice (can't prune!)
- Go: large board size (361 positions)

Solution: learning

AlphaGo

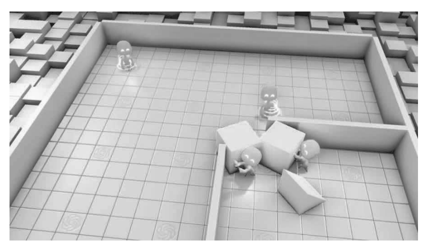


- Supervised learning: on human games
- Reinforcement learning: on self-play games
- Evaluation function: convolutional neural network (value network)
- Policy: convolutional neural network (policy network)
- Monte Carlo Tree Search: search / lookahead

Coordination games

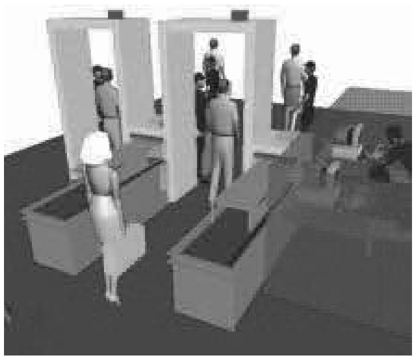
Hanabi: players need to signal to each other and coordinate in a decentralized fashion to collaboratively win.

Hide-and-Seek: OpenAI has developed agents with emergent behaviors to play hide and seek.



Other games

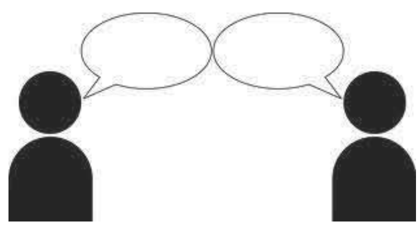
Security games: allocate limited resources to protect a valuable target. Used by TSA security, Coast Guard, protect wildlife against poachers, etc.



Other games

Resource allocation: users share a resource (e.g., network bandwidth); selfish interests leads to volunteer's dilemma

Language: people have speaking and listening strategies, mostly collaborative, applied to dialog systems



Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Markov networks

Bayesian networks

Reflex **States** **Variables** **Logic**

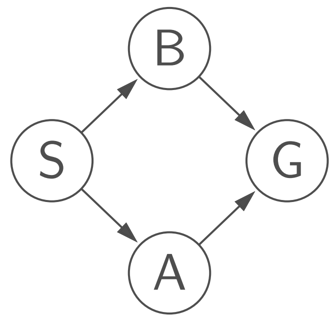
Low-level → High-level

Machine learning

State-based models

[Modeling]		
Framework	search problems	MDPs/games
Objective	minimum cost paths	maximum value policies
[Inference]		
Tree-based	backtracking	minimax/expectimax
Graph-based	DP, UCS, A*	value/policy iteration
[Learning]		
Methods	structured perceptron	Q-learning, TD learning

State-based models: takeaway 1

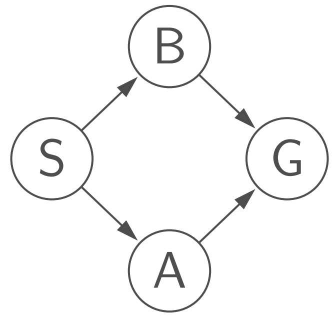


Key idea: specify locally, optimize globally

Modeling: specifies local interactions

Inference: find globally optimal solutions

State-based models: takeaway 2



Key idea: state

A **state** is a summary of all the past actions sufficient to choose future actions **optimally**.

Mindset: move through states (nodes) via actions (edges)

Homework

due: next week

作业 4(optional)周5-图像识别
作业 5(optional)-周5-语音识别
作业 6(optional)-周5-文本识别