CS 440: Introduction to AI

Lecture 2 : Uninformed Search

PEAS (how to describe AI problem)

- Performance measure:
- Environment: where it exists
- Actuators: actions available to agent (ex: ways chess piece can move)
- Sensors: what in environment you have access to

Reflex/Model-based vs. Goal-based Agents

- Reflex-based agent is not inherently aware of its goal
- Goal-based, aka problem solving agents is aware

Path-Finding Situation

- State: where you are now

Touring Situation

- State: current location and history (where you've been)

Search Tree

- Have a starting point and expand and search where the next state(s) can go
- **Fringe**: trailing edge of search tree, can always be pushed down further ,current limits of computation
    - Evolution of fringe will depend on which node you expand

    Algorithm 1: Tree Search
    - Doesn't check for possibility of having already been checked

    Algorithm 2: Graph Search

Comparing Search Strategies

- Completeness
- Optimal
- Time cost: # of nodes generated during search
- Space cost: max # of nodes stored in memory during search

Breadth First Search (BFS)

- Take entirety of fringe and expand it one by one
- FIFO / LIFO queue
- Bad b/c you have to store entire fringe

Uniform-Cost Search

- Expand node with current minimal cost
- Cannot guarantee it's optimal goal node so you expand next node
- But you always check to see if current path is less than next first

Depth-First Search

- Space advantage to DFS
- But DFS you might get stuck somewhere down low and you can't get back up

Iterative-Deepening Search

- It's like DFS but you put a limit to how far you wanna go down
- If you don't find anything, increase the limit of your search
- Guaranteed to find an optimal solution
- Every time you increase value of limit, you have to re-explore part of tree you've already seen
- However, not always terrible

Bidirectional Search

- Can start two searches, one from start one from goal
- When to two fringes meet, you've found route from start to end
- BFS: visit nodes O(b^d)
- BDBFS: visit nodes O(b^(d/z)) + O(b^(d/z))

Lecture #3 | 1.24.2017 - From Uninformed to Informed Search

E : discrete set of states, they exist in a state space X

A : successor function; $S: X \cdot U \rightarrow X$

S : full observability (we know the graph G)

P : one case is identification of a goal state (8 queens); alternative is an optimal path to a goal state (path planning, 15 puzzle)

- Where U = action space
- Together is a representation of the problem is a graph data structure
- Nodes: states $x \in X$
- Edges: valid actions form corresponding states

- Search algorithms build a "search tree" which maintains "visited states" or states for which we have discovered paths to them

Fringe

- Data structure
- Nodes generated during search process but not expanded yet

Ex: Search algorithm pseudo-code

```
Fringe ← x_o
Do
        If  (fringe is empty)  report failure;
        n ← get_node (fringe)
        If  (n is a goal)  report success, return n;
        neighbors ← expand(n)
        fringe ← fringe + neighbors                    // generation
```

BFS: fringe is queue (FIFO)
DFS: fringe is stack (LIFO)

	BFS	DFS	ID-DFS	Bi-BFS	Uniform-cost Search
Complete?	Yes, the search tree is finite or a goal exists in finite level	No	Yes, same as BFS	Yes, same as BFS	Yes, same as BFS
Optimal?	No, but it is optimal if edge costs correspond to a non-decreasing function of search tree layer	No	No, same as BFS	NO, same as BFS	Yes, require positive edge weights
Time Cost (# nodes generated)	$O(b^{d+1})$, d+1 because it is the fringe that has been generated when you're at level d	$O(b^m)$	$O(b^d)$	$O(b^{d/2+1})$	$O(b^{\frac{C*}{\varepsilon}+1})$
Space Cost	$O(b^{d+1})$	O(bm)	O(bd)	$O(b^{d/2+1})$	$O(b^{\frac{C*}{\varepsilon}+1})$

DFS - only need to keep track of ancestors and siblings
Bi-BFS - can identify a good node and can move backwards in X

Iterative-Deepening DFS (ID-DFS)

- Tries to combine advantages of BFS (completeness) and DFS (space complexity)
- Performs DFS up to certain limit
- If no goals discovered, increases limit and restarts

Principle of Best First Search

- For each node n, we can get a score f(n) according to an evaluation function f
- Data structure used: priority queue
- Uniform-cost search: f(n) = g(n)
- g(n): path cost from initial state x_0 to node n along the path corresponding to this node

Uniform-Cost Search

- Proof by Contradiction: g(G*) < g(G')
    - Assume uniform-cost search returns suboptimal goal G' before optimal G*
    - The moment UCS returns G', the fringe must contain a node n along the branch from xo to G*
    - And if all edges have positive weights, g(n) < g(G*)
    - Contradiction: UCS returned G' before n although g(n) < g(G') and UCS uses a priority queue according to g

*** All algorithms so far have ***uninformed***

- Uninformed means that have access only to input graph G
- BUT, many problems have access to external information in form of heuristic
    - h(n) : X ⟶ R
    - ^ estimate of how far away goal is from n

Greedy Best-First Search

- Uses priority queue
- f(n) = h(n)
- Heuristic will tell you estimated cost of the optimum path from n

A*

- Priority queue
- f(n) = g(n) + h(n)

Lecture #4 | Review of Informed Search - Intro to Adversarial Search

!! Extended Deadline for Phase 1: Sunday, February 5th / Demos: February 6-8th !!

Review

- Uninformed Algorithms: have access only to input graph G
  - BFS, DFS, ID-DFS, Bi-BFS
- Informed Algorithms: can also access h(n)
  - h(n) provides estimate of how far away a goal node is from n
  - $h(n_g) = 0$ where g is goal node
  - h(n) >= 0

g(n) = cost from initial node to node n
h(n) = heuristic estimate from n to goal

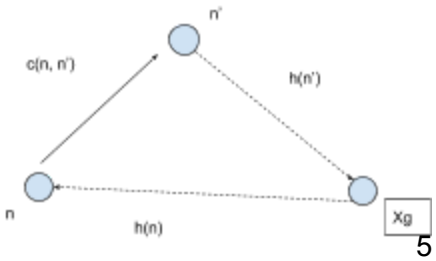
Greedy Best-First Search

- f(n) = h(n)

