2016-04-06b, CS7641

RL-1

- Decision-Making & Reinforcement Learning
  - Recall our types of learning:
    - Supervised, $y = F(x)$ - trying to approximate F given (x, y)
    - Unsupervised, $F(x)$ - trying to cluster/describe
    - Reinforcement, $y = F(x)$ but we are given (x, z), not y, and want to approximate F
  - Reinforcement learning is one mechanism of decision making
  - Markov Decision Processes (MDPs)
    1. States: S - A set of tokens representing every state we could be in.
    2. Model: $T(s, a, s') \sim Pr(s' | s, a)$ - or transition function.
       state ↵ ↳ action
    3. Actions: $A(s)$ or A
       - What actions you can do, sometimes depends on state, sometimes is just a set and some actions do nothing in certain states.
    - The model is sort of the 'physics' of the system, or 'rules'. It gives the probability that action 'a' will transition you to s', given that you are in state s. If a deterministic world, then all probabilities are 0 or 1.
    - We've encoded the Markovian property in this: Only the current state (the 'present') ever matters, not past states.
      - But note that nearly anything can be made Markovian by encoding all sorts of past stuff into state.
    - Also: Our rules are stationary (i.e. our model).
    4. Reward: $R(s)$, $R(s, a)$, $R(s, a, s')$ - some scalar value attached to a state, or state + action, or state + action + actually reaching state. (Actually, all equivalent.) Value can be negative to penalize too.
       - This is where one encodes domain knowledge.
    - The 'solution' to an MDP is the policy, $\pi(s) \rightarrow a$, telling an action to take for any given state. π^* is the optimal policy (maximizes expected long-term reward)

RL-1 (cont.)

- Basically, π^* is the f for $y = f(x)$. The reward is 'z', the actions are 'y', the states are 'x'.
  Note that in supervised learning we'd be given $\langle s, a \rangle$ from π, and need to estimate π. Here we're given $\langle s, a, r \rangle$ and need to estimate optimal $\langle s, a \rangle$ (i.e. π).
- Note that a policy doesn't tell a sequence of actions. It tells a single action, for a given state. This matters when the system is stochastic.
- What makes reinforcement learning MDP problems different is that reward is delayed; whole long strings of actions may give no information on their own & and have no reward, but set up for a state that does give reward.
  Minor changes may matter, but this is never reflected in the immediate rewards.
  - Name for this: "credit assignment problem" or "temporal credit assignment problem".

Credit covers blame too.

- A small negative reward in most states can be used to encourage shorter paths and to reach terminal states.
- And in general rewards matter a lot and are where your domain knowledge goes.
- We've been making two assumptions:
  - Infinite horizons (any number of actions can happen)
  - Utility of sequences
- Finite horizons matter. The policy may need to change if you have only, say, 3 or 4 moves left - even if in the same state. Maybe we need $\pi(s, t) \to a$, not $\pi(s) \to a$.
  (But we don't discuss this further in this course.)
- Utility of sequences: Consider utility function U.
  If $U(s_0, s_1, s_2, \ldots) > U(s_0, s_1', s_2', \ldots)$ then
  $U(s_1, s_2, \ldots) > U(s_1', s_2', \ldots)$.

"stationarity of preferences"

  - We've assumed a certain way of adding rewards.
  - Actually: Nothing but addition preserves this property.

2016-04-06c, CS7641

RL-1 (cont.)

- In math terms: $U(s_0, s_1, s_2, \ldots) = \sum_{s=0}^{\infty} R(s_t)$ ← rewards
- But this has problems...
  If it's an infinite sum of rewards (and all divergent) then what difference does it make? $U \to \infty$ anyhow.
  What we do is then irrelevant. The system's "immortal".

"Discounted rewards"

- One trick around this: $\sum_{t=0}^{\infty} \gamma^t R(s_t)$ where $0 \le \gamma < 1$

  This is $\le \sum \gamma^t R_{max}$ - geometric series, equaling $\frac{R_{max}}{1-\gamma}$
  It's a finite horizon, sort of, but it's always the same distance away.
- Optimal policies

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,\middle|\, \pi\right]$$

$$U^{\pi}(s) = E\left[\sum \gamma^t R(s_t) \,\middle|\, \pi, s_0 = s\right]$$ - Utility of policy at a state is the result of starting at s and following policy.

- N.B. $R(s)$ is immediate reward, $U(s)$ is <u>long-term</u> reward.
  → Delayed reward!

Then... $\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') U(s')$ for U using π^*. (Yes, that's recursive.)

Bellman Equation →

So... $U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$

Reward now — Reward later (discounted by γ)

(same as the 'curse of Dimensionality' guy)

If we can solve for $U(s)$ then we can solve for π^*.

- So, say we have n states, and thus $U(s)$ has n unknowns.
  But 'max' complicates this. One way to solve:
  - Start with arbitrary utilities.
  - Update utilities based on neighbors
  - Repeat!

  (until convergence)

→ While we add this in at every step, it's also discounted.

$$\hat{U}_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') \hat{U}_t(s')$$

for estimate $\hat{U}_t$ at time step t.

So this weight is based on both estimates & probabilities.

Value Iteration (a.k.a. backward induction)

-and can be turned to linear programming?

RL-1 (cont.)

- Value iteration works quite well.
- Example in the final quiz is a good illustration.
- Note that U contains much more information than π! U may have incorrect utilities - but the correct ordering of actions, enough to find π.
- Compare with the first part of this class where we could do correct classification from Bayesian learning & "good enough" probabilities.
- Here, "good enough" utility function might give us an optimal policy, and correct utility is unnecessary.

Policy Iteration

- A revised algorithm:
  - Start with $\pi_0 \leftarrow$ guess
  - evaluate: given π_t calculate $U_t = U^{\pi_t}$ (Utility of following π_t)
  - improve: $\pi_{t+1} = \underset{a}{\operatorname{argmax}} \sum T(s, a, s') U_t(s')$

We use $U_t(s) = R(s) + \gamma \sum_{s'} T(s, \pi_t(s), s') U_t(s')$

Yes. U_t is in both places.

Note that the above contains no 'max'. It is now linear. n linear equations, n unknowns - now solvable.

- This is like Value iteration but makes bigger jumps and works in policy space, not value space. Thus policy iteration.
- Guaranteed to converge (finite # of policies - same idea as in k-means) and usually does so more quickly than value iteration.
- N.B. We've still not covered reinforcement learning. That's for next time.

Also planning:
BFS, DFS, A* } Deterministic (VI, PI are stochastic)

N.B.
Planning algorithm
↓
≠
Learning algorithm

2016-04-06d, CS7641

RL-2

Reinforcement Learning in MDPs

- "API":

Model (T,R) → [Planner] → Policy (π)

- This is sort of what we covered (Value & Policy Iteration)

Transitions ⟨s, a, r, s'⟩ (many) → [Learner] → Policy

— Reinforcement Learning

- It really just means "reward maximization" but we call it reinforcement learning for historical-ish reasons

Transitions → [Modeler] → Model

Model → [Simulator] → Transitions

Famous backgammon learner (TDGAMMON) did this

- Simulator & Learner together (Model → Policy) might be: RL-based planner?
- Modeler & Planner together (Transitions → Policy) might be: model-based RL
- Three approaches to RL:

s → [π] → a

- Policy search
  - Direct use (we learn π) but indirect learning

s → [U] → V

- Value-function based
  - Turns to policy-search easily
  - Less-indirect learning, more-indirect use

s, a → [T,R] → s', r'

- Model-based
  - Could use Bellman equations to turn T & R to U/π - indirect use
  - But, we can solve this pretty directly as supervised learning (T&R)

- We focus on the middle method as a 'just right' one

2016-04-07, CS7641

RL-2 (cont.)

