CS 188: Intro to AI Lecture Notes

Week 1: Lecture 1 Introduction (1/20)

What is artificial intelligence?

Short History

- 1940s: McCUlloch & Pitts: Boolean circuit model of brain
- 1950-1970: Excitement: Early AI: chess, checkers, "complete algorithm for logical reasoning"
- 1970-1990: Knowledge based approaches: early development of knowledge-based systems,
- 1990 - 2012: Statistical approaches, probability uncertainty
- 2012 - : big data, big compute, deep learning, AI used in many industries

AI as Designing Rational Agents

- An *agent* is an entity that perceives and acts
- A *rational agent* selects actions that maximize its expected *utility*
- Characteristics of sensors, actuators

Week 1: Lecture 2 Agents and Environments (1/20)

What's bad about better AI?

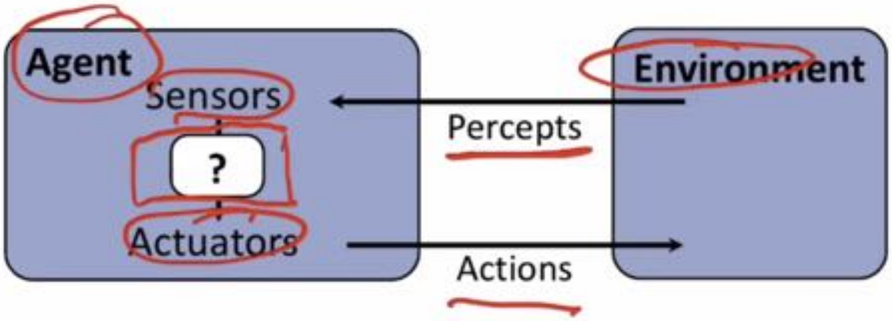
- AI is incredibly good at achieving something other than what we really want
- AI, economics, stat all assume utility to be fixed, known and exogenously specified
- Machines are *beneficial* to the extent that *their* actions can be expected to achieve *our* objectives

A new model for AI

1. The machines only obj is to max the realization of human preferences
2. The robot is initially uncertain about what those preferences are
3. Human behavior provides evidence about human preferences

Agents and environments

- Agent percieves its environment through sensors and acts upon it through actuators (or effectors)



Agent Functions

- Maps percept histories to actions:
- F: P* -> A

Agent programs

- L runs on some machine M to implement f
- F = Agent (l, M)
- Limited

Vacuum example

- Performance measure evaluates the environment sequence
    - One point per square cleaned up
        - NO rewards an agent who dumps dirt and cleans it up
    - One point per clena square per time step
- Rational agent choose action to maximize the expected value

Rationality

- Are rational agents omniscient
    - No they are limited by the available percepts
- Are rational agents clairvoyant?
    - No they may lack knowledge of the environment dynamics
- Do rational agents explore and learn?
    - Yes in unknown environments, these are essential
- Do rational agents make mistakes?
    - No but their actions may be unsuccessful

Week 2: Lecture 3 Agents and Environments (1/25)

PEAS: Automated taxi

- Performance measure
- Environment
- Actuators
- Sensors

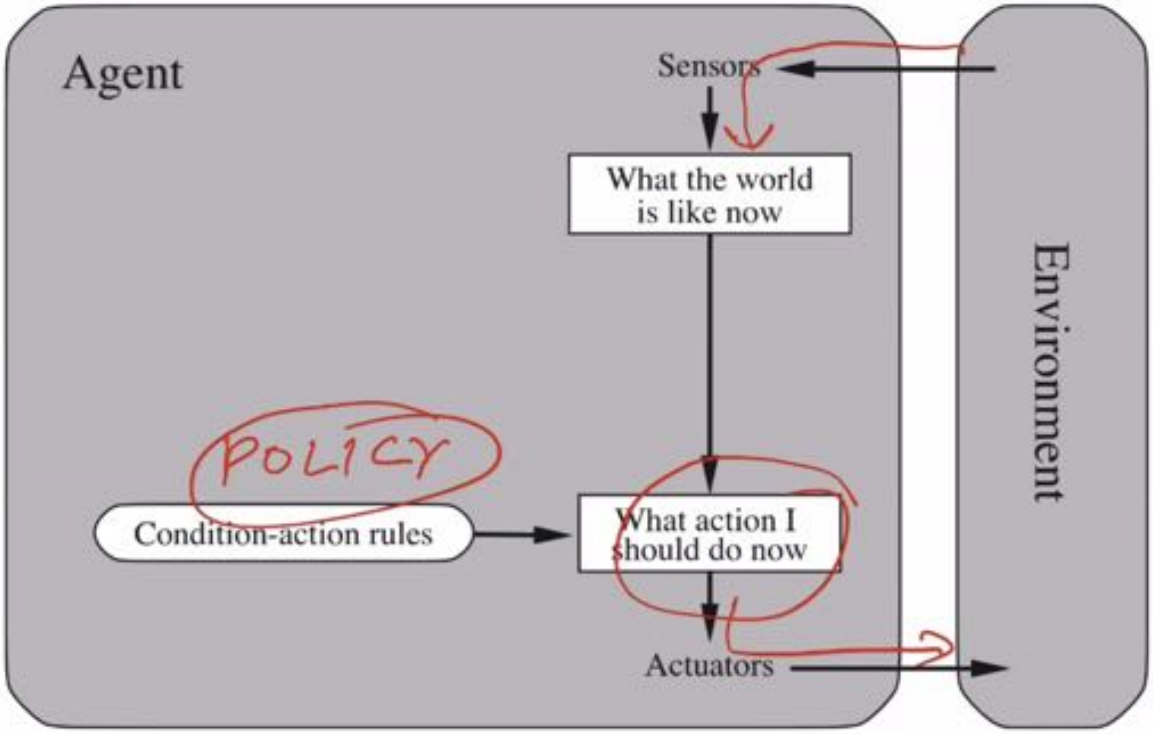
Environment types

- Fully or partially observable
- Single-agent or multiagent
- Deterministic or stochastic
- Static or dynamic
- Discrete or continuous
- Known physics?
- Known perf measure

Agent Design

- Partially observable -> requires memory
- Stochastic -> agent may have to prepare for contingencies
- Multi-agent -> agent may need to behave randomly

- Static -> agent has time to compute a rational decision



Can we extend reflex agent to behave well,

- Pacman is not fully observable power pellets

Week 2: Lecture 4 Uninformed Search (1/25)

Search Problems

- Consists of
    - State space S
    - Initial state s_0
    - Actions A9s) in each state
    - Transition model Result(s, a)
    - Goal test G(s)
    - Action cost c(s, a, s')
- Solution is an action sequence tha reaches a goal state
- Optimal solution has least cost among all solutions

State Space Sizes

- Agent positions: 120, Food count: 30, ghost positions: 12

State Space Graphs

- State space graph, each state occurs only once
- Can rarely build this full graph in memory

Search Trees

- Can be infinitely large, node for each path

General Tree Search

-

<u>Week 2: Lecture 5 Informed Search (1/27)</u>

Uniformed Algorithms

- Start out radially from the start

Informed ALgorithms

- Are guided toward the goal

<u>A*</u>

- Expand a node *n* most likely to be on the optimal path
- Expand a node n st the cost of the best solution through n is optimal
- Expand a node n with the lowest value of g(n) + h*(n)
    - g(n) is the cost from root to n
    - h*(n) is the optimal cost from n to the closest goal

Example: pathing in Pacman

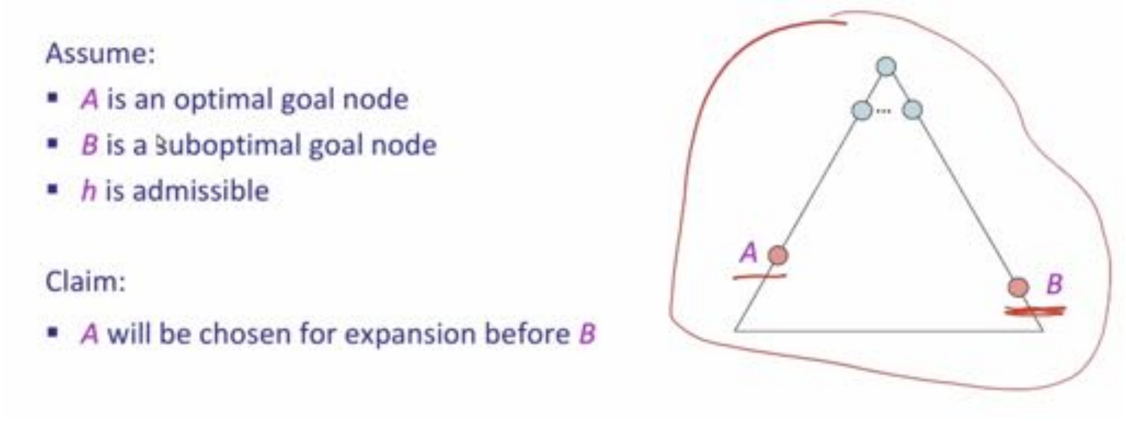
- h(n) = Manhattan distance = |delta x| + |delta y|

Bad solutions

- Actual bad solution cost < estimated good solution cost

Finding good, cheap admissible heuristics is the key to success

Optimality of A* Tree Search



Assume:

- *A* is an optimal goal node
- *B* is a suboptimal goal node
- *h* is admissible

Claim:

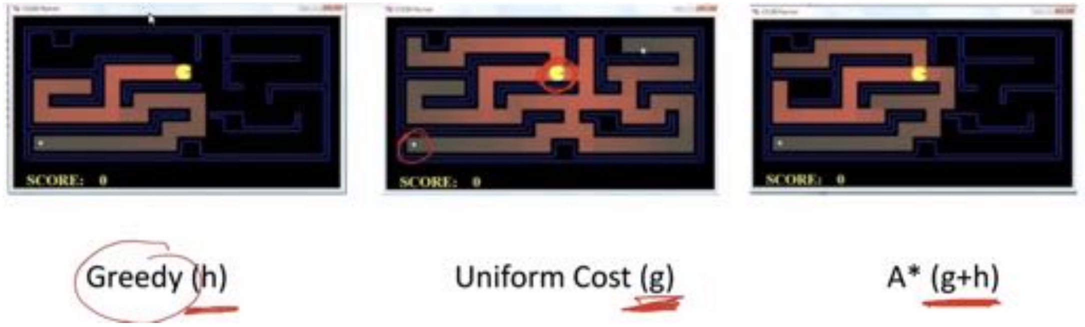
- *A* will be chosen for expansion before *B*

Proof, imagine B is on the frontier

- f(n) is less than or equal to f(A)
- f(A) is less than f(B)
- N is expanded

Uniform-cost expands equally in all directions, A* expands toward goal

Comparison



Creating Admissible Heuristics

- Often admissible heuristics are solutions to relaxed problems, where new actions are available ,
- Like going through walls in pacman

8 Puzzle

- States:
    - discrete tile locations
- How many States:
    - 9!
- What are the actions?
    - New tile for blank
- What are the step costs?
    - 1
- Heuristic:
    - Number of tiles misplaced
    - Admissible since needs to be moved to correct location

Combining Heuristics

- Dominance: $h_1 > h_2$
- Larger is better as long as both are admissible
- Zero heuristic is pretty bad
- Exact heuristic is pretty good but too expensive
- If two heuristics, neither dominates the other,
    - New heuristic: $h(n) = \max(h1(n), h2(n))$

Knights

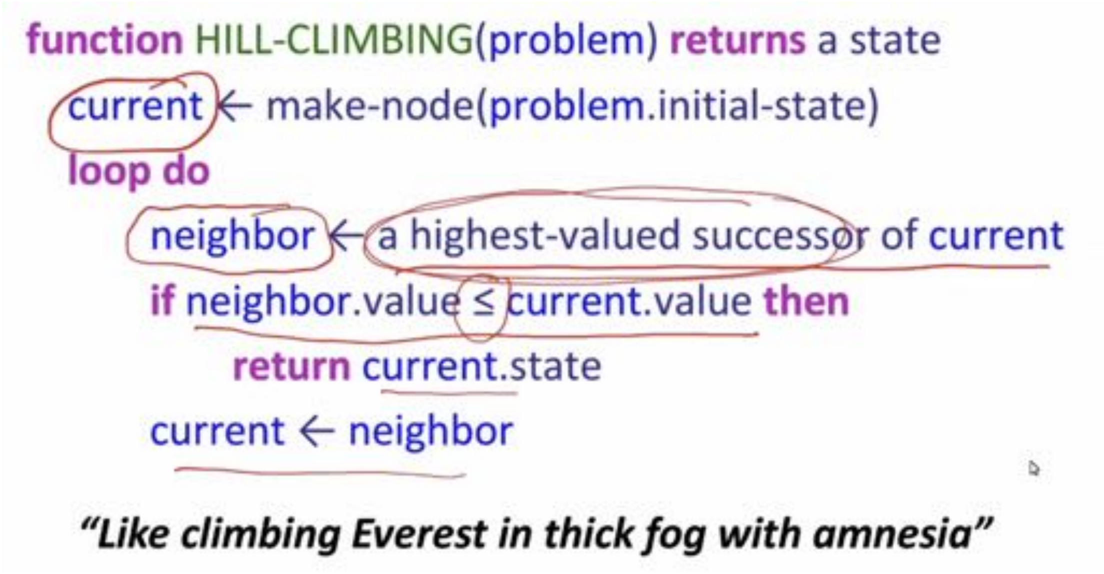
- If end square dif color from start square need odd
- H1 = manhattan distance /3
- H2 = euclidian distance / sqrt(5)

Week 3: Lecture 6 Local Search (2/1)

Hill Climbing

- Start wherever

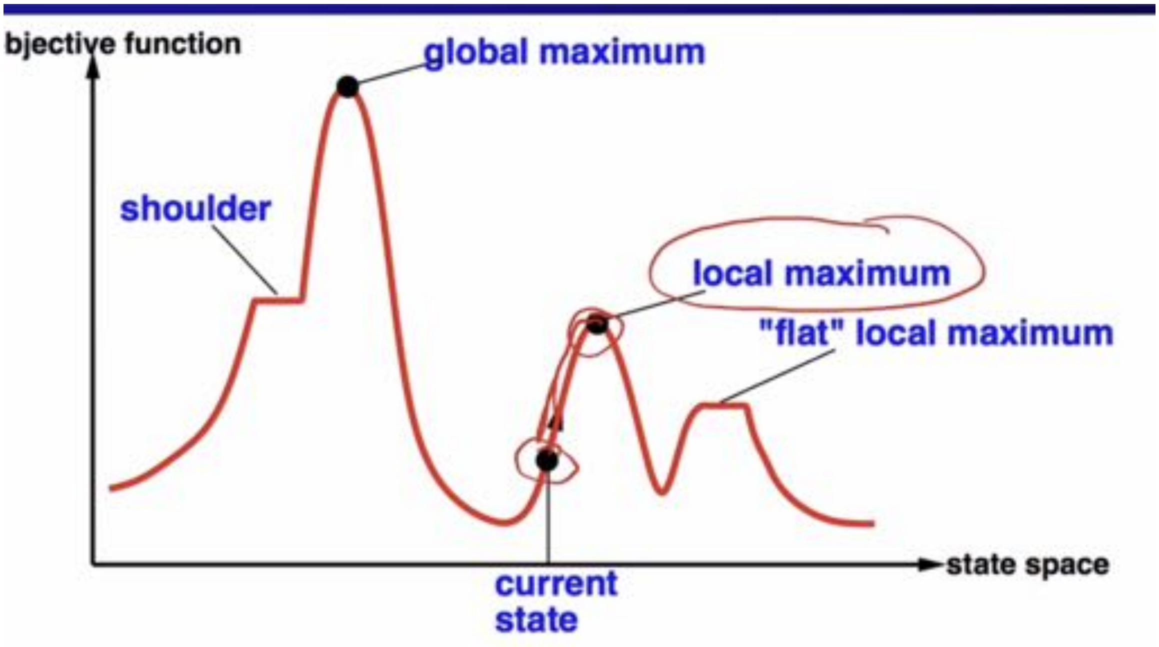
- Move to the best neighboring state



```
function HILL-CLIMBING(problem) returns a state
  current ← make-node(problem.initial-state)
  loop do
      neighbor ← a highest-valued successor of current
      if neighbor.value ≤ current.value then
          return current.state
      current ← neighbor
```

"Like climbing Everest in thick fog with amnesia"

- If no neighbors better, quit



- In order to find the global maximum, have random restarts
-

N Queens problem,

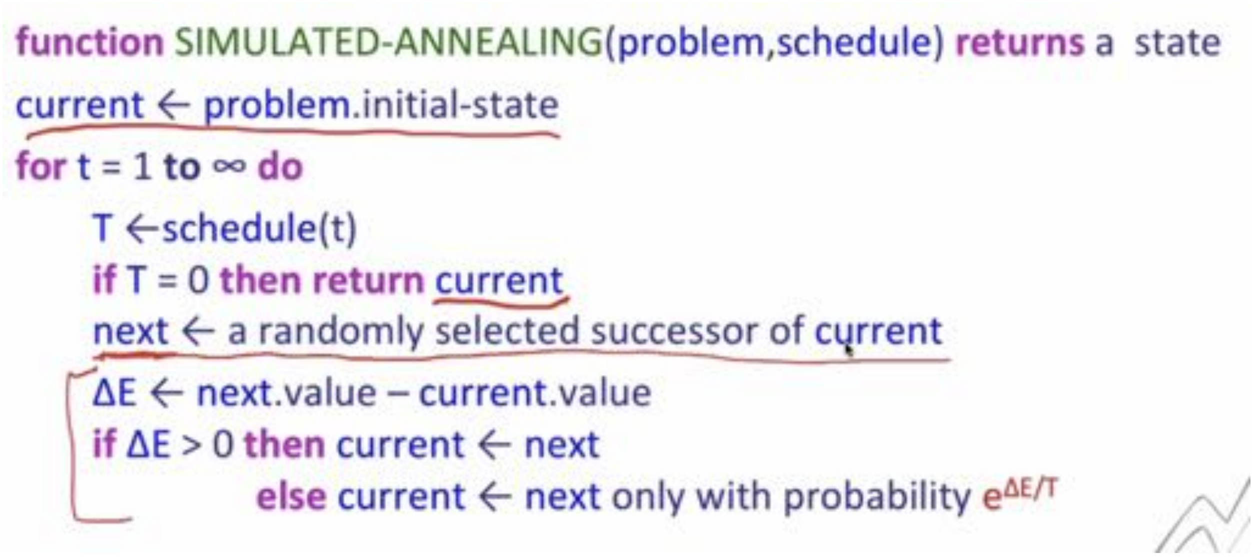
- heuristic value function: number of conflicts

Hill climbing on the 8 queens problem

- Algoirhtms with knobs to twiddle are irritating

Simulated annealing algorithm

- Have a temperature, if bad temperature, make it more variable



```
function SIMULATED-ANNEALING(problem,schedule) returns a state
current ← problem.initial-state
for t = 1 to ∞ do
    T ←schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.value – current.value
    if ΔE > 0 then current ← next
            else current ← next only with probability $e^{\Delta E/T}$
```

Week 3: Lecture 7 Adversarial Searches, Games: minimax (2/3)

A Brief History

- Checkers
    - 1950: First computer player
    - 1994: First computer world champion: Chinook defeats Tinsley
    - 2007: Checker's solved! Endgame database of 39 trillion states
- Chess
    - 1945-60: Zeus, Wiener, Shannon, Turning
    - 1997: Deep Blue defeats human champion Garry Kasparov
    - Expert level, better than humans
- Go
    - 1968: Zobrist's program plays legal Go
    - 2017: AlphaGo defeats human world champions
    - Expert level, better than humans