A* Search

- f(n) = g(n) + h(n)
- Show that under certain conditions for h(n) A* is complete, optimal, and optimally efficient
- **Case 1:** Allow repeated states to arise (referred to in book as tree search)
  - Requirement: h(n) is admissible (admissible = optimistic estimate)
  - $h(n) \leq C^*(n)$
- **Lemma**: If G2 is a node that corresponds to a goal state but a suboptimal path, show that A* will select a node n along the optimal path before selecting G2 for expansions
- Define C*: optimal solution from x0 to optimal goal
  - h(G2) = 0 because G2 corresponds to a goal state
  - f(G2) = g(G20 + h(G2) = g(G2) > C* : because G2 is along suboptimal path
  - Think of a node n along the optimal path (there must be one in the fringe)
    - f(n) = g(n) + h(n)
    - C* = g(n) + C*(n)
  - Since h(n) is admissible, i.e. h(n) <= C*(n) this means that f(n) <= C*

- Overall: f(n) <= C* < f(Gn)
- **Case 2:** Avoid Repeated States (GRAPH-SEARCH)
  - Approach 2a: Remember best path to each state and compare against it every time we revisit it. Lots of book-keeping
  - Approach 2b: Identify the conditions for the heuristic function so that the first time we visit every node n, we have discovered the optimum path to it
  - Requirement to be optimal: Heuristic is consistent



h(n) <= h(n') + c(n, n')

- **Lemma:** Consistent Heuristics are also admissible
    - Xg: goal state, h(Xg) = 0, so obviously admissibility holds h(Xg) <= C*(Xg)
    - N1: predecessor of goal node
    - Eq 1: C*(n1) = C(n1, Xg)
    - Eq 2: h(n1) <= C(n1, Xg) + h(Xg) = C(n, Xg)
    - Eq1 + Eq2 → h(n1) <= C*(n1)
- Let's consider general case node n' for which admissibility holds: h(n') <= C*(n') and a parent node n, given that heuristic is consistent:
    - h(n) <= h(n') + c(n, n') <= c(n, n') + c*(n') = C*(n) => h(n) <= C*(n)
- **Lemma:** If h(n0 is consistent, then f(n) is nondecreasing along any path
    - g(n') = g(n) + c(n, n')
    - f(n') = g(n') + h(n')
        = g(n) + c(n, n') + h(n') >= g(n) + h(n)
        = f(n) = f(n') >= f(n)
    - **Corollary:** sequence of nodes expanded by A** using GRAPH-SEARCH with consistent heuristic is nondecreasing order of f(n)
        - Based on this, you can say A* is complete, optimal, and optimally efficient
        - Time complexity is exponential still

Recitation #2 | 1.31.2017

Consistent Heuristics

- If for every node n, every successor n' of n generated by any action a
- If you estimate path cost from n node to goal, it should be less than cost of n → n' → goal node
- Basically, it follow the triangle equality

Lecture #5 | 1.31.2017 - Review of Informed Search, Adversarial Search, Into to Local Search

Conditions for h(n)

Tree Search	Allows revisiting states	h(n) has to be admissible h(n) <= C*(n)
Graph Search	Does not allow revisiting states	h(n) <= c(n, n') + h(n')

> Graph Search

- The first time you are expanding a node, you have discovered the shortest path to the corresponding state
- All nodes with f(n) < C* will be expanded
    - For those f(n) = C*, the breaking may be involved

Selecting Heuristics

- Note that heuristics h1 and h2 for the g-puzzle are optimal solutions to a "relaxed" version of the original problem
- h1: we are allowed to swap tiles
- h2: we can move over other tiles
- By default, satisfy consistency problem

Adversarial Games

- We can still approach them by building a search tree
    - Out the leaf nodes of the tree we have utility values
    - We have extra layers where the opponent is making a choice
        - Layers corresponding to the player that wants to maximize utility will be defined as MAX and the others as MIN

MIN_MAX_VALUE(n) =

Utility(n)	If n is a terminal node
Max, M_V(s) $s \in succ(n)$	If n is a max node
Min, M_V(s) $s \in succ(n)$	If n is a min node

Search Process

- Has to be DFS because we have to reach the terminal nodes first to compute the MIN_MAX_VALUE(n) at every intermediate node n
- Time: $O(b^m)$
- Space: $O(b^m)$
- We can perform some pruning of the search space (called $\alpha - \beta \; pruning$) and do not visit part of the tree
- Effective of alpha-beta pruning depends on the order you visit nodes
- If we examine first that successor that are likely to be best: complexity is $O(b^{m/2})$
- If successors are random: $O(b^{3m/4})$

Lecture #6 | 2.2.2017 - Local Search, Genetic Algorithms

EXPECTIMINIMAX(n) =

Utility(n) Or Heuristic(n) for a cut off	If n is a terminal node
Max, EMM(s) $s \in succ(n)$	If n is a max node
Min, EMM(s) $s \in succ(n)$	If n is a min node
$\sum P(s) \cdot EMM(s)$ $s \in succ(n)$	If n is a chance node

In many problems, the path to the goal is not important so local search may be preferred

Principles of Local Search

- Try to reduce memory requirements by remembering 1 state for now
- Make only local "moves" or state adaptations within a neighborhood of the current state
- Define a utility for each state and try to optimize this utility
    - Goal should be an optimum for this utility function

Benefits

- Little memory usage
- Has the potential of returning solutions even in large state spaces
- Can be applied to optimization problems

Issues

- Local minima
- Plateaux

Statistics

- For naive hill climbing (using 8 queens example) for 8^8
-

Random restarts up to 100	6% fails	86%	Local minimum	@ 3 steps

Random restarts up to 100	94% success	14%	works	@ 4 steps

- One approach for improving H.C. (random restarts / walks)

Hill Climbing w/ Random Restarts

- Probabilistically complete
- Probability of finding a solution goes to 1 as computation time increase
- While a solution has not been found, we do not know whether one exists
- If hill climbing has probability p of succeeding, then the expected # of restarts for finding a solution is 1/p

Simulated Annealing

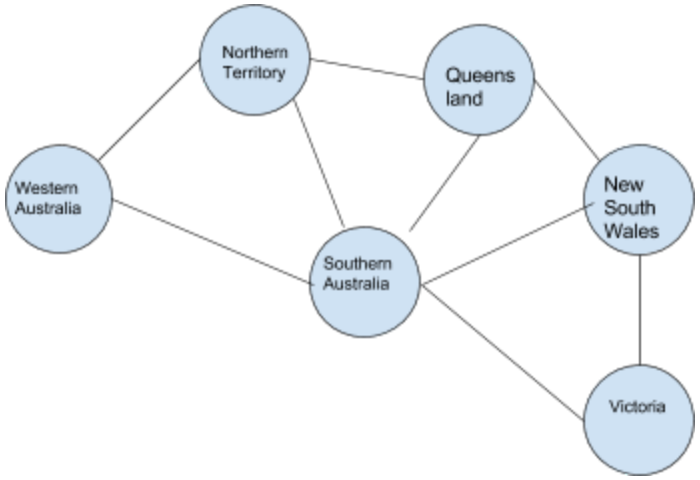
- Probabilistically complete hill climbing approach
- Idea: allows going against the gradient
  - Does so according to a probability:
    - a) higher in beginning of process
    - b) lower as time passes

