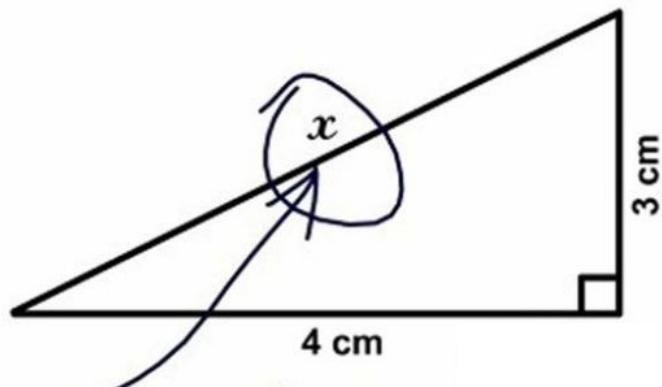


3. Find x.



Here it is

AI as Search

CS B551
Fall 2022

Announcements

- Canvas, Q&A Community, Slack, etc.
- Activity posted on Canvas – due Monday!
- Assignment 0 coming soon!
 - Practice with searching, and with Python.
 - Lots of online resources to learn Python: Google Code, CodeAcademy, many, many tutorials, etc.

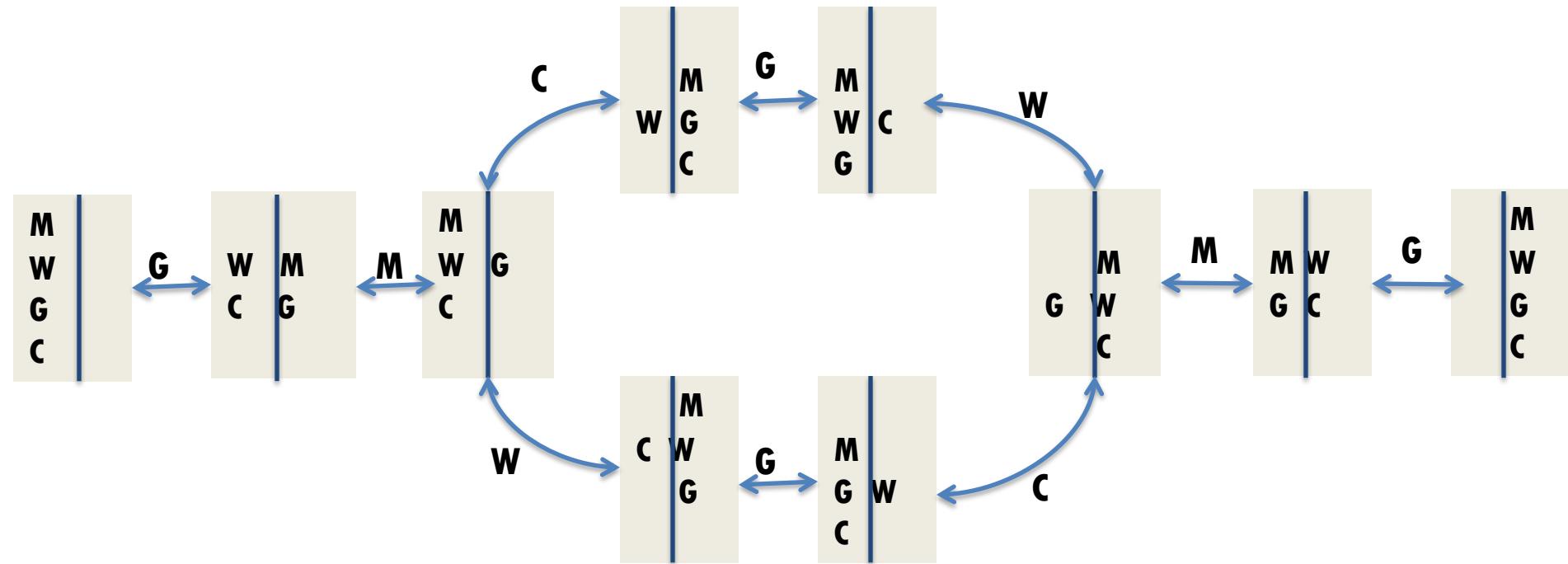
An old puzzle...



Puzzles and games have long been considered a challenge for human intelligence:

- **Chess** in Persia and India ~4000 years ago
- **Checkers** in 3600-year-old Egyptian paintings
- **Go** in China over 3000 years ago

A representation of the problem



M: Man goes alone
W: Man takes wolf
G: Man takes goat
C: Man takes cabbage

State space
represented as a
graph

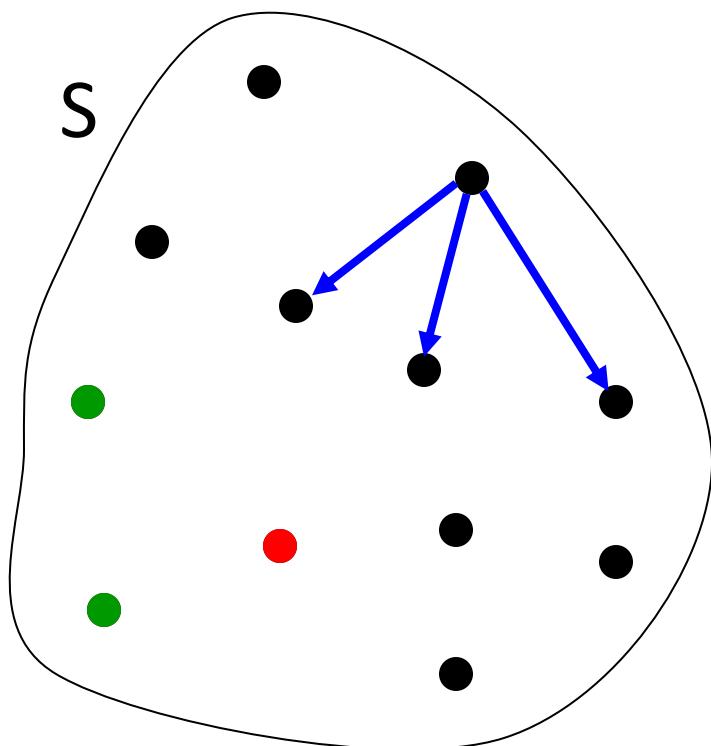
Exploring Choices

- Problems that seem to require intelligence usually require exploring multiple choices.
- Search:
 - A systematic way of exploring choices.
 - The process of looking for a sequence of actions that reaches the goal.

These abstractions have 5 parts:

1. Set of states S
2. Initial state s_0
3. A successor function $SUCC: S \rightarrow 2^S$ that encodes possible transitions of the system. A successor is any state reachable from a given state by applying a single action.
4. Set of goal states
5. A cost function that calculates how “expensive” a successor is

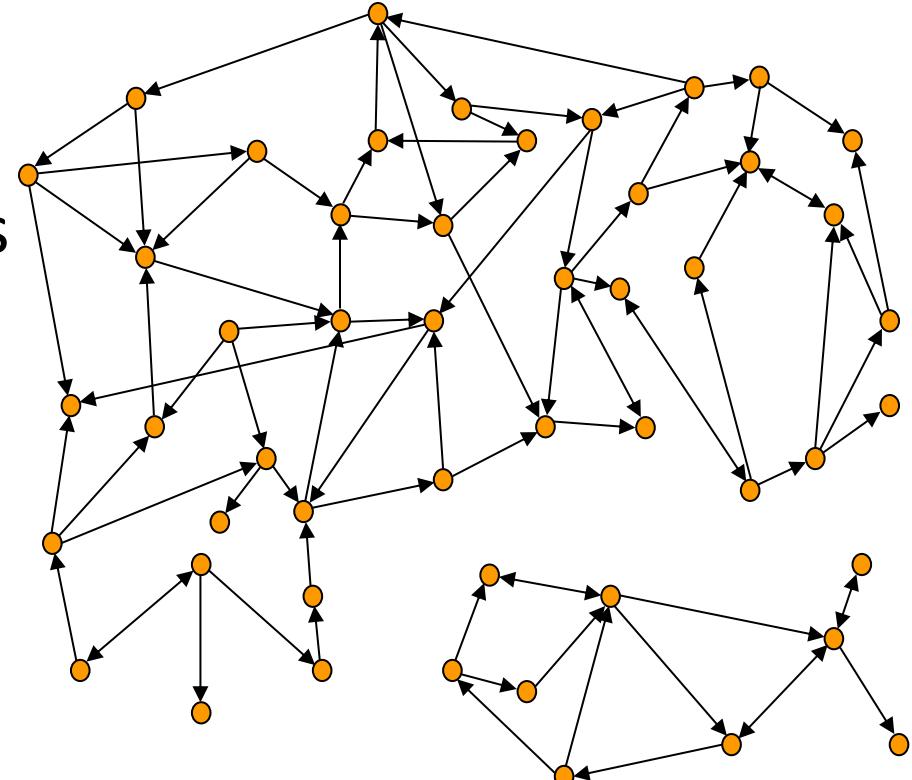
Defining a Search Problem



- State space S
- Successor function:
 $x \in S \rightarrow \text{succ}(x) \in 2^S$
- Initial state s_0
- Goal test:
 $x \in S \rightarrow \text{GOAL?}(x) = T \text{ or } F$
- Cost

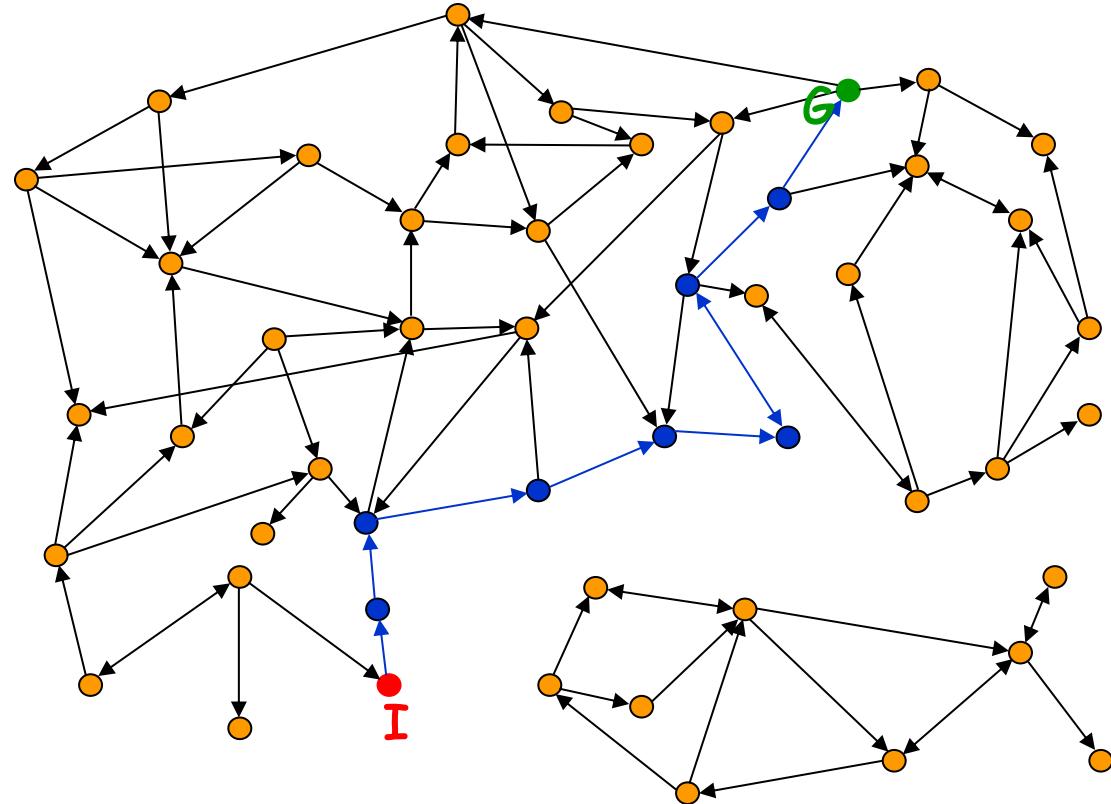
State Graph

- Each state is represented by a distinct node
- An edge connects a node s to a node s' if $s' \in \text{succ}(s)$
- The state graph may contain more than one connected component



Solution to the Search Problem

- A **solution** is a path connecting the initial node to a goal node (any one)
- The **cost** of a path is the sum of the edge costs along this path
- An **optimal** solution is a solution path of minimum cost
- There might be no solution !



Example: 8-Puzzle

8	2	
3	4	7
5	1	6

Initial state

1	2	3
4	5	6
7	8	

Goal state

Example: 8-Puzzle

8	2	
3	4	7
5	1	6

Initial state

1	2	3
4	5	6
7	8	

Goal state

State: Any arrangement of 8 numbered tiles and an empty tile on a 3x3 board

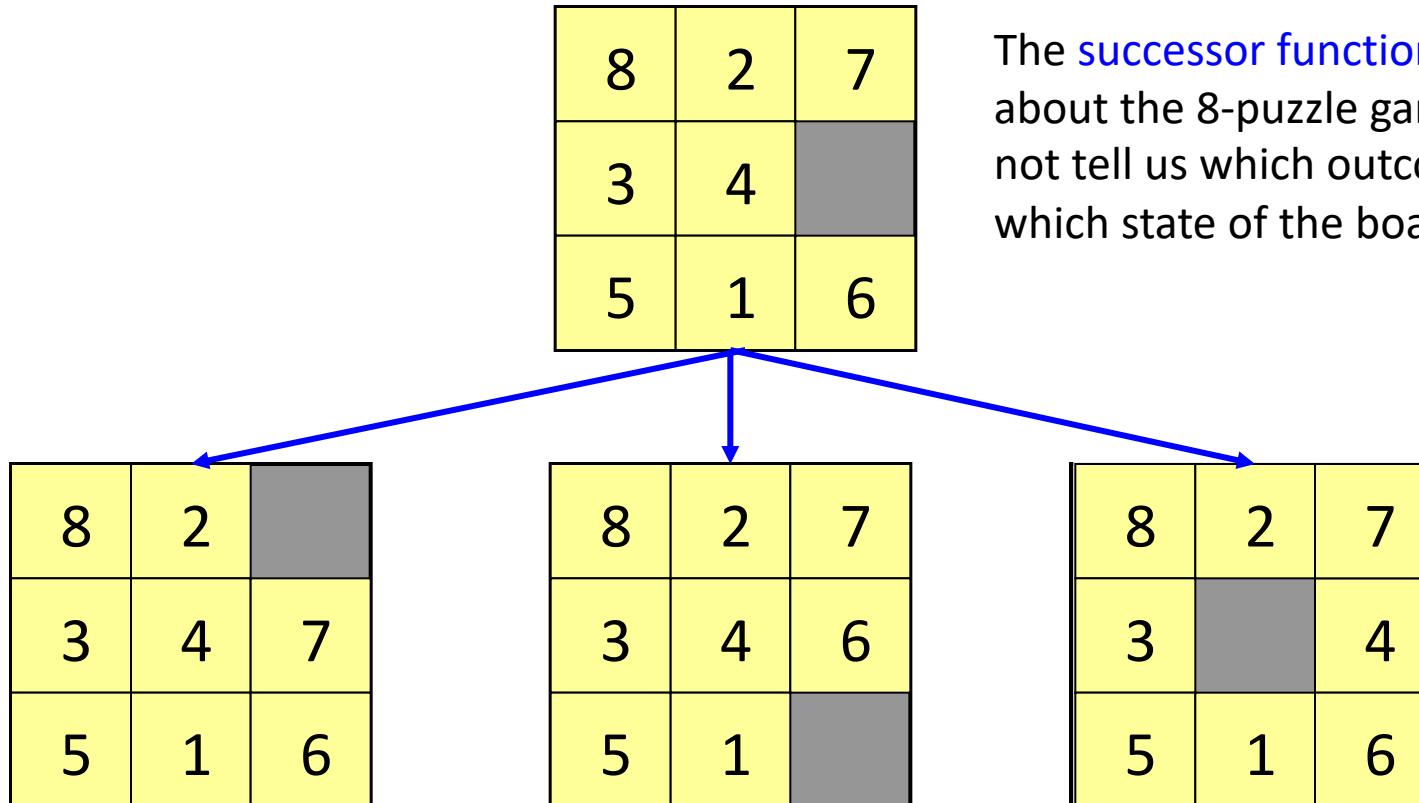
Successor function: given by available actions (sliding tiles) L, R, U, D.