- New kind of value function

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

- has elements of both U & π

→ Arriving in s, leaving via a, proceeding optimally thereafter

Note that $U(s) = \max_a Q(s,a)$ and $\pi(s) = \operatorname{argmax}_a Q(s,a)$

- So, how do we find Q?
- Q-learning: Estimating Q from transitions
  - It's easy if we have T & R but here we don't.
  - A transition has $\langle s, a, r, s' \rangle$.

Start with estimate $\hat{Q}$:

$$\hat{Q}(s,a) \xleftarrow{\alpha_t} r + \gamma \underbrace{\max_{a'} \hat{Q}(s',a')}_{\text{= utility of next state}}$$

where α_t = learning rate, $0 < \alpha < 1$

N.B. $V \xleftarrow{\alpha} x$ means $V \leftarrow (1-\alpha)V + \alpha x$ (moving α of the way from V to x)

- Interestingly, $V_t \xleftarrow{\alpha_t} x_t$, $x_t \sim X$ and $\alpha_t = \frac{1}{t}$, converges to $E[X]$

Actually this cheats a little because $\hat{Q}$ depends on time

- So, $\hat{Q}$ is then converging to $E[r + \gamma \max_{a'} \hat{Q}(s',a')]$
$= R(s) + \gamma E_{s'}[\max_{a'} \hat{Q}(s',a')] = R(s) + \gamma \sum_{s'} T(s,a,s') \hat{Q}(s',a')$
- $\hat{Q}$ can start nearly anywhere
  - Just update by the given rule, and it is guaranteed to converge, provided:

$$\sum \alpha_t = \infty, \quad \sum \alpha_t^2 < \infty$$

$s' \sim T(s,a,s')$, $r \sim R(s)$ — s & r are drawn from distributions

s,a need to be visited infinitely often. Must visit every (state, action) pair!

- Q-learning is actually a Family of algorithms.
  - How to initialize $\hat{Q}$?
  - How to decay α_t?
  - How to choose actions? We must try all of them infinitely often, and need to make use of $\hat{Q}$.

2016-04-07(b), CS7641

RL-2 (cont.)

- Greedy action selection has issue of running into local minima, especially if initial $\hat{Q}$ pushes you to "bad" actions and then just keeps reinforcing this.
  - One good way to work with this: random restarts. However, it's slow; it throws away everything each restart.
  - Perhaps, then, simulated annealing.
    $$\hat{\pi}(s) = \begin{cases} \arg\max_a \hat{Q}(s,a) & \text{with prob. } 1-\epsilon \quad (\text{small } \epsilon) \\ \text{random action} & \text{otherwise} \end{cases}$$
    - The randomness helps explore the whole space, while still we learn from iteration.
- ϵ-Greedy Expectation:
  - IF GLIE (greedy limit & infinite exploration), i.e. decayed ϵ, $\hat{Q} \to Q$ (learn) and $\hat{\pi} \to \pi^*$ (use) → Exploration vs. Exploitation again!
  - This is a pretty fundamental dilemma in RL.

RL is →

- Model learning + planning — But information must move between the two areas (well-studied in isolation)
- What we've not yet touched here: Where overfitting comes in.

Mitchell Ch. 13 (Reinforcement Learning)

- "Reinforcement learning addresses the question of how an autonomous agent that senses & acts in its environment can choose optimal actions to achieve its goals."
- Text uses similar definitions but has seemingly no transition matrix, and $V = U$ but not always identical?
- p377, section 13.3.4, gives proof that $\hat{Q}$ converges to true Q.
- Then p382, theorem 13.2, extends to nondeterministic.
- "Temporal Difference Learning"
- Q-learning approaches relate to dynamic programming approaches to solving MDPs.
  - See: Bellman-Ford shortest path algorithm.
- This chapter references many seminal works of the era at the end.

2016-04-11, CS7641

- Notes on: "Reinforcement Learning: A Survey" (Kaelbling, Littman, Moore, JAIR 1996)
  - Dynamic programming comes up frequently.
  - One classic problem: k-armed bandit (see p. 7)
    - Is this usable still with value & policy iteration?
- Model-free methods:
  - Adaptive Heuristic Critic
  - TD(λ)
- Model-based methods: "...especially important in applications in which computation is considered to be cheap & real-world experience costly." p19
  - Model-free methods make inefficient use of data gathered (which doesn't matter when that data is trivial to gather).
  - See Dyna algorithm & Prioritized Sweeping, Queue-Dyna p20
  - See RTDP (Real-Time Dynamic Programming) & Plexus planning system p23
- Problem with all methods so far: They assume that storing mappings like $S \to A$, $S \to \mathbb{R}$, $S \times A \to \mathbb{R}$, $S \times A \to S$, and $S \times A \times S \to [0,1]$ is feasible, & with large environments it is often not.

2016-04-12, CS7641

- Notes on "Reinforcement Learning" (Sutton & Barto) text
  - AI used to be seen as nearly entirely separate from control theory & statistics; it was logic and symbols and large LISP programs, not numbers, linear algebra, diff. eq., or statistics.
  - The modern view lacks this sharp distinction and acknowledges the usefulness of all those latter categories.
  - Reinforcement learning bridges AI with optimal control theory & stochastic approximation
  - All methods the text discusses further are based on finding value functions, but they need not be.

2016-04-12(b), CS7641

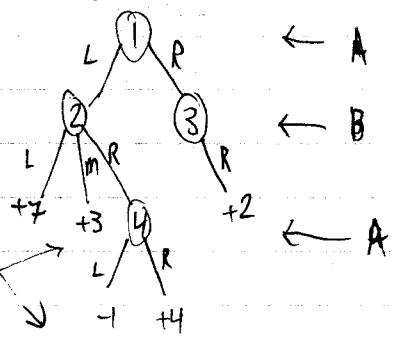
Sutton & Barto text

- For instance, we could use genetic algorithms, simulated annealing, or other optimization methods to search the space, without making explicit use of value functions.
- Generally, though, when we refer to reinforcement learning it involves interacting with some 'environment'. Evolutionary algorithms (as above) do not take advantage of this.
- Three threads combined in ~1980s to form modern RL:
  1. Early AI work
  2. The problem of optimal control & its solution with value functions & dynamic programming (1950s, Bellman equation)
  3. (Less distinct) Temporal-difference methods
- Dynamic programming is considered the only feasible way of solving general stochastic optimal control problems, despite suffering from curse of dimensionality.
- Though these all were developed separately to RL, the book considers them still as part of it.
- Minsky's 1961 paper, "Steps Toward Artificial Intelligence", was very influential & discussed credit assignment problem

2016-04-19, CS7641

RL-3, Game Theory

- Game Theory:
  - "Mathematics of Conflict" (or conflict of interest)
  - Single agent → Multiple agents (extend from RL)
  - Economics & politics - sometimes even sociology & biology
  - Increasingly a part of AI & ML!
- A simple game as an example; take agents A & B:
  - Shown is a game tree. Each node moves to another by the action shown on the branch. Agent A moves, then B. Rewards are
  - Agent A can move L or R in state 1...



