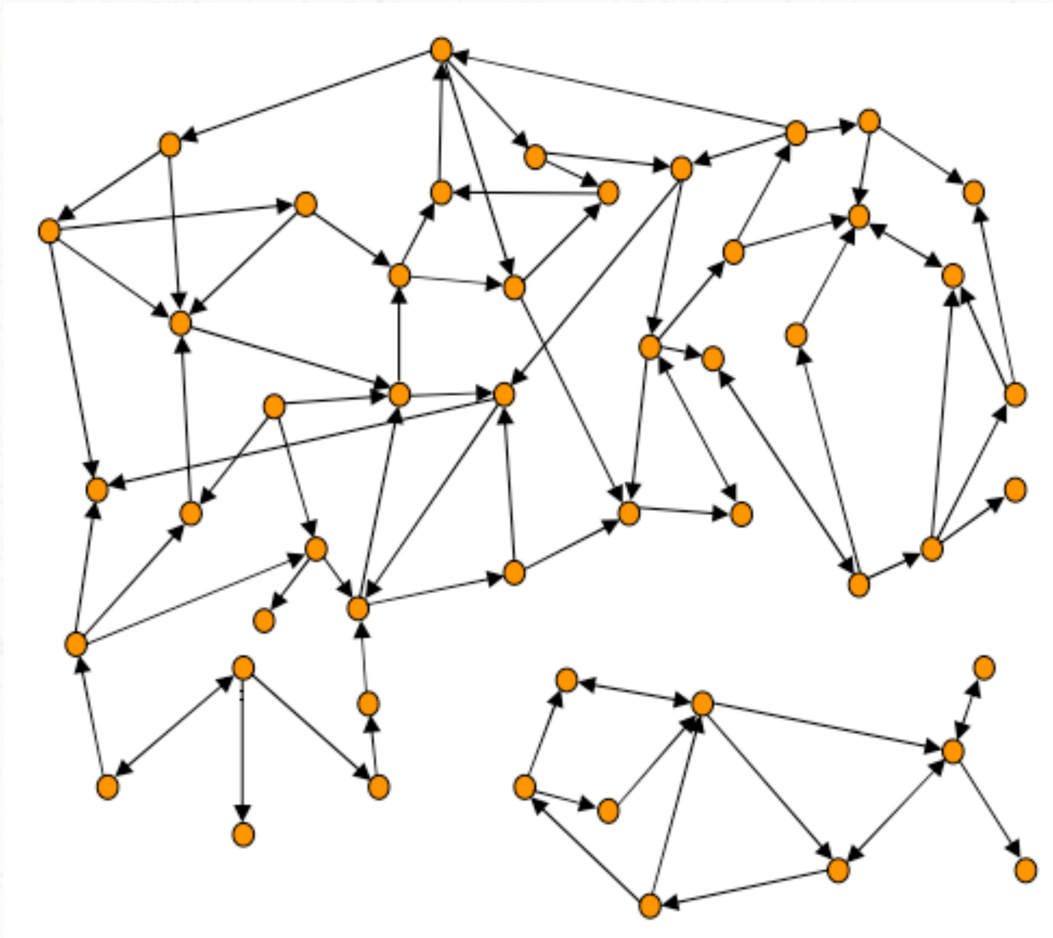
CS121

- Heuristic Search
- Planning
- CSPs
- Adversarial Search
- Probabilistic Reasoning
- Probabilistic Belief
- Learning

Heuristic Search

- First, you need to formulate your situation as a Search Problem
  - What is a state?
  - From one state, what other states can you get to (successor function)?
  - For each of those transitions, what is the cost?
  - Where is the start? What is the goal?