Cost: How many moves were performed

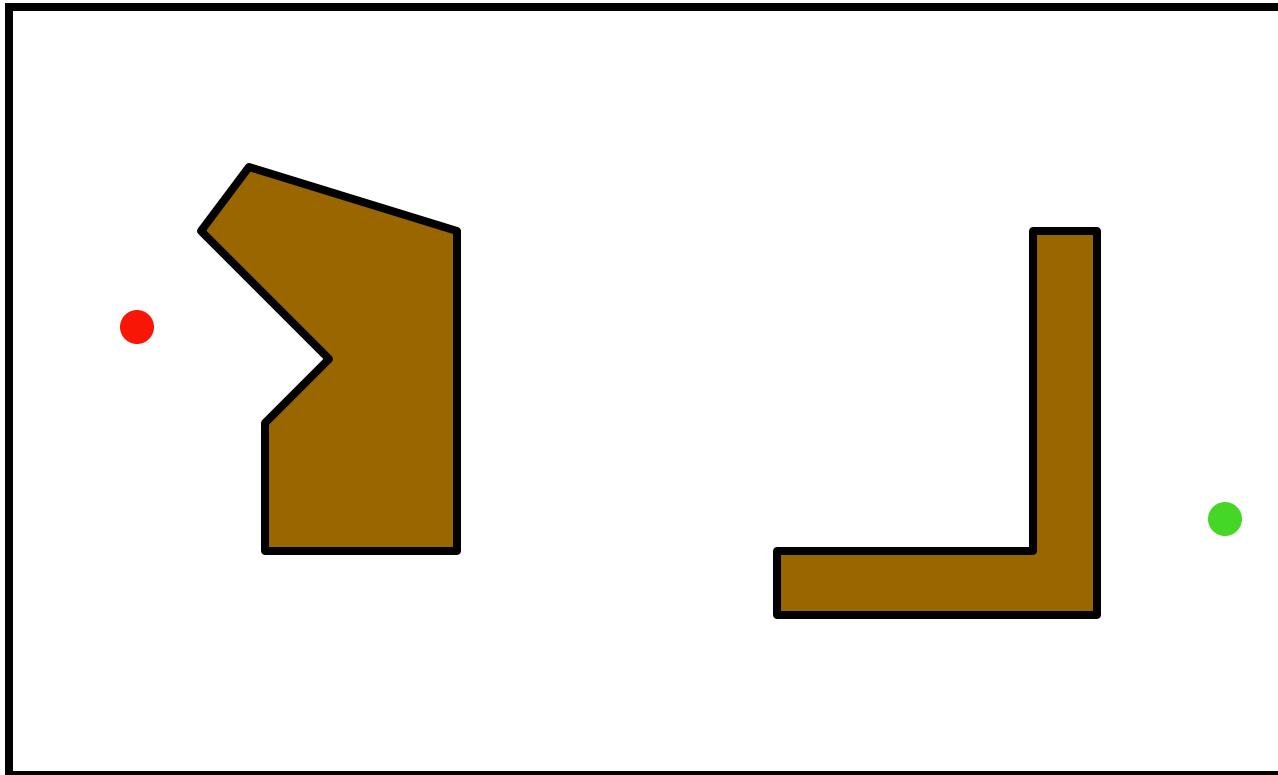
Successor Function: 8-Puzzle

$SUCC(state) \rightarrow$ subset of states

The **successor function** is knowledge about the 8-puzzle game, but it does not tell us which outcome to use, nor to which state of the board to apply it.

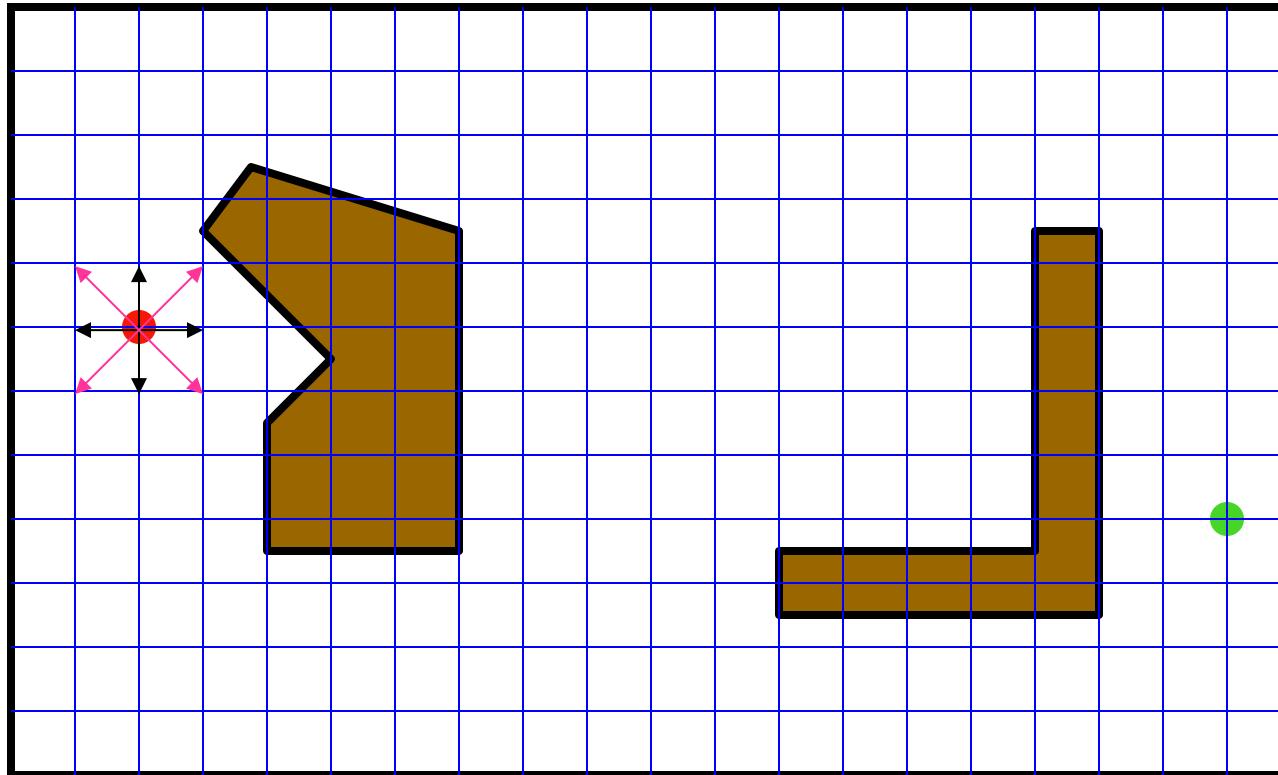


Path Planning



What is the state space?

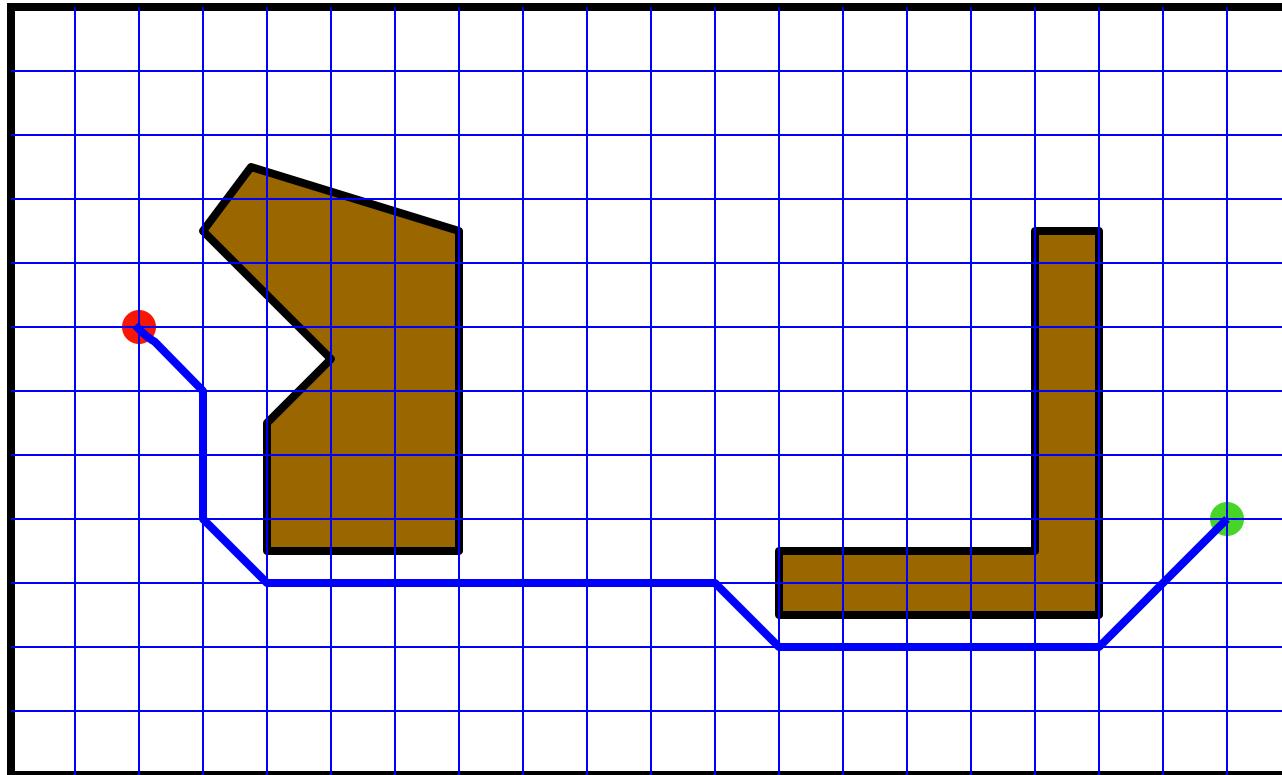
Formulation #1



Cost of one horizontal/vertical step = 1

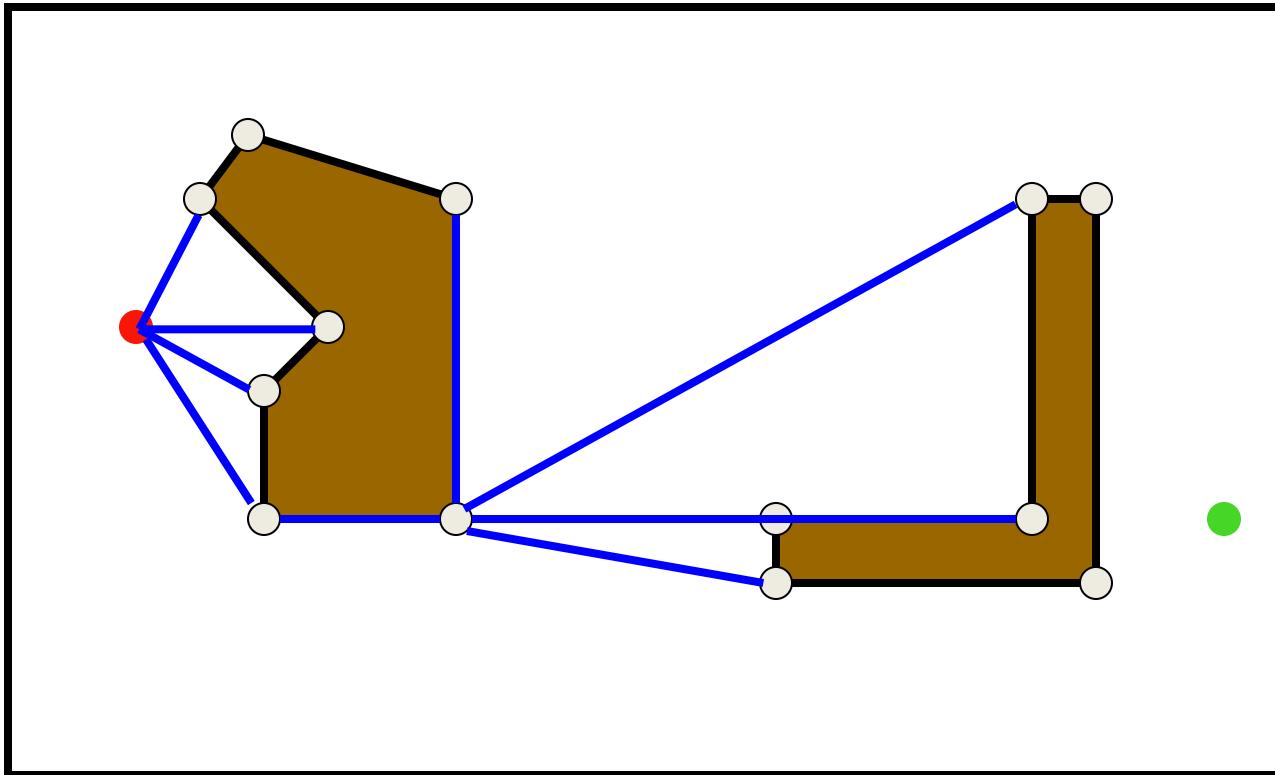
Cost of one diagonal step = $\sqrt{2}$

Optimal Solution



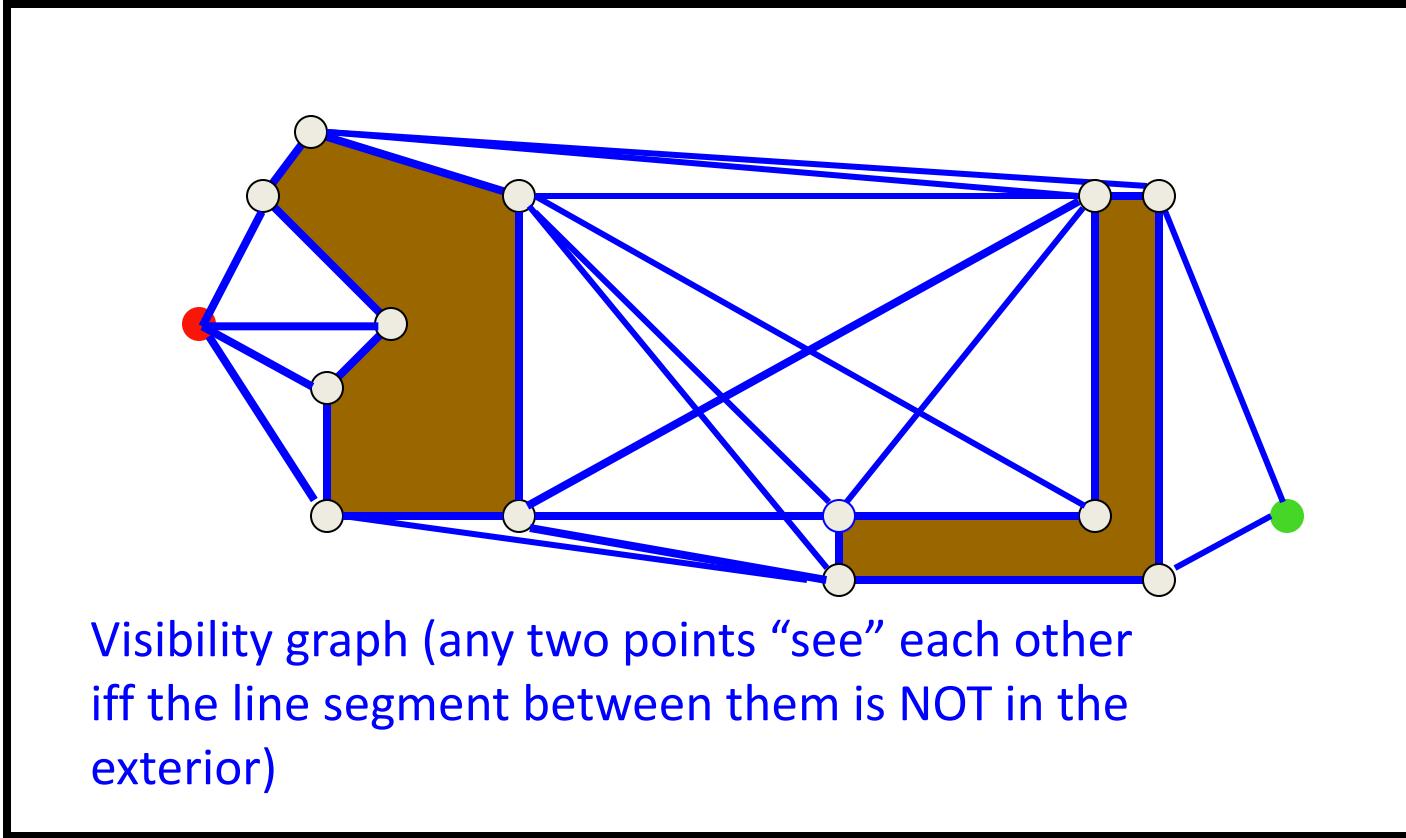
This path is the shortest in the discretized state space, but not in the original continuous space

Formulation #2



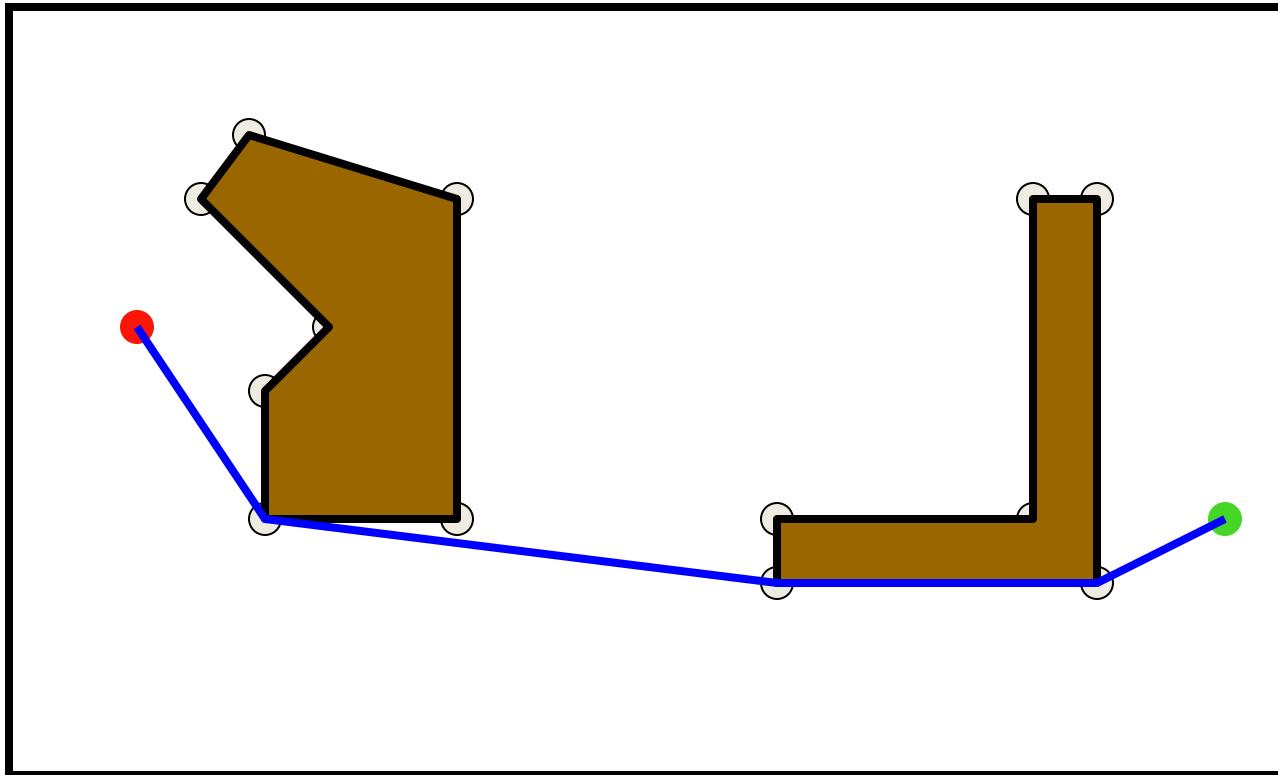
Cost of one step: length of segment

Formulation #2



Cost of one step: length of segment

Solution Path



The shortest path in this state space is also the shortest in the original continuous space

What is a State?