Types of Games

- Task environment with >1 agent
- Turn-taking or simultaneous, randomization, deterministic or stochastic
- Calculate a contingent plan recommending a move for every possible eventually

Game formulation

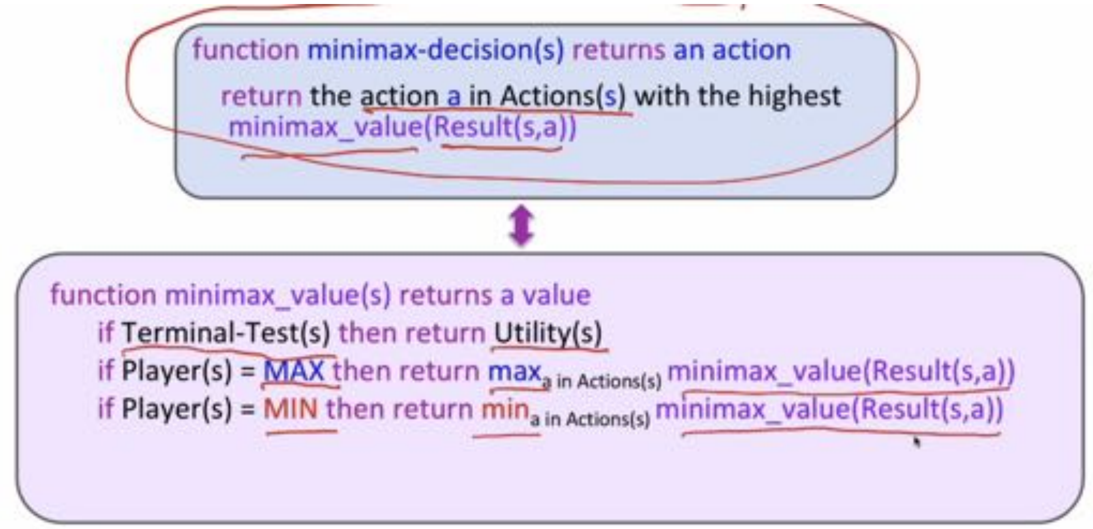
- Initial state: s0
- Players: Players indicates whose move it is
- Actions: Actions for player on move
- Transition model: Results(s, a)
- Terminal Test: Terminal Test
- Terminal Values Utility how good is the board for each player

Value of a State

- Value of a state is the best achievable outcome from that state

Minimax algorithm

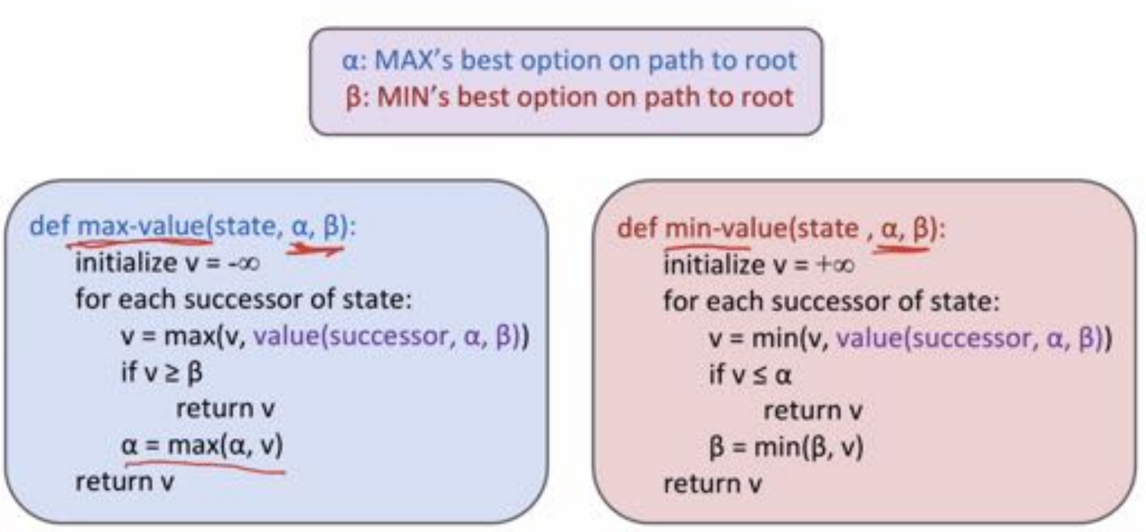
- Assume all future moves will be optimal
    - Rational against a rational player



```
function minimax-decision(s) returns an action
    return the action a in Actions(s) with the highest
    minimax_value(Result(s,a))
```

```
function minimax_value(s) returns a value
    if Terminal-Test(s) then return Utility(s)
    if Player(s) = MAX then return $\max_{a \text{ in Actions}(s)}$ minimax_value(Result(s,a))
    if Player(s) = MIN then return $\min_{a \text{ in Actions}(s)}$ minimax_value(Result(s,a))
```

-
- Exhaustive DFS
- Time: O(b^m)
- Space: O(bm)



α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, $\alpha$, $\beta$):
    initialize $v = -\infty$
    for each successor of state:
        $v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$
        if $v \geq \beta$
            return v
        $\alpha = \max(\alpha, v)$
    return v
```

```
def min-value(state , $\alpha$, $\beta$):
    initialize $v = +\infty$
    for each successor of state:
        $v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$
        if $v \leq \alpha$
            return v
        $\beta = \min(\beta, v)$
    return v
```

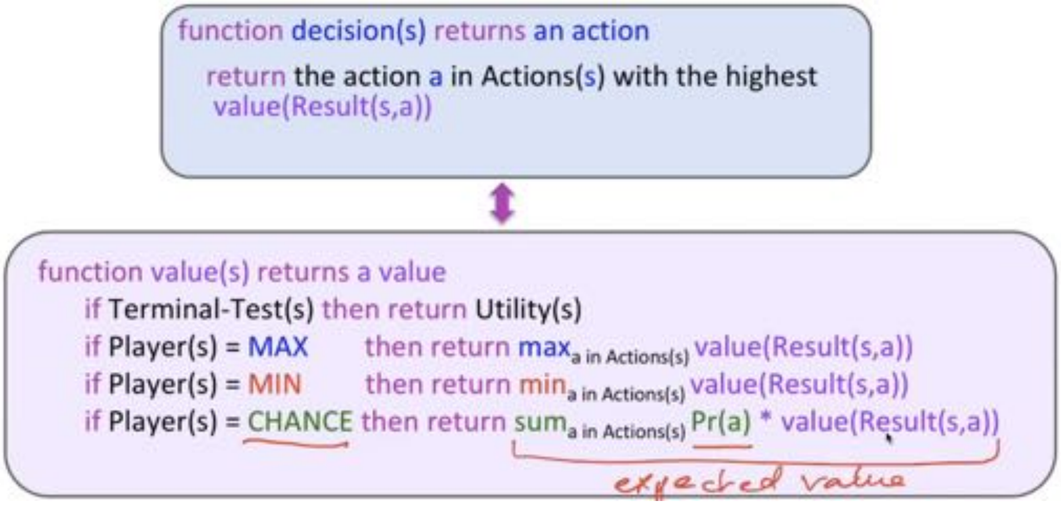
pruning of children of a minimizer node m is possible (for some assignment to the terminal nodes), when both of the following conditions are met: (i) the value of another child of m has already been determined, (ii) somewhere on the path from m to the root node, there is a maximizer node M for which an alternative option has already been explored. The pruning will then happen if any such alternative option for the maximizer had a higher value than the value of the "another child" of m for which the value was already determined.

Week 3: Lecture 8 Adversarial Searches II (2/5)

Resource Limits

- Bounded Lookahead (Shannon, 1950)
    - Search to preset depth limit
    - Need to have evaluation function at non leaf node
- Multiple player minimax
    - Terminals with utility tuples, node value whoever can choose

Games with uncertain outcomes

- Single player: Tetris, investing
- Multiple players: monopoly
- 

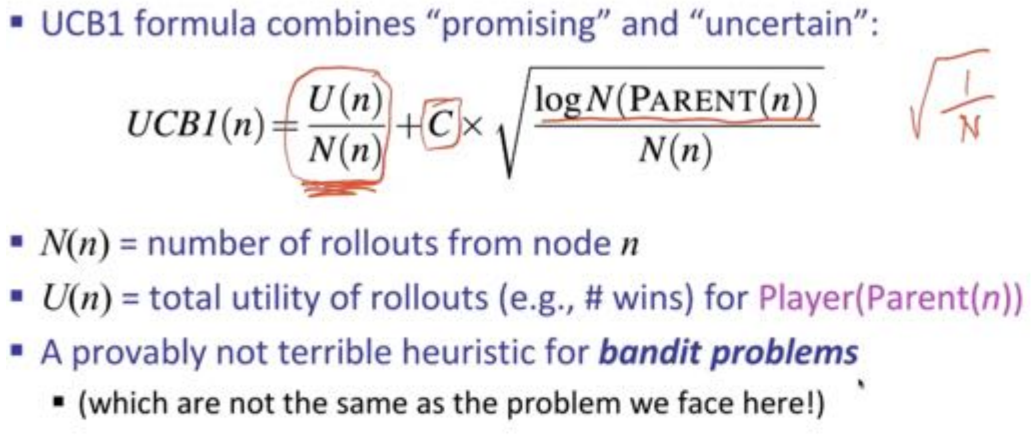
```
function decision(s) returns an action
    return the action a in Actions(s) with the highest
     value(Result(s,a))

