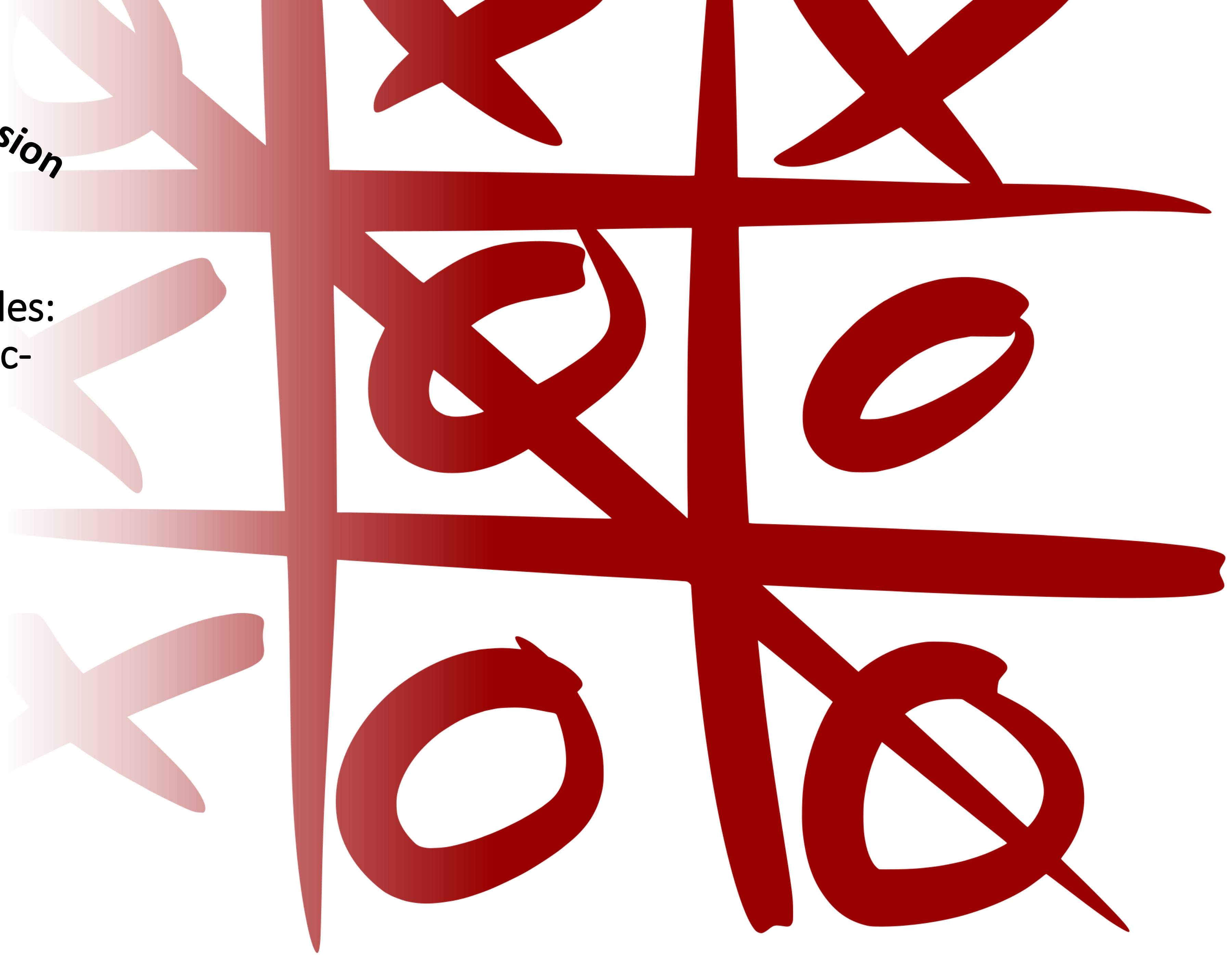
Discussion

CS 5/7320
Artificial Intelligence

Learning from Examples: Learning to Score a Tic-Tac-Toe board

AIMA Chapter 19

Michael Hahsler



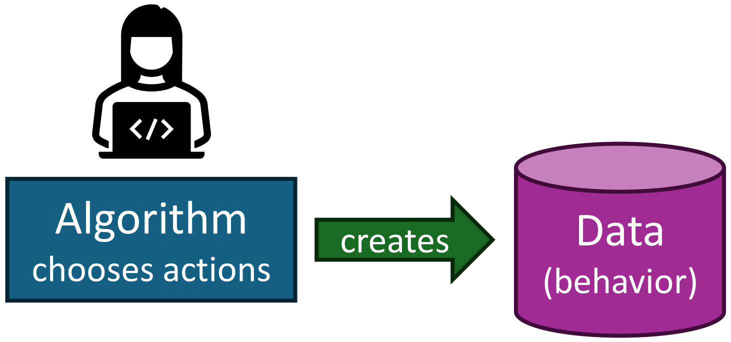
Intro: Learning from Examples

What can be learned?

Learning from Examples

Up until now in this course:

- **Hand-craft algorithms** to make rational/optimal or at least good decisions.
  Examples: Search strategies, heuristics, and constructing Bayesian networks.

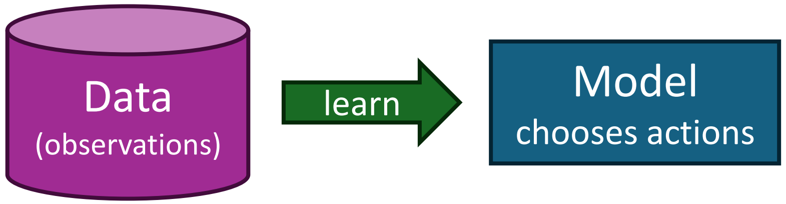


Issues

- We may not be able to anticipate all possible future scenarios.
- We may have examples, but we do not know how to implement a solution.