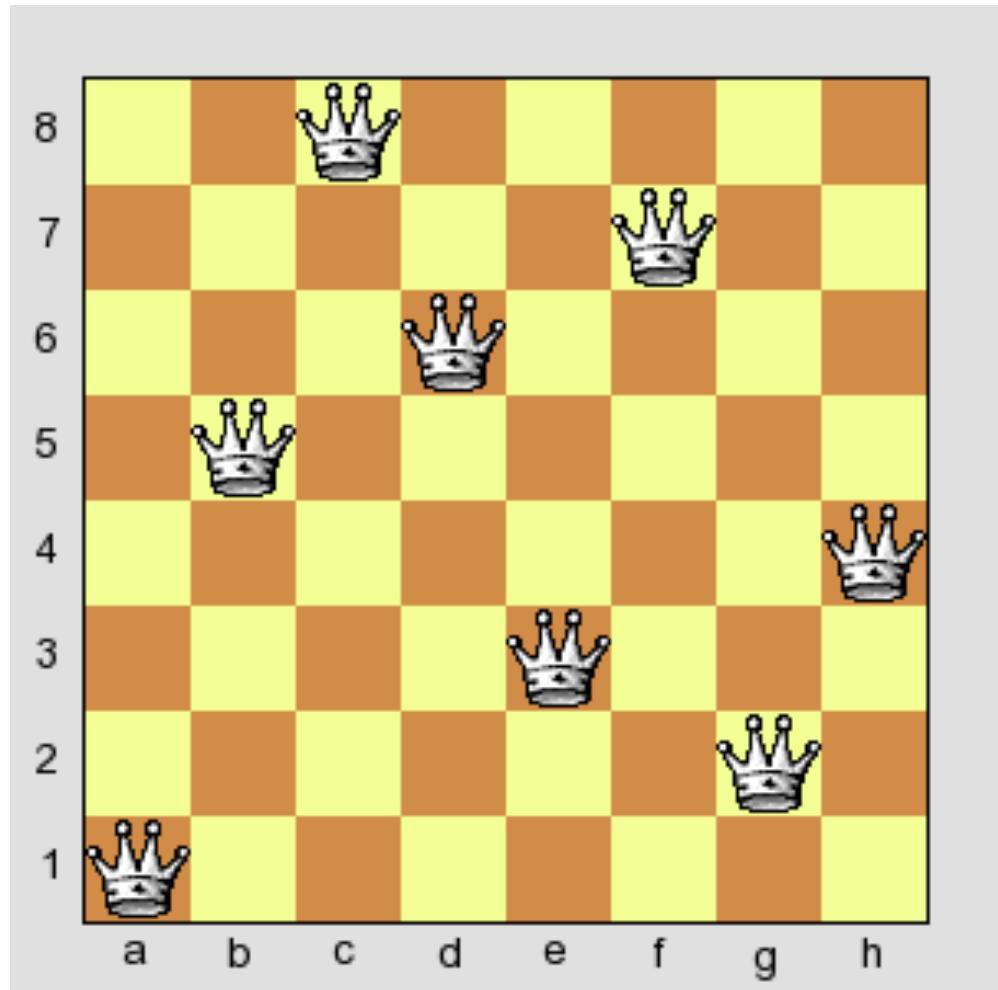
- **A state does:**
 - Represent all information meaningful to the problem at a given “instant in time” – past, present, or future
 - Exist in an *abstract, mathematical* sense
- **A state DOES NOT:**
 - Necessarily exist in the computer’s memory
 - Tell the computer how it arrived at the state
 - Tell the computer how to choose the next state
 - Need to be a unique representation

What is a State Space?

- An abstract mathematical object
 - E.g., the set of all permutations of (1,...,8,empty)
- Membership should be **trivially testable**
 - So $S = \{ s \mid s \text{ is reachable from the start state through transformations of the successor function} \}$ is *a bad definition* (not trivially testable).
- Edges should be **easily generated**
- *Again: the state space does NOT contain information about which edge to take (or not to take) in a given state*

8-Queens Problem

Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

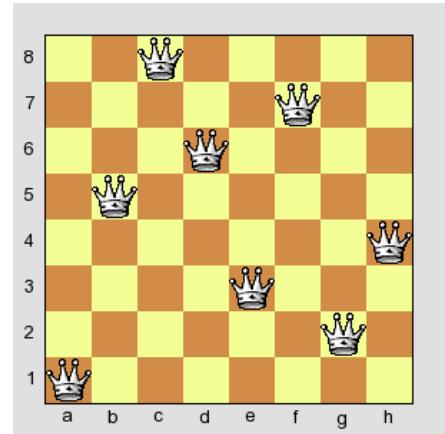


8-Queens Problem

Two main formulations of the problem:

1. An incremental formulation:

- **States:** Any arrangement of 0 to 8 queens on the board is a state.
- **Initial state:** No queens on the board.
- **Actions:** Add a queen to any empty square.
- **Goal test:** 8 queens are on the board, none attacked.



In this formulation, we have $64 \cdot 63 \cdots 57 \approx 1.8 \times 10^{14}$ possible sequences to investigate.

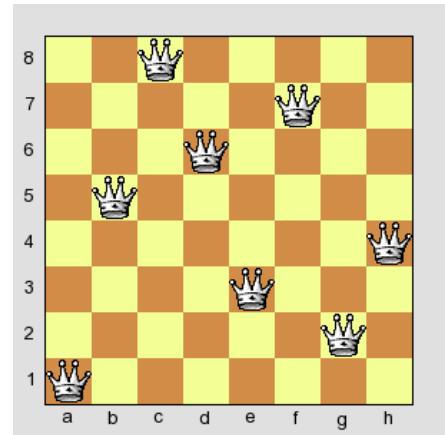
8-Queens Problem

Two main formulations of the problem:

2. Complete-state formulation: prohibit placing a queen in any square that is already attacked

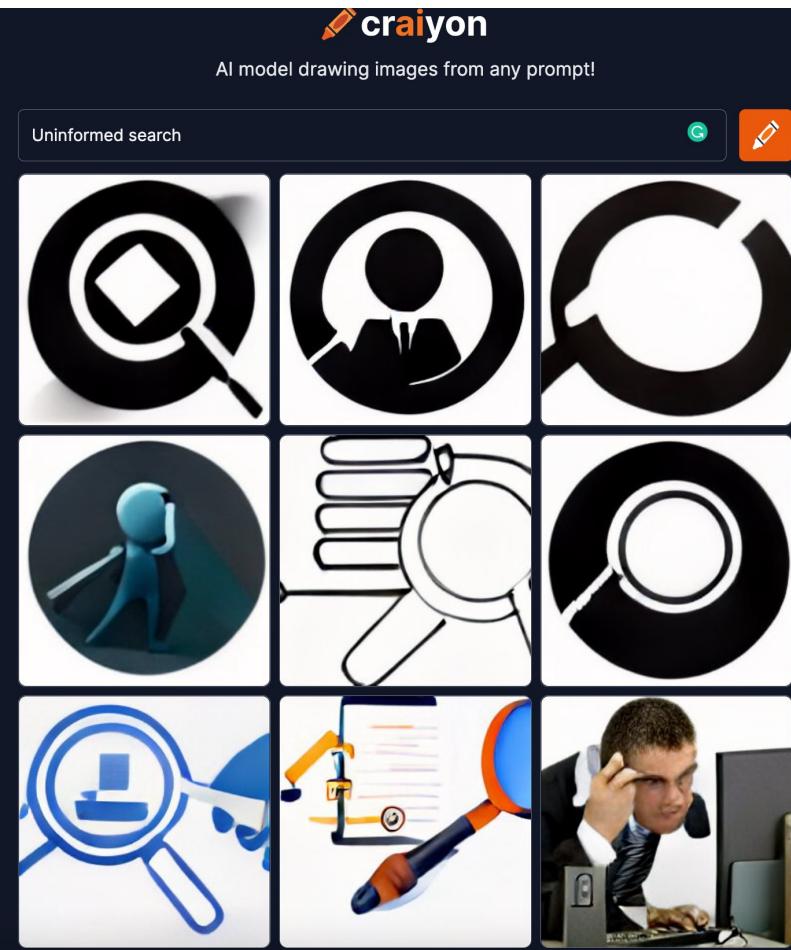
- States: All possible arrangements of n queens ($0 \leq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another.
- Actions: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.
- Goal test: 8 queens are on the board, none attacked.

This formulation reduces the 8-queens state space from 1.8×10^{14} to just 2,057, and solutions are easy to find.



Next class: Search algorithms (ch 3.4 in the book)

Search algorithms and uninformed search



CS B551
Fall 2022

Announcements

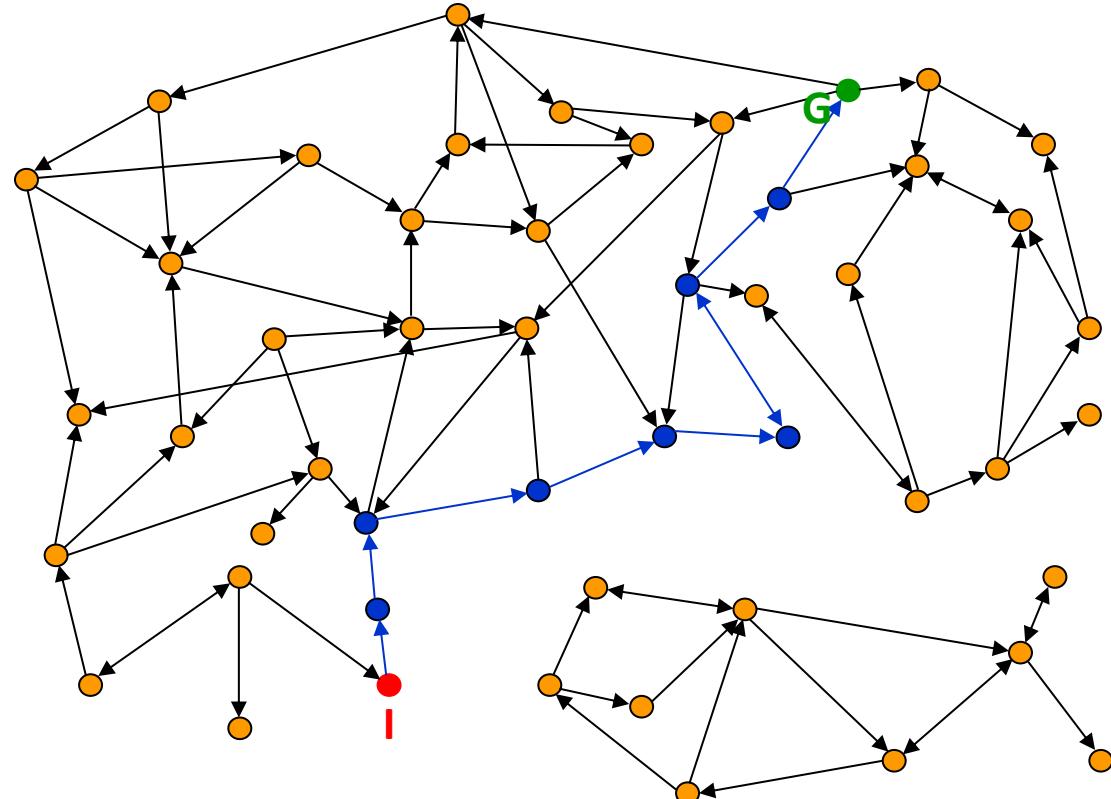
- AI's & Office hours
- Canvas, Q&A Community, Slack, etc. for Syllabus, videos, questions, slides, discussions, etc.
- Assignment 0 coming soon! (Wednesday)
 - Practice with searching, and with Python.
 - Lots of online resources to learn Python: Google Code, CodeAcademy, many, many tutorials, etc.

These abstractions have 5 parts:

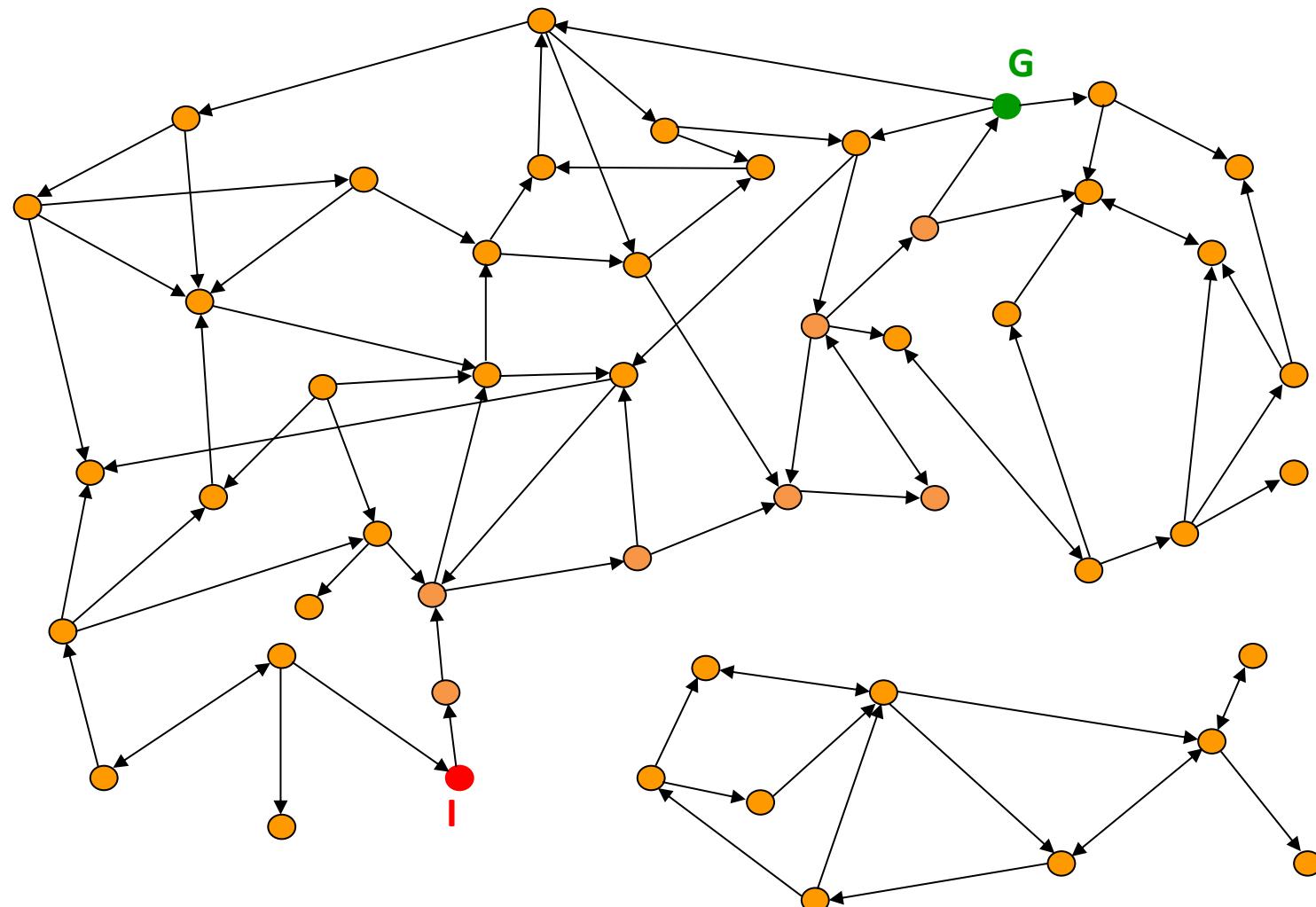
1. Set of states S
2. Initial state s_0
3. A function $\text{SUCC}: S \rightarrow 2^S$ that encodes possible transitions of the system
4. Set of goal states
5. A cost function that calculates how “expensive” a given set of moves is

Recall: Abstracting AI problems with graphs

- Nodes are states
- Start state, goal state(s)
- Edges encode SUCC
- Cost function
- A **solution** is a path from initial node to a goal
- The **cost** of a path is the sum of its edge costs
- An **optimal** solution is a path of minimum cost



Graph search



8-, 15-, 24-Puzzles

8-puzzle → 362,880 states

0.036 sec

15-puzzle → 2.09×10^{13} states

~ 55 hours

24-puzzle → 10^{25} states

> 10^9 years

100 million states/sec

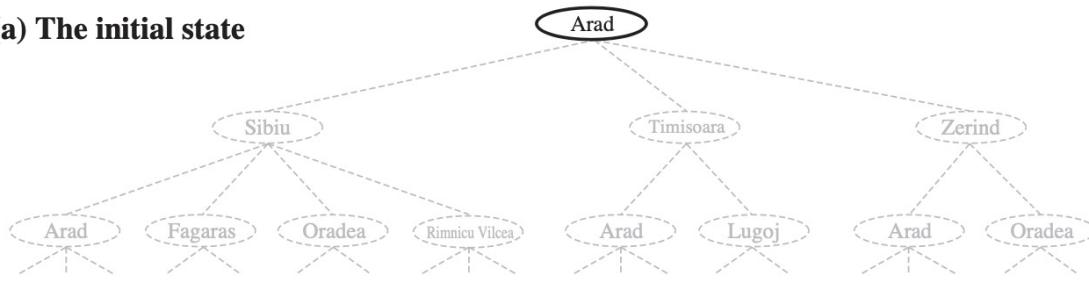
Tractability of search hinges on the ability to explore only a tiny portion of the state graph!