function value(s) returns a value
    if Terminal-Test(s) then return Utility(s)
    if Player(s) = MAX      then return max_{a in Actions(s)} value(Result(s,a))
    if Player(s) = MIN      then return min_{a in Actions(s)} value(Result(s,a))
    if Player(s) = CHANCE then return sum_{a in Actions(s)} Pr(a) * value(Result(s,a))
                                                        expected value
```

- Minimax decision invariant on monotonic transformations on values
- Invariant with respect to positive affine transformations

Monte Carlo Tree Search

- Developed for GO,
- Evaluation by rollouts
    - Play multiple games to termination from a state s and count wins and losses
- Selective search
    - Explore parts o the tree that will help improve the decision at the root
- Rollouts
    - For each rollout, repeat until terminal, fraction of wins correlates with true value of the position
    - Do N rollouts from each child and pick the move that gives the best outcome by this metric
    - Allocate rollouts to more promising nodes

- ALlocate rollouts to more uncertain nodes



- UCB1 formula combines "promising" and "uncertain":

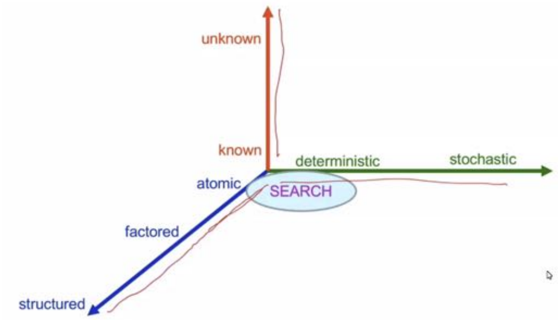
$$UCB1(n) = \boxed{\frac{U(n)}{N(n)}} + \boxed{C} \times \sqrt{\frac{\underline{\log N(\text{PARENT}(n))}}{N(n)}} \qquad \sqrt{\frac{1}{N}}$$

- $N(n)$ = number of rollouts from node n
- $U(n)$ = total utility of rollouts (e.g., # wins) for Player(Parent(n))
- A provably not terrible heuristic for ***bandit problems***
    - (which are not the same as the problem we face here!)

- UCT uses UCB and choose branches
- Value of a node U(n)/N(n) is a weighted sum of child values
- Rolllouts will typically be used on best children
- Games require decisions when optimality is impossible
    - Bounded depth search and approx evaluation functions

Hw: pruning

Week 4: Lecture 9 Logic 1: Introduction to Logic (2/8)



Agents that know things

- Knowledge of the effects of actions ("transitions model")
- Knowledge of how the world affects sensors ("sensor model")
- Knowledge of the current state of the world
- A single inference algorithm can answer any answerable question

Logic

- Syntax: what sentences are allowed?
- Semantics
    - What are the possible worlds, which sentences are true in which worlds

Relational databases

- Syntax: ground relational sentences
- Possible worlds: objects and relations

Propositional logic syntax

- Make sentences with conjunctions or logic
-

Week 4: Lecture 10 Logic 2: Introduction to Logic (2/10)

Partially Observable pacman

- Can see walls at different locations
- Doesn’t know where he is on the map

Pacman’s knowledge base: Sensor model

- State facts about how Pacman’s percepts arise
- Percept variable at t <-> some conditioni on world at t
- If there is a wall to the west at tiem t
- Blocked all around has wall

Pacman’s knowledge base: Transition model

- We care about location variables
- Make logic sentences, and send it to a sat solver

Reasoning tasks

- Localization with a map and local sensing
    - Given an initial KB, plus sequence of percepts and actions, where am I?
- Mapping with a location sensor
    - Given initial KB, plus sequence of percepts and actions, what is the map?
- Simultaneous localization and mapping
    - Where am I and what is the map?
- Planning
    - What action sequence is guaranteed to reach the goal?

Summary

- One possible agent architecture: knowledge + inference
- Logics provide a formal way to encode knowledge
    - syntax , set of possible worlds, truth condition
    - Initial state, sensor model and transition model
    - Logical inference computes entailment relations among sentences, enabling a wide range of task

Inference in Propositional logic

1. Model checking
    a. For every possible world, if alpha is true, make sure that beta is true too
2. Theorem proving
    a. Search for a sequence of proof steps leading from alpha to beta

Forward Chaining

- Applies modus ponens to generate nwe facts, applying rule until nothing more acan be added
- Given X1 and X2 ... Xn -> Y and X1, X2, Xn infer Y

Resolution

- Resolution inference rule takes two implication sentences and infers a new implication sentence

Satisfiability and entailment

- Whether sentence is satisfiable if it is true in at least one word

Week 4: Lecture 11 Logic 3 (2/10)

Conjunctive normal form (CNF)

- Each sentence can be expressed
- Each cause is a disjunction

Efficient SAT Solvers

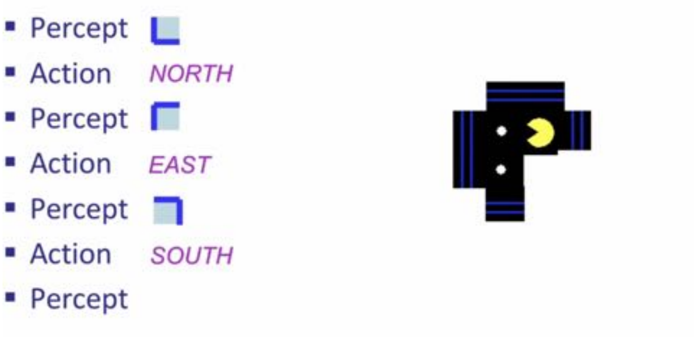
- DPLL core of modern solvers
- Recursive depth first search over models with some extras
    - Early termination: stop if
        - All clauses satisfied
        - Any clause is falsified
    - Pure literals
        - If all occurrences of a symbol in as yet unsatisfied caluses have the same sign
    - Unit clauses
        - If a clause is left with a single literal, set symbol is satisfy clause
- Smart variables and value ordering,
- Divide and conquer
- Caching unsolvable subcases as extra caluses to avoid redoing them
- SAT solvers in practice

Logical Agents

- Agent that operates in the pacworld
- State estimation means keeping track of what's true now
    - Ask whether actions percepts
- Localization demo
    - Percept, action, percept action
    - Can find out where you are

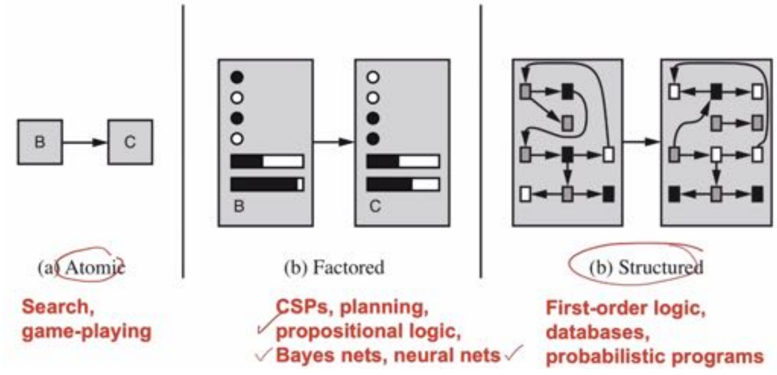
Mapping from a known relative location

- Knows where relative to start space
- Mapping



- Percept
- Action NORTH
- Percept
- Action EAST
- Percept
- Action SOUTH
- Percept

Week 5: Lecture 12 First Order Logic (2/17)



Expressive power

- Can write it out in logic of a problem

Possible Worlds

- Possible world for first order logic
    - Non-empty set of objects
    - For each k-ary predicate in the language, set of k-tuples of objects
    - For each k-ary function in the language, a mapping from k-tuples of objects to objects
    - For each constant symbol, a particular object
    - Infinitely many

Syntax and semantics: Atomic sentences

- An atomic sentence is an elementary proposition
    - Predicate symbol with terms as arguments
    - An equality between terms

Syntax and semantics: Complex sentences

- Sentences with logical connectives
- Sentences with universal or existential quantifiers
    - True in world w iff true in all extensions of w where x refers to an object in w

Inference in FOL

- Entailment is defined exactly as for propositional logic
- In FOL, we can go beyond just answering “yes” or “no” given an existentially quantified query, return a substitution for the variable such that the resulting sentence is entailed

Week 5: Lecture 13 Probability(2/19)

The real world is rife with uncertainty

- Probabilistic assertions summarize effects of ignorance and laziness

Basic Laws of probability

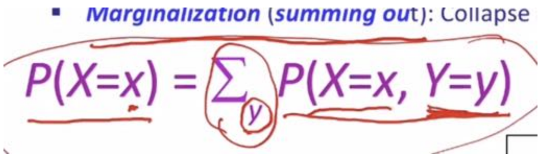
- Probability model assigns a number P(omega) to each world omega
- 0 <= P(omega) <= 1
- Sum P(omega) = 1

Probability of an event is the sum of probabilities over its worlds

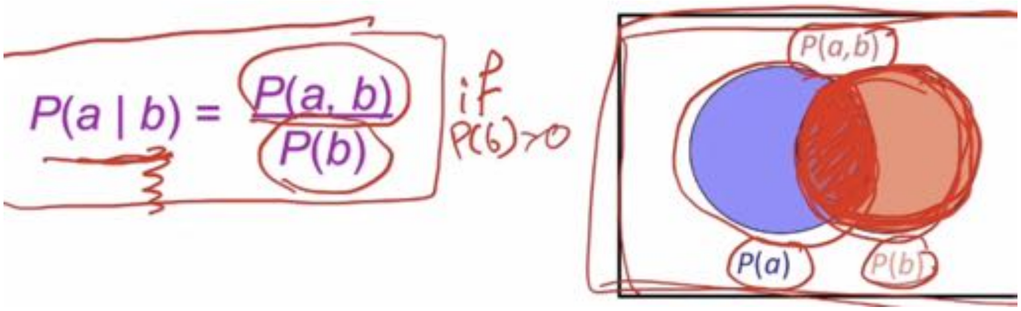
Random Variables

- Some aspect of the world about which we may be uncertain
- Formally a deterministic function of omega
- Range of a random variable is the set of possible values
- Probability distribution of a random variable X gives the probability for each value x in its range

Marginal Distributions

- 
  ▪ *Marginalization* (*summing out*): Collapse
  $$P(X=x) = \sum_{y} P(X=x, Y=y)$$
- Collapse a dimension by adding

Conditional probability

- 
  $$P(a \mid b) = \frac{P(a,b)}{P(b)} \quad \text{if } P(b) > 0$$

Normalizing a distribution

- To bring or restore to a normal condition
- Procedure:
    - Multiply each entry by alpha = 1/(sum of all entries)

Week 6: Lecture 14 Probability (2/22)

Inference by enumeration

- Evidence variables: E = e
- Query variables: Q
- Hidden variables: H

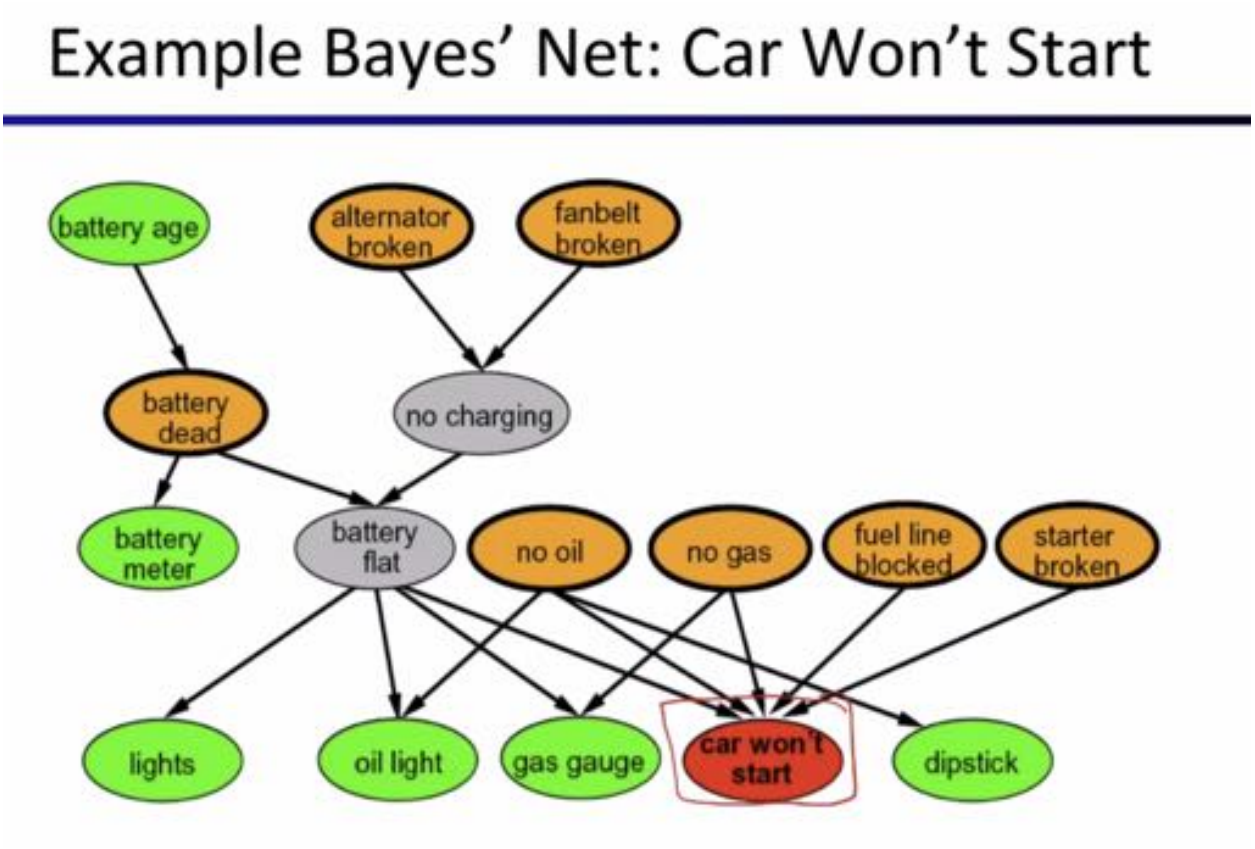
Ghostbusters model

- Uniform prior distribution over ghost location P(G)
- Sensor Model: P(Cxy | G) depends on distance to Gwhat is

Naive Bayes Model

- One discrete query variable
- All other potentially evidence variables
- Evidence vars conditionally independent given the query variable

Conditional Indpendence

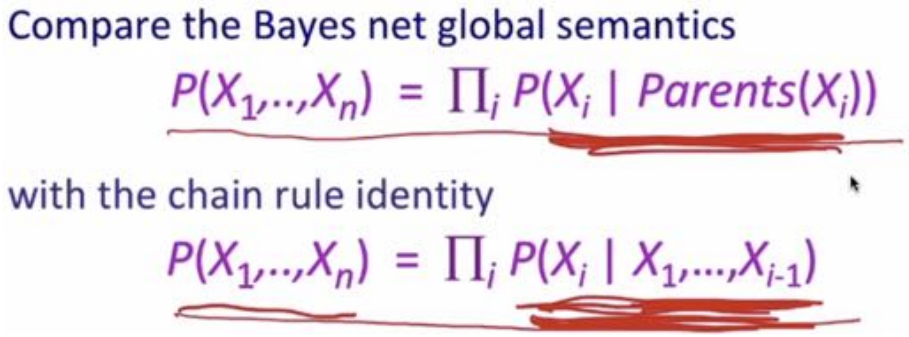


Example Bayes' Net: Car Won't Start

Week 6: Lecture 15 Probability (2/24)

Conditional independence in BNs

- Baynes net global semantics
- Chain rule identity



Compare the Bayes net global semantics

$$P(X_1,..,X_n) = \prod_i P(X_i \mid Parents(X_i))$$

with the chain rule identity

$$P(X_1,..,X_n) = \prod_i P(X_i \mid X_1,\ldots,X_{i-1})$$

Bayes Nets

- Inference: calculating some useful quantity from a probability model

Variable elimination

Operation 1: Pointwise product

- Pointwise product of factors

Operation 2: Summing out a variable

- Suming out
- Shrinks a factor to a smaller one

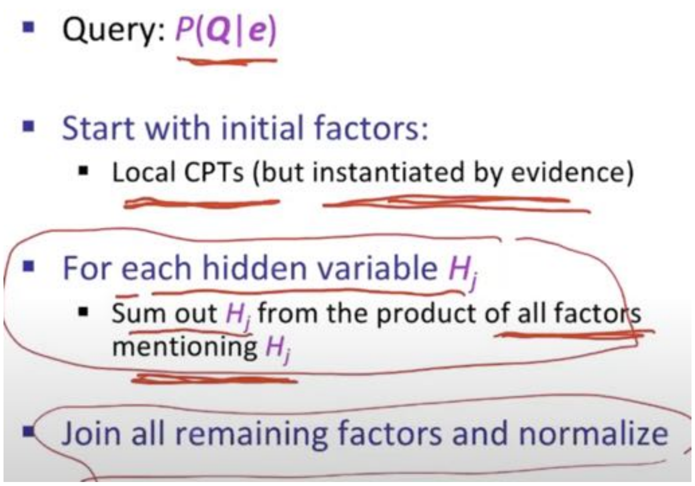
Week 6: Lecture 16 Probability (2/26)

Variable Elimination: Move summations inwards as far as possible

Operation 1: Pointwise product

- Pointwise product of factors (similar to database join, not matrix multiply)
- Increases: blows up
- P(A, J) * P(A, M) = P(A, J, M)

Operation 2: SUmming out a variable

- Summing out a variable from a factor
- Shrinks down
-  P(A, J) => P(A)

▪ Query: $P(Q|e)$

▪ Start with initial factors:
  ▪ Local CPTs (but instantiated by evidence)

▪ For each hidden variable H_j
  ▪ Sum out H_j from the product of all factors mentioning H_j

▪ Join all remaining factors and normalize

Polytrees

- Directed graph with no undirected cycles
- Linear in the network size if you eliminate the leaves towards the roots

Sampling basics: discrete categorical distributions

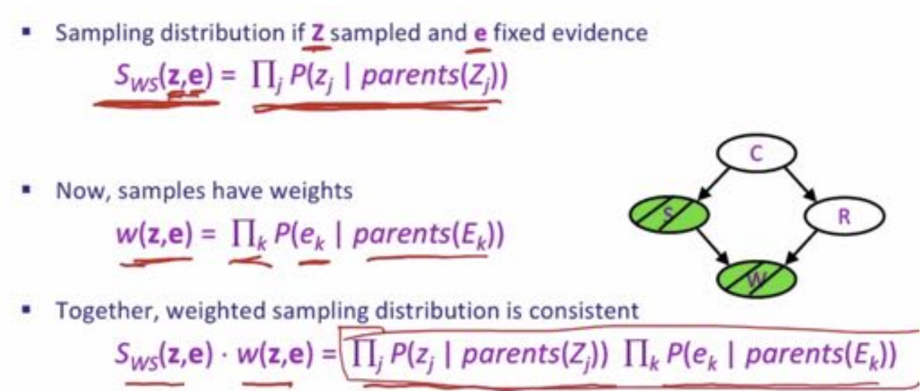
- Get a sample over uniform distribution
- Convert outcome into associating with subinverval

Week 7: Lecture 17 Sampling (3/1)

Rejection Sampling

- Application of prior sampling for estimating conditional probabilities
- Count the C outcomes for samples with r, w, and reject all other samples

Likelihood weighting



▪ Sampling distribution if **z** sampled and **e** fixed evidence