Lecture #8 | 2.14.2017

- Search problems: difficult to generalize effectiveness of heuristics

Constraint Satisfaction Problems (CSPs)

- 2 Approaches:
  1. Classical Search: DFS, BFS, etc.
     a. Start with empty assignment and incrementally make assignments while making sure they are consistent
  2. Local Search:
     a. Start with random, complete assignment that may violate constraints and incrementally adapt it until it becomes consistent
- Solution we want: A complete and consistent assignment of values to variables



- Western Australia Example: input representation for map crossing
    - DFS: appropriate since the depth of the search tree is limited (equal to # of variables)
    - This is also called backtracking search - every time a consideration of a node violates a constraint, we backtrack to the parents
- <u>Heuristic to Guide Search</u>
    1. Variable and value ordering
        a. Minimum Remaining Values heuristic (MRV)
            i. For instance, WA = red and NT = green, prefer the SA to color next as it has the MRV
        b. For the first variable: maximum degree heuristics
            i. Pick the variable that is involved in the maximum # of constraints
        c. Value selection: apply the “least constraining heuristic”
            i. Prefer the value that rules out the fewest choices for the neighboring variable
        d. We can use bookkeeping to keep track of the valid domain of each variable so as to detect values that allow choices for neighbors as well as to allow arc consistency checks (make sure all edges in latest assignment connect variables that are compatible to each other)

	WA	NT	Q	SA	NSW	V	T
Domain	RGB	RGB	RGB	RGB	RGB	RGB	RGB
WA = R	R	GB	RGB	GB	RGB	RGB	RGB
Q = G	R	B	G	B	RB	RGB	RGB
V = B	R	B	G	empty	B	B	RGB

- There is an issue because SA is left as empty because neighbors have already taken all colors
- Instead, you have to have WA and Q be different colors

Binary CSPs

- All CSPs can be turned into a binary CSP
- Wumpus example, neighboring cells can detect the monster's stench

True Map

stench		breeze	PIT
WUMPUS	Stench, breeze, gold	PIT	breeze
stench		breeze	
PLAYER	breeze	PIT	breeze

Sensed Map

PLAYER			

Building the Knowledge Base (focus on PITS)

P(i, j): true if there is a pit at cell i, j

B(i, j): True if we sense breeze at i, j

Rules

R1: $\neg$P(1,1)

R2: B(1,1) $\Leftrightarrow$ [P(1,2) $\vee$ P(2,1)] or

$(\neg B(1,1) \vee P(1,1) \vee P(2,1)) \wedge (\neg P(1,2) \vee B(1,1)) \wedge (\neg P(2,1) \vee B(1,1))$

R3: $\neg$B(1,1)

R4: B(1,2)

R5: B(1,2) $\Rightarrow P(1,1) \vee P(3,1) \vee P(2,2)$

R6: $\neg B(2,1)$

R7: $B(2,1) \Leftrightarrow P(1,1) \vee P(2,2) \vee P(1,3)$

R6, R7: P(3,1)

- In Rule 2, contains disjunctive clauses and all those together create conjunction of clauses; this is called **3-CNF**
- All propositional logic statements can be written in 3-CNF form

Representational Tool: Boolean Logic

- Binary problems: variables can only be TRUE / FALSE
- Operators: and, or, not, implication, biconditional
- Biconditional Elimination: $(a \Leftrightarrow \beta) \equiv (a \Rightarrow \beta) \wedge (\beta \Rightarrow a)$
- Implication Elimination: $(a \Leftrightarrow \beta) \equiv (\neg a \vee \beta)$
- De Morgan's

Lecture #9 | 2.16.2017

- Binary CSP continuation: Wumpus World

P(i,j): indicate if a PIT exists at cell (i, j)
B(i, j): indicate if a BREEZE can be sensed at (i, j)

Questions to Ask

- Is $\neg P(1,2)$ true?
- Is $\neg P(2,2)$ true?
- Is $\neg P(3,1)$ true?

Helpful Rules

Modus Ponens	$\frac{a \Rightarrow B,\ a}{B}$
And-Elimination	$\frac{a \wedge B}{a} \quad \frac{a \wedge B}{B}$

Bi-conditional elimination on R2

R6: $B(1,1) \Rightarrow (P(1,2) \vee P(2,1)) \wedge (P(1,2) \vee P(2,1) \Rightarrow B(1,1))$

And-elimination on R6

R7: $P(1,2) \vee P(2,1) \Rightarrow B(1,1)$

Contraposition

R8: $\neg B(1,1) \Rightarrow \neg(P(1,2) \vee PP(2,1))$

Modus Ponens

R9: $\neg(P(1,2) \vee P(2,1))$

De Morgan's

R10: $\neg P(1,2) \wedge \neg P(2,1)$

R11: $\neg B(1,\ 2)$

R12: $B(1,2) \Leftrightarrow (P(1,1) \vee P(1,3) \vee P(2,2))$

R13: $\neg P(2,2)$

R14: $\neg P(1,3)$

From R3: Bi-conditional elimination and given rules R1, R5, R14

R15: $P(1,1) \vee P(3,1) \vee P(2,2)$

Resolution on $\neg P(2,2)$ (R14)

R16: $P(1,1) \vee P(3,1)$

Resolution on $\neg P(1,1)$ (Rule 1)

R17: P(3,1)

- All logical sentences can be expressed in CNF (Conjunctive Normal Form)
- $(l1 \vee ... \vee lk) \wedge (m1 \vee ... \vee mr) \wedge (n1 \vee ... \vee n0)$
- Each one of these parentheses are called clauses, which are disjunction of literals

Ex: Turning sentence into CNF

$B(1,1) \Leftrightarrow P(1,2 \vee P(2,1)$

$(B(1,1) \Leftrightarrow (P1,2) \vee P2,1)) \wedge ((P(1,2) \vee P(2,1) \Rightarrow B(1,1))$

$(\neg B(1,1) \vee P1,2) \vee P(2,1)) \wedge (\neg (P1,2) \vee P(2,1) \vee B(1,1)))$