Intractability

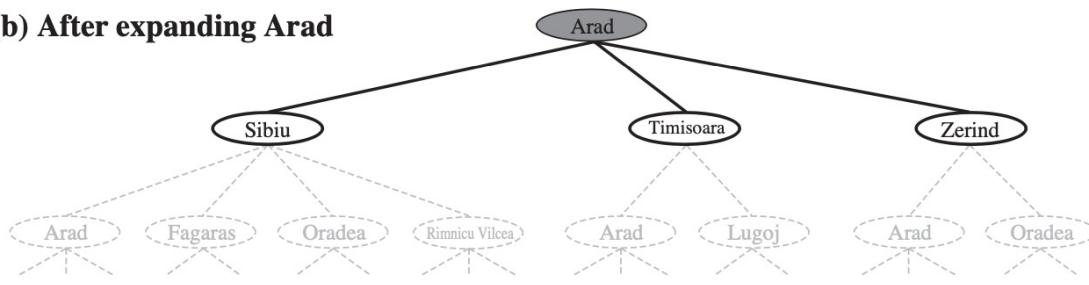
- Constructing the full state graph is intractable for most interesting problems
- n-puzzle: $(n+1)!$ states

Example of a search tree

(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu

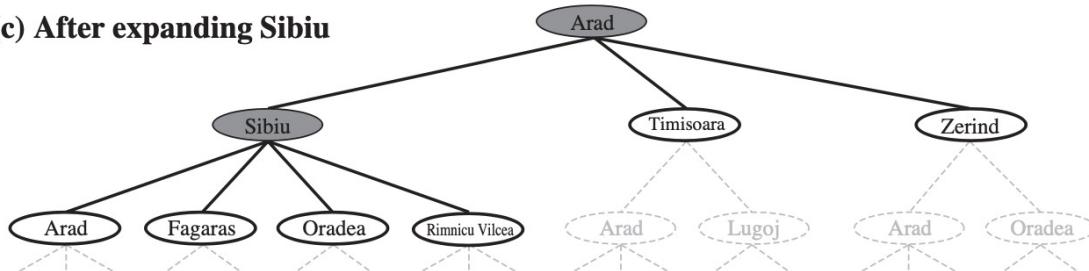


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

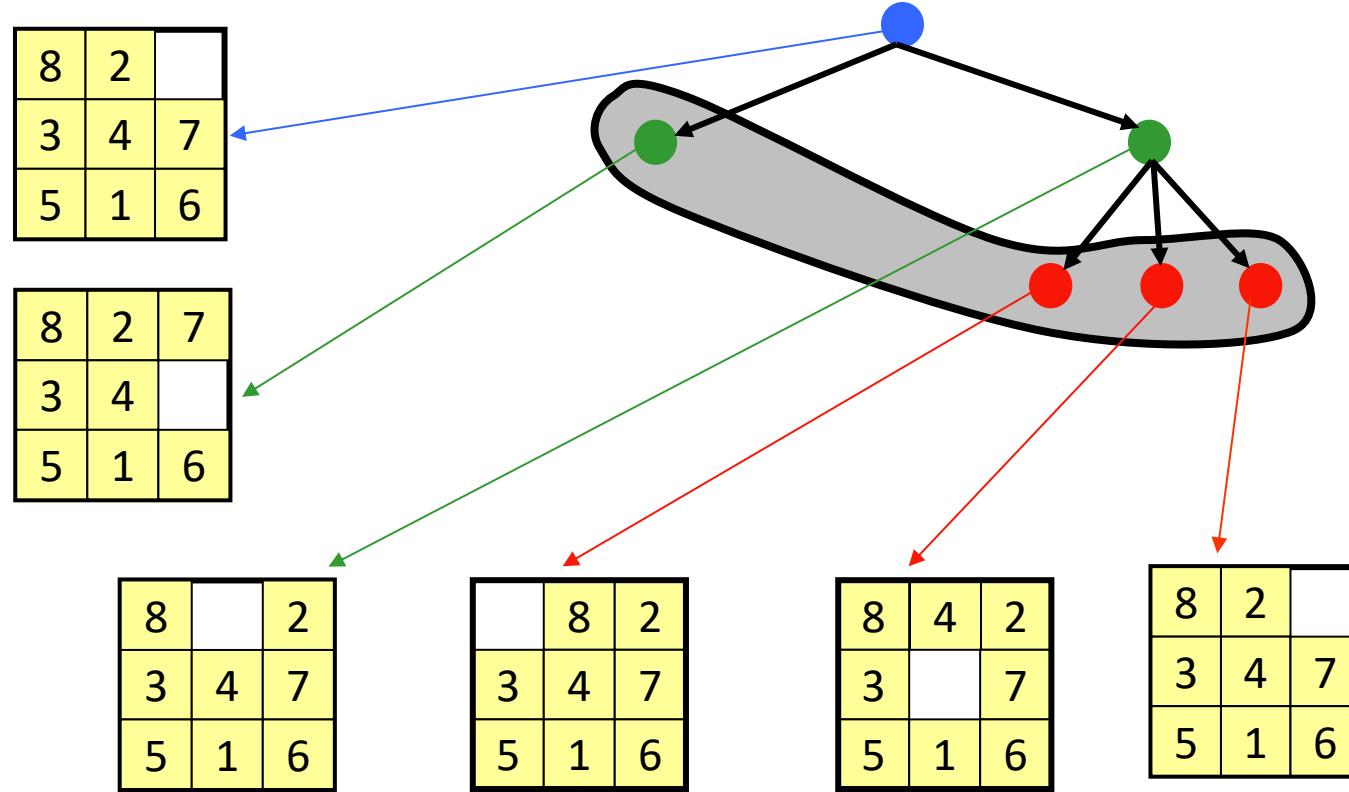
The root node of the tree corresponds to the initial state, $In(Arad)$.

We **expand** the current state; that is, applying each legal action to the current state, thereby **generating** a new set of states. In this case, we add three branches from the **parent node** $In(Arad)$ leading to three new **child nodes**: $In(Sibiu)$, $In(Timisoara)$, and $In(Zerind)$.

What to do now?

The set of all leaf nodes available for expansion at any given point is called the **frontier**.