$$S_{WS}(\mathbf{z},\mathbf{e}) = \prod_j P(z_j \mid parents(Z_j))$$

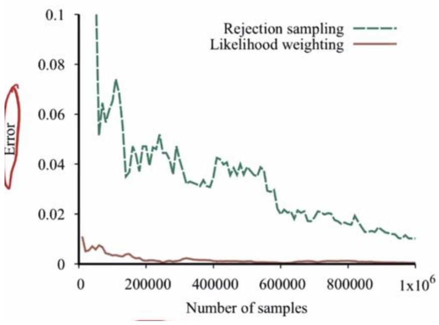
▪ Now, samples have weights

$$w(\mathbf{z},\mathbf{e}) = \prod_k P(e_k \mid parents(E_k))$$

▪ Together, weighted sampling distribution is consistent

$$S_{WS}(\mathbf{z},\mathbf{e}) \cdot w(\mathbf{z},\mathbf{e}) = \boxed{\prod_j P(z_j \mid parents(Z_j)) \; \prod_k P(e_k \mid parents(E_k))}$$

- Importance sampling, estimate some quantify based on p
- Likelihood is a lot better than rejection sampling
- Likelihood weighting is good
    - Values of downstream variables are influenced by upstream evidence



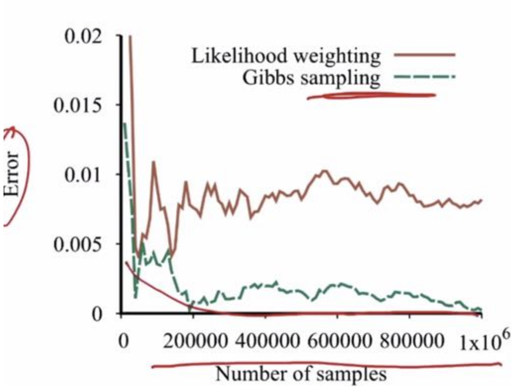
Markov Chain Monte Carlo

- MCMC is a family of randomized algorithms for approx some quantity of interest over a very large state space
- Markov chain = a sequence of randomly chosen states where each state is chosen conditioned on the previous state
- Monte Carlo = an algorithm that has some probability of producing an incorrect answer

MCMC = wander around for a bit, average what you see

Gibbs sampling

- States are complete assignments to all variables
- Evidence variables remain fixed, other variables change
- To generate next state, pick a variable and sample a value for it conditioned on all the other variables:
    - Tend to move towards states of higher probability, but can go down too
- Gibbs sampling is consistent
- Most widely used



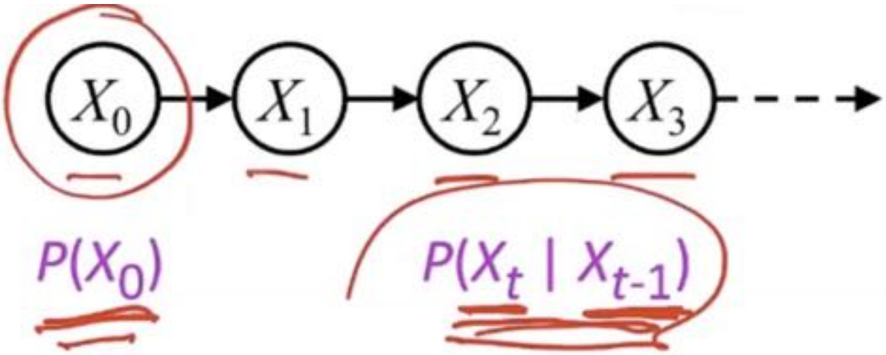
Week 7: Lecture 18 Markov Models (3/3)

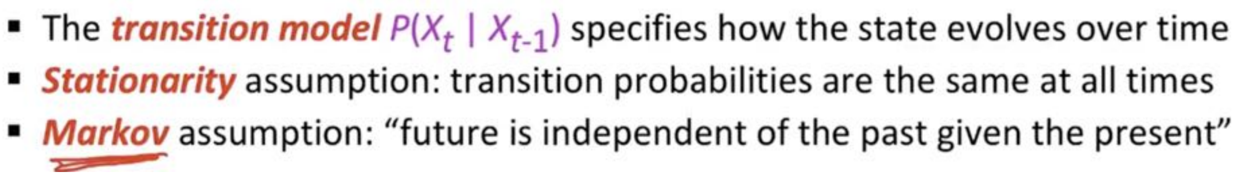
Reason about a sequence of observations

- Need to introduce time into our models

Markov Models (Markov chain/process)

- Value of X at a given time is called the state (usually discrete finite)





- The ***transition model*** $P(X_t \mid X_{t-1})$ specifies how the state evolves over time
- ***Stationarity*** assumption: transition probabilities are the same at all times
- ***Markov*** assumption: "future is independent of the past given the present"

Conditional independence assumption

Markov models a special case of Bayes net

- Directed acyclic graph, infinitely many variables,

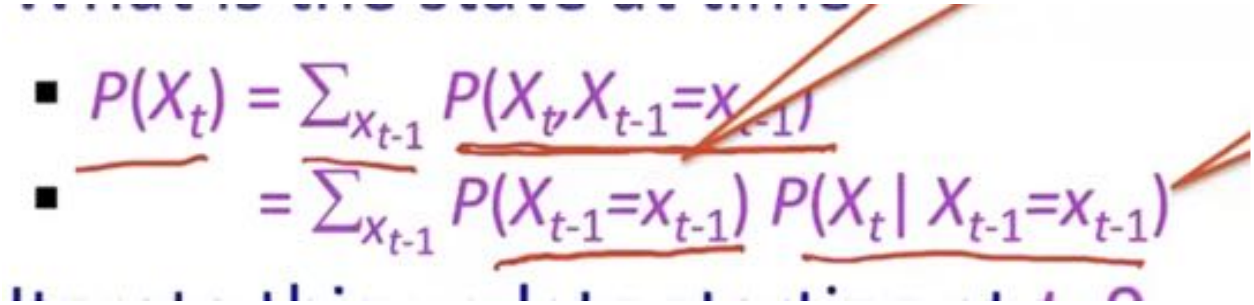
N-gram models

- State: word at position t in text (can build letter n-grams)
- Transition model
    - Unigram (zero-order): P(word_t = i)
        - Can tell language
    - Bigram(first-order): (Pword_t = i | Word_t-1 = j)
    - Trigram (second-order): P(Word_t = i | word_t - 1 = j, word_t-2 = k)
        - Sounds somewhat more like english
- Applications: author identification, spam

Example: web browsing

- State: URl visited at step t
- Transition model
    - With probability p, choose an outgoing link at random
- Application: google page rank

Forward algorithm: recursive network



- $\underline{P(X_t)} = \underline{\sum_{x_{t-1}}} \underline{P(X_t, X_{t-1}=x_{t-1})}$
- $= \sum_{x_{t-1}} \underline{P(X_{t-1}=x_{t-1})}\ \underline{P(X_t \mid X_{t-1}=x_{t-1})}$

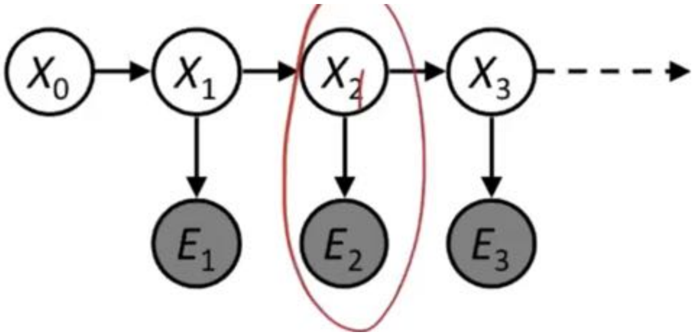
-

Limiting distribution is called the stationary distribution Pinfinity

<u>Hidden Markov Models</u>

Hidden Markov Models (HMMs)

- True state is not observed directly
- Underlying Markov chain over states X
- Observe evidence E at each time step
- X_t is a single discrete variable; E_t may be continuous and may consist of several variables
- Evidence is only depending variable

- 

$$P(X_0,X_1,\ldots,X_T,E_T) = P(X_0)\prod_{t=1:T} P(X_t \mid X_{t-1})\,\underline{P(E_t \mid X_t)}$$

Evidence variables are not independent from each other

- Speech recognition HMM,
    - Observations are acoustic signals
    - States Are specific positions in specific words

Inference tasks

- FIltering: P(X_t | e1:t)
    - Belief state -- input to the decision process of a rational agent
- Prediction: P(Xt+k | e_1:t) for k > 0
    - Evaluation of possible aaction sequences, filtering without the evidence
- Smoothing: P(X_k
    - Better estimate of past states,essentioal for leaning
- Most likely explanation: argmax
    - Speech recognition , deocidng with a noisy channel

Week 7: Lecture 19 Markov Models (3/5)

Filtering algorithm

Aim: devise a recursive filtering algorithm of form P(e|X)

- P(Xt+1 | e1: t+1) = g(e_{t+1} , P(X_t | e_{1:t})

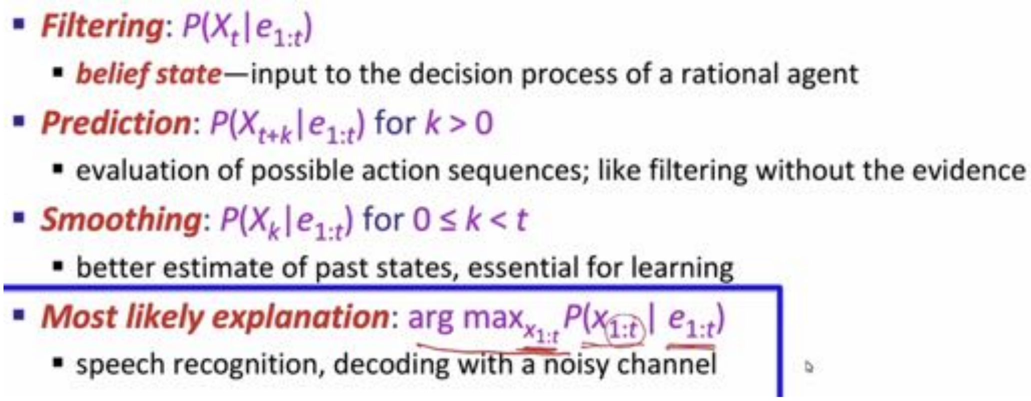
Transition matrix T, observation matrix Ot

Filtering algorithm becomes

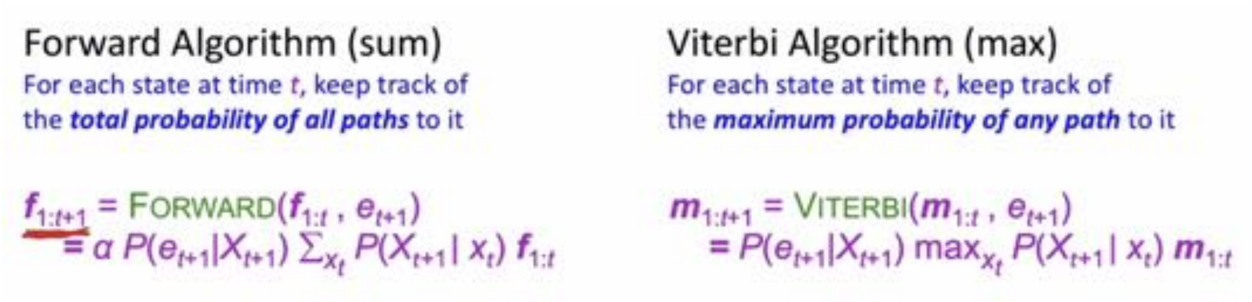
F_1:t+1 = alpha O_t+1 T^t f!:t

Predict -> update -> Normalize -> predict...

Most LIkely Explanation



- ***Filtering***: $P(X_t|e_{1:t})$
    - ***belief state***—input to the decision process of a rational agent
- ***Prediction***: $P(X_{t+k}|e_{1:t})$ for $k > 0$
    - evaluation of possible action sequences; like filtering without the evidence
- ***Smoothing***: $P(X_k|e_{1:t})$ for $0 \le k < t$
    - better estimate of past states, essential for learning
- ***Most likely explanation***: $\underline{\arg\max_{x_{1:t}}} P(x_{1:t} | \underline{e_{1:t}})$
    - speech recognition, decoding with a noisy channel



Forward Algorithm (sum)

For each state at time *t*, keep track of the ***total probability of all paths*** to it

$\mathbf{f}_{1:t+1} = \text{FORWARD}(\mathbf{f}_{1:t}, e_{t+1}) = \alpha P(e_{t+1}|X_{t+1}) \Sigma_{x_t} P(X_{t+1} | x_t) \mathbf{f}_{1:t}$

Viterbi Algorithm (max)

For each state at time *t*, keep track of the ***maximum probability of any path*** to it

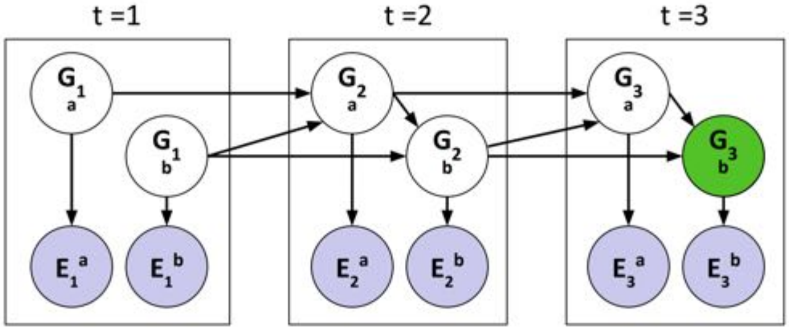
$\mathbf{m}_{1:t+1} = \text{VITERBI}(\mathbf{m}_{1:t}, e_{t+1}) = P(e_{t+1}|X_{t+1}) \max_{x_t} P(X_{t+1} | x_t) \mathbf{m}_{1:t}$

-

Week 8: Lecture 20 Dynamic Bayes Nets and Particle Filters (3/8)

Dynamic Bayes Nets (DBNs)

- Track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net structure at each time
- Variables at time t can have parents at time t-1



DBNs and HMMs

- Every HMM is a single var DBN

Exact Inference in DBNs

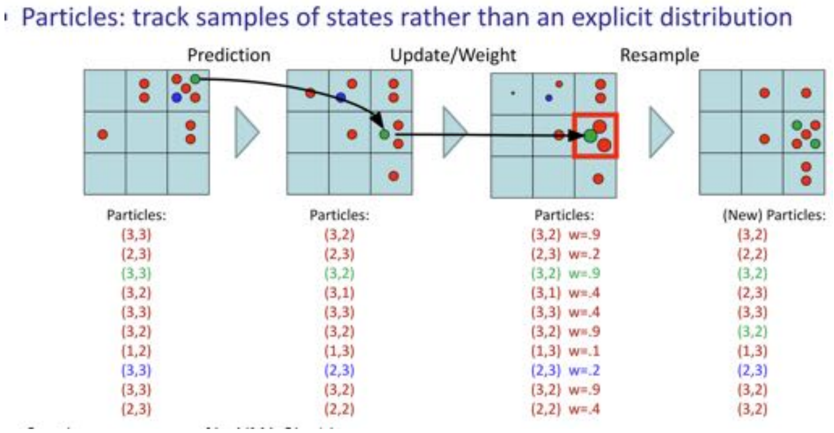
- Varirable elimination: “unroll” network for T time steps, elmiinate vars to find P(X | e)
- Online: eliminate all vars from previous time step
- Largest factor contains all vars for current time

Particle Filtering

- Particle FIltering
- Belief state by a set of samples
- Samples are particles, time per step is linear in the number of samples
- Number needed may be large

1. Prediction step
    - Particle j in state x_t samples a new state from transition model
        - $x_{t+1}^{(j)} P(X_{t+1} | x_t^{(j)})$
2. Update Step
    - After observing e_{t+1}
        - $w^{(j)} = P(e_{t+1} | x_{t+1}^{(j)})$
3. Resample
    a. N times, we choose from weighted sample distribution with replacement,

Simultaneous Localization and Mapping (SLAM)



- Particles: track samples of states rather than an explicit distribution

Particles:	Particles:	Particles:	(New) Particles:
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,3)	(2,3) w=.2	(2,2)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(3,2)	(3,1)	(3,1) w=.4	(2,3)
(3,3)	(3,3)	(3,3) w=.4	(3,3)
(3,2)	(3,2)	(3,2) w=.9	(3,2)
(1,2)	(1,3)	(1,3) w=.1	(1,3)
(3,3)	(2,3)	(2,3) w=.2	(2,3)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,2)	(2,2) w=.4	(3,2)

Week 8: Lecture 21 Rational Decisions (3/12)

Principle of Maximum Expected utility

- Rational agent should choose the action that maximizes its expected utility, given its knowledge

Wrost case minimax reasoning

- Better states to have higher evaluations, monotonic transformation
- Average case, we need magnitudes to be meaningful

Utilities

- Fuctions from outcomes (states of the world) to real numbers that describe an agent's preferences

Uncertain Outcomes

- Agent has preferences among prizes A, B, Lotteries: situations with uncertain prizes

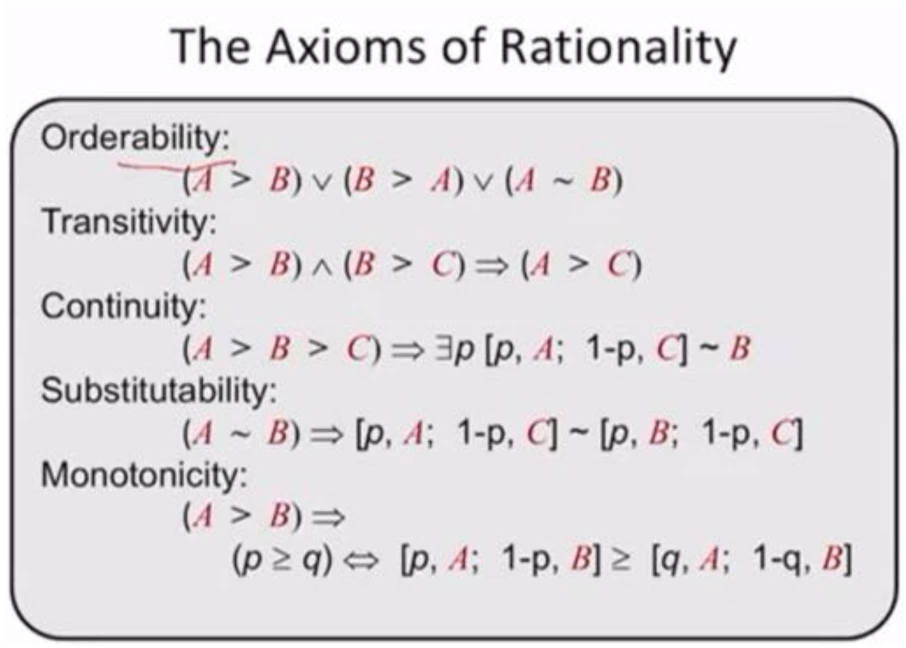
Notation

- Preference: A > B
- Indifference: A ~ B

Rationality

Rational Preferences

- Axiom of Transitivity: $(A > B) \wedge (B > C) \rightarrow (A > C)$



The Axioms of Rationality

Orderability:

$$(A > B) \vee (B > A) \vee (A \sim B)$$

Transitivity:

$$(A > B) \wedge (B > C) \Rightarrow (A > C)$$

Continuity:

$$(A > B > C) \Rightarrow \exists p\, [p, A;\ 1\text{-}p, C] \sim B$$

Substitutability:

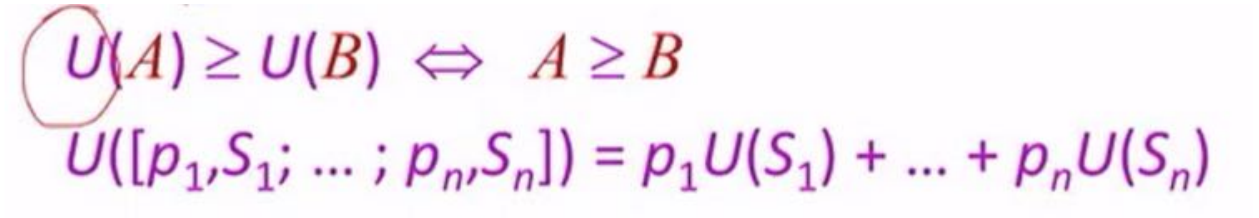
$$(A \sim B) \Rightarrow [p, A;\ 1\text{-}p, C] \sim [p, B;\ 1\text{-}p, C]$$

Monotonicity:

$$(A > B) \Rightarrow$$
$$(p \geq q) \Leftrightarrow [p, A;\ 1\text{-}p, B] \geq [q, A;\ 1\text{-}q, B]$$

Maximinzing Expected Utility

- Given any preferences satisfying constraints, there exists real valued function U st



$$U(A) \geq U(B) \Leftrightarrow A \geq B$$

$$U([p_1,S_1; \ldots ; p_n,S_n]) = p_1U(S_1) + \ldots + p_nU(S_n)$$

Optimal policy invariant under positive affine transformation $U' = aU + b,\ a > 0$

Human Utilities

- Compare prize A to standard lottery L_p between
  - Best possible prize and worst possible catastrophe

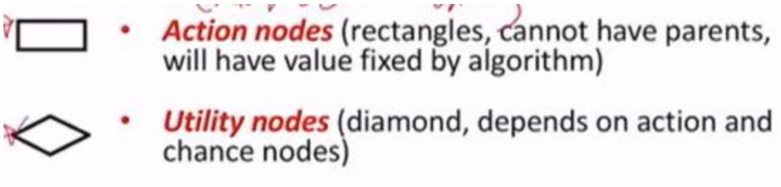
Money does not behave as a utility function, can talk about utility of having money

Post decision Disappointment: Optimizer's Curse

- Don't have access to exact utilities, only estimates
- Choose best one V* with high probability actual value is considerably less than V*

Week 9: Lecture 22 Decisions Networks and VPI (3/15)

Decision Networks

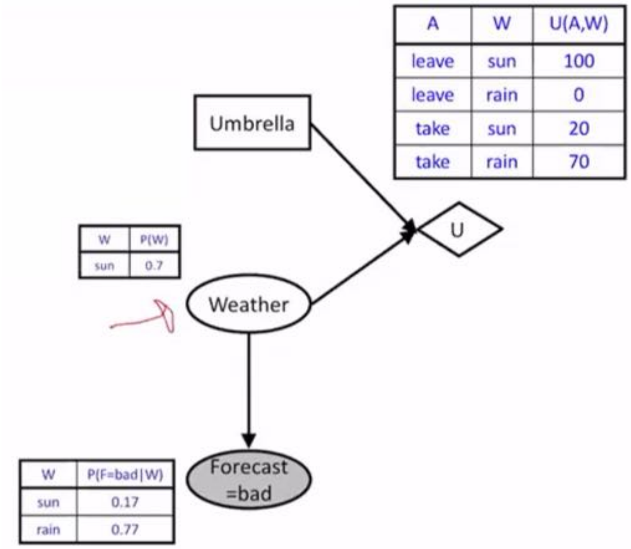
- Includes info about
  - Current state, possible actions, state that will result, utility of state
  - 

    - **Action nodes** (rectangles, cannot have parents, will have value fixed by algorithm)
    - **Utility nodes** (diamond, depends on action and chance nodes)

Decision Algorithm

- Fix evidence "e"
- For each possible action a
  - Fix action node to a
  - Compute posterior P(W | e, a) for parents of W or U
  - Compute expected utility

    $\sum_w P(w|e,a)\, U(a,\ w)$
- Return action with highest expected utility
- WIth action node as parents, it is sometimes called Q-value



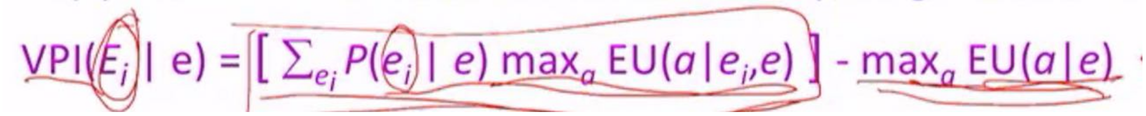
A	W	U(A,W)
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

W	P(W)
sun	0.7

W	P(F=bad|W)
sun	0.17
rain	0.77

Value of Information

Value of information = expected improvement in decision quality from observing value of a variable
Key Point: decision network contains everything needed to compute it