$(\neg B(1,1) \vee P(1,2) \vee P(2,1)) \wedge ((\neg P1,2)) \wedge \neg P(2,1) \vee B(1,1))$

$(\neg B(1,1) \vee P(1,2) \vee P(2,1) \wedge (\neg P(1,2) \vee B(1,1) \wedge (\neg P(2,1) \vee B(1,1))$

Inference by Resolution (Proof by Contradiction

- In order to show that KB (knowledge base) infers α (logical sentence to decide if it can be inferred)
- First show that the logical statement $(KB \wedge \neg\alpha)$

Steps

1. Represent the KB as a set of logical propositions
2. Turn the sentence in CNF
3. Apply the resolution rule over $KB \wedge \neg\alpha$ so as to get the empty clause (contradiction)

Boolean Satisfiability Problem

- Possible approaches:
    - Resolution Approach

- Classical Search: Backtracking
    - Davis-Putnam 1960's
- Local Search for Boolean Satisfiability
    - Start with a random assignment to the variables and then edit the assignment so as to decrease the number of unsatisfied clauses
    - $(A \vee D) \wedge (B \vee C) \wedge (B \vee \neg D) \wedge (\neg A \vee B) \wedge B$
    - Random initial assignment: A= F, B=F, C=T, D=F
    - Needs evaluation function: scores ⅗
    -

Neighbor 1	T	F	T	T	2/5
Neighbor 2	F	T	T	T	2/5
Neighbor 3	F	F	F	T	3/5
Neighbor 4	F	F	T	F	2/5

    -

- Heuristics:
    - No need to reach the leaf odes to realize satisfiability or not
        - E.g. (A ∨ B) ∧ (A ∨ C)
        - A is true
        - No need to branch out for B and C
    - A clause is true as long as 1 literal is true
    - A sentence is false as long as 1 clause is false

    > *Pure Symbol Heuristics*
        - A symbol is "pure" if it appears with the same "sign" in all clauses
        - Assign the value to the literal that satisfies the clauses
        - $(A \vee D) \wedge (B \vee C) \wedge (B \vee \neg D) \wedge (\neg A \vee B) \wedge B$
        - Here, set B= true; all clauses are satisfied where B appears

    > *Unit Clauses Heuristic*
        - A clause with one literal or clauses where all literals but one are false should indicate that the corresponding literal must be prioritized and set to a value that satisfies the values

Lecture #10 | 2.21.2017 - Classical Planning & PDDL, Intro to Probabilistic Reasoning

!!! EXAM I - MARCH 7TH, 3:20 - 4:40

- Propositional logic provides tools for representing problems but:
    - Time is not easily expressed

- You do not have operators like $\forall$, $\exists$ (need to go to higher-order logic)

Planning Domain Definition Language (PDDL)

- Another way to represent problems
- States are represented as conjunctions of positive literals
  - Everything that is not mentioned is assumed to be false
- Actions are specified by giving their *Preconditions* and their *Effects*

Actions

- Actions applicable when all preconditions are satisfied in current state
- Executing action *a* at state *s* results in a new state *s'* which is similar to *s* where fluents that appear negative in the EFFECTS(a) are removed from *s* and the positive ones are added

Time

- Represented implicitly, precondition refers to time *t* and effects refer to the time *t+1*

Variables

- Treated as existentially quantified
- $At(p, SFO) \wedge Plane(p) \rightarrow \exists \ a\ plane\ in\ SFO$

Building Block Example

- Stack blocks alphabetically from top to bottom of tower
- You have some Block(x), Clear(x), On(x, y)

Action: MoveOverBlock (x, previous, target)

- Precondition: Block(x) $\wedge$ Block(target) $\wedge$ Clear(x) $\wedge$ Clear(target) $\wedge$ x != target $\wedge$ On (x, previous) $\wedge$ x != previous $\wedge$ target != previous
- Have to check if this covers the case where target is table
- Effects: $\neg$ Clear(target) $\wedge$ On (x, target) $\wedge$ Clear(previous) $\wedge$ $\neg$ On(x, previous)

PDDL (cont)

- Given a PDDL problem specification, we can follow 2 high-level search procedures:

1. Forward Search
   - set as root of the search tree the initial state
   - define actions that are applicable to generate children states
   - successor states are generated by adding positive effects and deleting the negative
   - challenge: # actions tends to be very high

2. Backward Search (from goal)
   - start from goal state and considers actions whose effects agree with state specification

- typically, # of actions is smaller than forward search
- approach: try to express multiple valid states simultaneously through a specification of constraints over variables

- Current: In (x, P1) ∧ At(y, P2) ∧ At (P1, A1) ∧ At (P2, A2)
- Goal: At (x, A2) ∧ At (y, P2) ∧ At (P1, A2) ∧ At (P2, A2)
- Can guide search by defining heuristics for defining relaxed version of your problem
  - Can ignore preconditions
  - Consider only positive effects
  - Then count the number of actions that give you the goal state from the current state

Uncertainty & Probability

- So far, we had perfect knowledge of the state
- Variables were either true or false

First Order

- $\forall\ p \quad Symptom\ (p,\ toothache) \Rightarrow Disease\ (p,\ cavity)$
- This rule may be the most probability course by does not cover every case
- We will introduce probabilities to express our uncertainty about the world (i.e. how can we come up with these probabilities? data)

Unconditional Probability	P(cavity) = 0.1
Conditional Probability	P(a | b) = P (a ∧ b) / P(b)
Axioms	- P(true) = 1 - P(false) = 0 - 0 <= P(a) <= 1 - P (a v b) = P(a) + P(b) - P(a ∧ b) - P(¬ a) = 1 - P(a)

Full Joint Probability Distribution

- Know probability for any combination
- Through marginalization: $P(y) = \sum_z P(y,z)$
- Y = value you're interested in
- Ex: P(cavity and catch) = P(cavity and catch and toothache) + P(cavity, catch, and not toothache)

Lecture #11 | 2.23.2017

Decision Theory = Probability Theory + Utility Theory
Probability theory: representation and state estimates
Utility theory: decisions

2 choices:

Leave early for airport	1% lose airplane
	99% succeed - wait
Leave just on time for airport	5% losing airplane
	95% waiting short amount of time

Decision-Theoretic Agents

- Update our belief of where the agent is inside the state space
- Belief: a probability distribution over states, the belief is updated given sensing input and actions
- For each probable state for the current belief, we need to know its utility and we compute the expected utility according to the belief
- Then for all possible actions, compute the resulting expected utility
- Select action that maximizes expected utility