Fringe or Frontier



Informal description of a search algorithm

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
        only if not in the frontier or explored set
```

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

Evaluation criteria for search algorithms

- We will consider different algorithms and evaluate them in terms of:
 - **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
 - **Optimality:** Does the strategy find the optimal solution (minimum cost path)?
 - **Time complexity:** How long does it take to find a solution?
 - **Space complexity:** How much memory is needed to perform the search?
 - size of the state space graph, $V + E$, where V is the set of vertices (nodes) of the graph and E is the set of edges (links).

Note about complexity

- Complexity is usually expressed in terms of three quantities:
 - b , the **branching factor** or maximum number of successors of any node;
 - d , the **depth** of the shallowest goal node (i.e., the number of steps along the path from the root);
 - m , the maximum length of any path in the state space.
- Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory.

Stacks, Queues, PQs, Oh my!

- Stack



- Queue



- Priority Queue

- You put (item, priority) pairs into queue
- You remove the highest-priority item from the queue

Blind vs. Heuristic Strategies

- **Blind** (or un-informed) strategies do not use properties of states to order FRINGE. All states are treated the same.
- **Heuristic** (or informed) strategies order FRINGE so that more “promising” states are considered first

Example

1	2	3
4	5	6
7	8	

Goal state

8	2	
3	4	7
5	1	6

STATE

N_1

For a **blind strategy**, N_1 and N_2 are just two nodes (at some position in the search tree)

1	2	3
4	5	
7	8	6

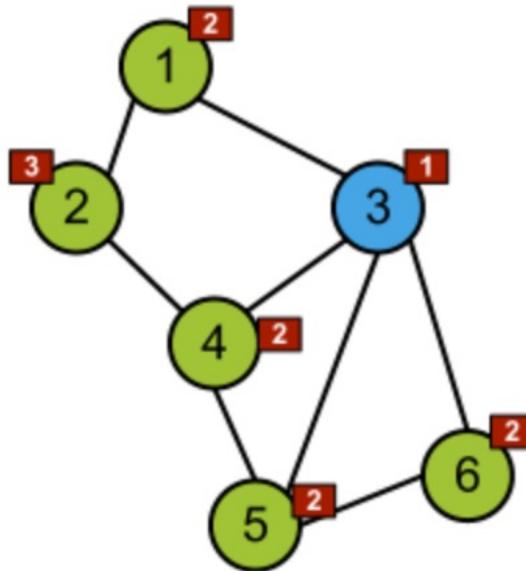
STATE

N_2

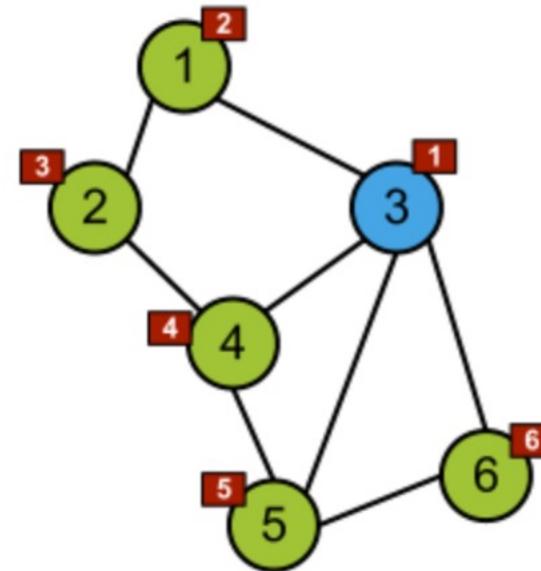
For a **heuristic strategy**, N_2 seems more promising than N_1 (fewer misplaced tiles)

Breadth-first search and Depth-first search

- We will cover two uninformed (blind) search algorithms:



Breadth-first search



Depth-first search

Breadth-first search

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

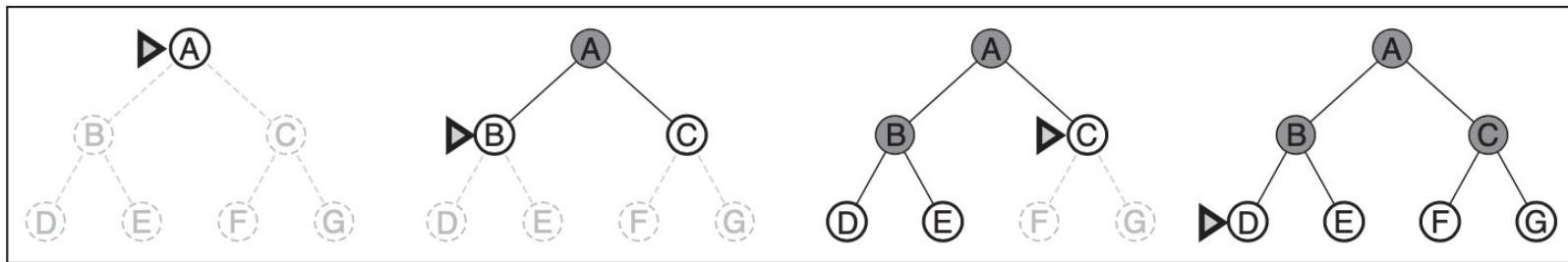


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Time and Memory Requirements

d	# Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	$\sim 10^6$	1 sec	100 Mb
8	$\sim 10^8$	100 sec	10 Gbytes
10	$\sim 10^{10}$	2.8 hours	1 Tbyte
12	$\sim 10^{12}$	11.6 days	100 Tbytes
14	$\sim 10^{14}$	3.2 years	10,000 Tbytes

Assumptions: $b = 10$; 1,000,000 nodes/sec; 100bytes/node

Depth-first search

Depth-first search always expands the *deepest* node in the current frontier of the search tree.

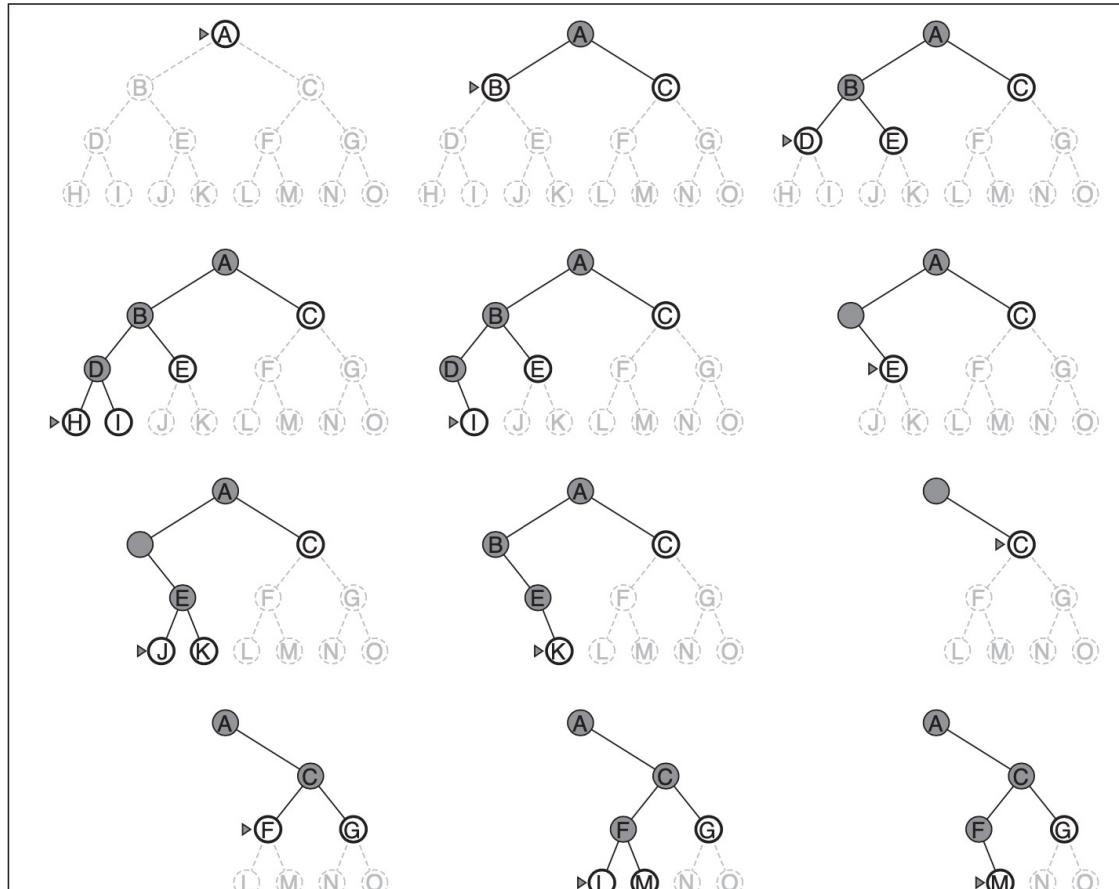


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Comparing uninformed search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.

Next class

- Heuristic (or informed) search

Refinements to BFS/DBS and informed (heuristic) search strategies

Announcements

- Assignment 0 (almost) released
 - Please make sure to login to your IU GitHub account <https://github.iu.edu/> before the end of the day (how about now?)
- Don't forget the Lecture 3 Review Quiz due Sep 5 at 11:59pm
- Are you getting Canvas notifications?
 - If not, you can change them; go to Profile -> Settings -> Notifications
 - Might consider notifications for Q&A Community and slack
- No class on September 5 (Labor day)

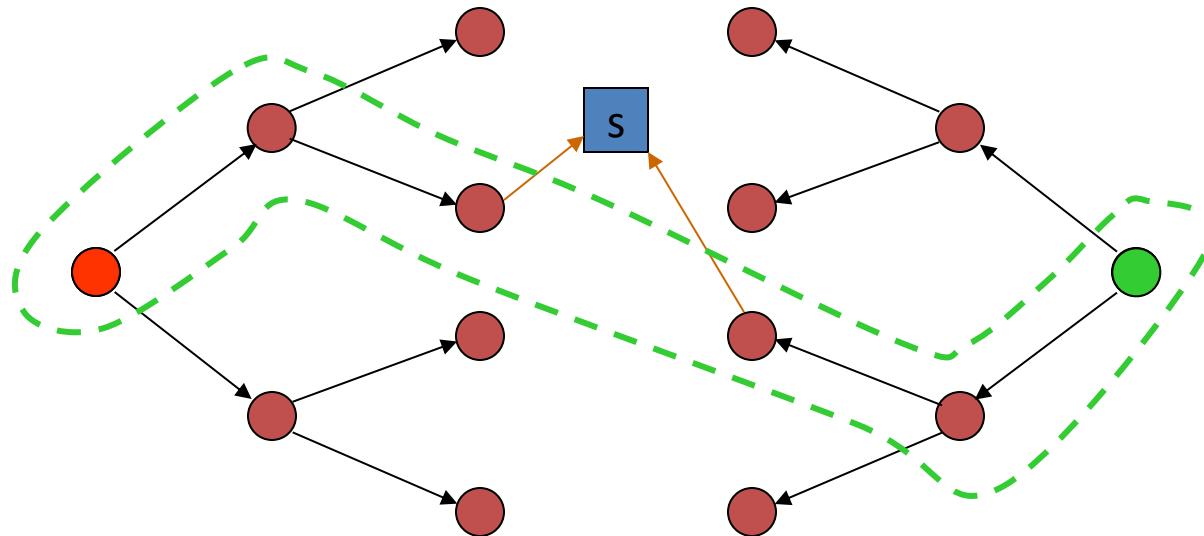
Refinements to BFS & DFS

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.

Bidirectional Strategy

2 fringe queues: FRINGE1 and FRINGE2



Time and space complexity is $O(b^{d/2}) \ll O(b^d)$
if both trees have the same branching factor b

Depth-Limited Search

For $k = 0, 1, 2, \dots$ do:

 Perform depth-first search with
 depth cutoff k

- Three possible outcomes:
 - Solution
 - Failure (no solution)
 - Cutoff (no solution within cutoff)

Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:
 - Store all states associated with **generated (expanded)** nodes in VISITED
 - If the state of a new node is in VISITED, then discard the node

Avoiding Revisited States

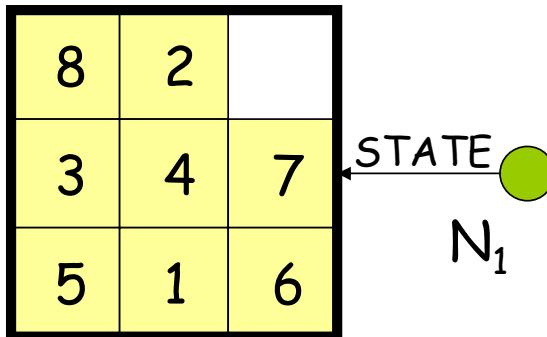
- Requires comparing state descriptions
 - Breadth-first search:
 - Store all states associated with **generated (expanded)** nodes in VISITED
 - If the state of a new node is in VISITED, then discard the node
- Implemented as hash-table
or as explicit data structure with flags