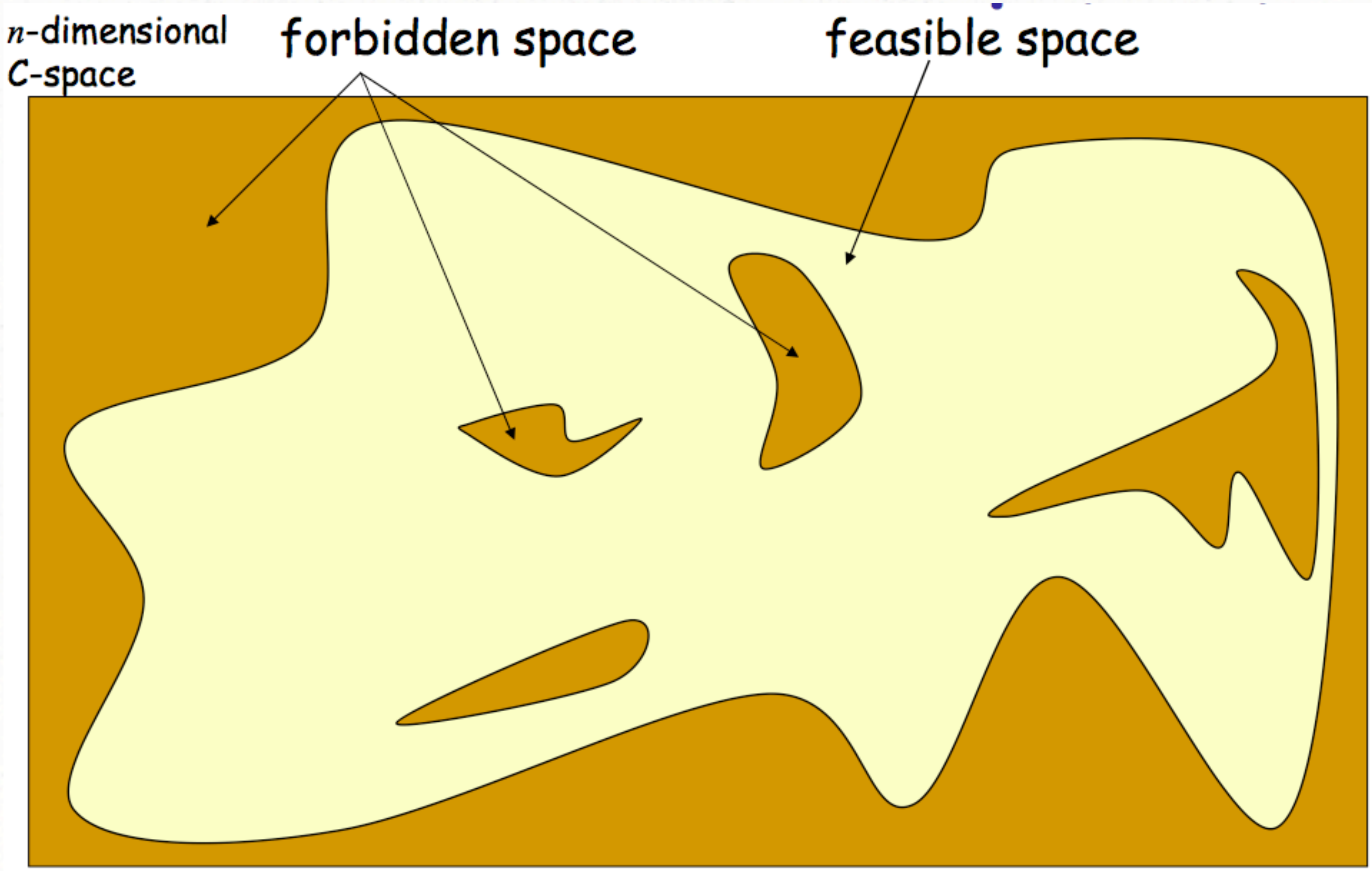
Heuristic Search



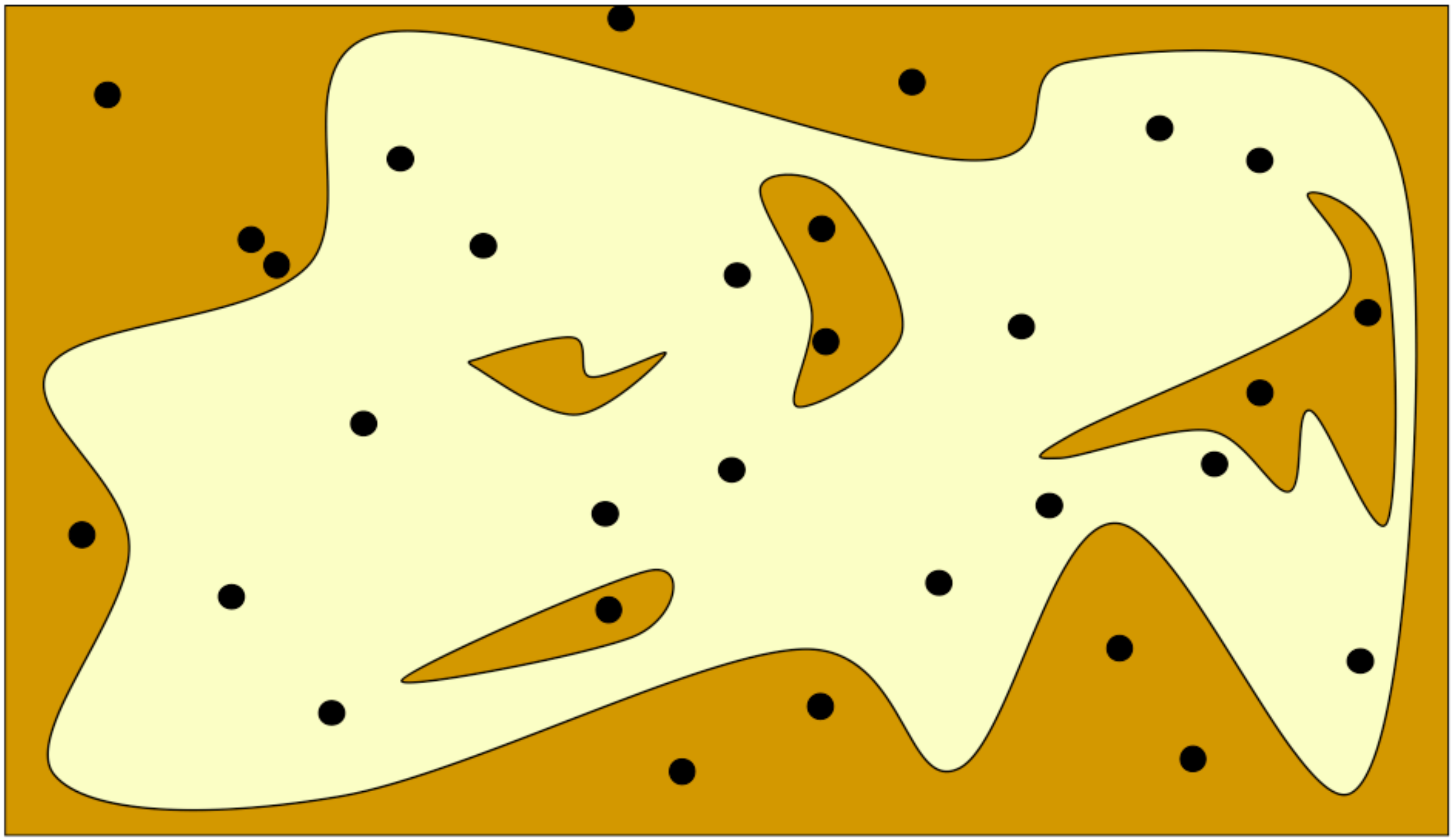
Heuristic Search

- Easy to formulate for problems that are inherently discrete
  - Solve a rubik's cube
  - Given all the flights of the airlines, figure out the best way (time/distance/cost) to get from city A to city B
- What about problems that have continuous spaces?
  - Maneuvering a robot through a building
  - Controlling a robot arm to do a task