Inference using Full Joint Probability Distribution

- We can do so through application:
  - Marginalization: $P(Y) = \sum_Z P(Y, Z) = P(Y, -Z) + P(Y, Z)$
  - Conditional: $P(Y) = \sum_Z P(Y \mid Z) \cdot P(Z)$

  Comparisons

P(cavity | toothache)	$\frac{P(cavity \wedge toothache)}{P(toothache)} = \alpha \cdot P(cavity \cdot toothache)$
P(¬cavity | toothache)	$\frac{P(\neg cavity \wedge toothache)}{P(toothache)} = \alpha \cdot P(\neg cavity \wedge toothache)$
P(toothache, cavity, ¬catch, weather=cloudy)	P(weather=cloudy | toothache, ¬catch, cavity)
P(toothache, ¬catch, cavity)	P(weather=cloudy) · P(toothache, ¬catch, cavity)

Independence

- a,b are independent

P(a | b) = P(a)
P(b | a) = P(b)
P(a ∧ b) = P(a) · P(b)

Bayes Rule

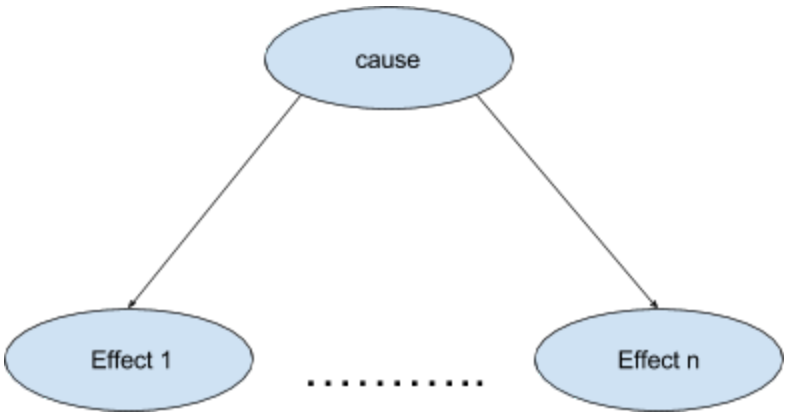
$$P(a \mid b) = \frac{P(b \mid a) \cdot P(a)}{P(b)}$$

$$P(Y \mid X, e) = \frac{P(X \mid Y, e) \cdot P(Y, e)}{P(X \mid e)}$$

Example

- Meningitis causes stiff neck 50%
- 1/50,000 people suffer from meningitis
- 1/20 suffer from stiff neck
- $P(m|s) = \frac{P(s|m) \cdot P(m)}{P(s)} = \frac{0.5 \cdot 1/50000}{1/220} = 0.0002$
- What you care about typically is whether:
  - P(m | s) is greater than P(¬ m|s)
  - P(m|s) = $\frac{P(s \mid m) \cdot P(m)}{P(s)} = \alpha \cdot P(s \mid m) \cdot P(m) P(m) = 1 - P(\neg m)$
  - $P(\neg m \mid s) = \frac{P(s \mid \neg m) \cdot P(\neg m)}{P(s)} = \alpha \cdot P(s \mid \neg m)(1 - P(m))$

- Bayesian Networks express the conditional independence properties among variables for our problems (e.g. toothache and catch are conditionally independent given cavity)



Naive Bayes Model

$$P(cause,\ effect_1,\ \ldots,\ effect_n) = P(cause) \cdot \prod_{i=1\ldots n} P(effect_i \mid cause)$$

Ex: P(cavity, catch, toothache) = P(cavity) * P(catch | cavity) * P(toothache | cavity)

Bayesian Network

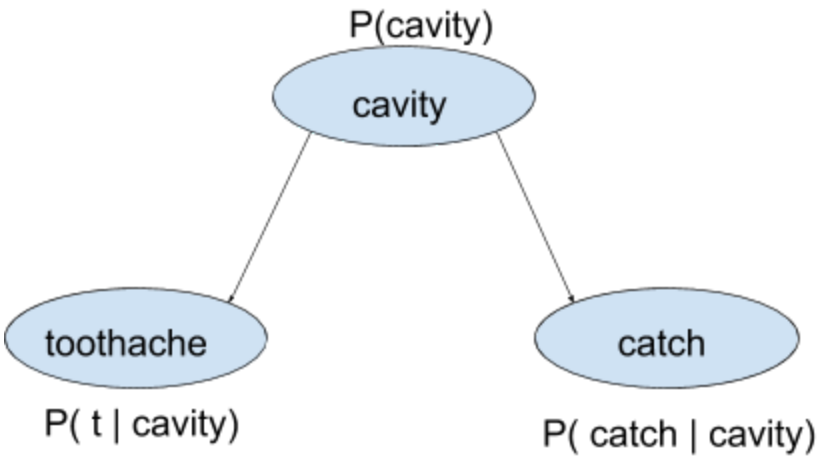
- Nodes are random variables
- Directed links connect pairs of nodes
  - X → Y (x is parent of y)
  - When X has direct influence over Y
- For each random variable Xi we have available the probability distribution
  - P(Xi | parents(Xi))
- Graph has no directed cycles

Lecture #12 | 2.28.2017

	toothache		¬ toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

Marginalization - way to answer probability and use to introduce variables

$$P(x) = \sum_y P(y,x) = \sum_y P(x \mid y) \cdot P(y)$$ ← second part is Conditioning



Bayesian Network

- Graphical representation that expresses conditional independence properties among variables of a problem involving uncertainty
- Nodes: random variables
- Edges: from variables X to dependent variables on X
- Conditional Probabilities: P(y| parents {Y})
- F.J.P.D: $2^n / 2^{30}$ (full joint probability distribution)
- Bayesian Network: $n \cdot 2^k / 30 \cdot 2^5$

$$P(X1, \ldots, Xn) = P(Xn \mid Xn-1, \ldots, X1) \cdot P(Xn-1, \ldots, X1)$$
$$= P(Xn \mid Xn-1, \ldots, X1) \cdot P(Xn-1 \mid Xn-2, \ldots, X1) \cdot P(Xn-2, \ldots, X1)$$
$$= \prod_{i=1\ldots n} P(Xi \mid Xi-1, \ldots, X1)$$

$P(Xi \mid Xi-1, \ldots, X1)$

- We know from the Bayesian Network that Xi is the conditionally independent given Parents {Xi}
- Then, if parents {Yi} is a subset of {Xi-1, … , X1}, we can rewrite as $P(Xi \mid Parents(Xi))$