Heuristic search

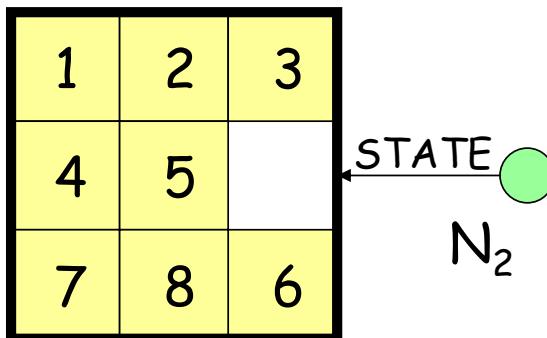
Heuristic vs blind search

1	2	3
4	5	6
7	8	

Goal state



For a **blind strategy**, N_1 and N_2 are just two nodes (at some position in the search tree)



For a **heuristic strategy**, N_2 seems more promising than N_1 (fewer misplaced tiles)

Best-First Search

- Explores most “promising” state from FRINGE first
 - Typically implemented with a priority queue
 - Requires *evaluation function* $f(s) \geq 0$ that estimates the **“cost” from initial state, through s, to a goal state**
- Typical choices of $f(s)$:
 - $f(s) = h(s)$, where h estimates the cost of reaching **a goal from s** (greedy best first search)
 - $f(s) = g(s) + h(s)$, where g is **cost of path from start state to s** and h estimates the **cost of reaching a goal from s**

Adding to our abstraction

1. Set of states S
2. Initial state s_0
3. A function $\text{SUCC}: S \rightarrow 2^S$ that encodes possible transitions of the system
4. Set of goal states
5. A cost function $c()$ that calculates how “expensive” a given set of moves is
6. A *heuristic function* $h()$ that estimates how “promising” a given state is

Heuristic function $h(s) \geq 0$

- Estimates the cost to go from state s to a goal state
 - The better the estimate, the more efficient the search
 - BUT we want to be able to compute it efficiently
 - Typically there are many possibilities, each with trade-offs
- How would you define $h(s)$ for the 8-puzzle?

5		8
4	2	1
7	3	6

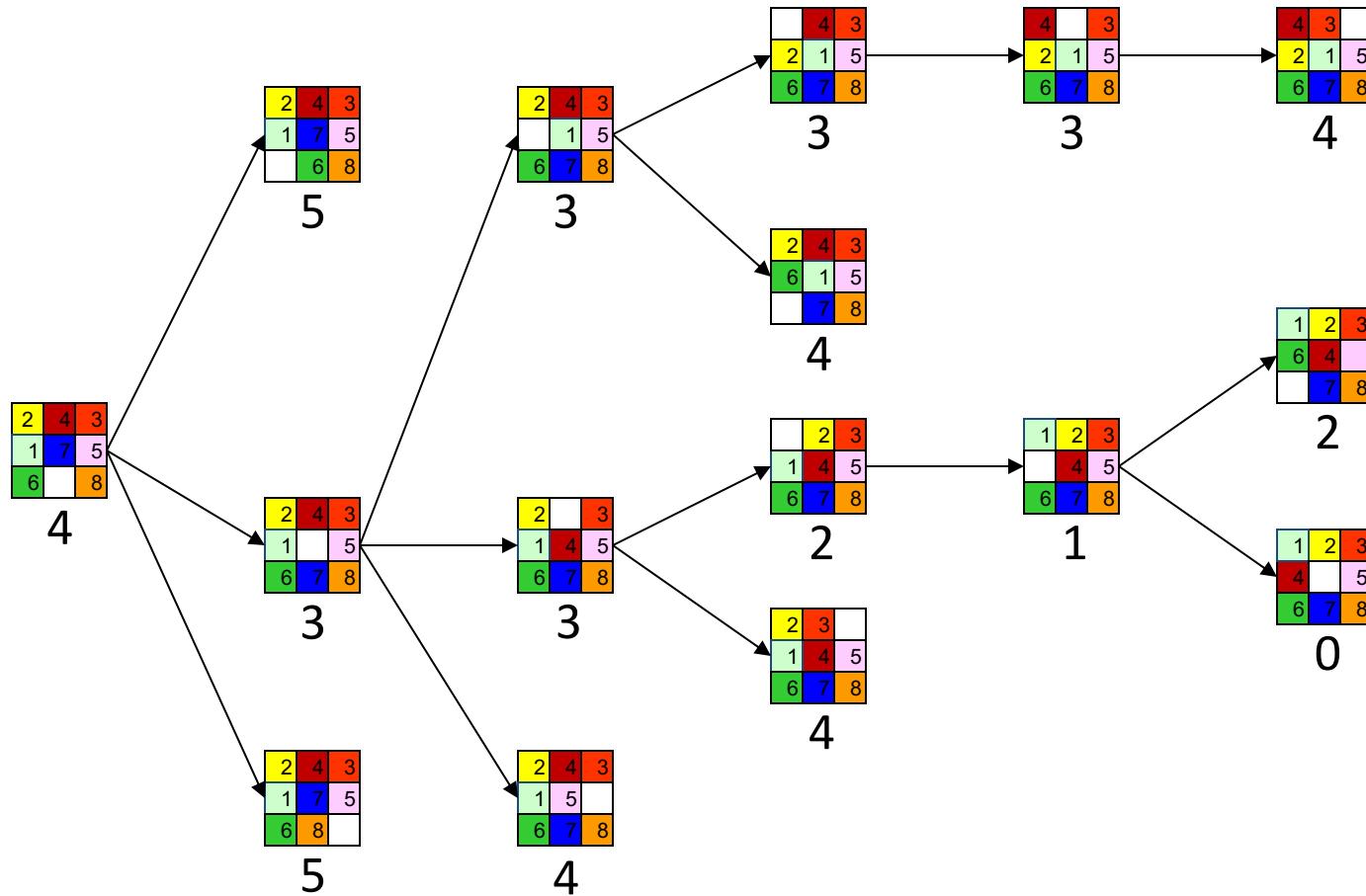
STATE(s)

1	2	3
4	5	6
7	8	

Goal state

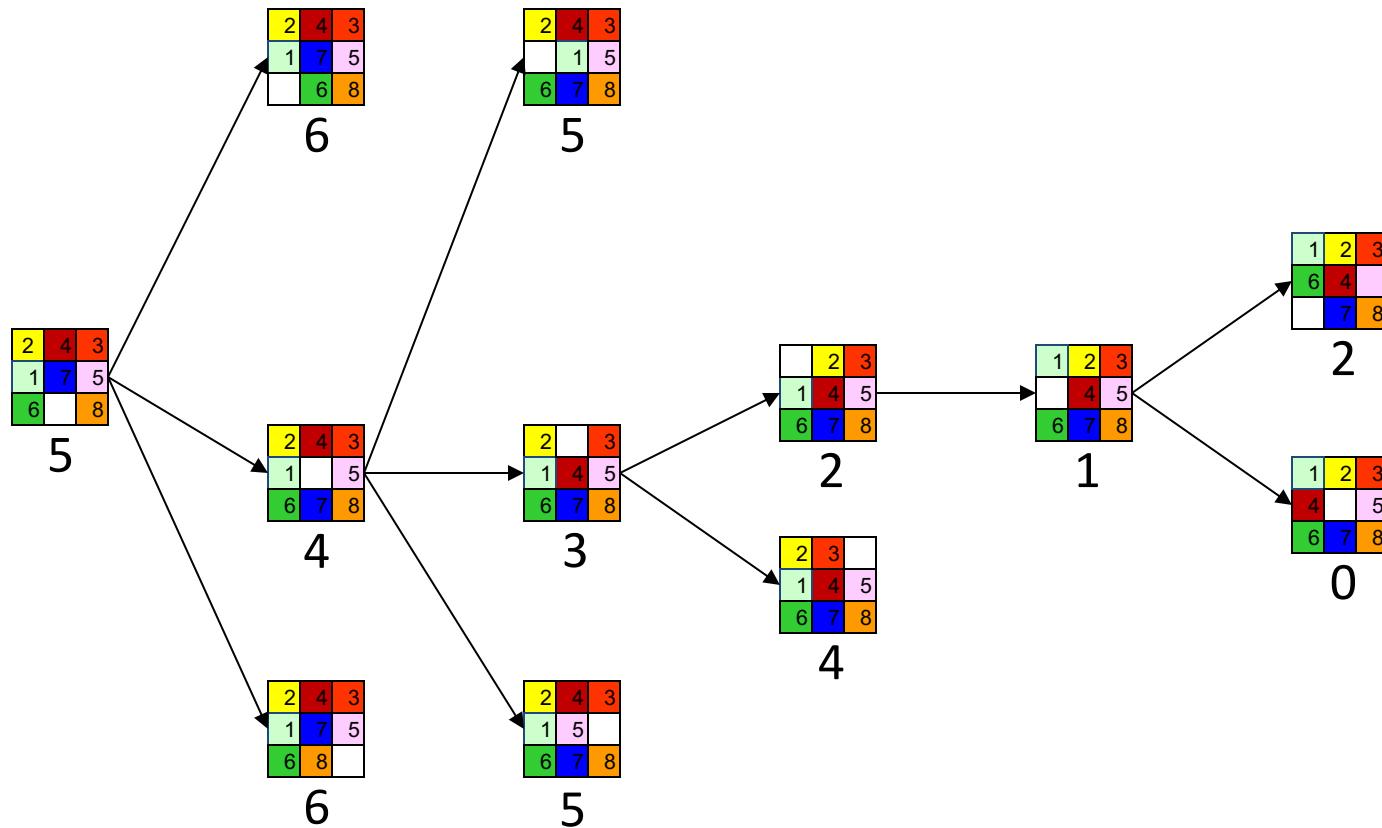
8-Puzzle

$f(s) = h(s) = \text{number of misplaced numbered tiles}$

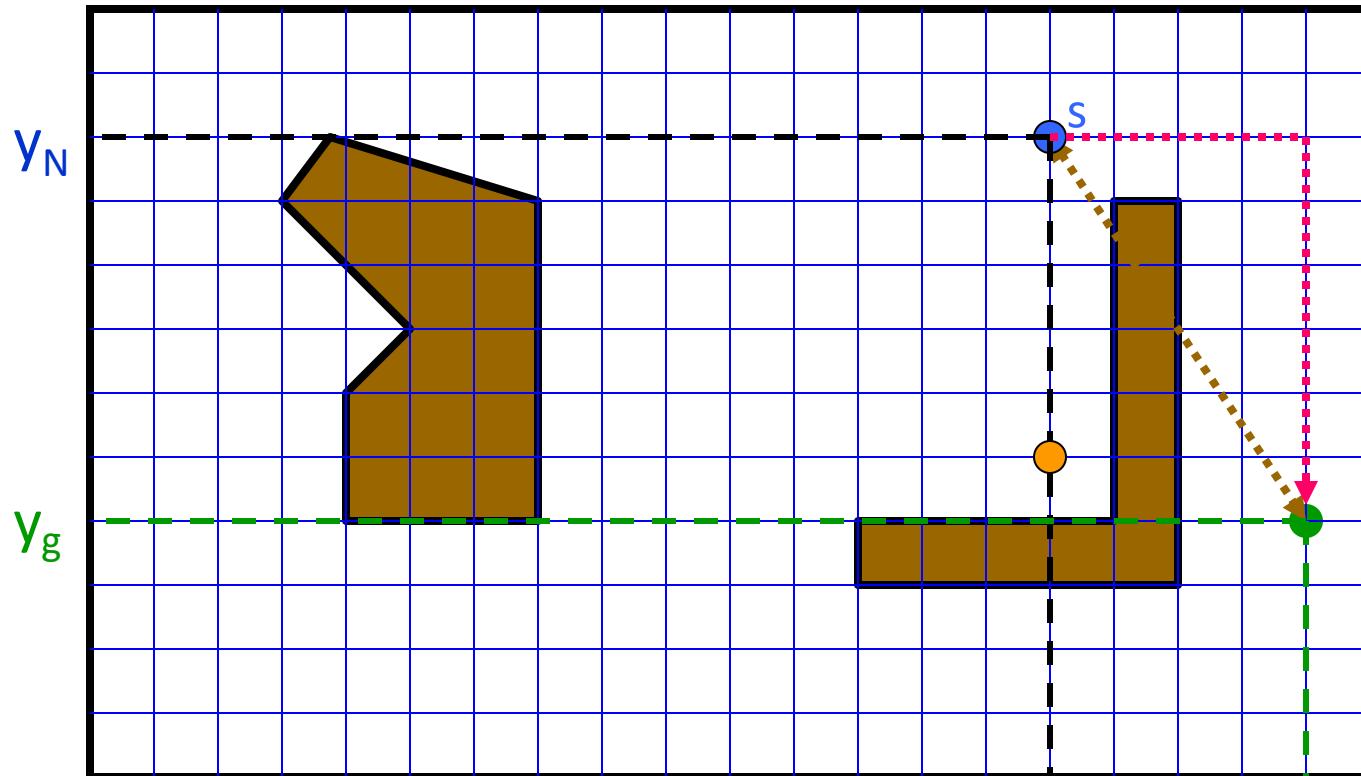


8-Puzzle

$f(N) = h(N) = \sum$ distances of numbered tiles to their goals



Robot Navigation



$$h_1(s) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$

x_N x_g
(L_2 or Euclidean distance)

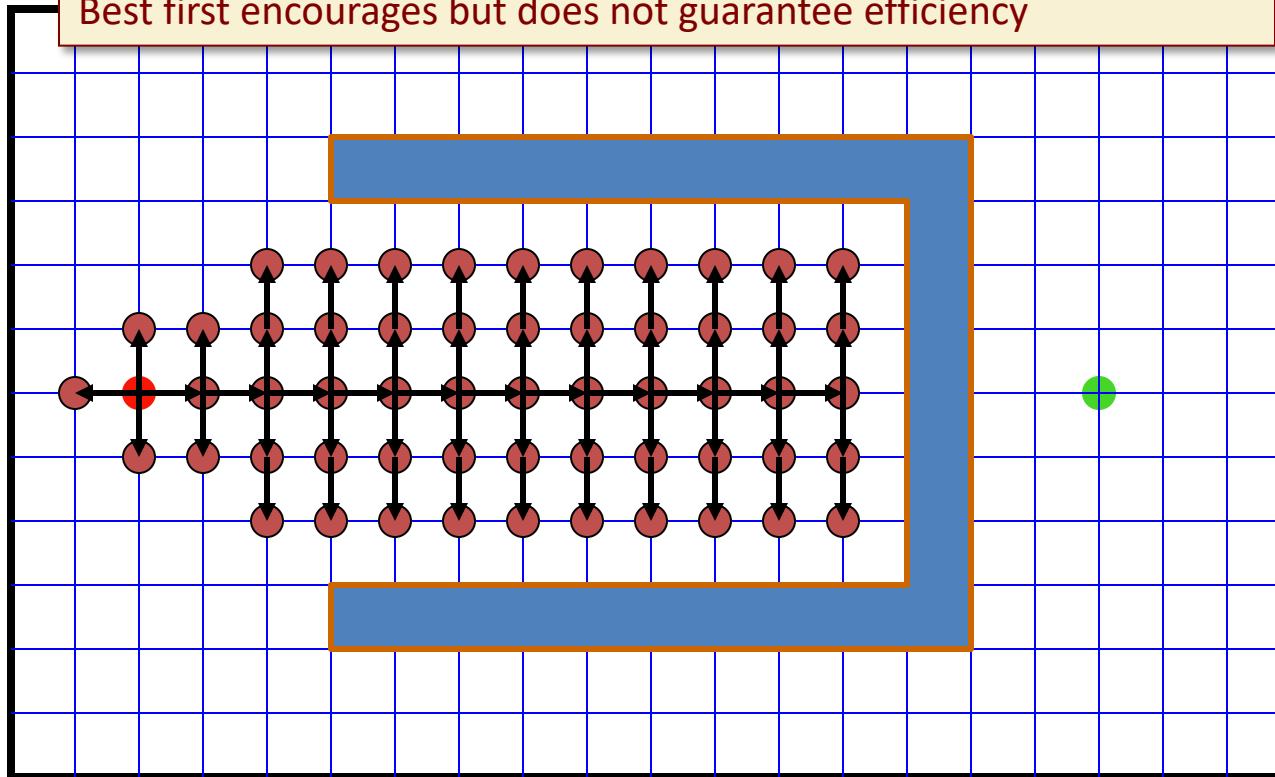
$$h_2(s) = |x_N - x_g| + |y_N - y_g|$$

y_N y_g
(L_1 or Manhattan distance)

What can go wrong?

Local-minimum problem:

Best first encourages but does not guarantee efficiency

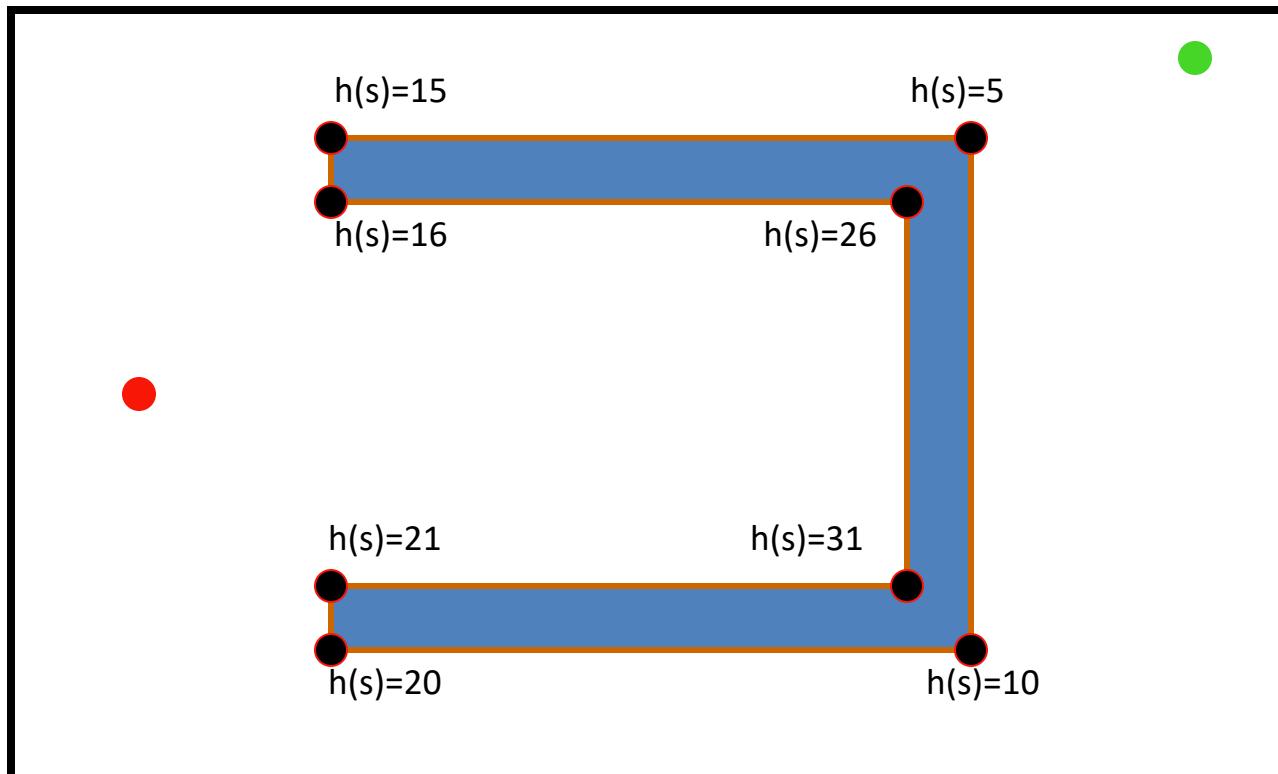


$h(s)$ = straight distance to the goal

Another idea: Pre-computation

E.g. assume environment is mostly static, use past best routes from some waypoints to estimate distances to goal

$h(s)$ = pre-computed distance from waypoint to goal



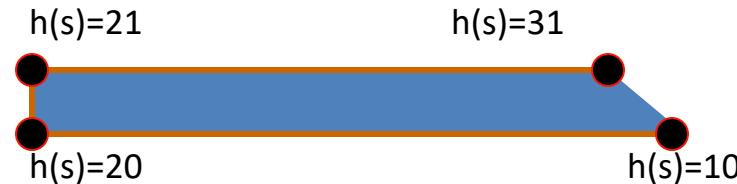
Another idea: Pre-computation

E.g. assume environment is mostly static, use past best routes from some waypoints to estimate distances to goal

$h(s)$ = pre-computed distance from waypoint to goal

Optimality problem:

Solutions found by best first are not necessarily optimal!



Admissible Heuristic

- An heuristic $h(s)$ is **admissible** if for any state s ,
$$0 \leq h(s) \leq h^*(s)$$
where $h^*(s)$ is the optimal cost from s to a goal.
- In other words, an admissible heuristic **never overestimates the cost to the goal**
 - We'll need to design $h(s)$ so that it's always less than $h^*(s)$, even though we don't know $h^*(s)$!

Which of these are admissible?

5		8
4	2	1
7	3	6

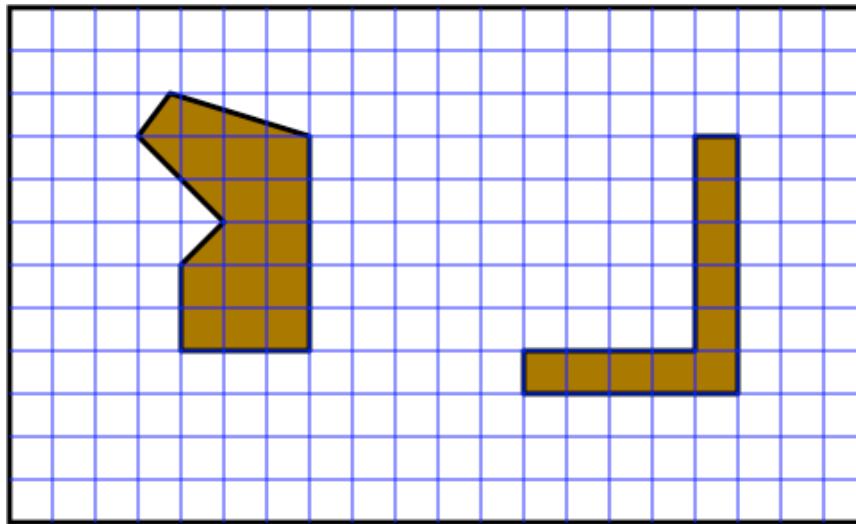
STATE(s)

1	2	3
4	5	6
7	8	

Goal state

- $h_1(N)$ = number of misplaced numbered tiles = 6
- $h_2(N)$ = sum of Manhattan distances of tiles to goal positions
 $= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13$

Which of these are admissible?



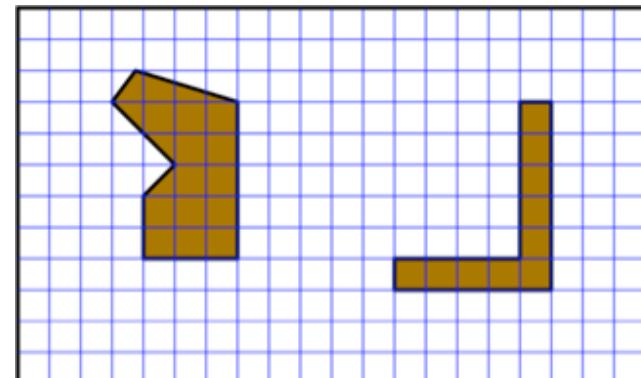
$$h_1(s) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$

$$h_2(s) = |x_N - x_g| + |y_N - y_g|$$

Assume that the agent can move diagonally

How to create an admissible h?

- Often a challenge! A good h is admissible, fast to compute, and still a good estimate
- Admissible heuristics are often optimal solutions to simplified (relaxed) problems (with constraints ignored). E.g. for robot navigation:
 - Manhattan distance ignores obstacles
 - Euclidean distance ignores obstacles and constraint that robot moves on a grid
- Much more on this later!



$$h_1(s) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$

$$h_2(s) = |x_N - x_g| + |y_N - y_g|$$

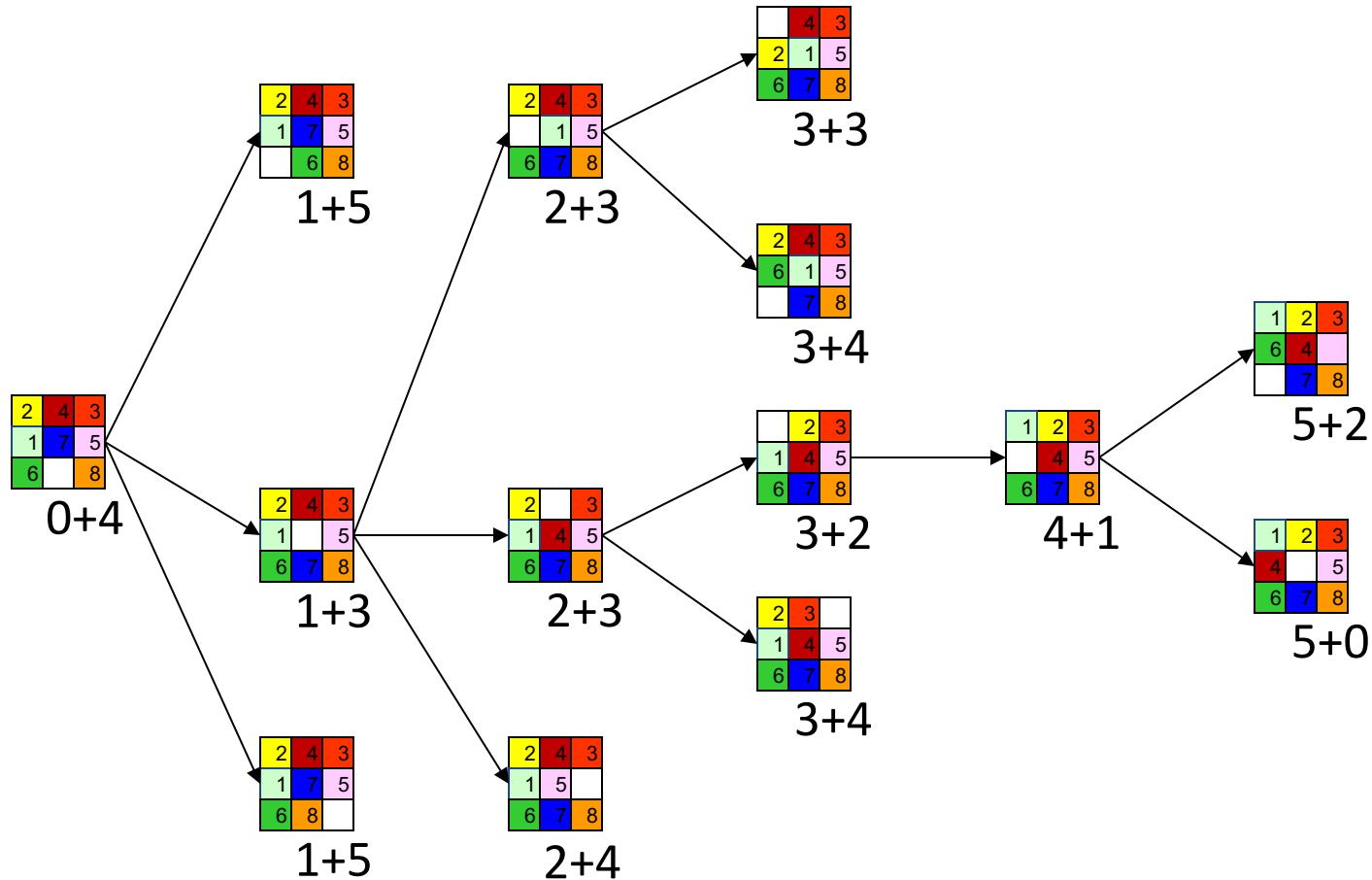
A* Search

- Best First Search with $f(s) = g(s) + h(s)$, where:
 - $g(s)$ = cost of best path found so far to s
 - $h(s)$ = **admissible** heuristic function
1. If $\text{GOAL?}(\text{initial-state})$ then return **initial-state**
 2. $\text{INSERT}(\text{initial-node}, \text{FRINGE})$