- These rewards always go to A, & negative of reward to B
- This is specifically: a 2-player zero-sum finite deterministic game of perfect information
- The tree is sort of an 'unrolled' MDP.
- MDPs have policies mapping states to actions. Games have strategies for the same concept.
  - Pure strategies - A has 4, B has 3

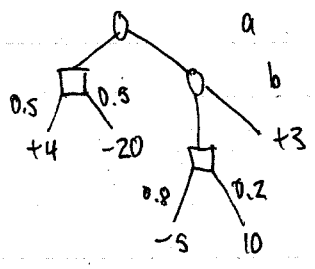
Written as a matrix:

	②	L	M	R
B	③	R	R	R

A \ B	① L	④ L	① L	④ R	① R	④ L	① R	④ R
L L	7	3	-1
L R	7	③	4
R L	2	2	2
R R	2	2	2

This is "the matrix form of the game" and it tells all that we need about the game.

But what do we do with this information? If we are agent A then how do we choose a strategy - knowing B will try to minimize A's reward?
A must consider worst-case counter by B; B also does the same for A. A wants to maximize, B wants to minimize...
→ Minimax strategy
the "value" of the game under this is 3 (circled)
- Result: In a 2-player, 0-sum deterministic game of perfect information, minimax = maximin. and an optimal pure strategy exists, always, for each player (& the matrix tells you it).
- Every row & column of matrix represent a strategy
- Another game tree:



the □ here is a chance node, with probabilities marked.

This is then a 2-player zero-sum, nondeterministic game of perfect information.

2016-04-19 (b), CS7641

RL-3 (cont.)

- So, what is the value of this game?

A		L	R
	L	-8	-8
	R	(-2)	3

B

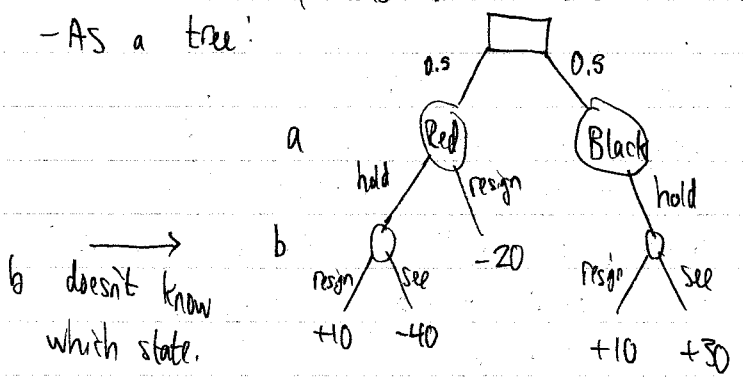
by weighted sums

- The probabilities just sort of fall out in the matrix. It uses the expected values here, as they are what matter.
- Value is -2.
- Given theorem still holds for non-deterministic games.
- This is <u>von Neumann's theorem.</u>

Yes, that von Neumann.

- Now relax another constraint - <u>hidden</u> information, not perfect.
  - Mini-Poker:
    - A is dealt card, 50% red or black. (red is bad for A, black is good)
    - A may resign if red: -20 cents for A (B doesn't see card)
      else A holds — B resigns, +10 cents (regardless of card)
      B sees — red — -40 cents
      black — +30 cents
    - Still zero-sum (B loses what A wins & vice versa)
    - As a tree:

(Andrew Moore example)



b doesn't know which state.

(A <u>could</u> resign on black but this is always a losing move.)

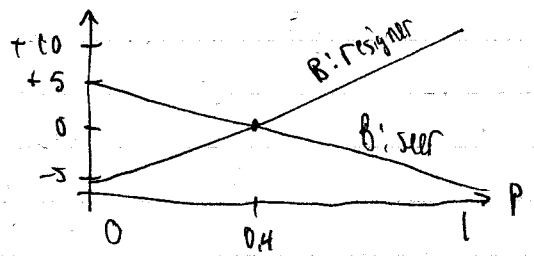
A:		B: resigner	B: seer
	resigner	-5	5
	holder	10	-5

A has really only 2 strategies (only choice is to hold/resign on red), likewise B has only 2.

- Use expected values again!

RL-3 (cont.)

- What is the value of the game?
  - A chooses first row, then B chooses first column (-5)
  - If B chooses first it picks 2nd column, then A picks 1st row (+5)
  - Minimax doesn't apply here due to hidden information!
  - A's strategy depends on B's strategy — and vice versa.
    No pure strategy exists.
- So we introduce <u>mixed strategy</u>: some distribution over strategies.
  - e.g. Here we'd pick prob. P of holding/folding
    - If A has prob. P of holding ...
      - If B always resigns: $15p - 5$ is A's expected profit
        If B always chooses to see: $-10p + 5$ " " " "
        (Try with $p=0$, $p=1$ to check with Matrix.)
      - Note that these are both lines:



p=0.4 is intersection.

      - Value of the game there is then 1. (Same in both strategies.)
    - But what if B also uses a mixed strategy?
      - This actually doesn't change the game's value. B would be picking a weighted average (i.e. "between") of the lines here. All weighted averages still would intersect at $p=0.4$ & value $=1$.
    - That intersection is optimal because it happens to be an extrema here (for area below all lines). It might be also at boundaries.
    - So this is basically minimax again: Taking Maximum of minimum because A wants to maximize assuming B wants to minimize.
- Next constraint to remove: Zero sum.
- Look at prisoner's dilemma example: Prisoner A defects first, he receives no punishment & prisoner B gets 9 months; vice versa for B.

RL-3 (cont.)

2016-04-20, CS7641

A \ B	cooperate	defect
cooperate	-1, -1	-9, 0
defect	0, -9	-6, -6

-9, 0 means reward to A, reward to B (since it's non-zero-sum)

- If A knows B will cooperate, it's in A's interest to defect
  - And vice versa if B knows A will cooperate
- If either one knows the other will defect, it makes sense to also defect.
- In no case does it make sense for either to cooperate. (Look at rewards to A and to B.)
- So the "best" strategy is for both to defect, but best for the group is to cooperate.
- Nash equilibrium
  - n players with strategies $S_1, S_2, \ldots S_n$ (sets)
    $S_1^* \in S_1, S_2^* \in S_2, \ldots S_n^* \in S_n$ are a <u>Nash equilibrium</u> iff:

$$\forall i \; s_i^* = \underset{s_i}{\text{argmax}} \; \text{utility}(s_1^*, \ldots s_i, \ldots s_n^*)$$

    - That is: For each chosen strategy, it is the strategy that maximizes the utility for that particular player, given all other strategies.
    - Or: If you randomly selected one player and gave them the opportunity to change strategies, they would have no reason to.
  - Works with pure strategies & mixed strategies.
  - Consider:

A \ B	-1, -1	-9, 0
0, -9	(-6, -6) Nash eq.	

A \ B	0, 4	4, 0	5, 3
4, 0	0, 4	5, 3	
3, 5	3, 5	6, 6	

Yes, "A Beautiful Mind" Nash

eliminate iteratively

- In n-player pure strategy game, if elimination of strictly dominated strategies eliminates all but one equilibrium, that combination is the unique Nash equilibrium
- Any Nash equilibrium will survive elimination of strictly dominated strategies
- For finite n, & $\forall i$ S_i is finite, $\exists$ mixed Nash equilibrium.