$$VPI(E_i \mid e) = \left[\sum_{e_i} P(e_i \mid e) \max_a EU(a \mid e_i, e) \right] - \max_a EU(a \mid e)$$

VPI properties

- VPI is non-negative!
- Not usually additive: $VPI(E_i, E_j \mid e) \neq VPI(E_i|e) + VPI(E_j|e)$
- Order independent: $VPI(E_i, E_j \mid e) = VPI(E_j, E_i \mid e)$

Week 9: Lecture 23 Markov Decision Process (3/17)

Decision network = Bayes Net + Actions + utilities
Decisions with unknown preferences

- Machine optimizing the wrong preferences causes problems

If robot lets human decide human will only allow robot to act if utility > 0
Sequential decisions under uncertainty

- Decision problem is one shot -- concerning one action

Markov Decision process (MDP)

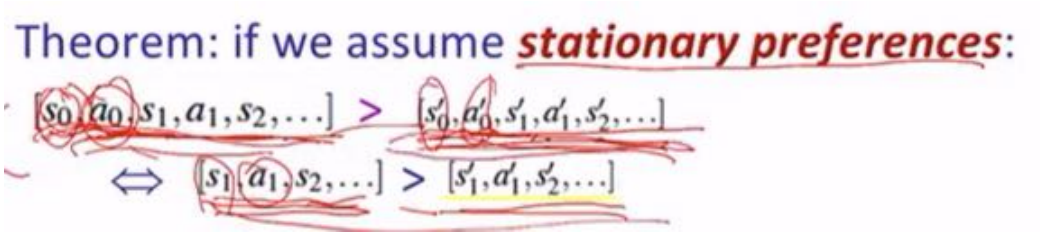
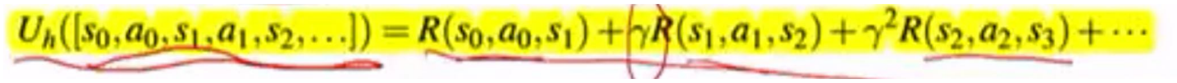
- ENvironment history, list of state transitions and actions
- “Markov” generally means given the present state, future, and past are independent
- “Markov” in MDP means actions outcomes depend only on the current state
- MDPs are fully observable but probabilistic search problems
- MDP defined by
    - Set of states s in S
    - Set of actions a in A
    - Transition model T(s, a, s′)
    - Reward function R(s, a, s′) for each transition
    - Start state
    - Terminal state or absorbing state
    - Utility function which is additive

Policies

- Policy π gives an action for each state S -> A
- We want an optimal policy π* S-> A
    - Maximizes expected utlity
    - Explicit policy defines a reflex agent

Utilities of Sequences

- What preferences should an agent have over reward sequences?
- More or less?

- Stationary Preferences: only one way to define utilities
    - 
      Theorem: if we assume ***stationary preferences***:
      $[s_0,a_0,s_1,a_1,s_2,\ldots] > [s'_0,a'_0,s'_1,a'_1,s'_2,\ldots]$
      $\Leftrightarrow [s_1,a_1,s_2,\ldots] > [s'_1,a'_1,s'_2,\ldots]$
    - Additive discounted utility
    - 
      $U_h([s_0,a_0,s_1,a_1,s_2,\ldots]) = R(s_0,a_0,s_1) + \gamma R(s_1,a_1,s_2) + \gamma^2 R(s_2,a_2,s_3) + \cdots$
    - $\gamma \in [0, 1]$ is the discount factor
    - Worth r now, yr next step, y^2r in two steps
    - $r_0 + \gamma r_1 + \gamma^2 r_2 +$ ... is bounded by $\pm R_{max}/(1 - \gamma)$
    -

Week 9: Lecture 24 MDP II: Value/Policy Iteration (3/19)

MDP

- Utility function is additive discounted rewards

The utility of a policy

- Executing policy πfrom any state $s_0$generates sequence
    - s_{0}., $\pi(s_9)$, s_1, $\pi(s_1)$
    - Sequence: $R(s_9, \pi(s_0), s_1)$, R...
    - $P(s_1 | s_0, \pi(s_0)) * P(s_2 | s_1, \pi(s_1)) *$
- Value (expected utility of πin $s_0$written as $U^{\pi}(s_0)$
    - Sum over all possible state sequences
- Optimal Quantities
    - Optimal policy
    - $\pi^*(s)$ = optimal action from state s
    - Gives highest $U^{\pi}(s)$ for any π
- Value (utility) of a state s:
    - U*(s) = U^{pi*} (s)
- Value (utility) of a q state (s, a)
    - Q*(s, a) = expected utility of taking action a in state s and thereafter acting optimally
    - U*(s) = max a Q*(s, a)

Bellman equations

- Value/utility of a state is

    - Expected reward for the next transition plus the discounted value/utility of the next state, assuming the agent chooses the optimal state

$$\mathrm{U(s)} = \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma U(s')]$$

-
- Bellman equations fo rQ functions

$$\begin{aligned} Q(s,a) &= \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma \boxed{U(s')}] \\ &= \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma \boxed{\max_{a'} Q(s',a')}] \end{aligned}$$

-

Solving MDPs

- Value Iteration
- Politcy iteration

Value iteration

- Value Iteration
- U_0 (s) = 0 and some termination paratemter ϵ
- Repeat until convergence (all updates smaller than ϵ
    - Do a bellman update (one ply of expectimax) from each state
    - Compute next utility function

Contraction

- Operator F is a contraction by a factor c, it brings any pair of objects closer to each other (according to some metric d(,))
- For any x, y d(E_x, F_y) \leq c, d(x, y) where c < 1
- Value iteration is just $U_{k+1} \leftarrow BU_k$
- The Bellman update B is a contraction by γ
    - Metric is max norm: $|| V - W || = max_s |V(s) - W(s)|$

Week 11: Lecture 25 MDP III: Value/Policy Iteration 2 (3/29)

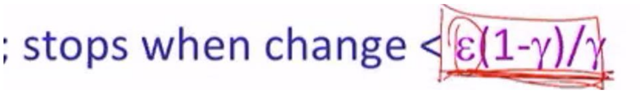
How fast does Value iteration converge

- Distance between $U_k$and $U*$

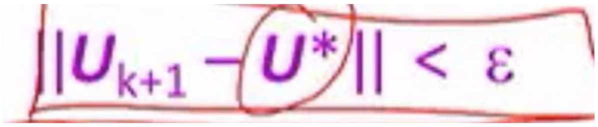
$$\boxed{\le \boxed{\gamma} \| U_k - U^* \|}$$

-

How do we know answer is (nearly) right

- 

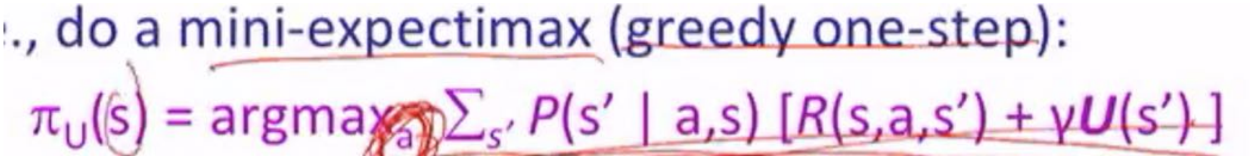
; stops when change < $\varepsilon(1-\gamma)/\gamma$



$||U_{k+1} - U^*|| < \varepsilon$

How should agent act given U(s)

- Max expected utility
- Policy extraction, finds policy implied by U

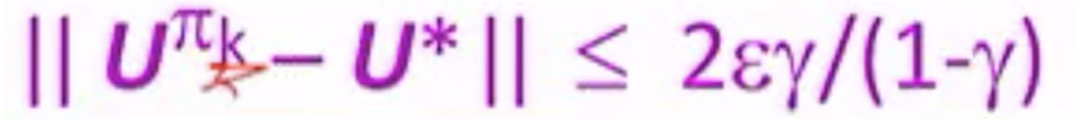


., do a mini-expectimax (greedy one-step):

$\pi_U(s) = \mathrm{argmax}_a \sum_{s'} P(s' \mid a,s)\,[R(s,a,s') + \gamma U(s')]$

Quality of a policy measured by policy loss $||U^{\pi} - U^{*}||$

Policy loss is bounded



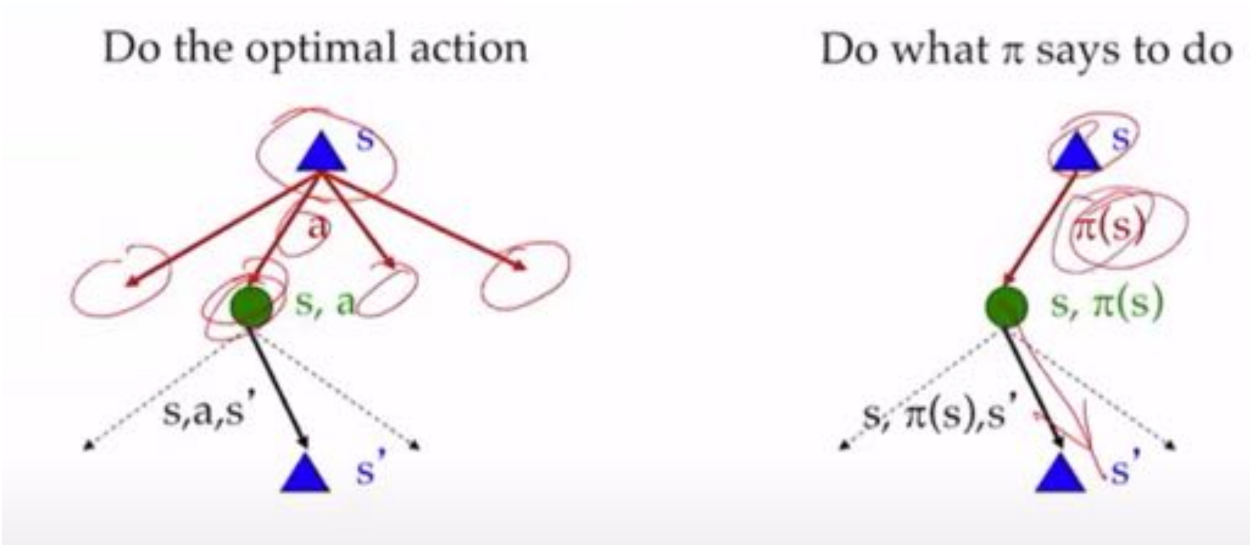
$|| U^{\pi_k} - U^* || \leq 2\varepsilon\gamma/(1-\gamma)$

Problems with Value iteration

- Value iteration repeats the Bellman updates
- Problem 1: slow: $O(S^2A)$ per iteration
- Problem 2: max at each state rarely changes
- Problem 3: policy often converges long before the values

Policy Iteration

- Basic idea: Make the implied policy in U explicit, compute long-term implications for value
- Repeat until no change in policy
    - Policy evaluation: calculate value $U^{\pi k}$ for current policy π_k
    - Policy improvements: extract new implied policy $\pi_{k+1}$from $U^{\pi k}$
- Still optimal
- Can converge much faster under some conditions

- 

Policy Evaluation

- Idea 1: Turn recursive Bellman equations into updates
    - $U_0^{\pi}(s) = 0$
    - Efficiency $O(S^2)$

$$U_{i+1}(s) \leftarrow \sum_{s'} P(s' \mid s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')]$$

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- Value iteration
    - Every iteration updates both the values and the policy
    - We don't track the policy, but taking hte max over actiosn implicitly recomputes
- Policy iteration
    - Passes to update utilities with fixed policy, only consider one action
    - After policy evaluation, new policy is chosen
    - New policy will be better
- Computer optimal values: value iteration or policy iteration
- Compute values for a particular policy: use policy evaluation
- Turn your values into a policy: use policy extraction
- Variaations of Bellman updates
- Use one step lookahead expectimax fragments
- Differ in fixed policy or max over actions

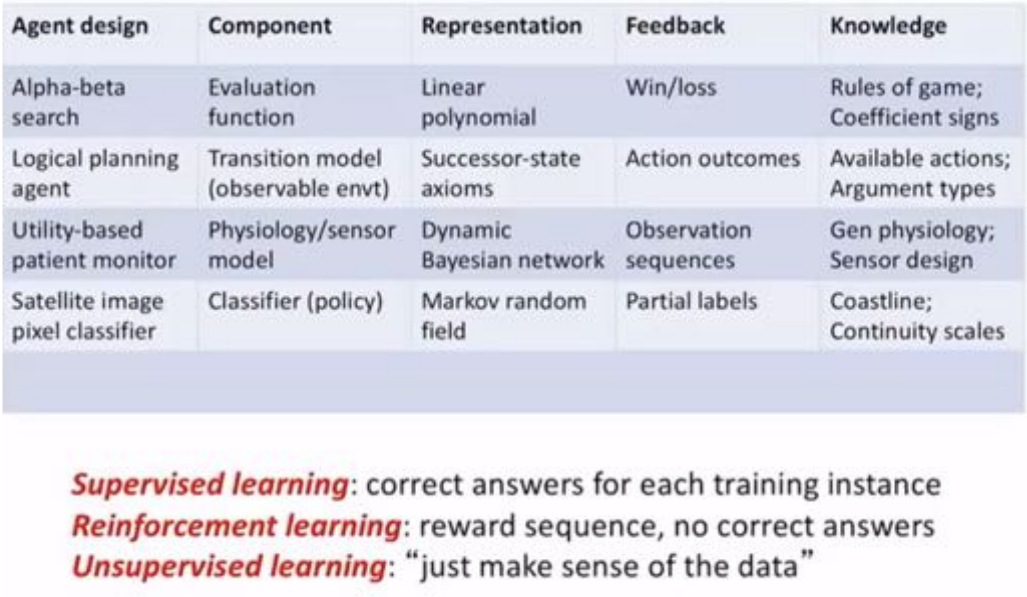
Week 11: Lecture 26 Learning 1: Decision Trees (3/31)

Learning: a process for improving hte performance of an agent through experience

- Supervised learning: classification and regression
- Learning is essential in unknown environments -- when the agent designer lacks omniscience
- Learning is useful as a system construction method, expose the system to reality rather than trying to write it down

Key Questions

- What is the agent design that will implement the desired performance?
- Improve the performance of what piece of the agent system and how is that piece represented?
- What data are available relevant to that piece?
- What knowledge available?



Agent design	Component	Representation	Feedback	Knowledge
Alpha-beta search	Evaluation function	Linear polynomial	Win/loss	Rules of game; Coefficient signs
Logical planning agent	Transition model (observable envt)	Successor-state axioms	Action outcomes	Available actions; Argument types
Utility-based patient monitor	Physiology/sensor model	Dynamic Bayesian network	Observation sequences	Gen physiology; Sensor design
Satellite image pixel classifier	Classifier (policy)	Markov random field	Partial labels	Coastline; Continuity scales

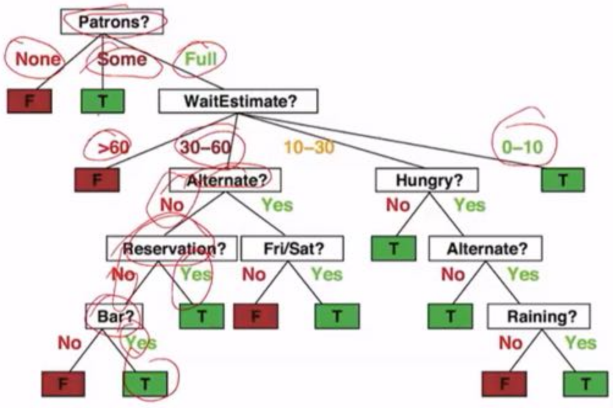
Supervised learning: correct answers for each training instance
Reinforcement learning: reward sequence, no correct answers
Unsupervised learning: "just make sense of the data"

Supervised learning

- To learn an unknown target function f
- Input: training set of labeled examples (xj, yj) where yj= f(xj)
- Output: hypothesis h that is "close" to f, predicts well on unseen examples ("test set")
- Many possible hypothesis families for h
    - Linear models, logistic regression, neural networks, decision trees, examples,
- Classification = learning f with discrete output value
- Regression = learning f with real-valued output value

Consistency vs Simplicity

- Fundamental tradeoff: bias vs. variance
- Usually algorithms prefer consistency by default
- Reduce hypothesis space
    - Fewer better features/ attributes
- Regularization
    - Smoothing, generalization parameters



Decision Trees

- Popular representations for classification
- Can represent any function of the input

Hypothesis Spaces in general

- Increasing the experessiveness of the hypothesis language
- With 2^{2^n} nypothesse, all but exponentially small fraction will require bits to express in any representation

Week 11: Lecture 27 Learning 2: Decision Trees (4/2)

Learning a Decision Tree

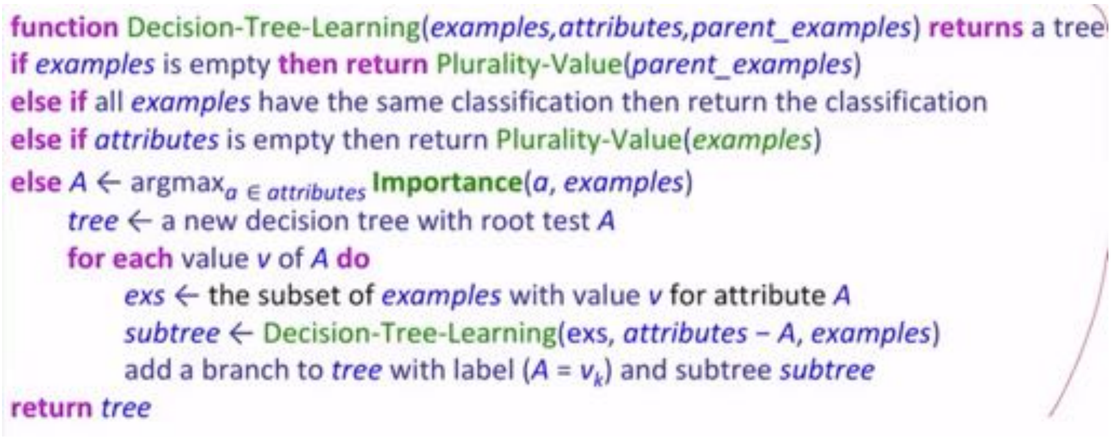
- Goal:
    - Given training data of a list of attributes & label
    - Hypothesis family H: decision trees
    - Find (approx) smallest decision tree that fits the training data
- Iterative process to choose the next attribute to split on

Information

- Info answers questions, the more clueless I am initially, the more information is contained in the answer
- Coin probability p heads, entropy
    - H(<p, 1-p>) = -plogp - (1 -p) log(1-p)
- Information in an answer when prior is $< p_1,..., p_n >$
    - Entropy of the prior: $H(< p_1,..., p_n >) = \sum_i - p_i \log p_i$
- p positive and n negative examples at the root
- An attribute splits the examples E into subsets $E_k$each of which needs less info to complete the classification
- Expected number of bits needed after split is $\sum_k (p_k + n_k)/(p + n)B(p_k/p_n + n_k))$

Information gain

- 1 - (sum expected bits)