 3. Repeat:
 4. If $\text{empty}(\text{FRINGE})$ then return **failure**
 5. $s \leftarrow \text{REMOVE}(\text{FRINGE})$
 6. If $\text{GOAL?}(s)$ then return **s** and/or path
 7. For every state s' in $\text{SUCC}(s)$:
 8. $\text{INSERT}(s', \text{FRINGE})$

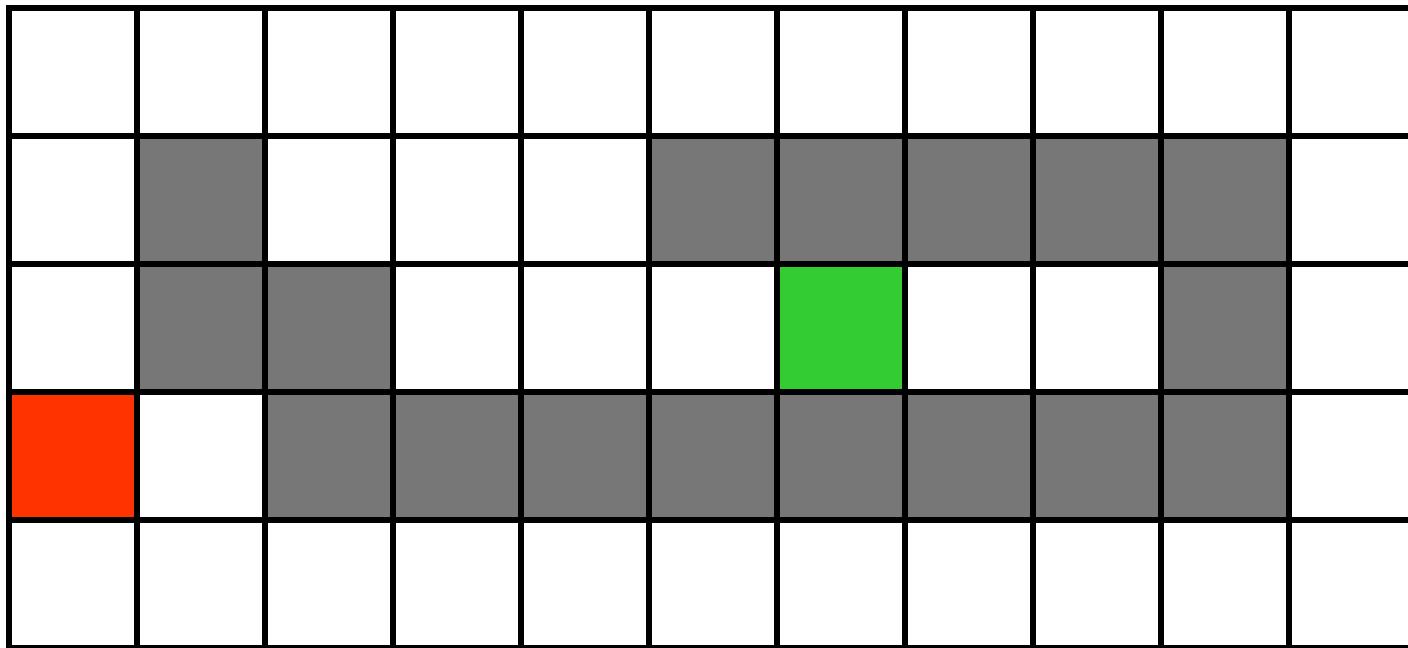
8-Puzzle

$$f(s) = g(s) + h(s)$$

with $h(s)$ = number of misplaced numbered tiles



Robot Navigation



Robot Navigation

$f(s) = h(s)$, with $h(s)$ = Manhattan distance to the goal
(not A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

$f(s) = h(s)$, with $h(s)$ = Manhattan distance to the goal
(not A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

$f(s) = g(s) + h(s)$, with $h(s)$ = Manhattan distance to goal (A*)

8+3	7+4	6+5	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

1. Is A* complete?
2. Is A* optimal?
3. What is the running time of A*?
4. What are the memory requirements of A*?

Answer in the next class
(and in the book, ch. 3.5)

Next class

- Finish heuristic (informed) search
- Local search

Heuristic search wrap-up, and local search

Announcements

- Assignment 0 released
- Survey released (closing tonight)

A* Search

- Best First Search with $f(s) = g(s) + h(s)$, where:
 - $g(s)$ = cost of best path found so far to s
 - $h(s)$ = **admissible** heuristic function
1. If $\text{GOAL?}(\text{initial-state})$ then return **initial-state**
 2. $\text{INSERT}(\text{initial-node}, \text{FRINGE})$
 3. Repeat:
 4. If $\text{empty}(\text{FRINGE})$ then return **failure**
 5. $s \leftarrow \text{REMOVE}(\text{FRINGE})$
 6. If $\text{GOAL?}(s)$ then return **s** and/or path
 7. For every state s' in $\text{SUCC}(s)$:
 8. $\text{INSERT}(s', \text{FRINGE})$

Reminder: Admissible Heuristic

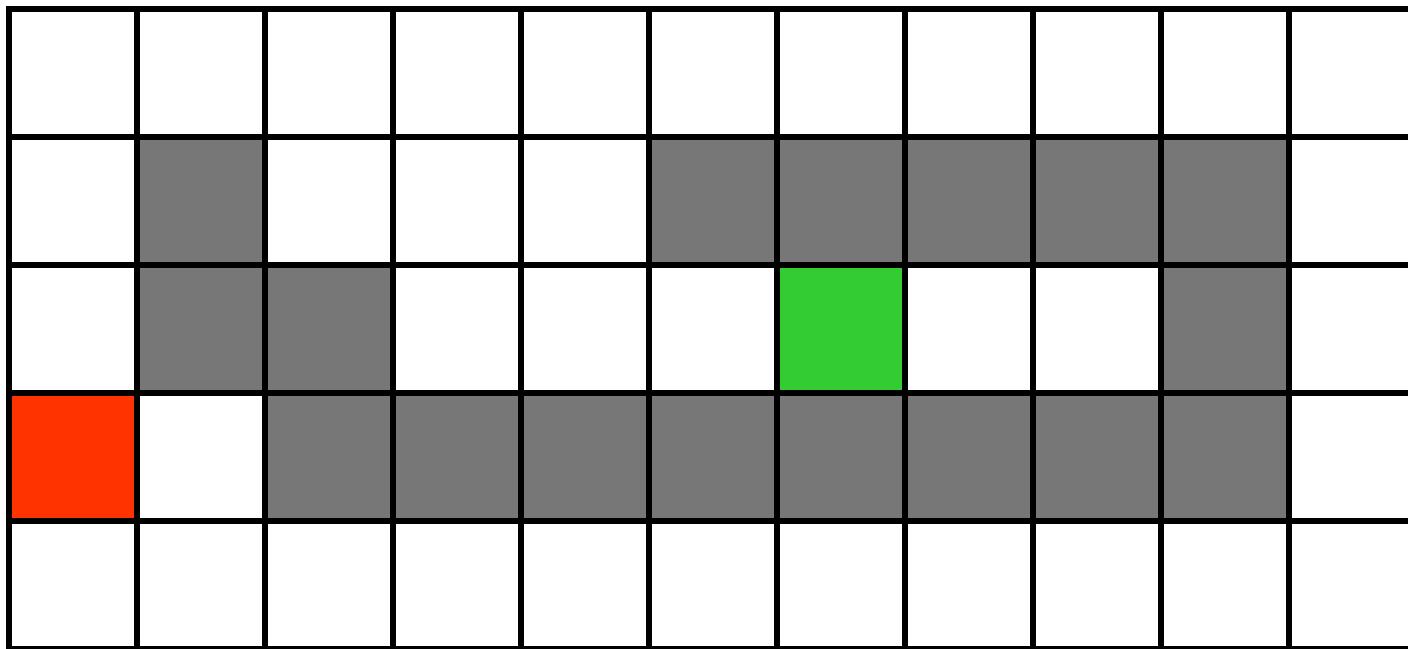
- An heuristic $h(s)$ is **admissible** if for any state s ,

$$0 \leq h(s) \leq h^*(s)$$

where $h^*(s)$ is the optimal cost from s to a goal.

- In other words, an admissible heuristic **never overestimates the cost to the goal**
 - We'll need to design $h(s)$ so that it's always less than $h^*(s)$, even though we don't know $h^*(s)!$

Robot Navigation



Robot Navigation

$f(s) = h(s)$, with $h(s)$ = Manhattan distance to the goal
(not A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

$f(s) = h(s)$, with $h(s)$ = Manhattan distance to the goal
(not A*)

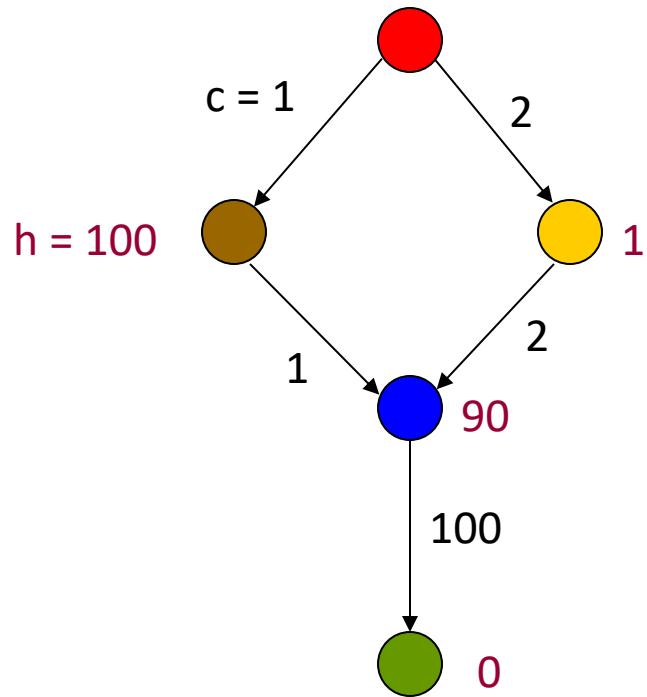
8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

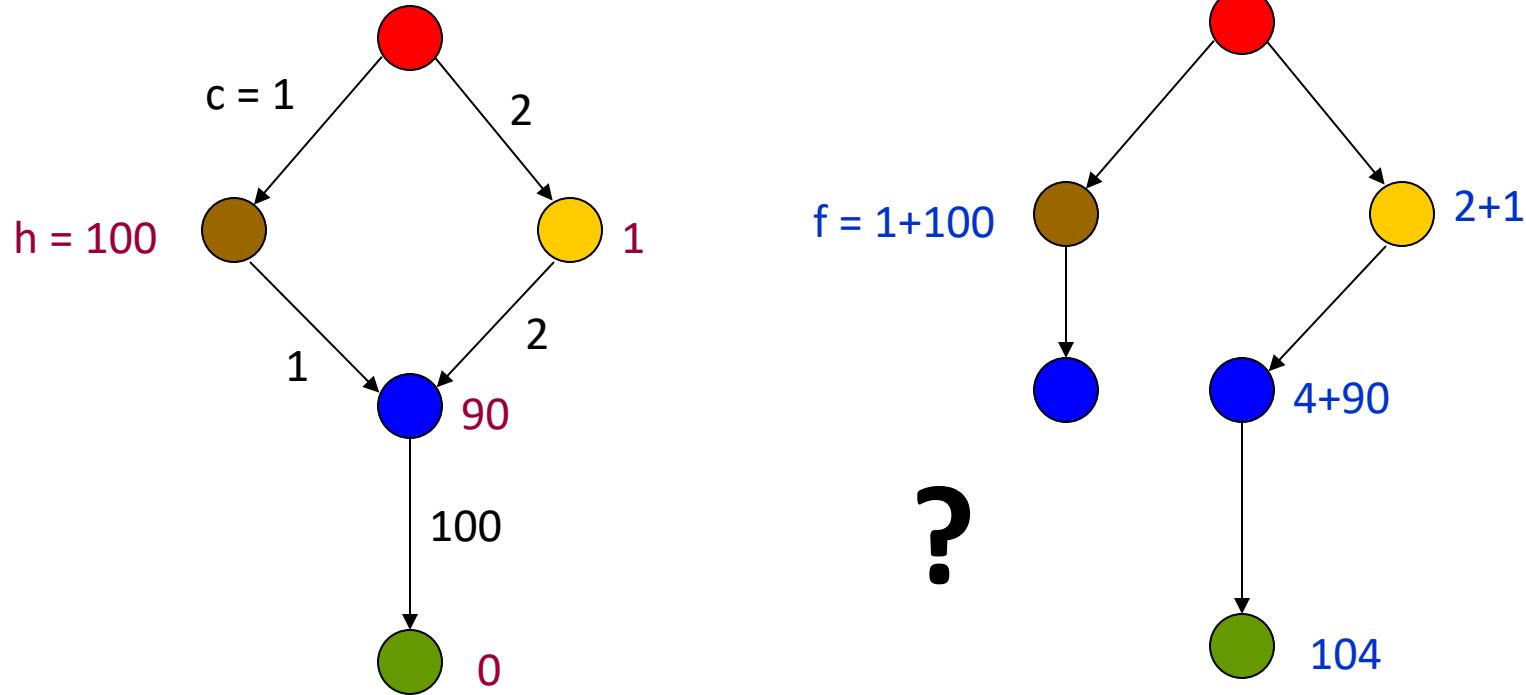
$f(s) = g(s) + h(s)$, with $h(s)$ = Manhattan distance to goal (A*)

8+3	7+4	6+5	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

What to do with revisited states?



What to do with revisited states?



If we discard this new node, A* expands the goal next, returning a non-optimal solution

1. Is A* complete?
2. Is A* optimal?
3. What is the running time of A*?
4. What are the memory requirements of A*?

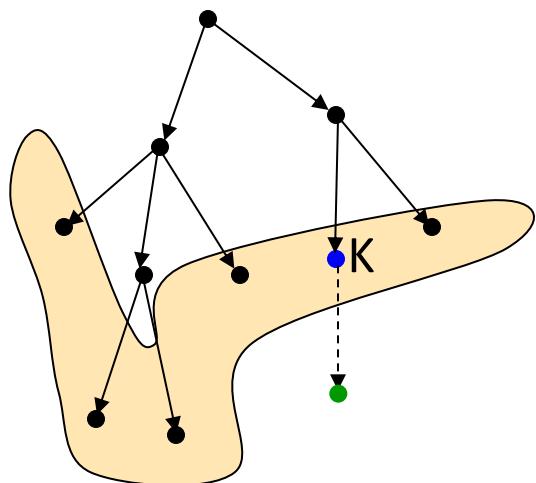
A* is complete and optimal if:

- Duplicate states are revisited, and
- h is admissible.
- Proof sketch:
 - First show that if a solution exists, A* terminates and finds a solution.
 - Then show that whenever A* expands a node, the path to that node is optimal.

A* is complete and optimal if h is admissible.

Proof sketch:

- If a solution exists, A* terminates and returns a solution

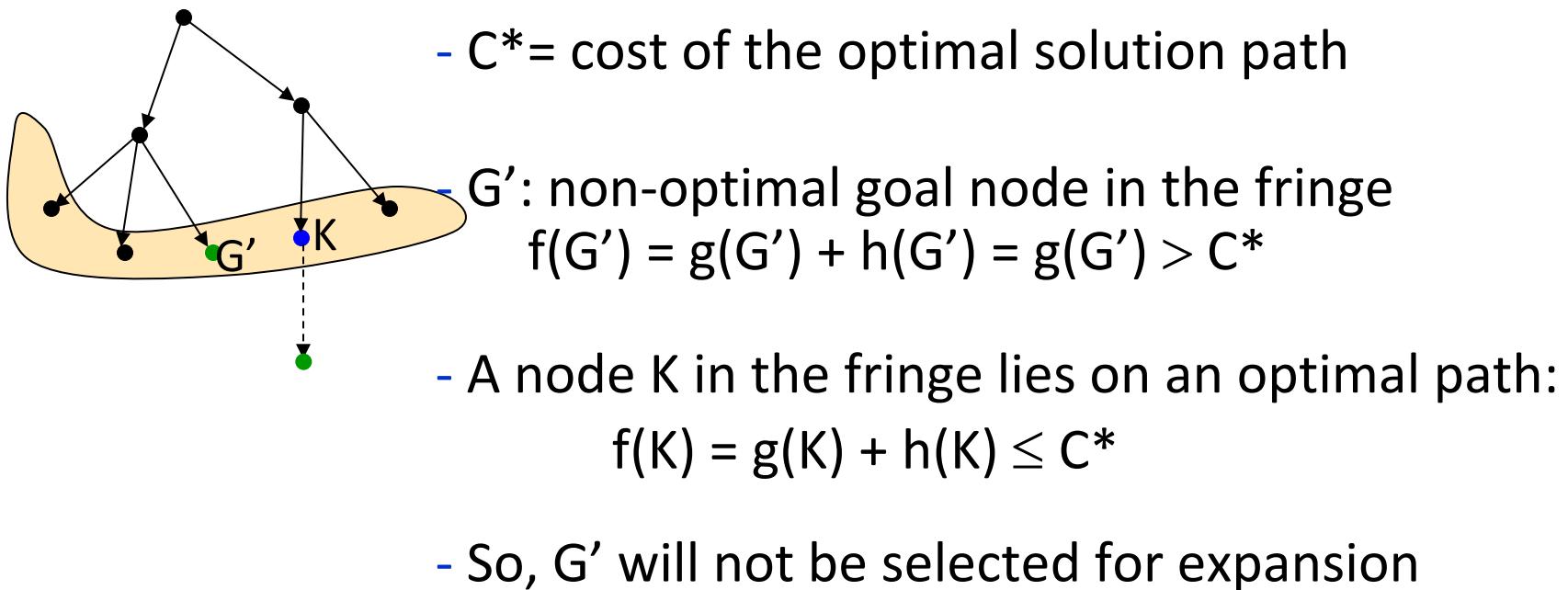


- As long as A* hasn't terminated, a node K on the fringe lies on a solution path
- Since each node expansion increases the length of one path, K will eventually be selected for expansion, unless a solution is found along another path

A* is complete and optimal if h is admissible.

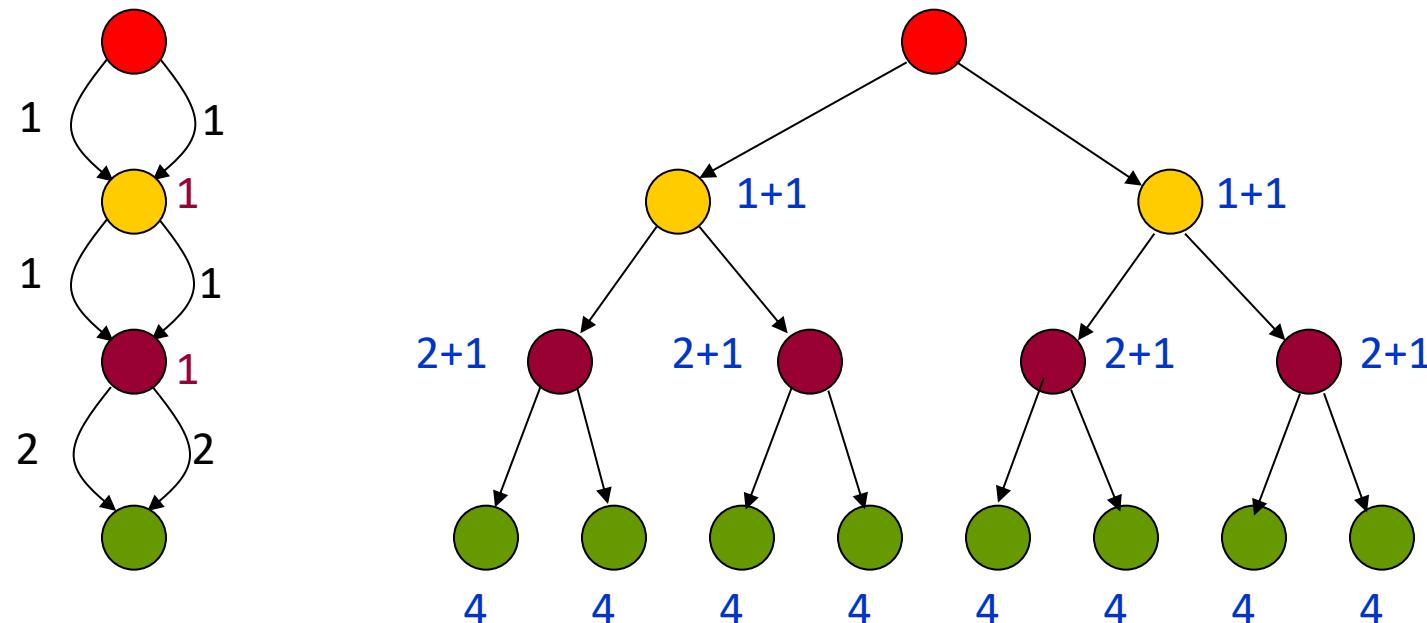
Proof sketch:

- Whenever A* chooses to expand a goal node, the path to this node is optimal



Complexity of A*

- A* expands all nodes with $f(s) < C^*$
 - May also expand non-goal states with $f(s) = C^*$
 - May be an exponential number of nodes unless the heuristic is sufficiently *accurate*

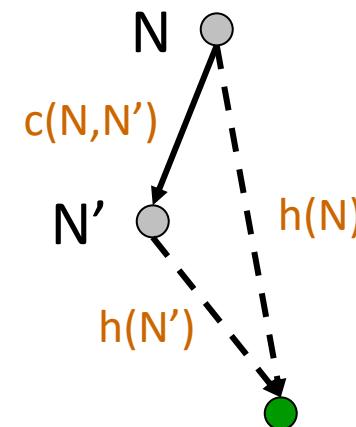


Consistent Heuristic

- An **admissible** heuristic h is **consistent** (or **monotone**) if for each node N and each child N' of N :

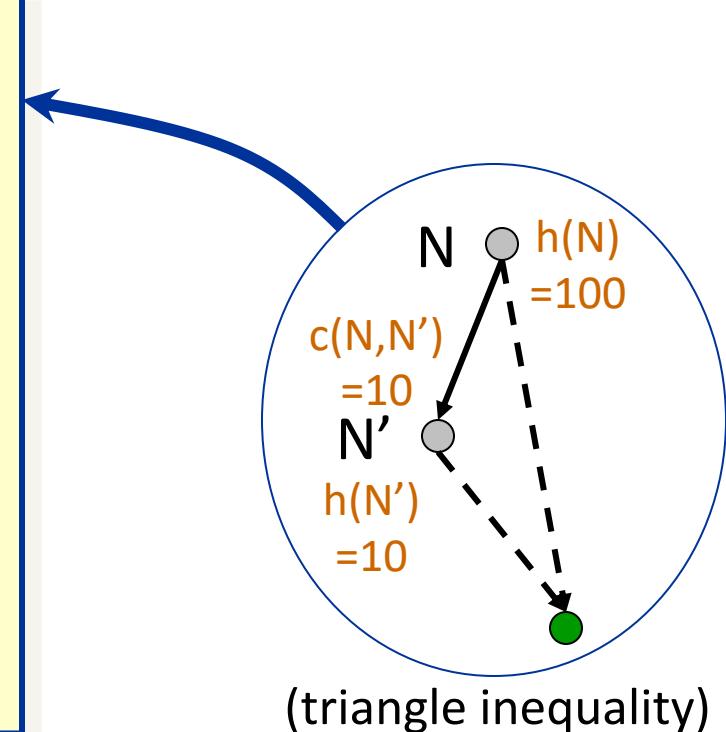
$$h(N) \leq c(N, N') + h(N') \quad (\text{triangle inequality})$$

Or: $h(N) - h(N') \leq c(N, N')$



Consistency Violation

If h says that N is 100 units from the goal, then moving from N along an edge costing 10 units should **not** lead to a node N' that h estimates to be 10 units away from the goal