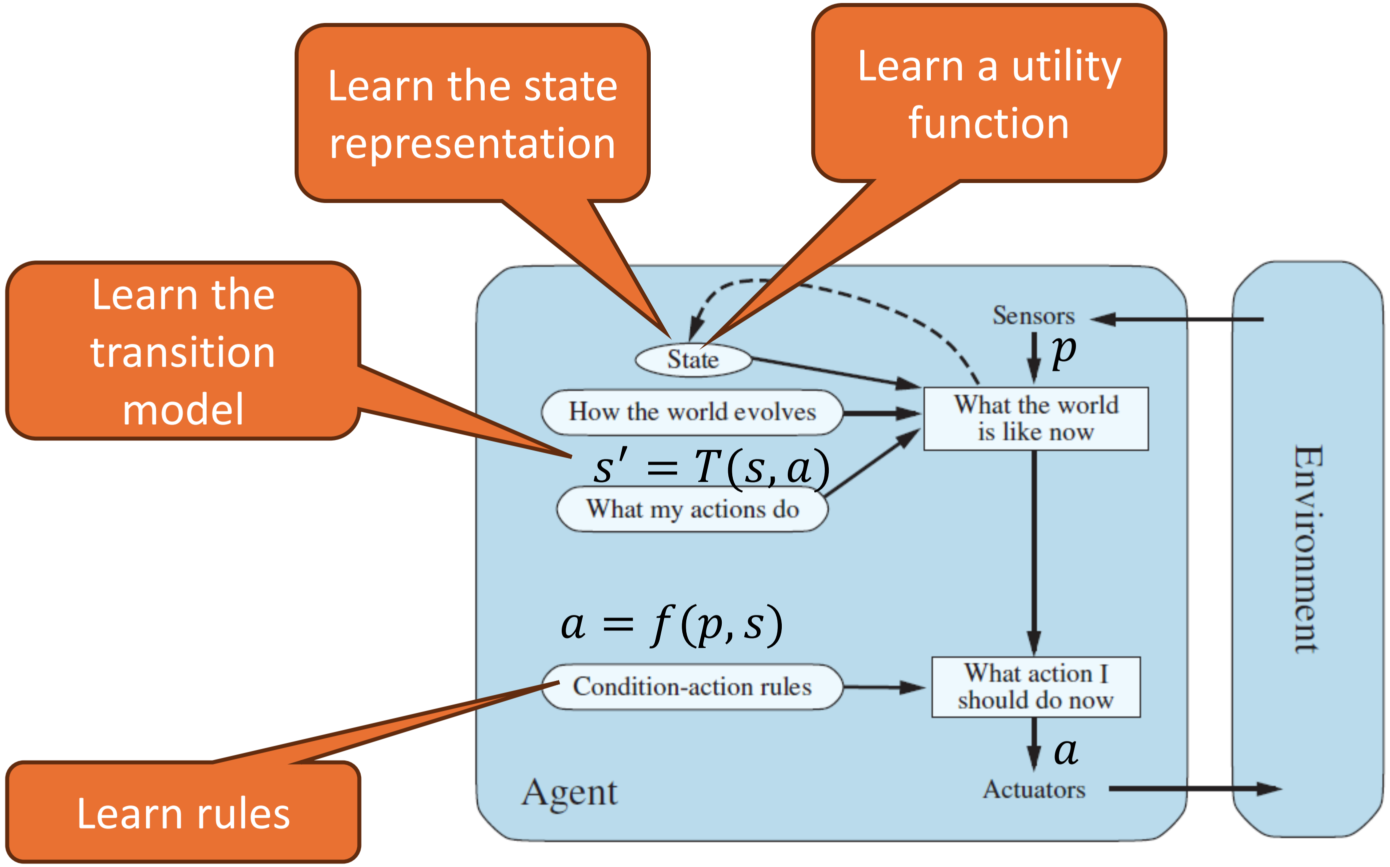
Supervised Machine Learning

- Uses observations: training data with the correct answers.
- Learn a function (model) to map an input (e.g., state) to an output (e.g., action) representing the desired behavior.
- Examples:
  - Use a naïve Bayesian classifier to distinguish between spam/non-spam.
  - Learn a playout policy to simulate games (current board -> good move)



Learning Components of an Agent

- We can learn many different components of an agent from examples
- **Example**: Learning components of a model-based reflex agent



Typical Use of Supervised ML for Intelligent Agents

Learn a Policy

- Classification: Directly learn the best action for each state from examples.

  $a = h(state\ features)$

- This model can also be used as a **playout policy** for Monte Carlo tree search with data from self-play.

Learn Evaluation Functions

- Regression: Learn evaluation functions to estimate state utilities.

  $eval = h(state\ features)$

- Can learn a **heuristic** for heuristic alpha-beta search.
- For reinforcement learning we can learn action values $q(state, action)$.

Learn Perception and Actuation

- **Natural language processing:** Use deep learning / word embeddings / language models to understand concepts, translate between languages, or generate text.