- 

```
function Decision-Tree-Learning(examples,attributes,parent_examples) returns a tree
if examples is empty then return Plurality-Value(parent_examples)
else if all examples have the same classification then return the classification
else if attributes is empty then return Plurality-Value(examples)
else $A \leftarrow \text{argmax}_{a \in attributes}$ Importance(a, examples)
    tree ← a new decision tree with root test A
    for each value v of A do
        exs ← the subset of examples with value v for attribute A
        subtree ← Decision-Tree-Learning(exs, attributes − A, examples)
        add a branch to tree with label ($A = v_k$) and subtree subtree
return tree
```

Important points about learning

- Data: labeled instances
    - Training data
    - Held out data
    - Test data
- Features: attribute-value pairs
- Experienctation cycle
    - Learn parameters on training set
    - Tune hyperparatmeters on held out set
    - Compute accuracy of test set
    - Never peek at the test set
- Evaluation:
    - Accuracy : fraction of instances predicted correctly
- Overfitting and gernalization
    - Want a classifier which does well on test data
    - Overfitting: fitting the training data very closely, but not generalizing well
    - Underfitting: fits the training set poorly

Week 12: Lecture 28 Learning 3: Linear Regression & Perception (4/4)

Entropy:

- Measure uncertainty in a probability distribution

Supervised Learning

- Classification = learning f with discrete output values
- Regression = learning f with real-valued output value

Linear Regression

- Least error or residual
- Actual - prediction

Loss function

- w* to minimize loss function

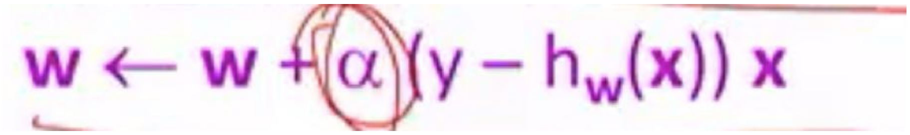
- Find the partial of loss with parameters and set to 0 to find optimal

Threshold perceptron as linear classifier

- Single unit that outputs y = 1 when x >= 0 and =0 when w.x < 0
- Multidimensional gieves hyper plane

Weight Updates

Perceptron learning rule

- If y ≠h(x)
- False negative
    - <0 but y = 1
    - Increase weights on positive inputs
    - Deceraes weights on negative inputs
- False positive
    - W.x > 0 but output should be y = 0
    - Decrease weights on positive inputs
    - Increase weights on negative inputs
- Perceptron learning rule
- 

  $\mathbf{w} \leftarrow \mathbf{w} + \alpha (y - h_{\mathbf{w}}(\mathbf{x}))\, \mathbf{x}$

Perceptron convergence theorem

- Linearly separable iff there is some hyperplane exactly separating ostivie from negative examples
- Convergence: if training data are sepratebl, perception learning applied regularly

Perceptron happy as loing as it separates the training data

- Sigmoud $g(x) \ = \ \frac{1}{1+e^{-x}}$
- Logistic regression

Week 12: Lecture 29 Learning 4: Statistical learning & Naive Bayes (4/9)

L2 loss for linear regression

Perceptions hopeless for XOR

Basic Questions

- Which hypothesis space H to choose
    - Probability model P(y, x, h)
- How to measure degree of fit
    - Data likelihood $\pi_j P(y_j|x_j,\ h)$
- How to rade off degree of fit vs. complexity
    - Regularization or prior arg max P(h) pi P(y | xj, h)

Bayesian: computing posterior over H

- Hypothesis space H to choose

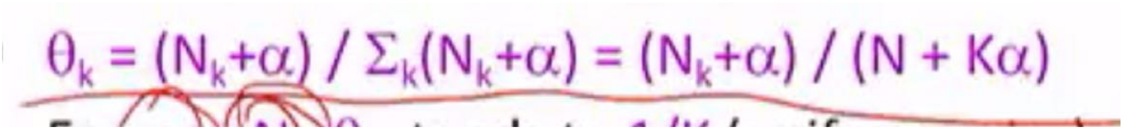
    - All hypotheses with nonzero a priori probability
- How to measure degree of fit
    - Data probability, as for MLE/MAP
- How to trade off degree of fit vs complexity
    - Use prior, as for MAP
- How to find a good h
    - Bayes predictor

Parameter Estimation

Maximum Likelihood parameter estimation

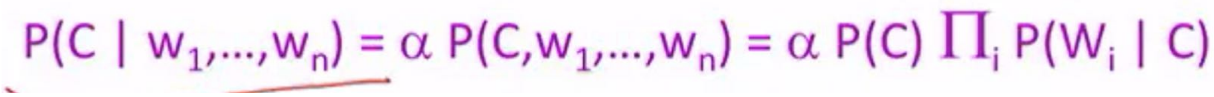
- Estimating distribution of a random variable
- Evidence x = x_1 , x_n
- Likelihood: probability of the evidence P(x_1 ... x_n)
- Log likelihood: L(x; θ) = log P(x, θ)
- Take derivative to find max loglikelihood

Laplace Smoothing

- Laplace smoothing strength α
- 

$$\theta_k = (N_k+\alpha) / \Sigma_k(N_k+\alpha) = (N_k+\alpha) / (N + K\alpha)$$

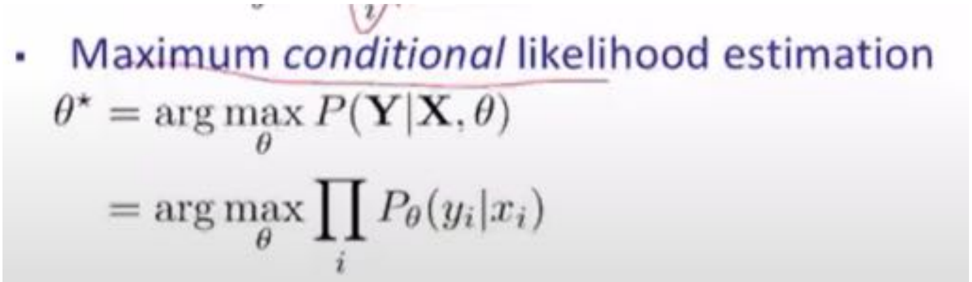
Probabilistic Classification

- Bayes net model for ham/spam
- Class C of document with prior P(C)
- Bag of words model: word W_i in the document generated independently from a class-specific distribution P(W|C) over words
- Naive bayes model
- For posterior distribution for class C
    - 

$$P(C \mid w_1,\ldots,w_n) = \alpha\, P(C,w_1,\ldots,w_n) = \alpha\, P(C) \prod_i P(W_i \mid C)$$

- Estimate prior over classes, parameter learning for naive bayes nets

Week 13: Lecture 30 Neural Nets (4/12)

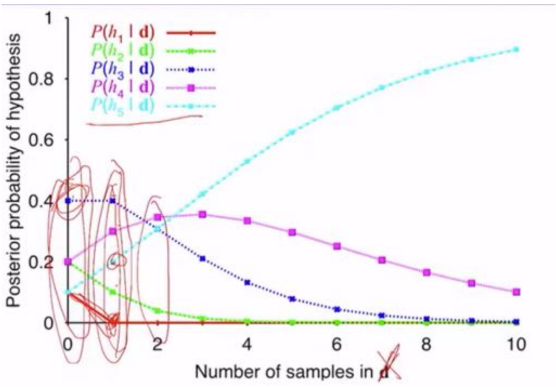


- Maximum *conditional* likelihood estimation

$$\theta^\star = \arg\max_\theta P(\mathbf{Y}|\mathbf{X},\theta)$$
$$= \arg\max_\theta \prod_i P_\theta(y_i|x_i)$$

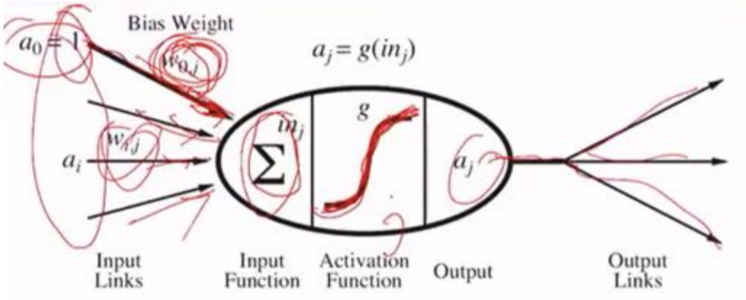
Bayesian Learning

- Learning = Bayesian updating of a probability distirbution over H
- Prior P(H) training data
- Predictions use a likelihood-weighted average over the hypotheses
- $$P(\mathbf{x}_{N+1}|\mathbf{X}) = \Sigma_k P(\mathbf{x}_{N+1}|\mathbf{X},h_k)P(h_k|\mathbf{X}) = \Sigma_k P(\mathbf{x}_{N+1}|h_k)P(h_k|\mathbf{X})$$
- Posterior probability of hypotheses changes

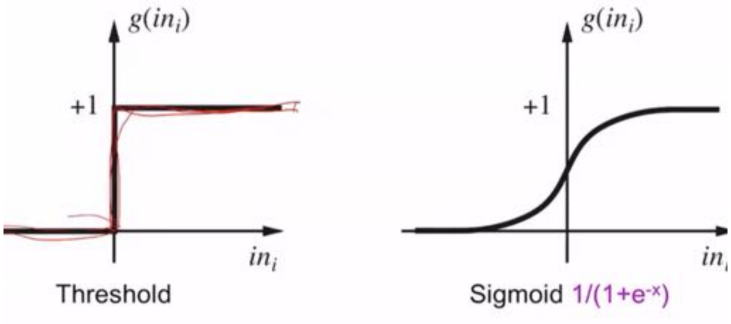


<u>Deep Learning/Neural Network</u>

Loose inspiration: human neurons



Activation functions g

-


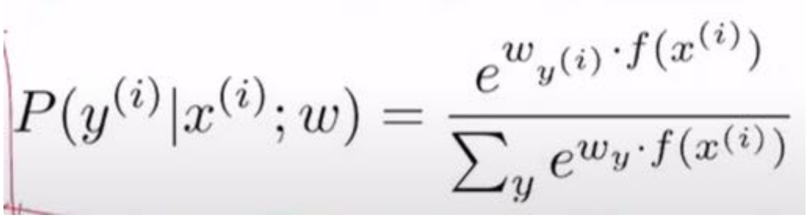
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation
- $$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$
- If activation positive, +1, negative -1

Best w?

- Max likelihood estimation
  - $$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Multiclass logistic regression

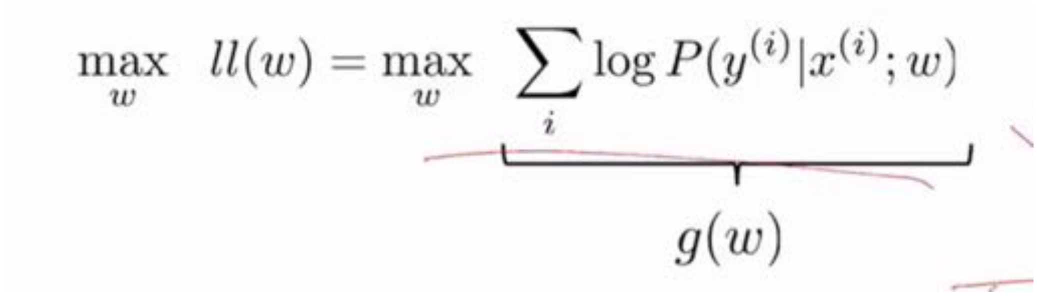
- Weight vector for each class: w_y
- Score (activation) of a class y: $w_y * f(x)$
- Prediction w highest score wins: $y = \arg\max_y w_y * f(x)$
- Max likelihood with p
- 

$$P(y^{(i)}|x^{(i)};w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

Optimization

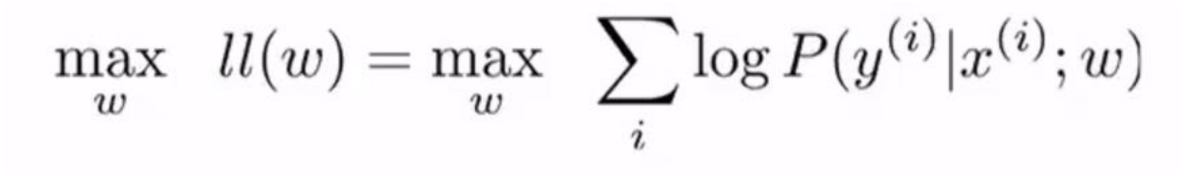
- How to solve equation getting closest
- Hill Climbing
    - CSP
    - Start from wherever
- Gradient ascent : steepest direction
    - Update in uphill direction
    - Sleeper the slope, the bigger the step

Batch Gradient Ascent on the Log Likelihood Objective



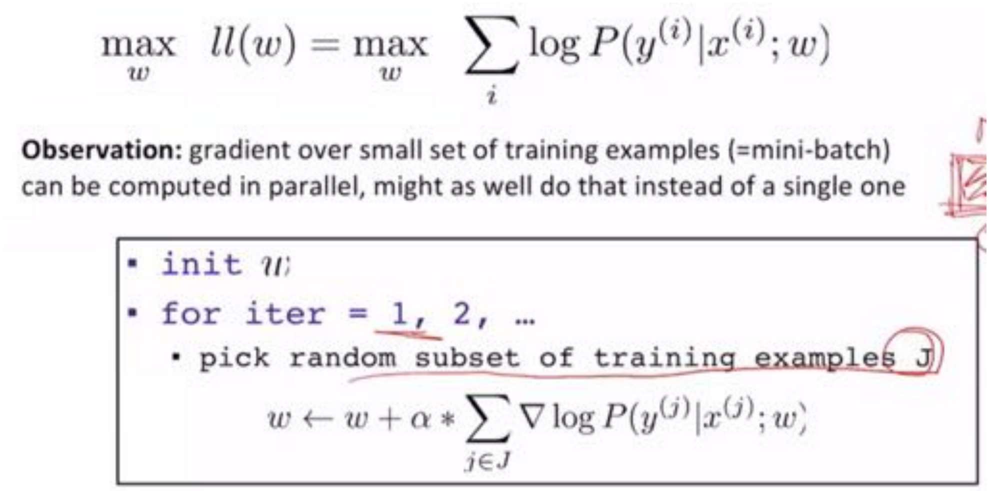
$$\max_w \; ll(w) = \max_w \; \underbrace{\sum_i \log P(y^{(i)}|x^{(i)};w)}_{g(w)}$$

Stochastic Gradient Ascent on Log Likelihood Objective



$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)};w)$$

Mini Batch Graident Ascent

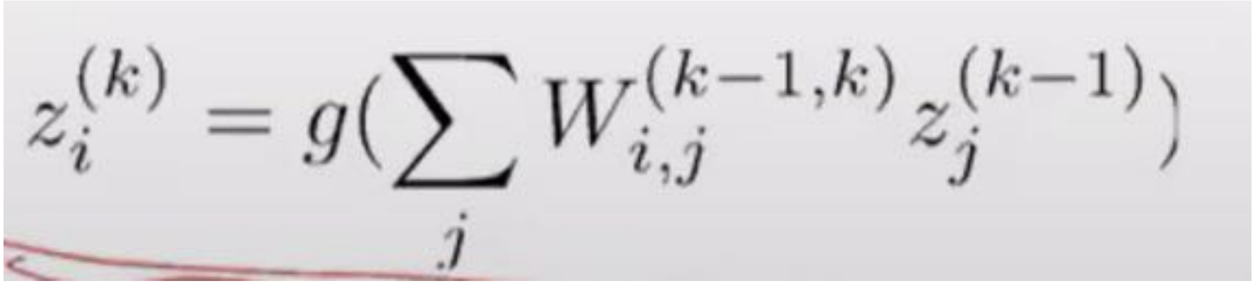


$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

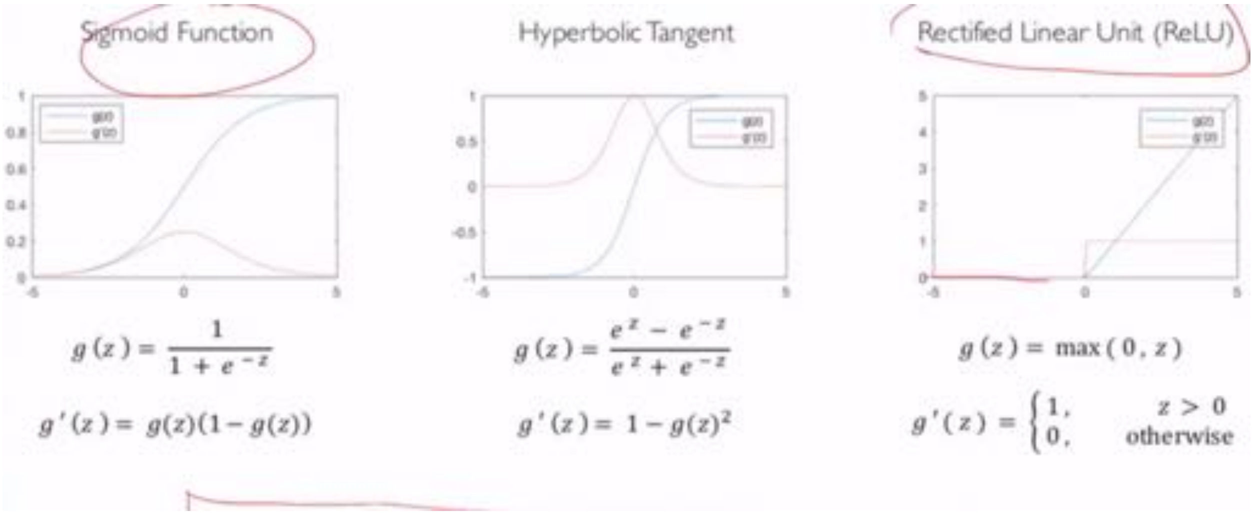
```
▪ init $w$
▪ for iter = 1, 2, …
    ▪ pick random subset of training examples J
        $w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$
```

General form for Deep Learning



$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

Common Activation functions



Sigmoid Function

Hyperbolic Tangent

Rectified Linear Unit (ReLU)

$g(z) = \frac{1}{1+e^{-z}}$

$g'(z) = g(z)(1 - g(z))$

$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

$g'(z) = 1 - g(z)^2$

$g(z) = \max(0, z)$

$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$

Deep Neural Networks,

Training is similar to logistic regression

- Run gradient ascent

Week 13: Lecture 31 Neural Networks (4/14)

Neural Network Properties

- Theorem (Universal Function Approximatiors)

- A two layer neural network with a sufficient amount of nodes can approx any function to desired accuracy

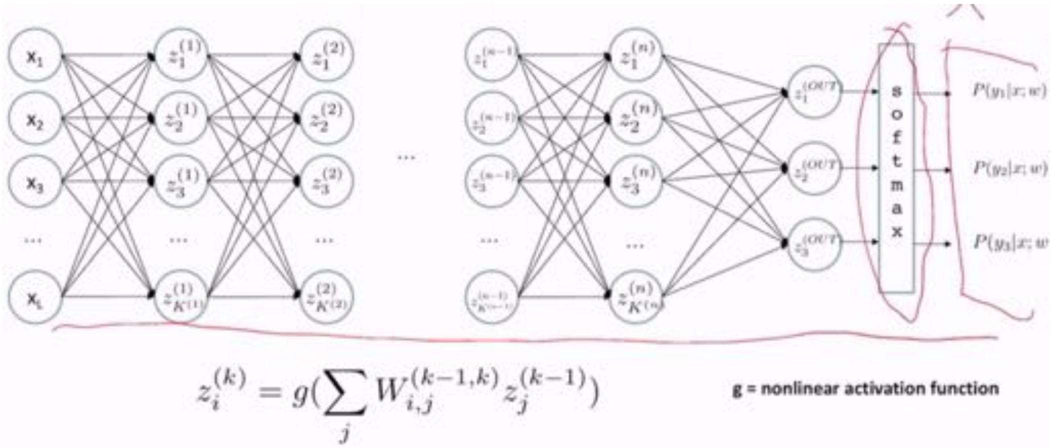
How to compute derivatores for all functions?

- Use chain rule

Automatic Differentiation

- Software only need to program the function g(x, y, w)
- Automatically compute all the derivatives wrt all entries in w

Training a Network

- 

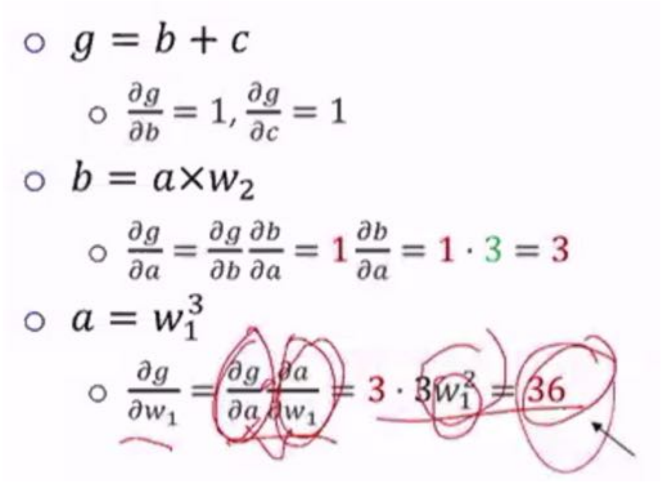
$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

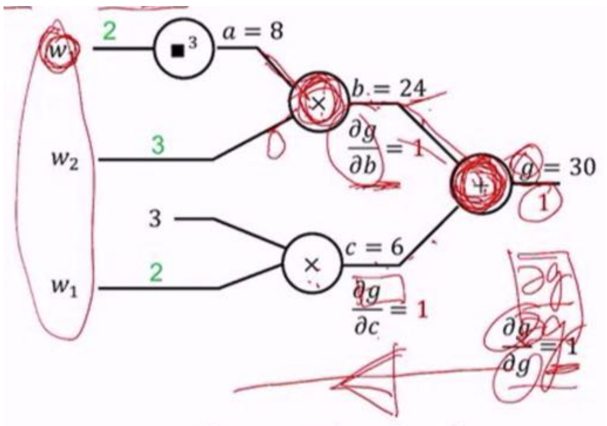
- Forward, Backwards, Gradient, Backprop

Back Propagation: $g(w) = w_1^3 w_2 + 3w_1$

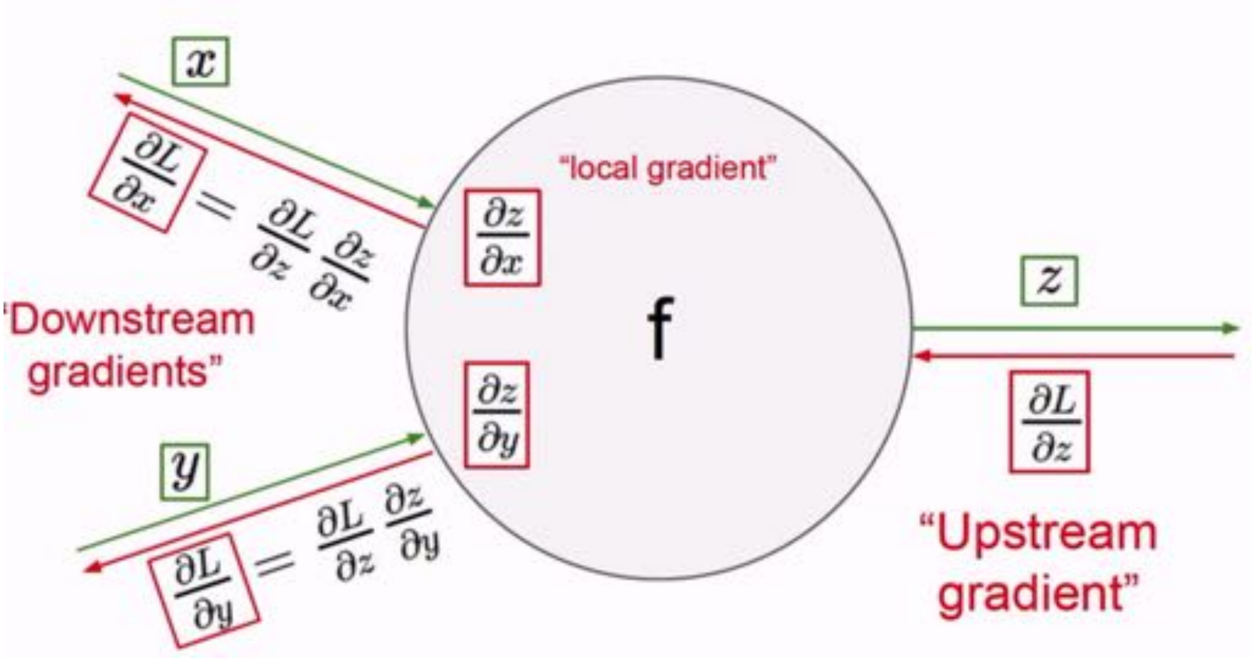
- Forward, inference putting in values
- Backprop start from the end
  - G =



- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1\frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$



Computation Graph



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
'Downstream gradients"
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
"Upstream gradient"

Summary of Key Ideas

- Optimize probability of label given input
- Continous optimization
  - Gradient ascent
    - Steepest uphill direction = gradient
    - Take step in gradient direction
    - Repeat
- Deep neural nets
  - Layered computation graph
    - Last year
  - Convolutional Network
    - alexNet input image
    - Weights
    - Loss
    - General computation graph

Neural Network architectures

- ResNet: Residual networks