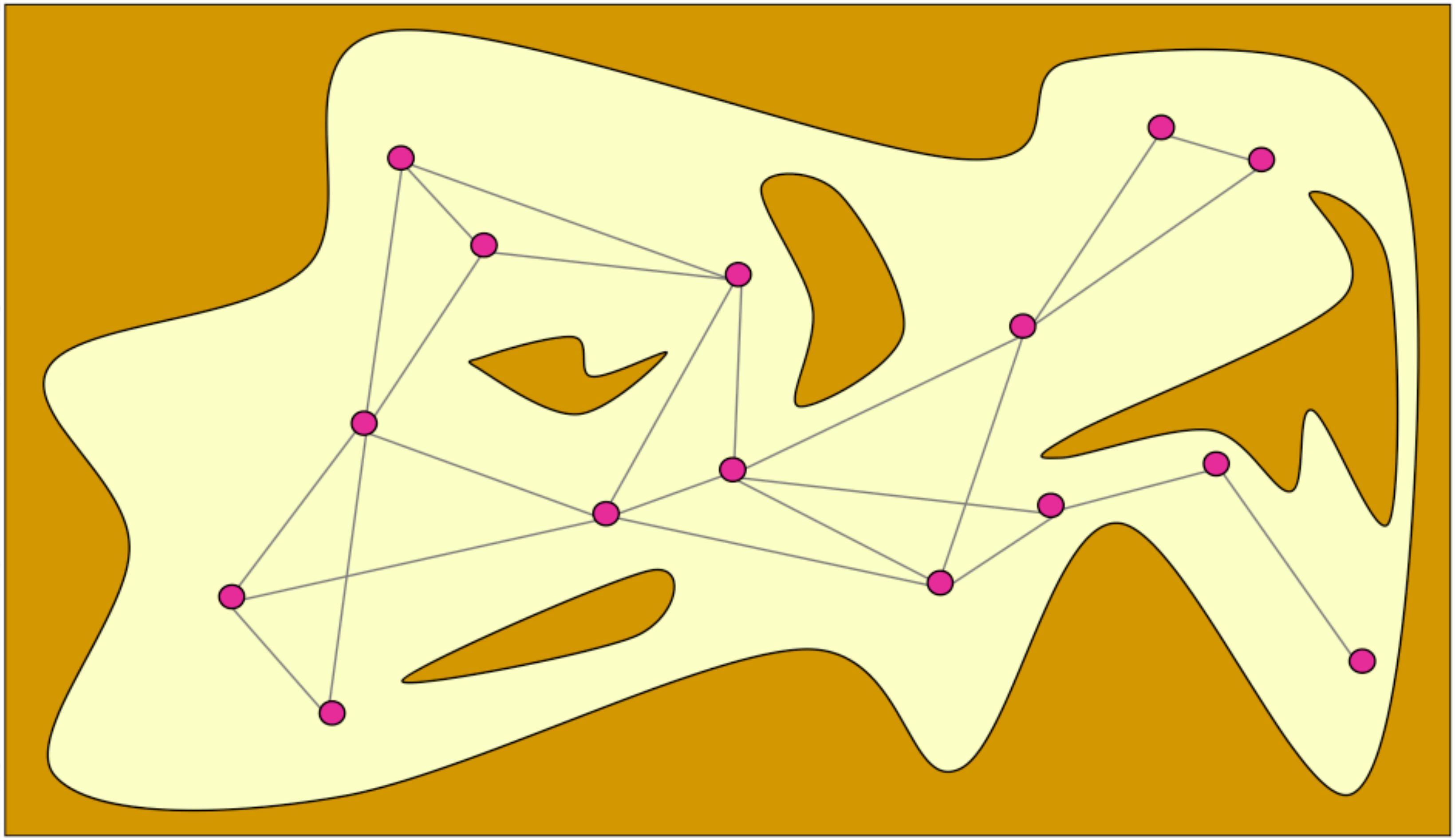
Heuristic Search



Heuristic Search



Heuristic Search



Heuristic Search

- No Heuristic
  - DFS, BFS, Iterative Deepening, Uniform Cost
- Heuristic
  - Have fringe sorted by $f = g + h$
  - Admissibility
  - Consistency

Planning

- Just a search problem!
- Use STRIPS to formulate the problem
    - A state is a set of propositions which are true
        - IN(Robot, R1), HOLDING(Apple)
    - Successor function given by Actions
        - Preconditions (which are allowed)
        - Add/Delete (what is the new state)
    - How do we get a heuristic?