- What if we have prisoner's dilemma run multiple times?
  - It doesn't actually matter. Start from the final dilemma: none of the prior ones mattered, the incentive at the end is exactly the same.
- n repeated game $\Rightarrow$ n repeated Nash equilibrium
- What to do with multiple Nash equilibria is out of the scope of this...
- Mechanism design?

2016-04-25

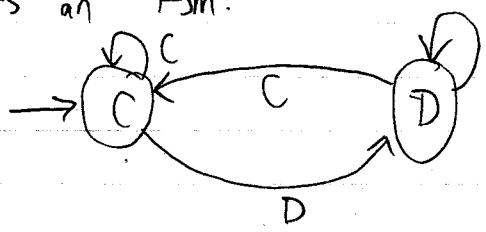
RL-4 (Game Theory Continued)

- Game Theory 2: The Sequencing
  - Consider prisoner's dilemma. We looked at iterated prisoner's dilemma too. But if the number of rounds is unknown, then it is different.
  - The variant we look at: Run a round of prisoner's dilemma. At the end, play again with probability τ, otherwise, game ends with probability $1-\tau$
    - This is the same τ as in discounted rewards because the math is the same.
    - Expected number of rounds is $\frac{1}{1-\tau}$ (geometric series)
- One famous strategy (tit for tat) for IPD:
  - Cooperate on first round.
  - Copy opponent's previous move thereafter.

2016-04-25, CS7641

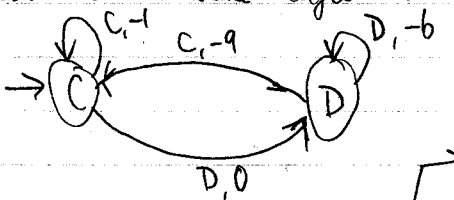
RL-4 (cont.)

- As an FSM?



Arrows represent opponent's move. Nodes are our move.

- Against these strategies, tit-for-tat does...
  - Always defect: C-D-D-D...
  - Always cooperate: Same!
  - Tit-for-tat: Always cooperate
  - D-C-D-C-D... : C-D-C-D-C-D...
- What's best response to TFT? → multiplied by γ because it's one timestep away.
  - If always defect: Reward = $0 + \frac{-b\gamma}{1-\gamma}$
    (We get 0 on first round because opponent cooperates, and thereafter we always get $-b$ because we both defect.)
  - If always cooperate: Reward = $\frac{-1}{1-\gamma}$ (both are always cooperating)
  - Thus, one's good for low γ, one for higher γ. That is: Always defecting is better for shorter games.
    At $\gamma = \frac{1}{b}$ the rewards are equal. (Solve $\frac{-b\gamma}{1-\gamma} = \frac{-1}{1-\gamma}$)
    $\gamma < \frac{1}{b}$, defect. $\gamma > \frac{1}{b}$, cooperate.
- But, we could conceive of all sorts of other strategies. If that's an FSM-based strategy and we're playing against it, how do we handle that? Our choices now impact opponent's future decisions.
  Suppose we annotate edges with rewards:



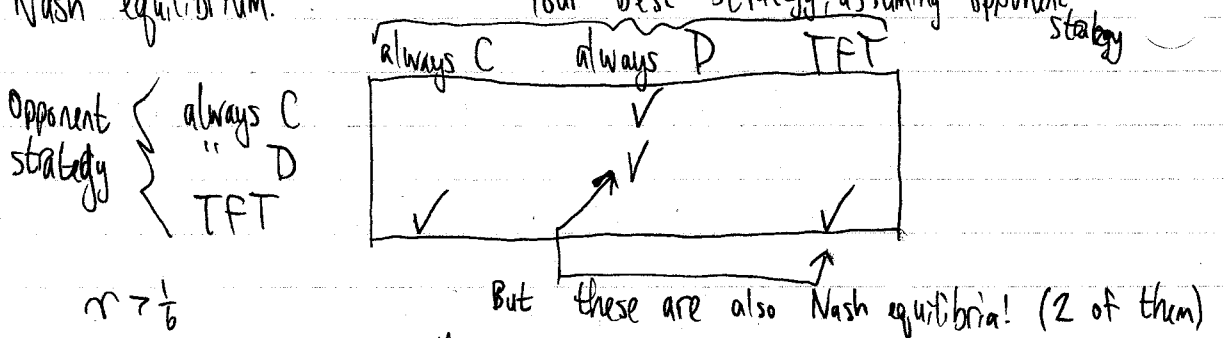
→ Discounted by γ, actually.

- But this is just... an MDP. Matrices only suffice for single rounds

RL-4 (cont.)

- Particularly: γ = discount factor
  Payoff matrix entries = rewards
  Our action = same thing.
  Opponent's internal state structure = our states
- So, how do we find an optimal strategy against an FSM?
  Solve the MDP. It doesn't matter that the states are our opponent's states - that doesn't change things.
- If we solve this MDP we get the same results as analyzing tit-for-tat earlier.
  An MDP always has a Markov optimal deterministic process, which puts a limit on the strategies.
- Mutual best response: Pair of strategies where each is the best response to the other. Also called... a Nash equilibrium.

Checkmark → Means: If opponent picks strategy in that row, your best response is that column



But these are also Nash equilibria! (2 of them)

Neither you nor opponent have any reason to switch for these. For the other two checks in the table, the opponent has reason to switch strategies.

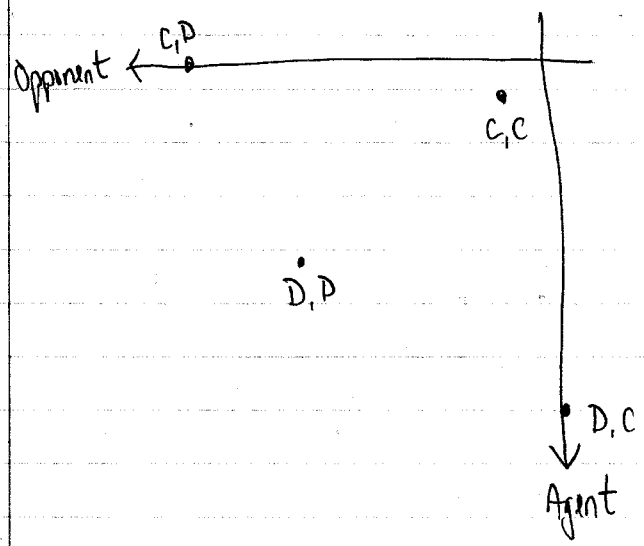
In mathematics, Folk theorem: "Results known, at least to experts in the field, and considered to have established status but not published in Complete form."

- Repeated games & the folk theorem
  - "In repeated games, the possibility of retaliation opens up the door to cooperation."
  - But in game theory, the Folke theorem is a specific Folk theorem that describes the set of payoffs that can result from Nash strategies in repeated games.

2016-04-25b, CS 7641

RL-4 (cont.)

- Two-player plot:



- For each joint strategy, plot the rewards for both players.

- The convex hull of these 4 points matters: It is what rewards (on average) some joint strategy can produce. It's a "feasible region".
- Minmax profile: "Pair of payoffs, one for each player, that represent the payoffs that can be achieved by a player defending itself against a malicious adversary."
  trying to lower score

N.B. minmax sometimes means 'pure strategy' and security level 'mixed strategy'

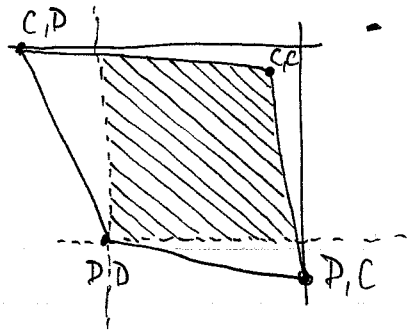
- One famous problem:

Agent 1 \ Agent 2	B	S
B	1,2	0,0
S	0,0	2,1

What is minmax profile of this for both players? Assume mixed strategies.