Alarm Example

- Let's try to compute $P(B = t \mid J = t, M = t)$

F.J.P.D: P(B,E,A,J,M)

[definition] $P(B = t \mid J = t, M = t) = \frac{P(B=t, \mid J=t, M=t)}{P(J=t, M=t)} = \alpha \cdot P(B = t, J = t, M = t)$

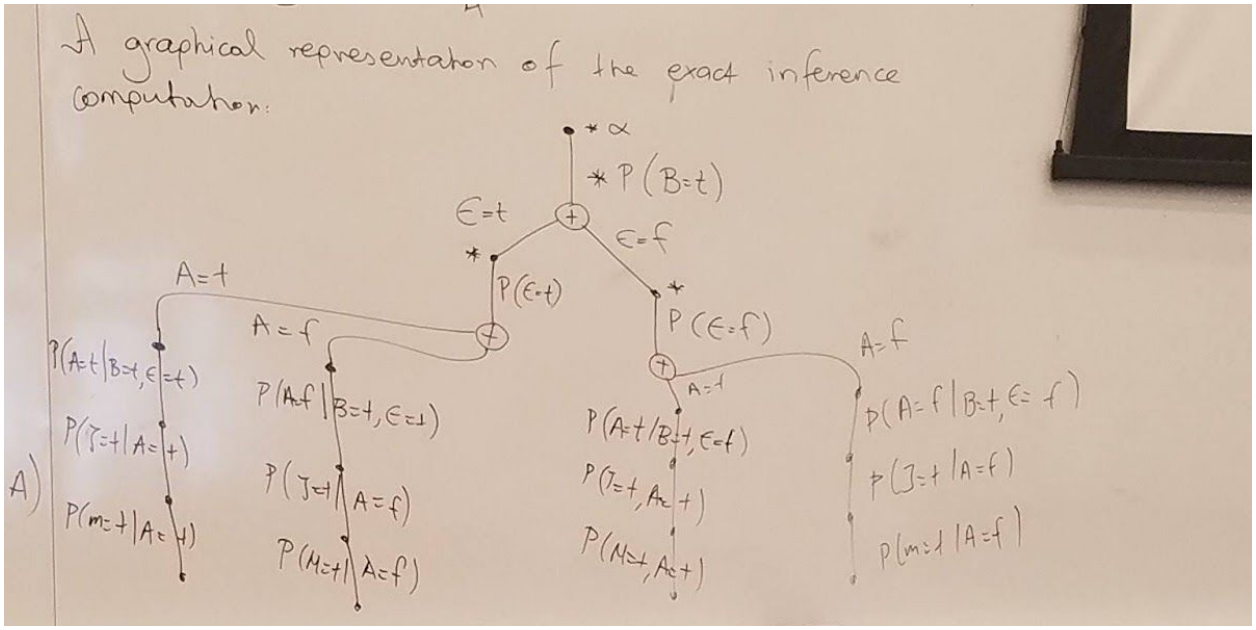
[conditional] $= \alpha \cdot \sum_E \sum_A P(B = t, E, A, J = t, M = t)$

$$= \alpha \cdot \sum_E \sum_A P(B = t) \cdot P(E) \cdot P(A \mid B = t, E) \cdot P(J = t \mid A) \cdot P(M = t \mid A)$$

- Worse Case: requires $n \cdot 2^n$ computations
- To reduce computations, can rewrite as:

$$\alpha \cdot P(B = t) \cdot \sum_E P(E) \cdot \sum_A P(A \mid B = t, E) \cdot P(J = t \mid A) \cdot P(M = t \mid A)$$

- Graphical representation of the exact inference computation



- Algorithm can follow a DFS to traverse the corresponding tree
- In worse case, still exponential complexity

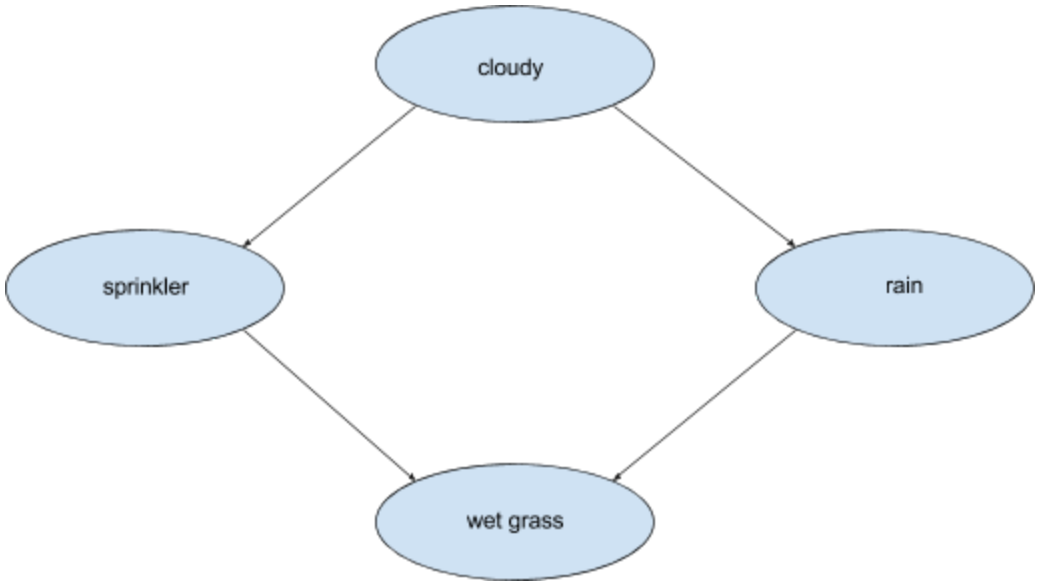
Variable Elimination

- Alternative approach for exact inference that is bottom-up
- In terms of alarm example:

Factor1(A)	$P(M=t \mid A)$	$P(M=t \mid A=t)$ $P(M=t \mid A=f)$
Facctor2(A)	$P(J=t \mid A)$	$P(J=t \mid A=t)$ $P(J=t \mid A=f)$
Factor3(A)	F2(A) * F1(A)	$P(J=t \mid A=t) \cdot P(M=t \mid A=t)$ $P(J=t \mid A=f) \cdot P(M=t \mid A=f)$
Factor4(A,E)	$P(A \mid B=t,\ E)$	$P(A=t \mid B=t,\ E=t)$ $P(A=t \mid B=t,\ E=f)$ $P(A=f \mid B=t,\ E=t)$ $P(A=f \mid B=t,\ E=f)$
Factor5(A,E)	F4(A,E) * F3(A)	F4(1,1)*F3(1) F4(1,2)*F3(1) F4(2,1)*F3(2) F4(2,2)*F3(2)
Factor6(E)	$\sum_A F5(A,E)$	F5(1,1) + F5(2,1) F5(1,2) + F5(2,2)