Planning

- Given some state s, how many actions will it take to get to a state satisfying g?
- Planning Graph
  - Initialize to S_0 all the proposition in s.
  - Add the add lists of actions that apply to get S_1
  - Repeat until convergence
- Find the first S_i where the g is met

Planning

- Forward Planning
  - Start initial node as initial state
  - Find all successors by applying actions
  - For each successor, build a planning graph to determine heuristic value
  - Add to fringe, pop, repeat
- Problems
  - branching factor,
  - multiple planning graphs

Planning

- Backward Planning
  - Construct planning graph from initial state
  - Start initial node as goal
  - Find successors by regressing through relevant actions
  - Look up heuristic values in planning graph
  - Add to fringe, pop, repeat

Constraint Satisfaction

- Formulation
  - Variables, each with some domain
  - Constraints between variables and their values
  - Problem: assign values to everything without violating any constraint
- Again, just a search problem (Backtracking)
  - State: Partial assignment to variables
  - Successor: Assign a value to next variable without violating anything
  - Goal: All variables assigned

Constraint Satisfaction

- No sense of "optimal" path.. we just want to cut down on search time.
- How to choose variable to assign next?
  - Most constrained variable
  - Most constraining variable
- How to choose the next value?
  - Least constraining value

Constraint Satisfaction

- To benefit from these heuristics, should update domains
  - Forward Checking
    - After assigning a value to a variable, remove all conflicting values from other variables
  - AC3
    - Given a set of variables, look at pairs X,Y
      - If for a value of X, there is no value of Y that works, remove that value from X

Adversarial Search

- Game tree from moves performed successively by MAX and MIN player
- Values at “bottom” of the tree – end of game, or use evaluation function.
- Propagate values up according to MIN/MAX
- Tells you which move to take
- Alpha-Beta pruning
  - Order of evaluation does matter

Probabilistic Reasoning

- Assume there is some state space
- Now actions are probabilistic
  - If I do action A, there are several different possible states I may end up in
  - There is a probability associated with going into each state (they must sum to 1)
- Some states have rewards (positive or negative)
- We would like to calculate utility for each state, and use that to determine what action to take.

Probabilistic Reasoning

$$U(s) = R(s) + \max_{a \in Appl(s)} \sum_{s' \in Succ(s,a)} P(s'|a.s)U(s')$$

Appl(s) is the set of all actions applicable to state s

Succ(s,a) is the set of all possible states after applying a to s

P(s'|a.s) is the probability of being in s' after executing a in s

Probabilistic Reasoning

- How do you calculate the Utilities?
  - If no cycles, can back values up the tree
  - Otherwise, can use Value Iteration
    - Start all utilities as 0, calculate new utilities, repeat until convergence
  - Or, Policy Iteration
    - Pick a random policy, solve utilities for it, calculate new policy until convergence

Probabilistic Belief

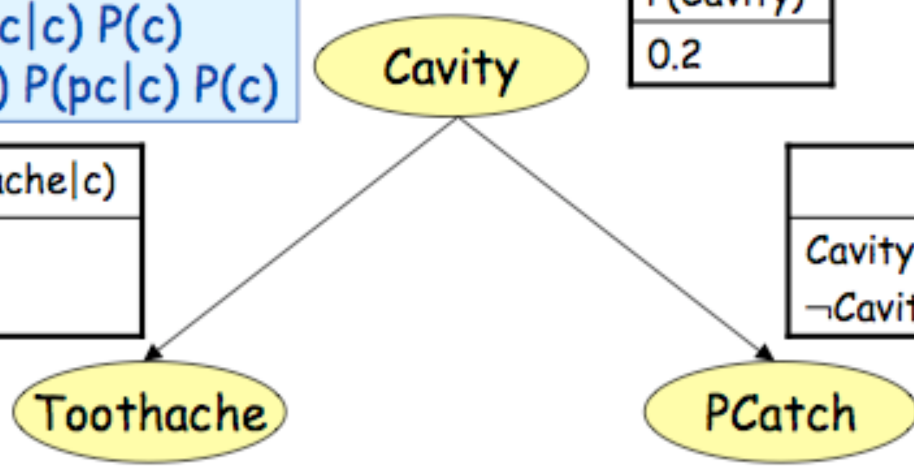
- Say N variables, each with 2 values, joint probability table has 2^n entries.