It's 1,1 for pure strategies.

- Agent 1 chooses B: Agent 2 always chooses S
- Suppose agent 2 always chooses B with probability x, S with $1-x$. Agent 1 gets reward x on average for choosing B, or $2(1-x)$ for choosing S. So agent 2 can minimize this at $x = \frac{2}{3}$ (solve $x = 2(1-x)$...), then agent 1 can get average payoff of $\boxed{\frac{2}{3}}$. It's symmetric, so answer is $2/3, 2/3$.

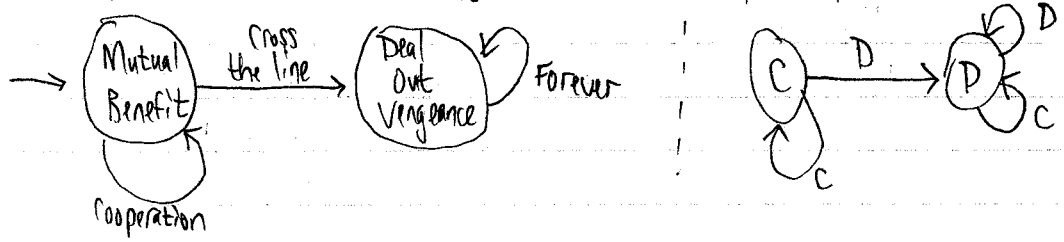


RL-4

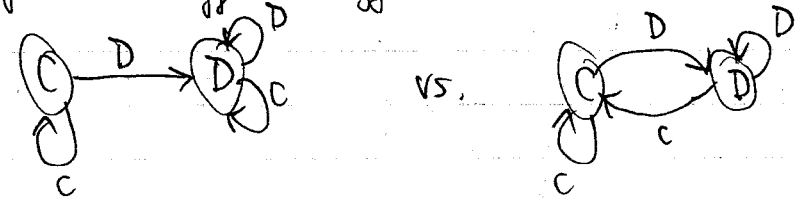
Back to this plot:
- Feasible region is in yellow highlighter.
- Now, look at dashed lines. These are minmax profile for this problem.
- Region above these (both axes) is the acceptable region. Anywhere in this region is either player getting a higher payoff than possible from a malicious adversary (by definition of minmax profile).

Folk. or Folke?

- The intersection of feasible & acceptable (marked with diagonal lines) then matters for Folk theorem...
- "Any feasible payoff profile that strictly dominates the minmax security profile can be realized as a Nash equilibrium payoff profile, with sufficiently large discount factor.
Proof: If it strictly dominates the minmax profile, can use it as a threat. Better off doing what you're told!"
- Another proof uses "Grim Trigger": ; Or, for prisoner's dilemma:



- If we know that we both follow this strategy, neither of us have reason to switch.
- Problem: "Implausible Threats"
  - If the ~~[illegible]~~ opponent would carry out threat, it may be far worse for both. The threats not plausible.
- Plausible threat corresponds to subgame perfect: always best response, independent of history.
- Imagine 'Grim Trigger' strategy vs. TfT:



2016-04-25c, CS7641

RL-4 (cont.)

- These strategies are in Nash equilibrium. Player with Grim trigger has no incentive to switch, against TFT – it cannot improve on it. Player with TFT is in same situation against Grim trigger.
- But, are they subgame perfect?
  How we show this: Would either player ever undertake a series of moves that could be improved on? If this is never the case in the game history, it is subgame perfect.
- For instance:

  Grim ... C D D D...
  TFT ... D C D D...

  - Had Grim chosen only to cooperate here instead of following through on its threat, this would have improved its payoff. That is: Grim's threat, when followed, also hurt Grim. It's not a plausible threat
  - So these are <u>not</u> subgame perfect, though in Nash equilibrium.
- Is TFT vs. TFT subgame perfect?
  - Suppose we see:

    ... D C D C ... (TFT #1)
    ... C D C D ... (TFT #2)

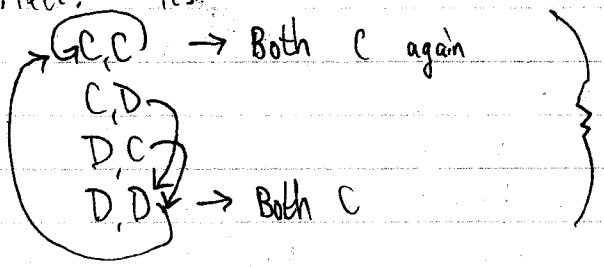
    - Only one D is needed to start this pattern.
    - Average reward is $\frac{0-9}{2} = -4.5$
    - But if TFT #2 cooperated at next step instead of defecting, its average reward would be higher, at -1!
    - Thus: It's not a plausible threat & this isn't subgame perfect. ... D D D...
  - Likewise for: ... D D D...
    - Either one switching to C, C, ... would improve.
- So: Can we make a subgame perfect strategy for prisoner's dilemma?

- Pavlov:

Basically:
cooperate if we agree.
defect if we disagree.
It's a little strange.

RL-4 (cont.)

- Is Pavlov v. Pavlov in Nash equilibrium?
  - Yes: Both start out cooperating & have no reason not to.
- Is it subgame perfect? Yes!
  - Check 4 cases:

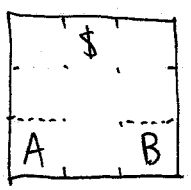


It always makes its way back to C,C.

  - Basically: The threat is plausible, and the punishment isn't bad in the long run for the player doing the punishing.
- Computational folk theorem
  - For any 2-player bimatrix (non-zero-sum) game, with average reward over many repeated games, can build Pavlov-like machines. Construct subgame perfect Nash equilibrium for any game in polynomial time.
  - It's Pavlov if possible, and if it's not, it's zero-sum-like.
  - We get it by solving a linear programming problem
  - If it doesn't work, then at best one player improves (and the result of that is a Nash equilibrium).

Peter Stone & Michael Littman

- Stochastic Games (or Markov games) & Multiagent RL
  - MDP is to RL as stochastic game is to multiagent RL.
  - Example:



← Dashed lines = semiwalls

- 2 agents, 3x3 grid. Actions: N, S, E, W, X (X = stay put)
- Actions are deterministic, except for with semiwalls (50% success)
- Goal: Get to $, & game ends.

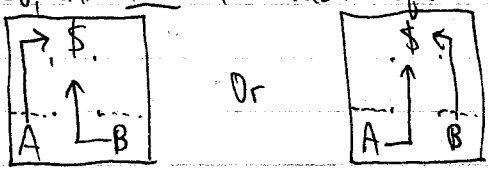
Assume that if they try to move to same space, one of them is randomly chosen to move, and other stays put.

  - If B ignores A: Best course is W, N, N. (For A: E, N, N).
  - But: A & B can't occupy same space, and game ends when either reaches $.

2016-04-25d, CS7641

RL-4 (cont.)

