L22 – A Quick Introduction to Game Theory

AIMA4e: Ch 5.1–2, 5.4–5; Ch 18.1–3

What you should know after this lecture

- Two-player zero-sum complete information alternating games can be handled similarly to MDPs
- Fully cooperative games can be handled as POMDPs with execution-time constraints

Problem setting

- Centralization: Is the process of selecting actions performed jointly for all the agents or separately?
  - Online centralization: basically, then, we just have a single-agent decision problem with multiple outputs
  - Offline centralization: we can run a single policy-optimization process that produces a set of policies for de-centralized execution
  - No centralization: agents make completely independent decisions
- Payoffs:
  - In an zero-sum (usually two-player) game, player A's loss is always exactly equal to player B's game.
  - In a fully-cooperative game, all players have the same payoff for all states/actions.
  - Otherwise, it is a general-sum game.
- Turn-taking:
  - In an alternating game, players take actions sequentially in a fixed order.
  - Otherwise, they may decide and execute in parallel.
  - Very little work lets them execute completely asynchronously.
- Observations: may have complete, partial, or no observability of
  - World state
  - Other agents' actions
- Communication
  - May have extra "side channels" for communication
  - Can understand actions taken by one agent and observed by another as communication

Some simple cases

- Centralized decision-making (and observations): really a path-search, MDP, or POMDP with large action space
- Decentralized decision-making but with centralized "planning/optimization" time: do some kind of policy optimization to find individual policies for the agents that perform well under decentralized execution.

Game theorists use "cooperative games" to handle much more complex situations in which players don't really have the same payoffs, but can make strategic alliances.

Mechanism design

Mechanism design is a way of setting up decision-making processes that have the property that if agents collectively make their decisions in that way, certain fairness-type outcomes will be guaranteed. Various types of auctions are an example of this.

Example: a sealed-bid second-price auction: the highest bidder wins but pays the second highest price.

The optimal strategy is for each player to bid their actual utility.

Value to the seller is the second-highest utility (which is actually also true, in a certain limit, for first-price auctions).

Two-player, turn-taking, zero-sum, perfect information games

Chess, Go, Backgammon, etc.

- Generally discrete state and action spaces
- Zero-sum: $U(s, p_1) = -U(s, p_2)$
- Added state variable: $\textit{to\_move} \in \{p_1, p_2\}$

Big question: what does it mean for a plan or policy to be optimal in this case?

If you know your opponent's policy, then it becomes a path-search problem (or MDP if the game has stochastic transitions).

Otherwise, we often look for the minimax optimal strategy: each player assumes the other player is perfectly rationally trying to win, and acts rationally under that assumption.

Note that the minimax optimal strategy isn't necessarily the best thing to play against a stupid opponent!

Minimax optimal strategy

- Solving online: Can make a "game tree"
  - Similar to alternating layers of expectimax, but this time we alternate between min and max
  - Max is the player selecting their current move, assuming the other player will take actions to try to minimize their score
  - Will find the optimal strategy if the whole tree can be searched
  - alpha-beta pruning is a cool strategy for reducing search-space size while retaining optimality
- If the game has a stochastic transition function, then you need to add expectation layers, between the min and max layers.
- If the tree is too big to search, then we are generally left with MCTS-type strategies
- Learned heuristic guidance (in the form of policies for biasing which actions to expand during search and estimated Q values) are very helpful! (See alpha-Go-zero, etc!)

Offline minimax strategies

Just as for MDPs, it's theoretically possible to compute an optimal value function, and hence, policy, using dynamic programming! Here's the value function for player max, assuming deterministic transitions, and a terminal 0-1 reward:

$$V_{\max}(s) = \begin{cases} +1 & \text{if } s \text{ is a win for max} \\ -1 & \text{if } s \text{ is a win for min} \\ \max_a V_{\min}(T(s,a)) & \text{otherwise} \end{cases}$$

$$V_{\min}(s) = -V_{\max}(s)$$

- Depending on the game, the state might need to include steps left to go (if finite horizon).
- Just need to compute one value function!