- **Speech recognition**: Identify the most likely sequence of words.
- **Vision**: Object recognition in images/videos. Generate images/video.
- **Robotics**: Learn how to move safely.

Compressing Tables

- Neural networks can be used as a compact representation of tables that do not fit in memory. E.g.,
  - Joint and conditional probability tables
  - State utility tables (i.e., an evaluation function)
  - Q-Value tables in reinforcement learning

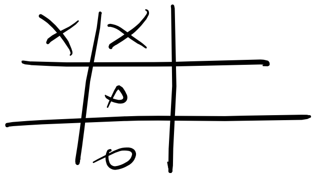
Bottom line: Learning a function is often more effective than hard-coding it. However, we do not always know how it performs for rare and edge cases!

Example: Tic-Tac-Toe

Approaches to implement/learn an evaluation function.

Utility of Being in a State: State Value

State s



$value(s)$

Approach: Try all actions and pick the action that leads to the state with the largest state value.

$$argmax_{a \in actions(s)}\ value(result(s,a))$$

The state value is defined as:

$$value(s) = \mathbb{E}_{\pi_x,\pi_o}[U(s)],$$

with the playout policies π_x, π_o of player x and o.

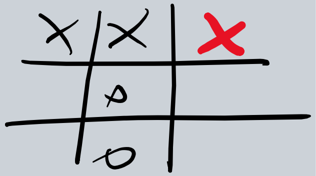
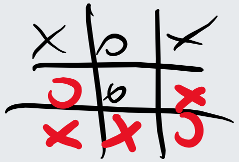
Expectation is used for stochastic playout strategies!

Questions:

- What playout policies do we use? Random, optimal, …
- How do we estimate the expected value?