- What might a Nash equilibrium mean here?
  - Pair of policies such that neither would want to switch.
- Consider strategy: A - Go N until top wall, then E
  B - Go N until top wall, then W
  A has 50% chance of being through semiwall as does B.
  A & B have 25% chance of both going through (& 25% of neither - then it's just back at start.)
  A gets to goal 2/3 of the time, as does B.
- But, knowing that A has this strategy, B could change to W, N, N and get to goal 100% of time. (likewise A ↔ B).
- So, this isn't a Nash equilibrium. But this is:



A gets there 1/2 of time and B all of the time. If A switches strategies, it's still 1/2 of time. thus no incentive → If B switches, it goes from all the time to 1/2 of the time. Thus, no incentive.

Actually a generalization of MDPs, but actually predate MDPs

- Stochastic games (Shapley): $\langle S, A_i, T, R_i, \gamma \rangle$
  S: States s
  A_i: Actions for player i - typ. a, b, $a \in A_1$, $b \in A_2$
  T: Transitions - $T(s, (a,b), s')$
  R_i: Rewards for player i - $R_1(s,(a,b))$, $R_2(s,(a,b))$
  γ: Discount
- We can constrain stochastic games to be more specific things we've already encountered. If $R_1 = -R_2$: ~~zero~~-sum stochastic game. If $T(s,(a,b),s') = T(s,(a,b'),s') \forall b'$ & $R_2(s,(a,b)) = 0$, $R_1(s,(a,b')) \forall b'$, MDP. If $|s| = 1$, repeated game.

RL-4 (cont.)

- Zero-sum stochastic games
  - Actions players take influence not just rewards, but future states (as in MDPs)
  - Start with Bellman equation:

$$Q_i^*(s,(a,b)) = R_i(s,(a,b)) + \gamma \sum_{s'} T(s,(a,b),s') \max_{a',b'} Q_i^*(s',(a',b'))$$

(Replace w/ minimax; joint action of a&b; we have basically a matrix here)

  - Problem: this assumes with $\max_{a,b}$ that joint actions always benefit us, which is not reasonable. This needs to be something like minimax - assuming other players are trying to minimize.

minimax-Q

$$\rightarrow \quad \langle s,(a,b),(r_1,r_2),s' \rangle : Q_i(s,(a,b)) \overset{\alpha}{\leftarrow} r_i + \gamma \operatorname*{minimax}_{a',b'} Q_i(s',(a',b'))$$

  - Update Q with current reward + discounted V in new state
- Value iteration works. minimax-Q converges. Q^* is unique. Policies for 2 players can be computed independently. minimax-Q can be updated efficiently (uses LP). Q functions are sufficient to specify policy.
- General-sum, instead of zero-sum:
  - Can't do minimax here (it's meaningless without zero sum). What we want is actually Nash equilibrium. Replace 'minimax' with Nash, Nash-Q - it's a defined operator.
  - Value iteration no longer works. Nash-Q doesn't converge (thus no unique Q^*). Policies aren't independent anymore - two halves of a Nash equilibrium don't make another Nash equilibrium. Update isn't efficient (it's in problem class PPAD, believed to be as hard as NP), and Q functions - if we could even find them - no longer are sufficient to specify policy.
- Repeated stochastic games are more surmountable w/ some folk theorems
  - cheaptalk → correlated equilibria - If players can communicate, then even if talk is not binding, it can allow equilibria

Dr. Isbell worked here

2016-04-25e, CS 7641

RL-4

- cognitive hierarchy: instead of solving equilibrium, treat each player as assuming that all <u>other</u> players have more limited computational resources, and compute best response under this. This turns out to be closer to how humans do it.
- side payments (coco values) - players can't just communicate, but can make payments to pay another player back for helping.

coco = cooperative competitive values

- But, these are all still open problems.

Outro

- Topics not covered
  - Big data: What do you do when datasets are so large that even linear-time is too slow?
    - This has fundamentally changed how things are done in science
  - Deep learning / deep neural nets
    - We knew a decade or two ago that neural nets could have many hidden layers but didn't see until recently. Some of the tricks to make this useful and feasible
  - Semi-supervised learning
    - What if we have lots of data but can only get labels for 10% of it? Perhaps it must be done by hand & is too time-consuming.
    - Unsupervised learning can get structure out of unlabeled data that may make all of it more amenable to supervised learning.
    - Big deal in late '80s, early '90s.
  - "The assumption of balanced labels" - This was implicit in what we do; we assumed that we have roughly even numbers of labels.

- But suppose we have an extremely rare label (say, 0.01% of the time). A learner that just always reports a negative is "correct" 99.99% of the time then!
  - But, this may totally fail to take the cost of a false negative into account (e.g. what if that 0.01% is terrorists?)
  - A seminal paper on this: "Cascade Learning"
    - I think CS6476 spoke of this?
    - & similar to boosting.
    - If we repeatedly remove true negatives (assuming it is lots of negatives, and almost no positives) we can eventually balance those labels.
      - We may apply more and more computational power at each level, and end up with a cascade of learners, each one handling harder and harder cases that earlier learners couldn't handle
- TFIDF: term frequency inverse document frequency
  - How close is a query to documents in your collection?
  - This makes use of the frequency in the document of a term in the query - but if a lot of documents have the term, then it considers it less important. Hence, inverse document frequency.

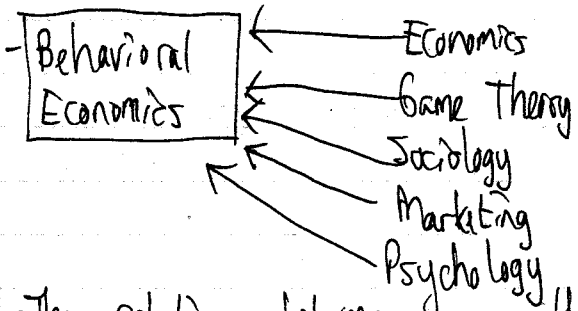
e.g. the word "the" is less important, despite high frequency

- Spectral clustering (less stuck in local minima, based more on linear algebra than search)
- Cross-entropy works well in lots of settings.
  - Littman considers it quite a lot like MIMIC.

- Can we apply function approximation with reinforcement learning?
- POMDPs - partially observable MDPs
  - In 'normal' MDPs agent always has full information on state, but this is unrealistic!

2016-04-25F, CS7641

Outro



Also "neuroeconomics" ?

- The relation between game theory/reinforcement learning and humans is very important (and all that interaction)
- Inverse reinforcement learning: Observe behavior and try to estimate the reward function

---

Andrew Moore slides

- "Games With Hidden Information"
  - Pure strategy - Mapping from all the possible states that player could see, to the move that player would make
  - Optimal strategy for n x m game (mixed strategy):
    - n x m game has matrix form, A has n pure strategies, B has m.
    - We want a mixed strategy like... Player A uses strategy 1 with prob p_1
      " " " 2 " " p_2
      . . .
    - As expected gains if B uses pure strategy i:
      $eg_i = m_{1i}p_1 + m_{2i}p_2 + \dots + m_{ni}p_n$
    - So choose $p_1, p_2, \dots p_n$ to maximize $\min(eg_1, eg_2, \dots eg_n)$ subject to $\sum_i p_i = 1$ and $p_i \in [0, 1]\ \forall i$.

  → Linear programming

- "Non-zero sum Game Theory, Auctions, & Negotiation"
  - Strictly Dominated: If one player's strategy is never the right thing to do, regardless of what opponents do

2016-05-02, CS7641 (sort of. Finishing up textbook.)