	Toothache		¬Toothache	
	PCatch	¬PCatch	PCatch	¬PCatch
Cavity	0.108	0.012	0.072	0.008
¬Cavity	0.016	0.064	0.144	0.576

Probabilistic Belief

- If variables are independent, can represent this table more compactly

$$P(c \wedge t \wedge pc) = P(t \wedge pc|c)\,P(c) = P(t|c)\,P(pc|c)\,P(c)$$



P(Cavity)
0.2

	P(Toothache|c)
Cavity	0.6
¬Cavity	0.1

	P(PClass|c)
Cavity	0.9
¬Cavity	0.02

(Supervised) Learning

- We are given a bunch of examples, where each example has values X1.. XN and Y
- We want to create some function H(X), that will take all the X's and output a single value
- The goal is that given some partial example X1... XN, we can use H(X) to guess Y
- This should work well for X's from the training set, but also for X's never seen before!

(Supervised) Learning

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

(Supervised) Learning

Some types of functions we can use:

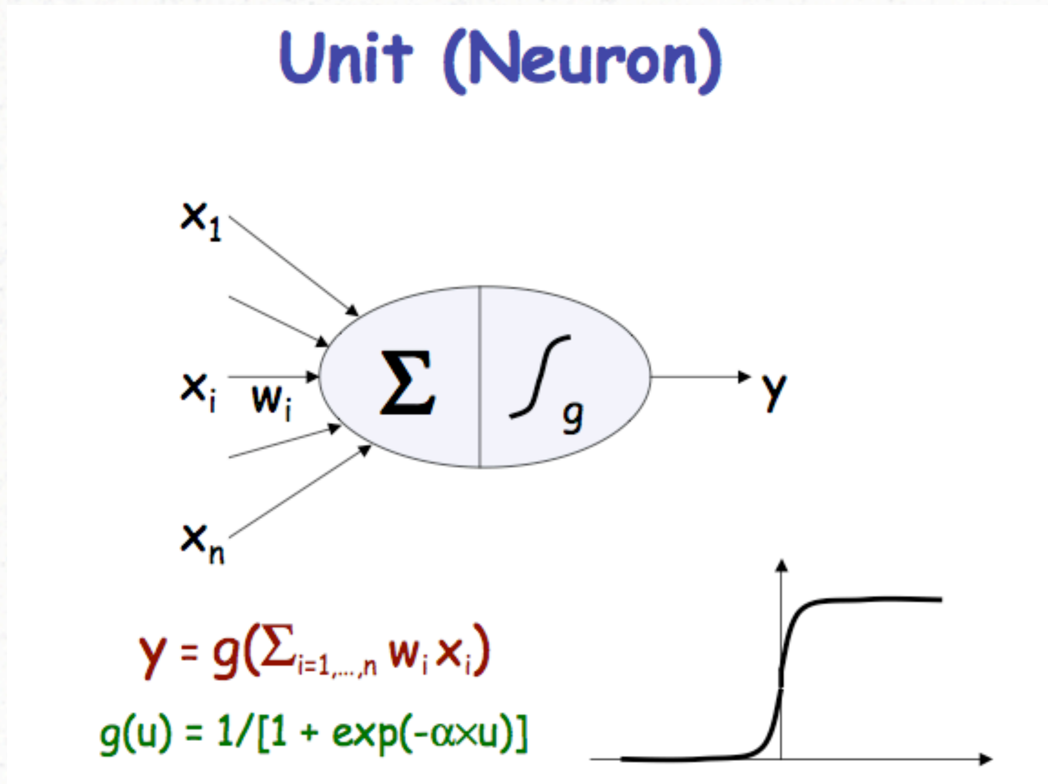
- Data Cache
- Linear Regression
- Decision Tree
- Neural Net

(Supervised) Learning

- Decision Tree
  - At each non-terminal node in tree, branch according to the value of one of the Xi's
  - A leaf node should output a value for Y
- Building the Tree (Greedy)
  - Look at all examples at current node
  - Choose Xi to split on that will allow you to classify the most number of examples correctly

(Supervised) Learning

- Neural Net



(Supervised) Learning

- Neural Net