Minimax and Variations

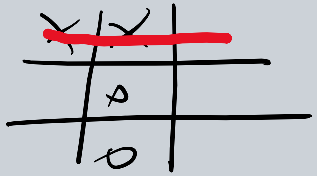
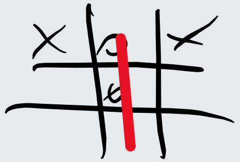
$value(s) \approx \mathbb{E}_{\pi_x,\pi_o}[U(s)]$

State s	$minimax(s)$	Expert code $eval(s)$		
	1			
	0			
⋮				

Playout policy π_x, π_o	optimal play
Method	Tree search

Expert Evaluation Function

$value(s) \approx \mathbb{E}_{\pi_x,\pi_o}[U(s)]$

State s	$minimax(s)$	Expert code $eval(s)$	Monte Carlo simulation $eval(s)$	
	1	+0.4		
	0	-0.4		
⋮				

Expert Heuristic

+.4 for each winning opportunity

-.4 for each loosing opportunity

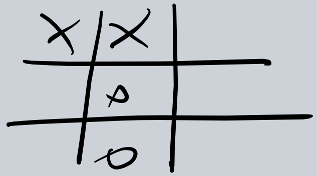
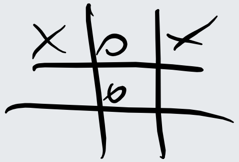
Playout policy π_x, π_o	optimal play	none
Method	Tree search	Code + cutoff search

Monte Carlo Simulation

$$value(s) \approx \mathbb{E}_{\pi_x,\pi_o}[U(s)]$$

Monte Carlo Sim.

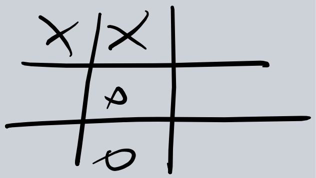
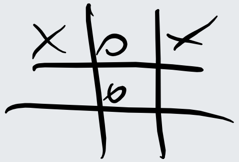
$$\approx \frac{1}{N}\sum_{i=1}^{N} u_i(s)$$

State s	$minimax(s)$	Expert code $eval(s)$	Monte Carlo simulation $eval(s)$	Machine Learning $h(s)$
	1	+0.4	0.634	
	0	-0.4	-0.021	
⋮				

Playout policy π_x, π_o	optimal play	none	Random or other
Method	Tree search	Code + cutoff search	Simulation

Machine Learning

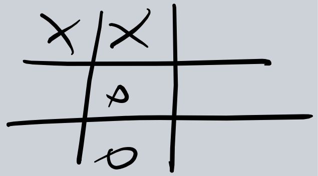
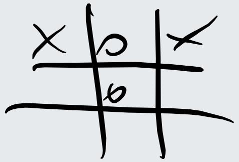
$value(s) \approx E_{\pi_x,\pi_o}[U(s)]$

State s	$minimax(s)$	Expert code $eval(s)$	Monte Carlo simulation $eval(s)$	Machine Learning $h(s)$
	1	+0.4	0.634	
	0	-0.4	-0.021	
⋮				

Playout policy π_x, π_o	optimal play	none	Random or other
Method	Tree search	Code + cutoff search	Simulation

Machine Learning

$value(s) \approx \mathbb{E}_{\pi_x,\pi_o}[U(s)]$

State s	$minimax(s)$	Expert code $eval(s)$	Monte Carlo simulation $eval(s)$	Machine Learning $h(s)$
	1	+0.4	0.634	0.325
	0	-0.4	-0.021	0.042
⋮				

train ML

Playout policy π_x, π_o	optimal play	none	Random or other	For training data
Method	Tree search	Code + cutoff search	Simulation	Machine learning

All methods hopefully produce a consistent state order.

Reinforcement Learning (RL)

Tries to learn state values directly from rewards resulting from

- **Model-based RL**: Uses a complete probabilistic model of the environment.
- **Model-free RL**: Learns from interactions with the environment.
    - Online learning
    - Offline learning (can use supervised learning from a sampled dataset)

Machine learning for an evaluation function can be seen as a special case of model-free offline RL called (neural) fitted Q-learning with replay memory.

We will talk about other RL methods later.