Ch. 2 (Mitchell)

- Concept Learning & the General-to-Specific Ordering
  - ↳ "problem of searching through a predefined space of potential hypotheses that best fits the training examples"
  - A "general to specific" ordering tends to occur naturally in hypothesis space
  - Concept learning is "inferring a boolean-valued function from training examples of its input & output"

Ch. 5

- Evaluating Hypotheses
  - Two difficulties in estimating accuracy: Bias in the estimate, & variance in the estimate.
  - Two questions:
    1. Given hypothesis h & data sample with n examples drawn at random according to distr D, what is best estimate of the accuracy of h over future instances drawn from same D?
    2. What is the probable error in this accuracy estimate?
  - $error_S(h)$ = sample error of hypothesis h w.r.t. sample S of instances drawn from X = fraction of S that it misclassifies, $\equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$
    $n = |S|$, $\delta(a,b) = 1$ if $a \neq b$, else 0
  - $error_D(h)$ = true error of hypothesis h w.r.t. target function f & distr. D = probability that h will misclassify a single randomly-drawn instance (drawn according to D).
    $\equiv Pr_{x \in D}[f(x) \neq h(x)]$

  - But we can usually only measure $error_S(h)$. So, how good an estimate of $error_D(h)$ does it provide?

Does this assume Gaussian?

  - See table 5.1 for $z_N \leftrightarrow$ confidence level. 68% → $z_N = 1.00$, 95% → $z_N = 1.96$. Then with ~N% probability, $error_D(h)$ lies inside: $error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$

    (good for $n \geq 30$)

2016-05-02(b), CS7641 ish

Ch. 5 (cont.)

- this relates to the well-studied statistics problem - estimating what part of population exhibits some property, given that we've observed it over some random sample
- Note that $error_S(h)$ is a random variable. Specifically, $error_S(h)$ is an <u>estimator</u> of $error_D(h)$, and <u>estimation bias</u> is $E[Y] - p$ for estimator Y & param p.
  For unbiased estimator, $E[Y] - p = 0$. This is the case for $error_S(h)$, provided h & S are chosen independently.
  - Don't confuse this number with inductive bias, which is not a number (rather, a set of assertions).
  - In addition, <u>variance</u> of an estimator is important. Lower variance → lower expected squared error.
- N% <u>confidence interval</u> for some parameter p is an interval expected with probability N% to contain p.
  - e.g. for $r = 12$ errors in sample $n = 40$ independent examples, interval 0.30 ± 0.14 contains $error_D(h)$ with prob. 95%.
- Examples here all use Binomial distribution & central limit theorem to assume normal distribution.
- We've done two-sided bounds, but sometimes we want one-sided, e.g. what's probability that $error_D(h) \neq \leq U$?
  Since normal distribution is symmetric, we can just take twice the confidence - e.g. $100(1-\alpha)\% \rightarrow 100(1-\frac{\alpha}{2})\%$
- Section 5.4 generalizes this to other distributions.
- This chapter is the tip of the iceberg. See sampling theory and more generally everything about statistics.

Ch. 10

- Learning Sets of Rules
  - While in Ch. 3 we looked at learning decision trees that can be turned to sets of rules, here we look at directly learning sets of <u>first-order</u> rules - not just propositional (much more expressive - can use variables)

- For instance: If Parent(x,z) then Ancestor(x,y)
  If Parent(x,z) ∧ Ancestor(z,y) then Ancestor(x,y)
  - This is recursive, & very hard to express in a decision tree or other propositions.
  - Note similarities to PROLOG!

Table 10.1

- "sequential covering algorithms": Learn one rule, remove data it covers, then iterate.
- beam search comes up here: search maintaining k best candidates at each step; then, generate k descendants (one for each). See CN2 (Clark & Niblett, 1989), and table 10.2
- Entropy/information gain is common to use here.
- For learning first-order rules:
  - Sometimes called ILP, inductive logic programming.
  - Note how much more general 1st-order rules can be (& how this avoids overfitting with too-specific rules that fail to generalize!)

Chapter 11

- Analytical Learning
  - Input to learner includes same hypothesis space H & training examples D as inductive learning. But, learner has additional input: Domain theory B consisting of background knowledge that can be used to explain observed training examples. Output needs to be consistent with both B & D!

Prolog-EBG actually is deductive, not inductive

  - Look up: Prolog-EBG (Kedar-Cabelli & McCarty 1987).
  - A training example may have many features, but only some relevant. The learner's explanation answers directly which are relevant by which features the explanation even mentions.
  - It can also provide a justification or 'proof' of an example.
  - Note that in explanation-based learning, the learner is just restating what it already "knows" - but this reformulation can still be meaningful.
  - Inductive bias of Prolog-EBG is simply domain theory B.
  - "search control knowledge" benefits here - simplifying searches through huge spaces by restricting by domain theory (e.g. chess rules)

2016-05-04, CS 7641 ish

Ch. 11 (cont.)

- See: PRODIGY (Carbonell et al. 1990)
  - "domain-independent planning system that accepts the definition of a problem domain in terms of the state space & operators"; finds sequence of operators leading from init. state s_i to state satisfying goal predicate.
- See: SOAR (Laird et al. 1986)

Ch. 12

- Combining Inductive & Analytical Learning
  - Inductive learning methods give a <u>statistical</u> justification of their learned hypothesis.
  - Analytical learning methods (e.g. PROLOG-EBG) give a <u>logical</u> justification (it follows deductively from domain theory).
  - But if we combine, how do we balance error with respect to the data (inductive learning) with error with respect to the domain theory?
  - Note that we can think of domain theory in the form of prior knowledge that we use with Bayesian techniques (i.e. ch. 6) – $P(h)$, $P(D)$, $P(D|h)$.
    However, it doesn't handle when these are imperfectly known.
  - Systems to look up: KBANN, EBNN, FOCL, ML-SMART
  - KBANN initializes network so that network always classifies the same way domain theory says it should – and then runs backprop over training examples.
    - This then handles imperfect domain theory to an extent.
    - The initial weights (when encoding domain theory) have some influence
    - This typically generalizes better than purely inductive methods (e.g. random weights & backprop)
    - However, KBANN handles only propositional domain theories (variable-free Horn clauses, specifically).

neural network, that is

- KBANN starts with hypothesis that fits domain theory, and then fits it to training data. Another approach is to use that domain theory in the error function minimized, so that it takes that and training data into account.

Simard et al. 1992

- One form of this: TangentProp & EBNN
  - "accomodates domain knowledge expressed as derivatives of the target function with respect to transformations of its inputs" that is, each instance is $\langle x_i, f(x_i), \frac{\partial}{\partial x} f(x) \mid_{x_i} \rangle$
  - Back-propagation normally favors smooth functions. The derivative helps remove this bias in some cases.
  - This can encode domain knowledge though, e.g. if we have additional features and want to convey that those features should produce smaller or larger changes in f. (They use example of handwriting & small rotational changes being irrelevant.)
  - Instead of $E = \sum_i (f(x_i) - \hat{f}(x_i))^2$ as in backprop, this uses: $E = \sum_i \left[(f(x_i) - \hat{f}(x_i))^2 + \mu \sum_j \left(\frac{\partial f(s_j(\alpha, x_i))}{\partial \alpha} - \frac{\partial \hat{f}(s_j(\alpha, x_i))}{\partial \alpha} \right)^2_{\alpha=0} \right]$

    where s_j is some transformation - $s_j(\alpha, x)$ where α is continuous & $s_j(0, x) = x$

    μ just biases derivative error & squared error
  - TangentProp isn't very robust to error in priors.
- EBNN (Mitchell & Thrun) builds on TangentProp.
  - Computes derivatives on its own & sets μ per-instance
  - Handles approximate domain theory better.
- FOCL extends purely-inductive FOIL. (Ch. 10)
- 12.6 says, "The topic of combining inductive & analytical learning remains a very active research area." That was from the late 1990's so I suspect, if it was correct, much else has occurred.