- Networds with attention
- Transformer networks

Neural Network architecture search

Really Large models

- GPT2, GPT3
- CLIP

Week 14: Lecture 32 Reinforcement Learning (4/19)

Related: Markov Decision Process (MDP)

- Fully observable but probabilistic search problems

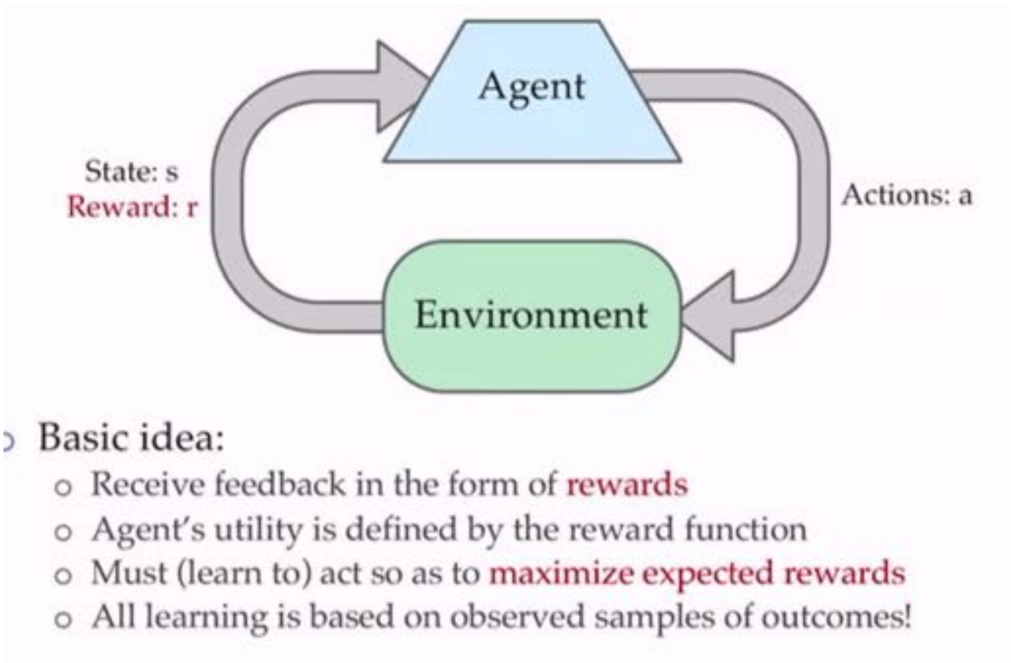
Reinforcemnt Learning

- Assumes Markov Decision Process (MDP)
- Set of states $s \in S$
- Set of actions (per state) A
- Model T
- Reward function R(s, a, s')

Still looking for policy $\pi(s)$

New Twist: don't know T or R

- Don't know which states are good or what the actions do,
- Must actually try actions and states to find out



- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

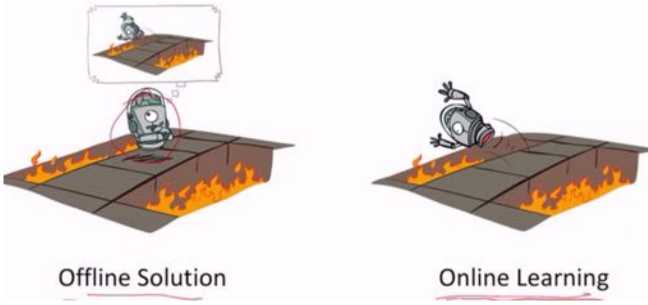
Reinforcement Learning

- Basic Ideas

    - Exploration: have to try unknown actions to get information
    - Exploitation: eventually, use what you know
    - Sampling: repeat many times to get good estimates
    - Generalization: what you learn in one state may apply to others too
    - Regret: How efficient doing exploration

Offline (MDPs) vs Online (RL)

- RL doesn't know what each action will do



Model-Based Learning

Model-Based Idea:

- Learn an approx model based on experiences

- Solve for values as if the larned model were correct

Step 1: Learn empirical MDP model

- Count outcomes s′ for each s, a
- Normalize to give an estimate T(s, a, s′)
- Discover each R(s, a, s′) when we experience

Step 2: Solve the learned MDP

- Use value iteration or another way to solve

Example

- Want to learn $\hat{T}(s,\ a,\ s')$ transition model and reward model $\hat{R}(s,\ a,\ s')$
- Find average of the transition state

Pro:

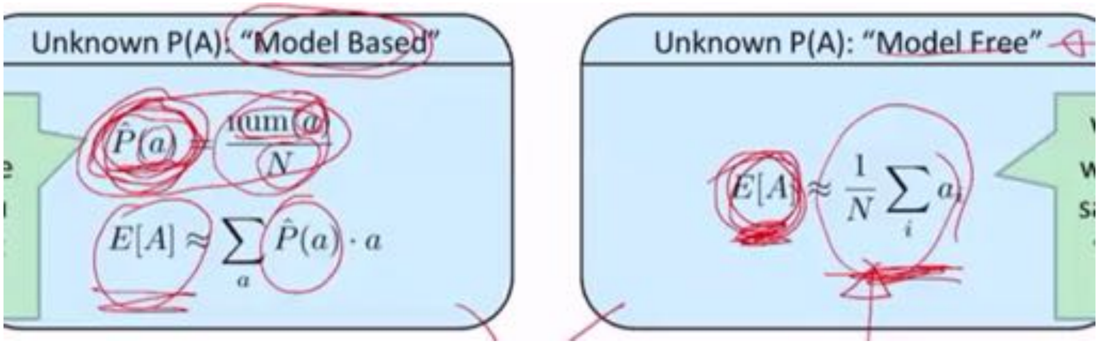
- Makes efficient use of experiences

Con:

- May not scale to large state spaces
- Learns model one state-action pair at a time
- Cannot solve MDP for large |S|

Analogy: Expected Age

- Collects samples a1, a2,...
- Unknown P(a): “Model Based”
    - Compute P(a)
    - Then compute expected value
- Unknown P(A): “Model Free”
    - Samples aeppear with the right rfrequencies
    - Compute expected value directly without probaiility



Unknown P(A): “Model Based”

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Unknown P(A): “Model Free”

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Passive

Reinforcement Learning

- Simplified task: Policy evaluation
- Input: fixed policy $\pi(s)$
- Learn the state values

Learner is “along for hte ride”

- No choice about what actions to take
- Execut ethe policy and learn from experience

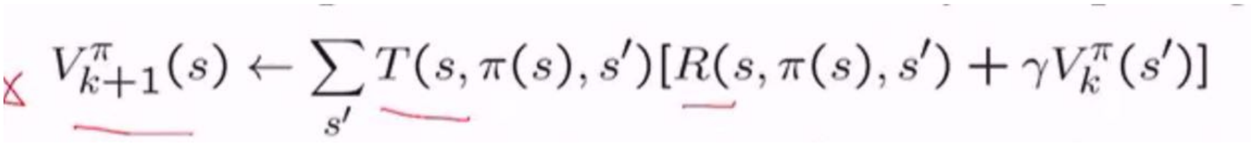
Direct Evaluation

- Goal: Compute values for each state under π
- Idea: Average together observed sample values
    - Act according to pi
    - Every time you visit a state,w rite down what the sum of discounted rewards turned out to be
    - Avg those samples
- Pros:
    - Easy to understand, doesn’t require any knowledge of T, R
    - Eventually computes the correct avg values, using sample transition
- Cons:
    - Waste information about state connections
    - Each state must be learned separately
    - Takes long time to learn

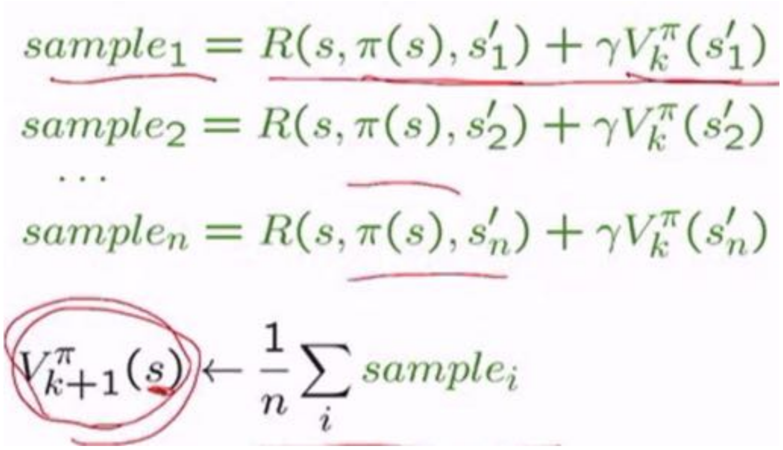
Why Not use policy evalutation

- Need T and R to do policy evaluation

Sample-Based Policy Evaluation

- Improve by computing averages
- 

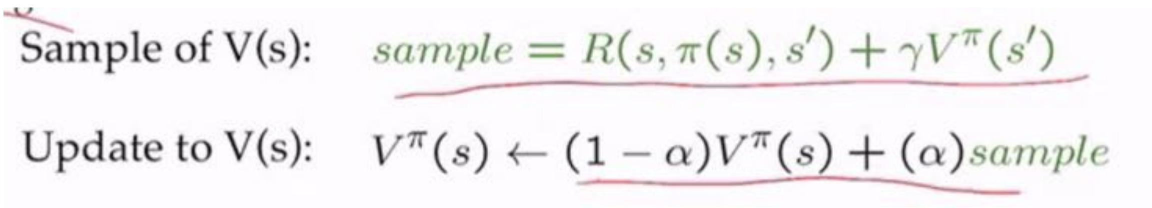
$$\underline{V^{\pi}_{k+1}}(s) \leftarrow \sum_{s'} \underline{T}(s, \pi(s), s')[\underline{R}(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')]$$

- Idea: Take samples of outcomes s’ and average
- 

$$\underline{sample_1} = \underline{R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)}$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\ldots$$
$$sample_n = R(s, \underline{\overline{\pi(s)}}, s'_n) + \gamma V^{\pi}_{k}(s'_n)$$
$$\boxed{V^{\pi}_{k+1}(s)} \leftarrow \frac{1}{n}\sum_{i} sample_i$$

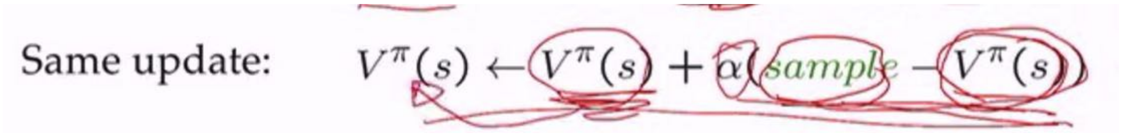
Temporal Difference Learning

- Big Idea: Learn from every experience
    - Update V(s) each time experience a transition (s, a, s’, r)
    - Likely outcomes s’ will contribute updates more often
- Temporal difference learning of values
    - Policy still fixed, still doing evalution
    - Move values toward value of whatever success occurs: runnign avg



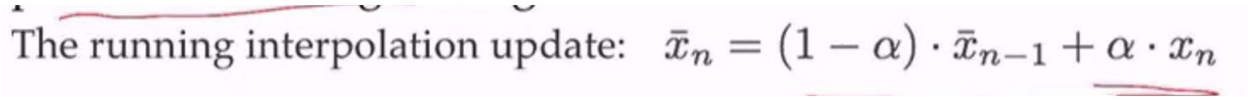
Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha)sample$



Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Exponential Moving Average

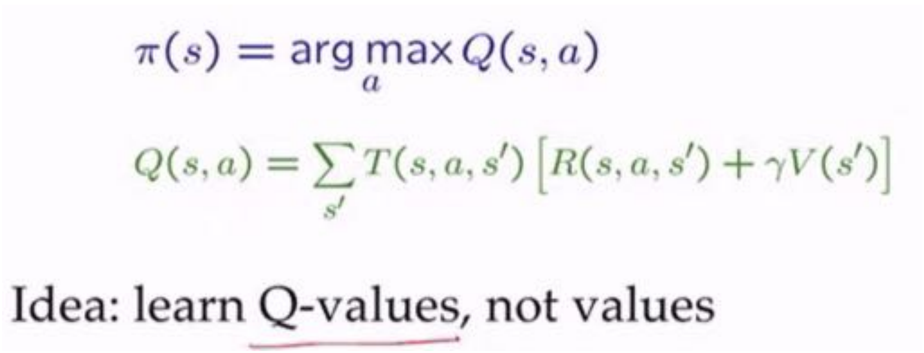


The running interpolation update: $\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$

-
- Recent samples more important, foregets about past
- Decreasing learning rate can give converging averages

TD Value Learning

- Model free wya to do policy evlaution
- Mimicking Bellman updates with running sample averages



$$\pi(s) = \arg\max_a Q(s, a)$$

$$Q(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V(s')\right]$$

Idea: learn Q-values, not values

- Make action selection model-free too

Week 14: Lecture 33 Reinforcement Learning 2 (4/21)

Passive Reinforcement learning

- Fixed policy that determines its behavor

Model-based learning

- Learn an approx MDP model based on experience

Model-free learning

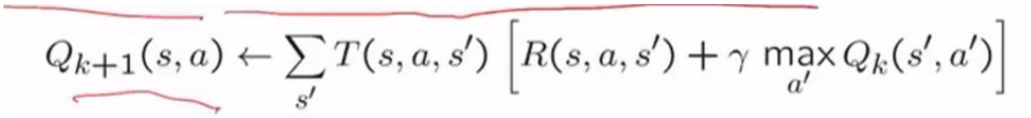
- Do not learn a specific MDP

TD Value Learning

- Q-value: state s action a first step is given
    - Find action to maximize Q value
- Value iteration: successive (depth-limited) values
    - Start with V(s) = 0 and calculate depth k+1 for values for all states
- Q values more useful can compute them instead

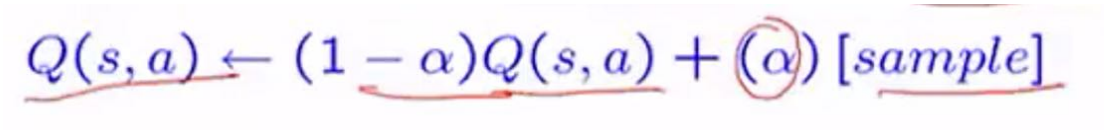
    - Start with Q(s,a) = 0
    - Given, Q, calc depth K + 1 q values for all q states

Q - Learning



$$\underline{Q_{k+1}(s,a)} \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Q-lenaring: sample-based Q value iteration
- Learn Q(s, a) values as you go
    - Get sample (s, a, s′, r)
    - Consider old estimate
        - Sample = R(s, a, s′) + γmaxQ
    - Incorporate new estimate into running avg



$$\underline{Q(s,a)} \leftarrow (1 \underline{- \alpha)Q(s,a)} + (\alpha)\ [\underline{sample}]$$

Q-Learning

- Full reinforcement learning: optimal policies
- Goal: learn optimal policy, values

Q-Leanring Properties

- Q-learningconverges to optimal policy even if you're acting suboptimally
- Off-policy learning
- Cavets
    - Have to explore enough
    - Make the learning rate small enough
    - Not decreases it too quickly
    - Doesn't matter how you select actions

Exploration vs Exploitation

- Explore to find new places, explotation always going to same place

How to Explore

1. Simplest: random action
    - Small probability ϵact randomly

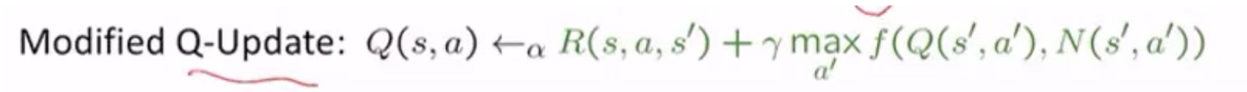
- Problems with random actions
    - Keep thrashing around once learning is done
    - Lower epsilon over tiem

Exploration Functions

- When to explore: random actions: explore a fixed amount
- Explore areas whose badness is not yet established, eventually stop exploring

2. Exploration function: explore badness not established
    - Takes a value estimate u and visit count na nd returns optimistic utility predetermined constant

        - f(u, n) = u + k/n,
    - Regular Q-Update: Q(s, a) <- R(s, a, s') + γ



Modified Q-Update: $Q(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

    - N(s, a) : number of times q state (s, a) visited)

Regret

- Even if you learn optimal policy, you still make mistakes along the way
- Can look at mistakes made during learning processs
- Regret: measure of total mistake cost: difference between (expected) rewards, youthful suboptimality, optimal rewards
    - Requires optimally learning to be optimal
    - Random exploration and exploration functions both end up optimal but ranodm exploration has higher regret

Approximate Q-Learning

Generalizing Across States

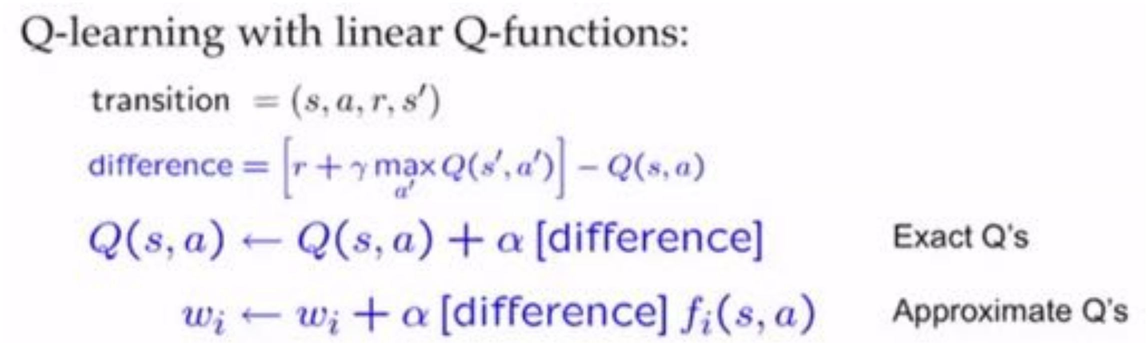
- Basic Q elearning keeps table of all q values
- Cannot possibly learn about every single state
    - Too many states to visit
    - Too many states to hold q tables in memory
- Want to generalize
    - Learn about small number of training states from experience
    - Generalize experience to new, similar situations

Feature -Based Representations

- Solution: describe a state using a vector of features (properties)
- Q(s, a) : describe a state using a vector of features (properties)
    - Distance to closest ghost, etc
- Linear Value Function
    - Can write q function for any state using a few weights
        - $V(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$
        - $Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a)...$
    - Advantage: experience summed up in a few powerful numbers
    - Disadvantage: states may share features but actually be very different

Approximate Q-Learning

- Q-Learning with linear Q -functions