Adding stochasticity

$$V_{\max}(s) = \begin{cases} +1 & \text{if } s \text{ is a win for max} \\ -1 & \text{if } s \text{ is a win for min} \\ \max_a \sum_{s'} P(s' \mid s, a) V_{\min}(s') & \text{otherwise} \end{cases}$$

$$V_{\min}(s) = -V_{\max}(s)$$

- Still just need to compute one value function!
- Can use systematic value iteration to compute this
- In large spaces, do "self-play":
  - Basically RL in simulation, but alternating player perspectives
  - Still need to worry about function approximation strategy and exploration strategy

Normal-form games

One-step, simultaneous actions, no constraint on payoffs. Much harder! What does it mean to act rationally?
Prisoner's dilemma: two players, each can **Cooperate** or **Defect**:

Player A \ Player B	**Cooperate**	**Defect**
Cooperate	(3, 3)	(0, 5)
Defect	(5, 0)	(1, 1)

- Dominant strategy for player A: **Defect**, maximizes utility independent of other player's choice (same for B)
- If we allow binding agreements or repeated play, we get a higher payoff for everyone.

Nash equilibrium

What if there's no dominant strategy? Can look for a pair of strategies (π_1, π_2) such that:

- If player 1 knows that player 2 is going to do π_2 then π_1 maximizes player 1's expected payoff, and
- If player 2 knows that player 1 is going to do π_1 then π_2 maximizes player 2's expected payoff.

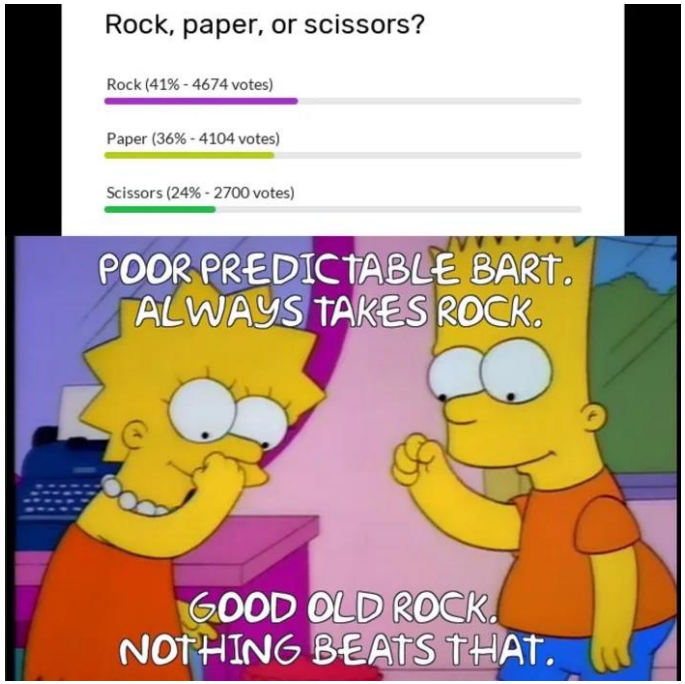
Player A \ Player B	**Left**	**Right**
Left	(0, 0)	(1, 1)
Right	(1, 1)	(0, 0)

- Two Nash equilibria: (L, R) and (R, L)
- Requires coordination to get a good payoff

Randomized policies

In MDPs and games like Backgammon, there is an optimal deterministic policy. But not necessarily true in matrix games. Here the Nash equilibrium is a mixed (randomized) strategy.

Bart \ Lisa	Rock	Paper	Scissors
Rock	(0, 0)	(-1, 1)	(1, -1)
Paper	(1, -1)	(0, 0)	(-1, 1)
Scissors	(-1, 1)	(1, -1)	(0, 0)



Straightforward linear algebra proof that there's a single Nash equilibrium in which each player chooses uniformly at random.

So much more!

- Ways of solving coordination problems
- What if we iterate a matrix game?
- What if two players are interacting in an MDP-like situation? Can seek a Nash equilibrium in the space of simple policies, for example.
- What if we have sequential games with partial information? (e.g. Poker)

Take Gabriele Farina's multi-agent learning class!

L22 – Learning and learning!

May 11, 2026

These are lpk's informal notes
3 relevant kinds of learning

- models : "synthetic"
- speed-up search / planning / infr : "analytic"
- amortized inference avoid online interest : "analytic"

1 Model learning

Propositional logic

- given many exmaple possibly partial interpretations
- discover some clauses that are always true algorithms: "rule mining"
- generate and test (unsupervised)
- pick one proposition and try to find sparse predictor as logical function of others (e.g. as decision tree)

First-order learning Similar idea. inductive logic programming

```
child(mary, bob).
child(bob, pat).

parent(bob, mary).
grandparent(pat, bob).

grandparent(A, B) :-
  parent(A, C),
  parent(C, B).

parent(A, B) :-
  child(B, A)
```

Can also do things like sorting algorithms!

```
sorted([1, 4, 6]).
not sorted([4, 2, 9]).
```