Factor7(E)	P(E)	$P(E=t)$ $P(E=f)$
Factor8(E)	F6(E)*F7(E)	F6(1)*F7(1) F6(2)*F7(2)
F9	$\sum_E F8(E)$	F8(1) + F8(2)
F10	P(B=t) * F9	

- Variable Elimination reduces the amount of computation necessary
- Avoid repeating the same computations
- Can be shown that on "polytrees", it can achieve polynomial complexity
- Exact inference in the general/ worst case remains exponential complexity
- This complexity issue motivates the consideration of approximate inference methods (e.g., Monte Carlo methods)



- P(C) = 0.5
- Sprinkler

S	C
0.9	F
0.1	T

Rain

P(R)	C
0.2	F
0.8	T

WG	S	R
0.99	T	T
0.90	T	F
0.90	F	T
0.00	F	F

$P(C = t,\ S = f,\ R = t,\ WG = false)$

We are going to follow a sampling process to compute the F.J.P.D from the Bayesian Network

- Sample each variable in topological order according to the conditional probability in the Bayesian Network
- First, sample from cloudy: <0.5, 0.5>, assume this returns true
- Then, consider Sprinkler and sample according to <0.1, 0.9>, assume this returns false
- Then, consider Rain and sample according to: <0.8, 0.2>, assume this returns true
- Then, consider Wet Grass and sample according to: <0.9, 0.1>, assume this returns true
- 1,000 samples: 500 cloudy is true, 150 WG is false, 350 WG is true

Lecture #13 | 3.2.2017

- Variable Elimination
    - Alternative to naive inference by enumeration
    - Benefits: drops irrelevant variables, avoids repeated computations

- Time & Space Complexity
    - Depends on largest factor that arises (makes matrix large)
    - Depends on order of elimination
    - If there is only one path between any 2 variables in the tree, the complexity of variable elimination is no longer exponential

- General Case: Exact Probability Inference
    - Implies exponential complexity

Approximate Inference via Sampling-Based Processes

1) Direct Sampling

- Sample Cloudy first <0.5, 0.5>, assume false
- Sample Sprinkler from <0.5, 0.5>, assume false

- Sample Rain from <0.2, 0.8>, assume false
- Sample Wet Grass from <0.0, 0.0>, false
- <false, false, false, false>
- 0.5 * 0.5 * 0.8 = 0.2

- Suppose we generate __ M total samples and let N (X1, … , Xn) be the # of samples corresponding to the atomic event <X1, … , Xn>
    - We expect: $\lim_{(\ blank)\ M \to \infty} \frac{N\ (X_1, \dots, X_n)}{(\ blank\ )\ M} = P(X_1, \ \dots . , \ X_N)$

2) Conditional Probabilities P(X | e)

A) Rejection Sampling
- Follow first direct sampling to approximate the F.J.P.D
- Ignore all the atomic events which disagree with the evidence
    - P(Rain | Sprinkler=true)using 100 samples
    - 73: sprinkler is false
    - 27: sprinkler is true → 8 has rain true, 19 has rain false
    - Total : 8/27
- Issues: wasted computations, may not provide enough samples in useful part of state space

B) Likelihood Weighting
- Fix the evidence variables and sample only the remaining ones
- Not all event, however, are equally likely when we sample in this manner
- For each event we will keep track of the likelihood in the F.J.P.D
    - P (Rain | Sprinkler=true, Wet Grass = true) where likelihood = 9.0
    - Sample Cloudy from <0.5, 0.5>, assume true where likelihood=1.0
    - Set Sprinkler to true with likelihood = 0.1 * 1.0 = 0.1
    - Sample Rain from <0.8, 0.2>, assume true
    - Set Wet Grass to true with likelihood = 0.99 * 0.1 = 0.099
    -

C	S	R	WG	Prob
F	T	F	T	0.45
F	T	T	T	0.495
T	T	F	T	0.09
T	T	T	T	0.099

    - (0.49 + 0.099) / sum of probabilities = rain is true or not
    - Answer: to query will be:

$$\frac{\sum \text{ weights of atomic events with rain = true}}{\sum \text{ weights of all atomic events}}$$

3) Markov Chain Simulation

- So far we have generating atomic events one variable at a time
- Can also sample values for all variables and then consider the transition to a new atomic event and the corresponding probability for this transition
    - P (Rain | Sprinkler=true, Wet Grass=true)
    - Initially, randomly select a valid atomic event: Cloudy=true, Rain=false
    - Initial State:
        -

C	S	R	WG
T	T	F	T

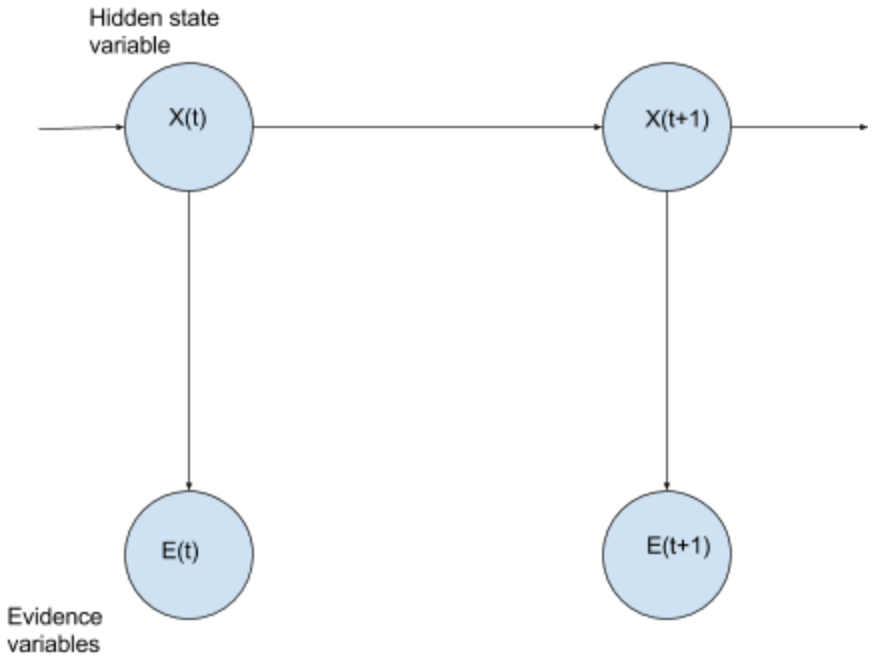
    - Then, we consider a potential change to one variable and sample according to its Markov Blanket
    - P(X | everything else) = P (X | M.B.(X))

$$= P(X \mid Parents\{X\}) \cdot \prod_{Y_i \in \text{children of } \{x\}} P(Y_i \mid Parents\{Y_i\})$$