Q-learning with linear Q-functions:

transition $= (s, a, r, s')$

difference $= \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s, a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$ Exact Q's

$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$ Approximate Q's

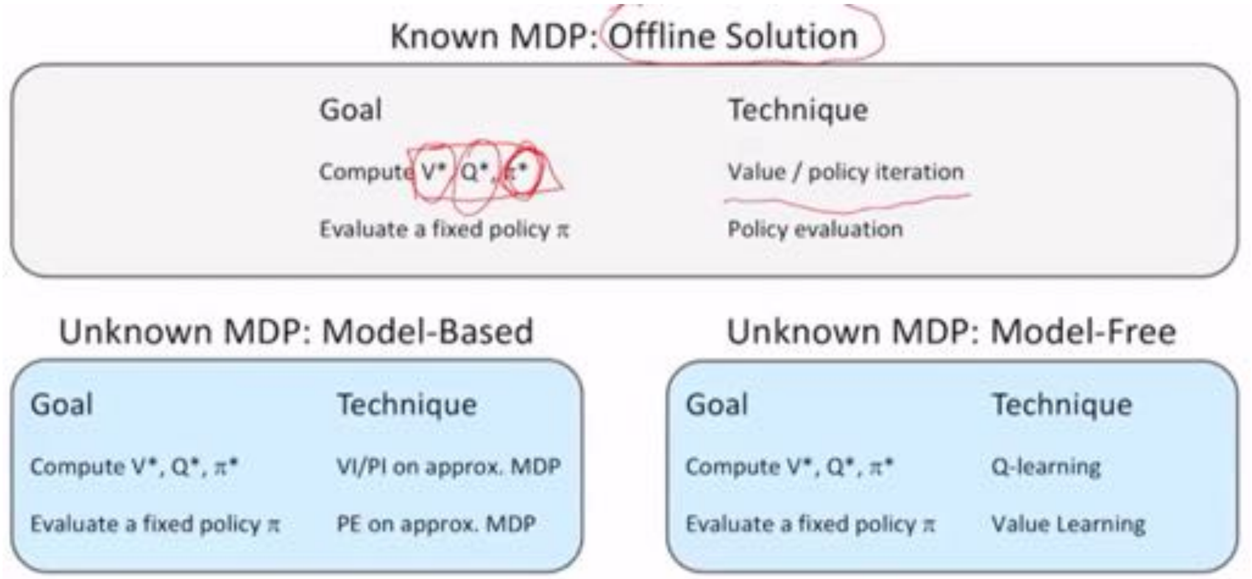
-
- Adjust weights of action features
- If somehtin bad happened, blame features
- Online least squares

Q-Learning and Least Squares

Linear Approx: regression

- Can reduce loss

Summary: MDPs and RL



Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

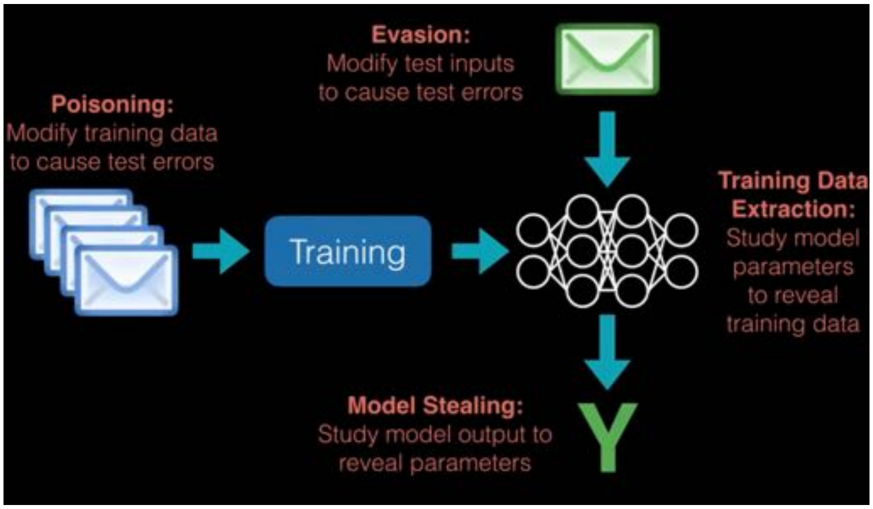
Week 14: Lecture 34 Guest Lecture: Adversarial ML (4/23)

Machine learning not as great in applications

Data -> Training -> Neural Networks -> test input gives Y

1. Poisoning: modify training data to cause test errors
2. Evasion: modify test inputs to cause test errors
3. Model Stealing: study model output to reveal parameters

4. Training Data Extraction: Study model praremeters to reveal training data



Evasion: Modify test inputs to cause test errors

Poisoning: Modify training data to cause test errors

Training

Training Data Extraction: Study model parameters to reveal training data

Model Stealing: Study model output to reveal parameters

Act I: Evasion

Already trained model, find inputs that cause errors

Can Adversarial Perturbation is you flips the bits on some

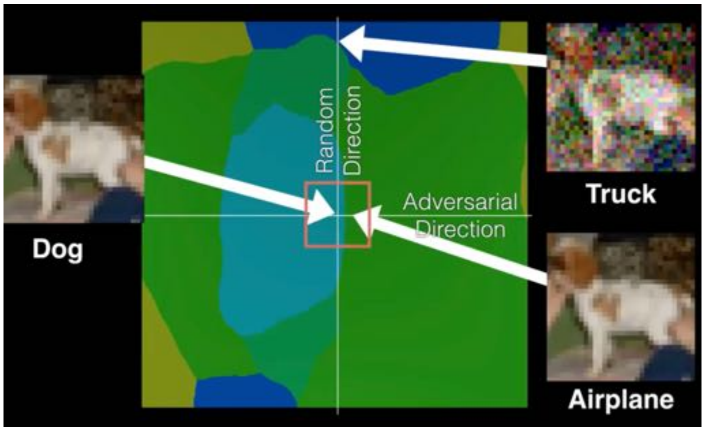
Vision problems

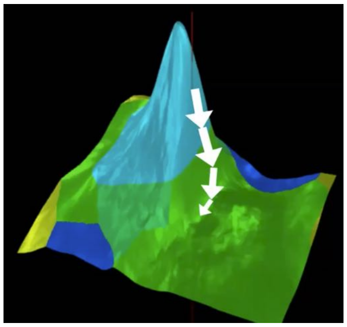
How do these attacks work?

- Classifys the dog by and the strength of the model is the peak
- Little change in pixel space makes huge loss

How to defend it against attacks

1. Train it on adversarial training
   a. There is still a region which is misscalssified

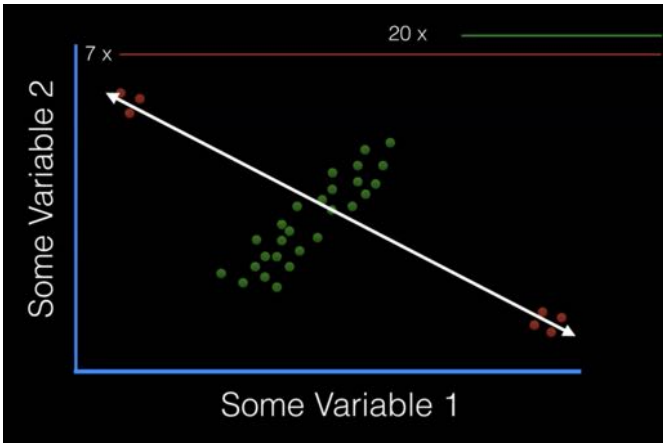


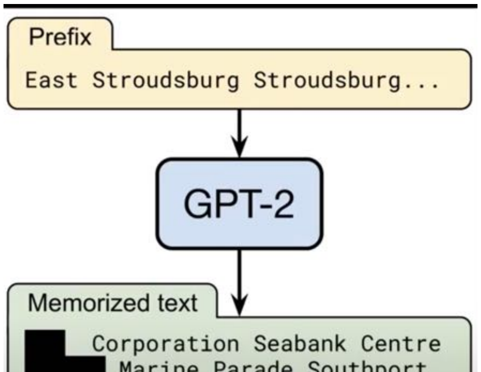


Act II: Poisoning

Modify training data to cause test errors,

Have ver small data that can mess up the model very much




Prefix
East Stroudsburg Stroudsburg...
GPT-2
Memorized text
[redacted] Corporation Seabank Centre
Marine Parade Southport

<u>Act III: Training Data Extraction</u>

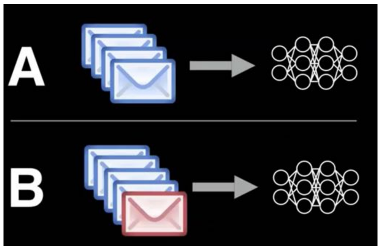
Study model parameteres to reveal training data

Can reveal data about the training

Model can try to guess the answer, loss might be better than if you guess

How do we prevent this?

- Differential using two different models and see if adversarial can find something about the model

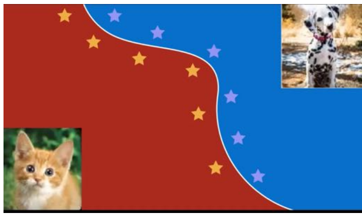


<u>Act IV: Steal the Model</u>

Given i/o learn the parameters

Find points around the boundaries to reverse find the model

Useful to set up future attacks



Week 15: Lecture 35 Guest Lecture: Fairness and ML: Hardt (4/26)

Live in a world of inequality, oppression, discrimination but ML can perpetuate existing pattersn of injustice

Ruha Benjamin: employment of new technologies reflect and produce existing inequalities

Unjustified discrimination in consequential decision making settings

Discrimination in US

1. Disparate treatment
    - Purposeful consideration of gorup membership
    - Goal: Precedual faireness
2. Disparate impact
    a. Avoidable or unjustified harm
    b. Goal: Distributive justice, minimize differences in outcomes

Failure of fairness through unawareness

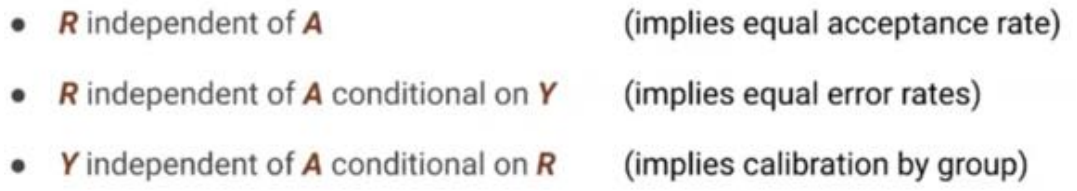
- Removing or not including sensitive data
- Amazon was predicting number of purchases correlates with affluence

We don't consider that in our data is never a valid argument

<u>Part 1: Fairness criteria in classification</u>

Urgency, scale, reach, and impact of algorithmic decisions

1. Equalizing acceptance rate
    a. Equal positive reate: for any two groups, a, b, require
    b. $P[D = 1 | A = a] = P[D = 1 | B = b]$
    c. Does not rule out unfair practices
        i. Make good decision in one group, equaize positive rate, less data or poor data in one group
    d. Adversarial learning to train representation of data independent of group memeberhsip



- ***R*** independent of ***A*** (implies equal acceptance rate)
- ***R*** independent of ***A*** conditional on ***Y*** (implies equal error rates)
- ***Y*** independent of ***A*** conditional on ***R*** (implies calibration by group)

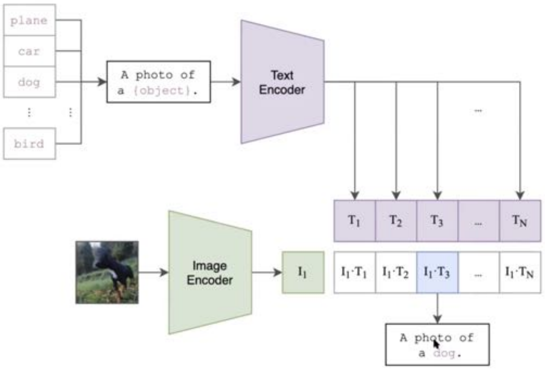
Error rate parity vs calidbration

- Can't have them all, mutually exclusive

Essence of COMPAS debate

- Risk score used by many jurisdictions to assess "risk of recidivism". Judges may detain defendant in part based on this score

Week 15: Lecture 36 Guest Lecture: CLIP Learning Transferable Visual Models From NL Supervision (4/28)



One model can be adapted to variety of tasks, recognizes things in a viaul scene learns about images from free-form text

Vision models led to the deep learning boom

CLIP special

- Instead of using a fixed set of labels, get supervision from natural language

CLIP: Contrastive Language-Image Pretraining

- Training
    - 400 M image-text pairs form the internet
    - Batch size of 32,768
    - 32 epochs over the dataset
    - Cosine learning rate decay
- Architecture
    - Resnet-based or ViT based

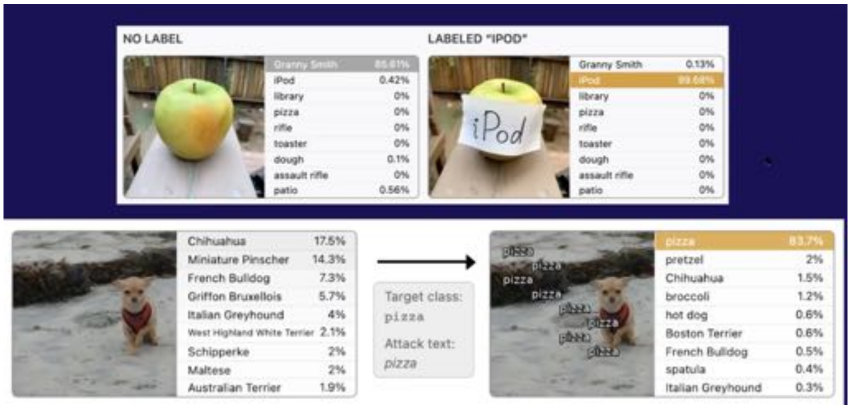
Representation Learning

Zero-Shot Transfer

Quantifying unsafety of CLIP models

- Race, gender, restricted to research

Typographic Attacks



NO LABEL

Granny Smith	85.61%
iPod	0.42%
library	0%
pizza	0%
rifle	0%
toaster	0%
dough	0.1%
assault rifle	0%
patio	0.56%

LABELED "IPOD"

Granny Smith	0.13%
iPod	99.68%
library	0%
pizza	0%
rifle	0%
toaster	0%
dough	0%
assault rifle	0%
patio	0%

Chihuahua	17.5%
Miniature Pinscher	14.3%
French Bulldog	7.3%
Griffon Bruxellois	5.7%
Italian Greyhound	4%
West Highland White Terrier	2.1%
Schipperke	2%
Maltese	2%
Australian Terrier	1.9%

Target class:
pizza

Attack text:
pizza

pizza	83.7%
pretzel	2%
Chihuahua	1.5%
broccoli	1.2%
hot dog	0.6%
Boston Terrier	0.6%
French Bulldog	0.5%
spatula	0.4%
Italian Greyhound	0.3%

Week 15: Lecture 37 The Future of AI (4/30)

Way we train AI is fundamentally flawed

Real world performance far worse than test set

Overinterpret how well our systems are doing

Behavior is fragile

Deep learning and infinitum

- Francois Chaollet(2017) :

    - Many more applications are completely out of reach for current deep learning techniques even given vast amounts of human-annotated data

Probablilstic programming

- Universal languages and algirhtms for prabailistic modeling, learning, and reasoning

Likely Developments in 2020s

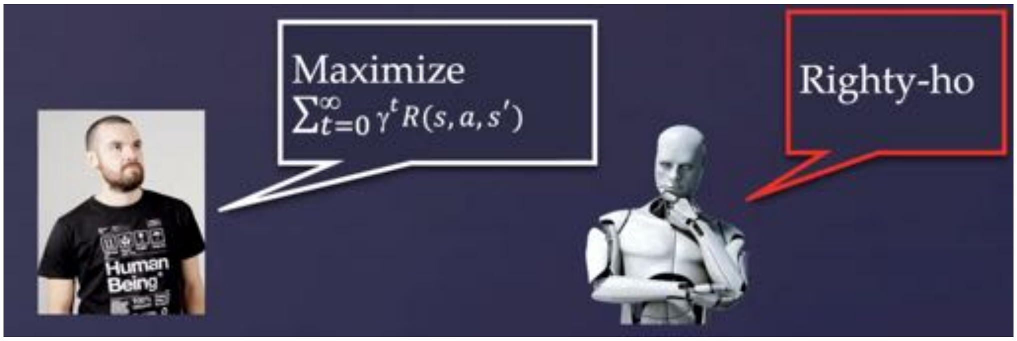
- Robots for war, roads, warehourses, mines, fields, home
- Personal digital assistants for all apsects of life
- Commercial language systems
- Global vision system via satellite imagery

General-purpose AI still missing

- Real understanding of language
- Integration of learning with knowledge
- Long range thinking at multiple levels of abstraction
- Cumulative discovery of concepts and theories

Standard model for AI

- Human gives maximze reward function



-

- Fine for artificially defined game like chess
- Hard to define function for other tasks like AI reward function for driving
- King Midas: Cannot specify R correctly
- Becareful what you wish for
- Succeed in getting objective
- Smarter AI -> worse outcome

Social media

- Optimizing clickthrough
    - ~~Learning what people want~~
    - Modify people to be more predictable

How we got into this mess

- Humans are intelligent to the extent that our actions can be expected to achieve our objects
- ~~Mahcines are intelligent to the extent that their actions cna be expected to achieve their objectives~~

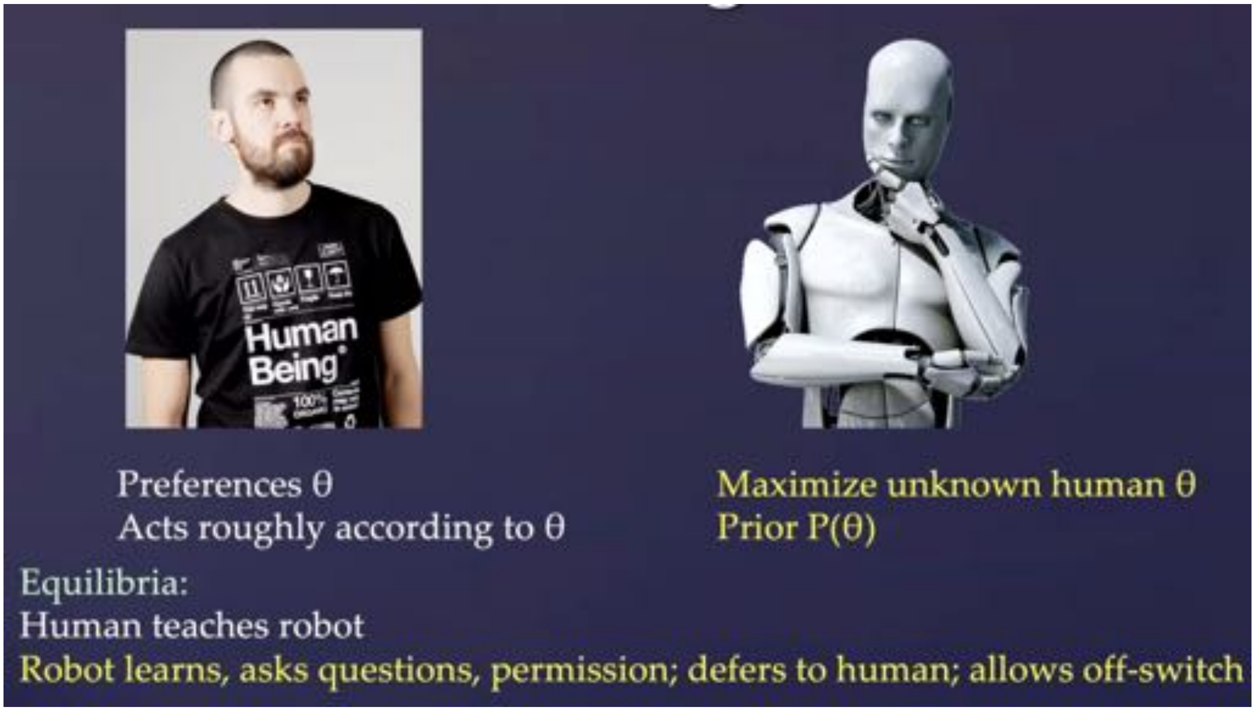
- Mahcines are beneficial to the extent that their actions can be expected to achieve our objectives

New model: Provably beneficial AI

1. Robot goal: satisfy human preferences
2. Robot is uncertain about human preferences
3. Human behavior preovides evidence of preferences

-> Assistance game with human and machine players

Smarter AI -> better outcome



Preferences θ
Acts roughly according to θ

Maximize unknown human θ
Prior $P(\theta)$

Equilibria:
Human teaches robot
Robot learns, asks questions, permission; defers to human; allows off-switch

Machine has positive incentive to allow itself to be switched off

“Imperfect” human behavior

- Computationally limited
- Emotioinally driven behavior

Many humans

- How to make decisions based on my human preference s
- Commonalities and difference in preferences
- Aggregate individual preferences
- Aggregation over individuals with diff beliefs
- How to have robots make decisions for us

Summary

- Standard model for AI leads to loss of human control over increasingly intelligent AI systems
- Provably beneficial AI is possible and desirable
    - It isn’t “AI safety” or “AI Ethics” its AI

- Problems of misuse and overuse