Top-down v bottom-up Bottom-up : Consume single data point at a time, move upward in a lattice
Unification : P(X, Y, X), P(a, Z, H) yields most general specialization P(a, Y, a)
Inverse unification : P(a, z, a) , P(b, 1, b) yields least general generalizer P(X, Y, X)
Top-down : first-order decision-tree learning e.g. FOIL Tests can be quantified: Ex. P(X, a)

Bayes Nets

1. Given structure, learn parameters : counting + laplace correction
2. Learn structure
   - Try to detect conditional independence relations
   - Local search in structure space

   Max likelihood of data but pressure complexity

   $$argmax_M P(D|M) - \lambda|M|$$

   Local greedy search:
   - don't have to recompute all the tables
   - respect equivalence classes of structures

   Chow/Liu
3. Learn parameters when some values are unobserved optimize $\sum_h P(\theta, h|D)$

   EM: $P(h|\theta, D)$, $\theta = argmax P(D, h|\theta)$

   Parameter estimation for a Hidden Markov Model

   Can also do gradient descent

   Can also add hidden structure, nodes, etc.

Markov Random Fields Counting doesn't work. Really just do gradient descent on likelihood.

Learning PDDL

- Roughly an ILP problem
- Look at what fluents changed (these have to be results)
- Find examples with conforming results
- Intersect the preconditions

Something about NSRTs if we have time

- grounding classifiers
- continuous parameters
- samplers

Learning an MDP

- If everything is small, it's just counting (like BN)
- If big, then we can use a neural network, but how to learn to predict a probability distribution?
  - Parametric representation with an "calibrated" loss function: logistic regression if discrete; or output $\mu\Sigma$. Train with samples of (x, y)
  - Generators: map noise and x into samples of P(Y — x) GAN, VAE, diffusion model
- If we only have a generator, then we could use it to make a simulator and do RL.
- Learn a big joint generative model (s, a, r, s, a, r, s, a, r, s) ...
  - Generate from it, conditioned on high reward, and s = s0
  - Need to do some kind of search at generation time EBM or diffusion
- Value iteration networks (instance of general strategy)

- Find an algorithm that works well, but models hard to specify
- Rather than estimating models using likelihood, but it all into a network and train end-to-end to perform well, coming up, implicitly, with parameters

VIN: input is map and your location, output is action Box includes transition model + VI algorithm Differentiate the whole thing! (T is convolutional in that case)

2 Speed-up learning

We have the model (learned or hand-written). We can do inference / search, but it's slow So: let's learn to reason more efficiently

- heuristic function / value function
- successor ordering in search (action distribution)
- value / var ordering in CSP
- search control in FOL theorem provers
- generating decompositions / abstractions (PLOI / CAMP)
- Alpha Go / Zero

Amortized inference Learn to answer queries of a specialized kind (e.g. always same query and evidence variables)

Can frame as supervised learning, but may be useful to keep some aspects of the inference algorithm, to help generalization

neural logic machines: Learn to infer truth value of complex FOL expression

3 What classes to take next?

Not exhaustive! Lots of options!

Logic

- 24.141 Logic I
- 24.242 Logic II
- 24.251 Modal Logic
- 24.251 Intro to Philosophy of Language
- 24.253 Philosophy of Mathematics
- 18.510 Intro to Mathematical Logic and Set Theory
- 18.515 Mathematical Logic

Probabilistic inference and statistics

- 6.7480 Information Theory: From Coding to Learning
- 6.7800 Inference and Information
- 6.7810 Algorithms for Inference
- 6.7830 Bayesian Modeling and Inference
- 14.38 Inference on Causal and Structural Parameters
- 14.39 Large-Scale Decision-Making and Inference
- 15.C08 Causal Inference

Decision and Game thy

- 6.3950 AI, Decision Making, and Society
- 16.410 Principles of Autonomy and Decision-Making (maybe too much overlap)
- 16.412 Cognitive Robotics
- 16.420 Planning Under Uncertainty (maybe too much overlap)
- 6.5340 Topics in Algorithmic Game Theory
- 14.12 Economic Applications of Game Theory
- 14.16 Strategy and Information
- 14.126 Game Theory
- 6.3260 Networks
- 14.163 Algorithms and Behavioral Science

Cognition and Philsophy

- 9.66 Computational Cognitive Science
- 24.233 Rationality
- 24.XXX AI and Rationality (to be approved); to be offered by lpk and Brian Hedden in Fall 25

Robotics

- 2.160 Identification, Estimation, an dLearning
- 2.165 Robotics
- 6.8200 Sensorimotor learning
- 6.4210 Robotic Manipulation
- 16.412 Cognitive Robotics

Optimization

- 6.C57 Optimization Methods