    - Sample Cloudy next
        - P(C | M.B. (C)) = P(C) · P(S=true | C) · P(R=false | C)
    - Suppose Cloudy is sampled False (1 iteration)
        - New State: [F, T, F, T]
    - Sample Rain next (1 iteration)
        - P(R | M.B. (C)) = P(R | C=false) · P(WG=true | R, S=true)
        - Assume rain is sampled false
            - New State: [F, T, F, T]

Lecture #15 | 3.21.2017

Temporal Probabilistic Estimation



- Markov Assumption
    - Each state/ observation depends on a finite history
    - First-order: depends only on the last state
- Stationary Process
    - The observation model and the transition model remain the same over time

<u>Define a DBN</u>

- Input necessary:
    - Prior belief: P(Xo)
    - Transition Model P(X(t) | X(t-1))
    - Observation Model: P(E(t) | X(t))
- Access To:
    - At each step we can observe E(t)
- Problems:
    - Filtering: P(X(t) | E(1=t))
    - Prediction: P(X(t+k) | E(1=t); k >= 1
    - Smoothing: P (X(k) | E(1=t) ; 1 <= k < t
    - Most Likely Explanation: what are the most likely sequence of states that explains the observations that we have seen up to this point
        - P (X(1=t) | E(1]t))
    - One Step Prediction: P(X(t+1) | E(1]t))
        - Let's assume we have access to the last filtering estimate P(X(t) | E(1=t)
        - Apply conditioning to the prediction probability

- $P(X_{t+1} \mid E_{1=t}) = \sum_{X_t} P(X_{t+1} \mid X_t, E_{1=t}) \cdot P(X_t \mid E_{1=t})$
- $= \sum_{X_t} P(X_{t+1} \mid X_t) \cdot P(X_t \mid E_{1=t})$

Final Review Session

SVMs

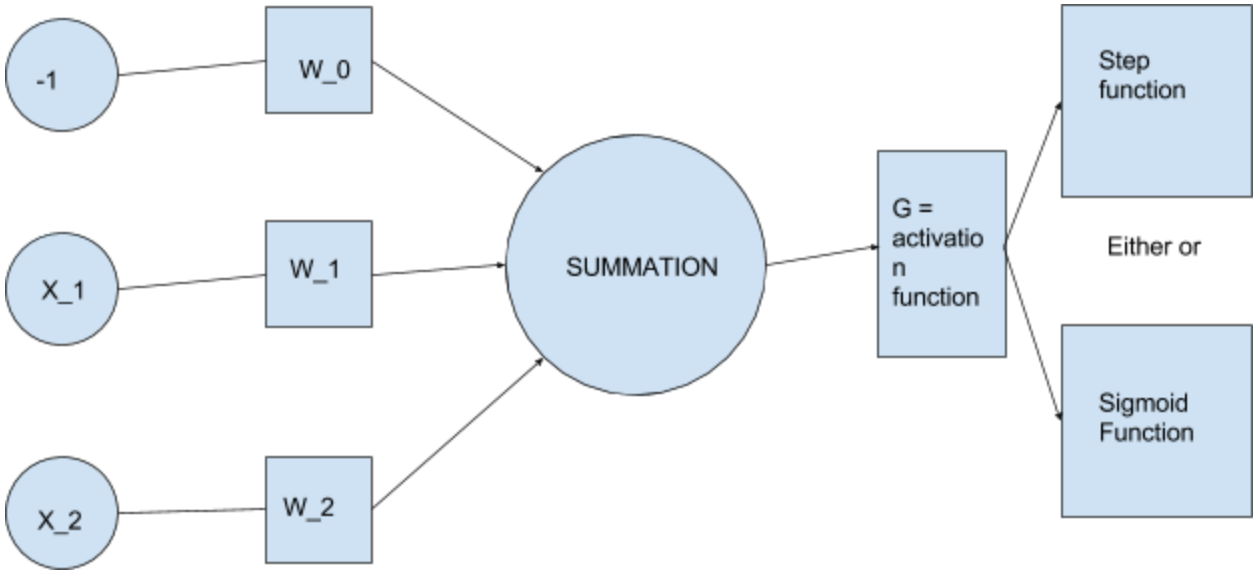
- Assumption: data points are linearly separable
- Problem: constraint optimization
- Optimization component: maximizing margin
- Constraint: correctly classify the data points
- Solution: depends only on dot products
- If data points are not linearly separable
    - N data points are always linearly separable in an N-1 dimensional space
    - Idea: Map points in a high enough dimension so they can be separated
    - Instead of explicitly doing the mapping, can perform "kernel trick"
    - Kernel Trick - use a function $K(x_i, x_j) = F(x_i) \cdot F(x_j)$
        - K is equivalent to $(1 + x_i x_j)^d$

Overfitting

- May perfectly explain your own data points but may lose generality
- Need third evaluation test aside from testing and training sets to report ratio of success

Artificial Neural Network (ANNs)

- Artificial neuron output = $g(\sum_{i=0}^{k} x_i \cdot w_i)$
- Prefer sigmoid function because you can take derivatives
- Want derivatives because the learning process where we compute weights, can find error between expected vs actual output



- Come up with a maximum something about neural net
- How can we learn wait of perceptron?
- Do hill climbing, goal is to minimize error
  - $Error = y - h_w(x)$
- Come up with weight for specific node that performs some function

- Compute utility for what happens after
- What is the approximation
- Describe this
- Explain this
- Tiger problem
- Requirements of _____: must be gaussian, process(transitioning and observation models) must be linear and gaussian noise added to it
- Viterbi Algo - most likely explanation (type of problem we solve with dynamic bayesian nets)

<u>Types of Problems</u>

Argmax P(X_i | E_i=1	Filtering
	Smoothing

What is the trajectory that maximizes this probability - argmax

No fun and game rule - merge into one individual lottery