BIC = Bayesian Information Criterion

$$BIC = -2 \cdot \ln \hat{L} + k \ln(n)$$

- n → sample size
- k → free parameters
- $\hat{L}$ → Likelihood, $\hat{L} = p(x \mid \hat{\theta}, M)$ where $\hat{\theta}$ = parameter set that maximizes likelihood function

- Lower BIC is preferred. It is written to penalize more complex models.
- Relates to MDL but with negative sign?

ICA

- Performs "pre-whitening" first with PCA
  - A & S are estimated 'source' & 'mixing' matrix (un... respectively. Reverse those two.)
  - $XKW = S$, X = preprocessed data matrix, W = un-mixing matrix
  - Trying to estimate under assumption $X = SA$

$X_{n \times p}$, $K_{p \times L}$, $W_{L \times L}$, $A_{L \times p}$, $S_{n \times L}$

SA seems to give a passable reconstruction. But what can I do with S that's interesting? What are the 'sources'?

Let SVD of $A = U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T$, S has positive singular values of A on its diagonal.

Then pseudoinverse $A^+ = V \begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T$ (Note A^T and A^+ are same-size.)

I should put this in my notes...

t-SNE:

- Vaguely reminds me of SVM & RBF Kernel.
- Seems to be more suited to visualization in 2D\3D than to dimensionality reduction?
- Perhaps I should still use it to visualize.

LDA

- Relies on labels. Is that okay? Do I split training\test here?
- R has it in MASS.
- http://www.r-bloggers.com/computing-and-visualizing-lda-in-r/

For $T = XW$, X = data matrix, $n \times p$ $\quad 1 \leq L \leq p$
W = weights/loading, $p \times L$
T = "scores", $n \times L$
"Full" PCA has $L = p$, so W is $p \times p$.
W's columns are eigenvectors of $X^T X$.

? Sum of squares within component = eigenvalues = components' variances
$X^T X$ is proportional to covariance matrix of D.
PCA transform can be seen as one that diagonalizes the covariance matrix.
'Covariances' section on Wikipedia sort of confirms this

2016-03-24
- Notes on: "Silhouettes etc." (Rousseuw, 1986)
Wikipedia formula: $s(i) = \dfrac{b(i) - a(i)}{\max(a(i), b(i))}$ (This is actually identical to the paper's formula.)
where $a(i)$ = average dissimilarity of i with all other data in same cluster, $b(i)$ = lowest average dissimilarity of i to any <u>other</u> cluster.
- The distribution of $s(i)$ within each cluster gives some useful information
$s(i)$ closer to 1 is preferred and indicates better-defined regions.
$s(i) < 0$ can indicate that i maybe should have been clustered elsewhere, like its nearest neighboring cluster

- Average silhouette width should be as large as possible. Within a cluster it can also identify a "strong" or "weak" cluster.

- MMDS, section 9.4:
  - UV-decomposition for dimensionality reduction.

$Me = \lambda e$, e = eigenvector, λ = eigenvalue } eigenpair

M is $n \times n$, e is $n \times 1$, λ is scalar

$\rightarrow (M - \lambda I)e = 0 \rightarrow \det(M - \lambda I) = 0$

- Eigenvectors have length ambiguity (only direction matters), so by convention we use unit vectors with 1st element positive. (Or, 1st nonzero element)
- That determinant = nth degree polynomial if $= 0$, thus n solutions, each with an eigenvector
- Let $E = [e_1\ e_2\ e_3\ \dots\ e_n]$ where e_i = ith eigenvector.

Symmetric $M \rightarrow E$ is orthonormal - all eigenvectors are orthogonal & unit vectors.

Note that nTn and nnT are symmetric!

- PCA can work by finding eigenvectors of MM^T or M^TM, if M contains tuples of data.
- That eigenvector matrix is then a rigid rotation (→ orthonormal matrix) in some high-dimensional space.

  Principal eigenvector: Axis corresponding is one with points the most 'spread out'. This continues through 2nd, 3rd, etc. eigenvectors.

2016-03-24

- Suppose E is eigenvectors of M^TM, largest eigenvalue first.
  Let L have the corresponding eigenvalues on its diagonals, 0s elsewhere.
- Note that $(M^TM)e = \lambda e = e\lambda$ for each eigenpair (e, λ).
  Thus, $(M^TM)E = EL$
- ME corresponds to rotation to some n-dimensional space, with dimensions ordered by variance.
  So if E_k has only the first k columns of E (i.e. first k eigenvectors), E_k is $p \times k$ so ME_k is $n \times k$ rather than $n \times p$ - giving a k-dimensional approximation.
- SVD
  - Let M be $m \times n$ matrix with rank r.
  - Then U, Σ, V exist such that:
    $U_{m \times r}$ is column-orthonormal - each column is unit vector, and dot product = 0 for any two columns.
    $V_{n \times r}$ is column-orthonormal. We always use V in transposed form - so V^T's <u>rows</u> are orthonormal.
    Σ is diagonal matrix, on whose diagonals are M's <u>singular values.</u>
  - And $M = U\Sigma V^T$ (to what numerical precision allows)
  - Σ in a way gives the strengths of actual 'concepts' in M, while U relates rows to concepts and V relates concepts to columns.

See: 'Matrix Computations' (G.H. Golub & C.F. van Loan)

- We can reduce dimensionality by zeroing smaller values in Σ, and if we do so, we can simply shrink Σ and remove the corresponding rows of U & V.
  - That provides a more compact representation that loses some information.
  - Typical guideline: Retain enough singular values to make up 90% of energy of Σ (sum of squares).
- p424 explains why (in terms of error) this works.
- Start with $M = U\Sigma V^T$. Then $M^T = (U\Sigma V^T)^T = V\Sigma^T U^T = V\Sigma U^T$.
  $M^T M = (V\Sigma U^T) M = V\Sigma U^T U\Sigma V^T$ but since U is orthonormal
  $M^T M = V\Sigma^2 V^T$ and multiplying by V:
  $M^T M V = V\Sigma^2 V^T V$ and again V is orthonormal so
  $M^T M V = V\Sigma^2$ and note that Σ^2 is diagonal also,
  so, V then by definition is the eigenvectors of $M^T M$ & Σ^2 is eigenvalues!
  so Σ is the square roots of $M^T M$'s eigenvalues.
- $MM^T = U\Sigma V^T (U\Sigma V^T)^T = U\Sigma V^T V\Sigma V^T = U\Sigma^2 U^T$
  $MM^T U = (U\Sigma^2 U^T) U = U\Sigma^2$
  → U is matrix of eigenvectors of MM^T.
- But $M^T M$ is $n \times n$, MM^T is $m \times m$ - but rank $r \leq m$ & $r \leq n$. So, $M^T M$ and MM^T have $(n-r)$ and $(m-r)$ extra eigenpairs which don't show up in U, V, Σ; their eigenvalues will be 0.
- See CUR decomposition for a faster approximation for large sparse matrices