Should satisfy: $h(N) - h(N') \leq c(N, N')$

8-Puzzle

5		8
4	2	1
7	3	6

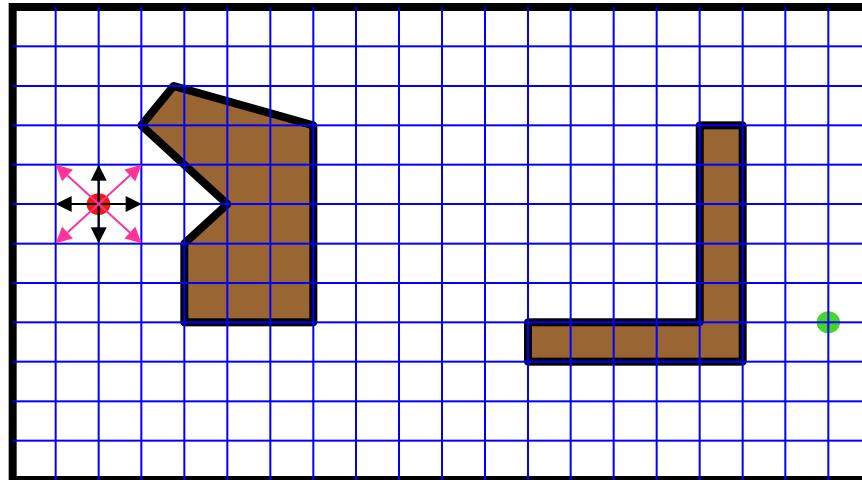
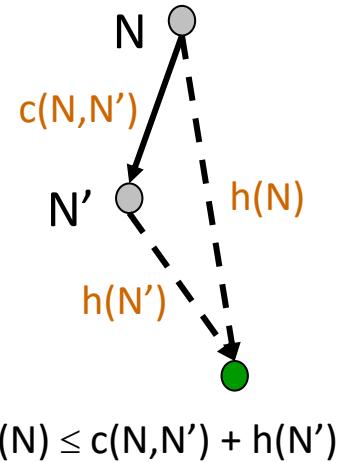
STATE(N)

1	2	3
4	5	6
7	8	

goal

- $h_1(N)$ = number of misplaced tiles
- $h_2(N)$ = sum of the (Manhattan) distances
of every tile to its goal position

Robot Navigation



Cost of one horizontal/vertical step = 1
Cost of one diagonal step = $\sqrt{2}$

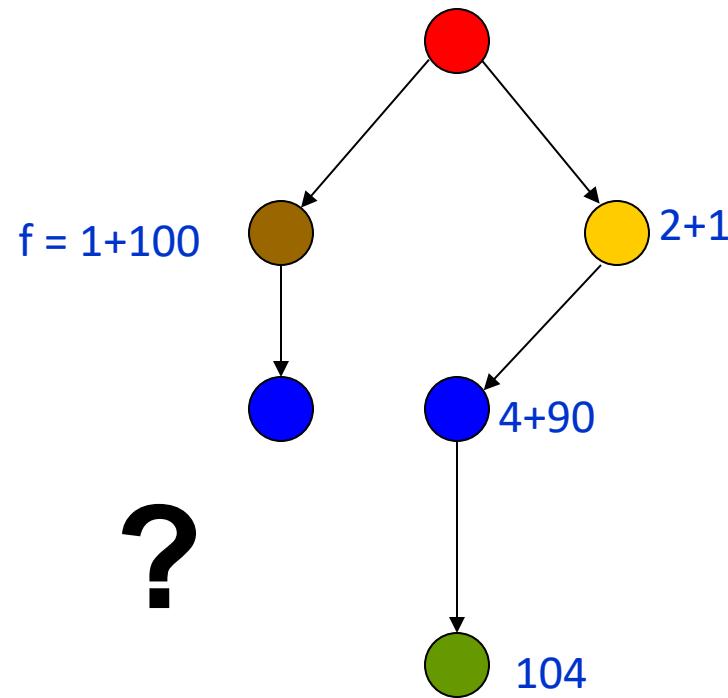
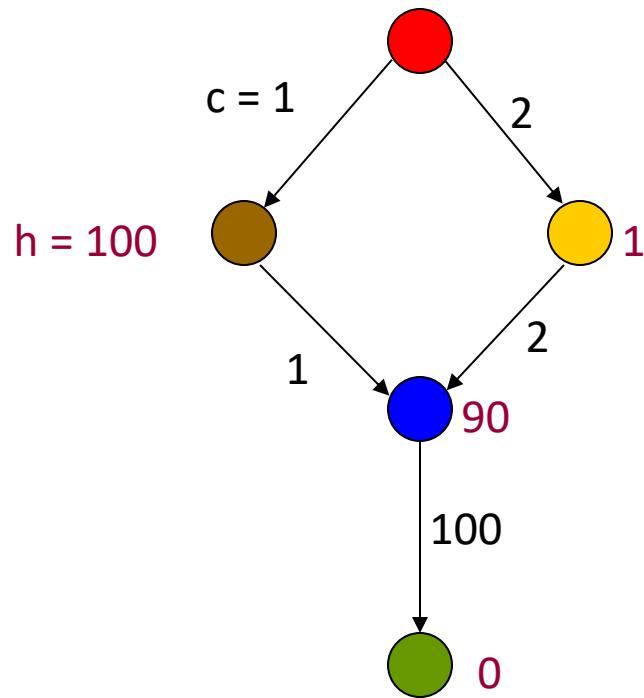
$$h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$

$$h_2(N) = |x_N - x_g| + |y_N - y_g|$$

Admissibility and Consistency

- A consistent heuristic is also admissible
- An admissible heuristic may not be consistent, but many admissible heuristics are consistent

Revisiting - Is $h(\cdot)$ consistent?



Updated Algorithm

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node, FRINGE)
3. Repeat:
4. If empty(FRINGE) then return failure
5. $s \leftarrow \text{REMOVE}(\text{FRINGE})$
6. INSERT(s , CLOSED)
7. If GOAL?(s) then return s and/or path
8. For every state s' in SUCC(s):
 9. If s' in CLOSED, discard s'
 10. If s' in FRINGE with larger s' , remove from FRINGE
 11. If s' not in FRINGE, INSERT(s' , FRINGE)

A* is optimal if

- h is **admissible** (but not necessarily consistent)

– Revisited states not discarded

- h is **consistent**

– (Many) revisited states discarded:

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node, FRINGE)
3. Repeat:
 4. If empty(FRINGE) then return failure
 5. $s \leftarrow \text{REMOVE}(\text{FRINGE})$
 6. If GOAL?(s) then return s and/or path
 7. For every state s' in SUCC(s):
 8. INSERT(s' , FRINGE)

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node, FRINGE)
3. Repeat:
 4. If empty(FRINGE) then return failure
 5. $s \leftarrow \text{REMOVE}(\text{FRINGE})$
 6. INSERT(s , CLOSED)
 7. If GOAL?(s) then return s and/or path
 8. For every state s' in SUCC(s):
 9. If s' in CLOSED, discard s'
 10. If s' in FRINGE with larger s' , remove from FRINGE
 11. If s' not in FRINGE, INSERT(s' , FRINGE)

Heuristic Accuracy

- If h_1 and h_2 are consistent heuristics such that $h_1(N) \leq h_2(N)$ for all nodes N , then h_2 is more **accurate** (or **informative**) than h_1
 - The more accurate h is, the less work A* has to do!

5		8
4	2	1
7	3	6

STATE(N)

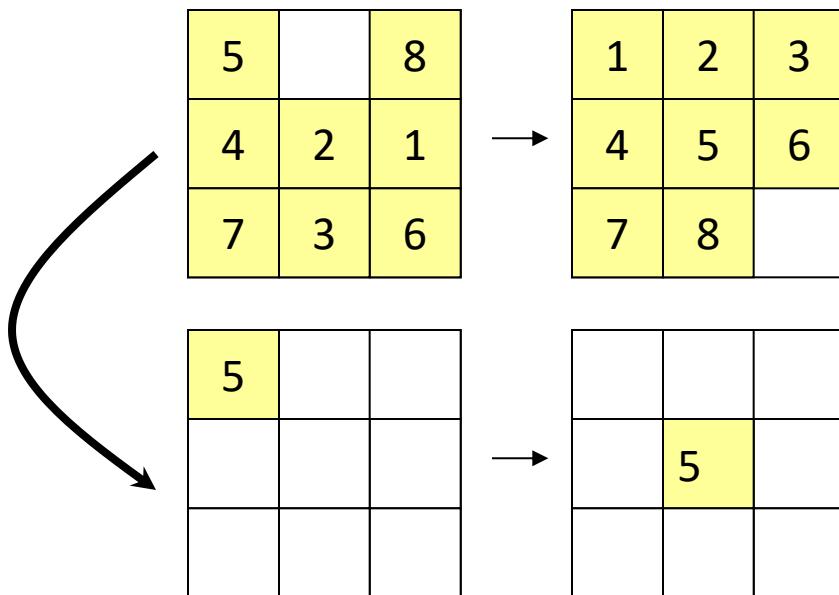
1	2	3
4	5	6
7	8	

Goal state

- $h_1(N)$ = number of misplaced tiles
- $h_2(N)$ = sum of distances of every tile to its goal position
- Which is more accurate?

How to create good heuristics?

- One approach: solve “relaxed” problems that ignore some constraints
 - E.g. ignore interactions among parts of the problem
 - In the 8-puzzle, the sum of the distances of each tile to its goal position (h_2) corresponds to solving 8 simple problems:

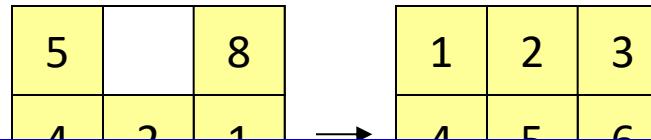


d_i is length of shortest path to move tile i to its goal position, ignoring the other tiles, and
 $h_2 = \sum_{i=1, \dots, 8} d_i$

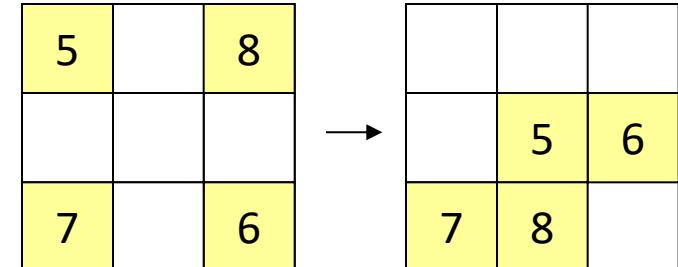
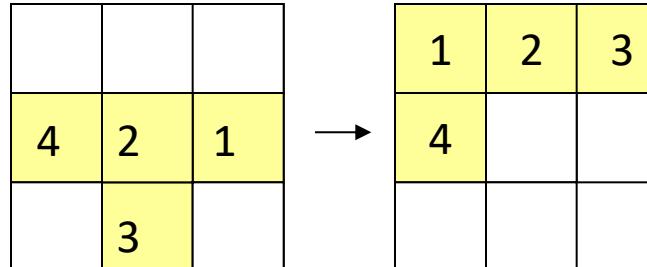
Can we do better?

- For example, we could consider two more complex relaxed problems:

d_{1234} = length of shortest
path
to go
Ignore



→ Several order-of-magnitude speedups
for the 15- and 24-puzzle (see R&N)



- $h = d_{1234} + d_{5678}$ [disjoint pattern heuristic]
- These distances can be pre-computed and stored
[Each requires generating a tree of 3,024 states (breadth-first search)]

Experimental Results

(see R&N for details)

- Random 8-puzzles with:
 - h_1 = number of misplaced tiles
 - h_2 = sum of distances of tiles to their goal positions
- Average “effective branching factors” based on actual # of nodes expanded:

d	IDS	A_1^*	A_2^*
2	2.45	1.79	1.79
6	2.73	1.34	1.30
12	2.78 (3,644,035)	1.42 (227)	1.24 (73)
16	--	1.45	1.25
20	--	1.47	1.27
24	--	1.48 (39,135)	1.26 (1,641)

Next class

- Local search
- Start with adversarial search and game playing

Local search

Announcements

- Assignment 0 due on Friday
- A1 coming next week
 - Please watch out for a team info

Local Search

- No search tree – just remember current state
 - No need for fringe = much less memory
 - Applicable to pathless problems (e.g. 8-queens)
 - Practical examples: integrated-circuit design, factory floor layout, telecommunications network optimization, and portfolio management
- Key idea: Try to find s that minimizes $h(s)$
 - Since $h(goal) = 0$

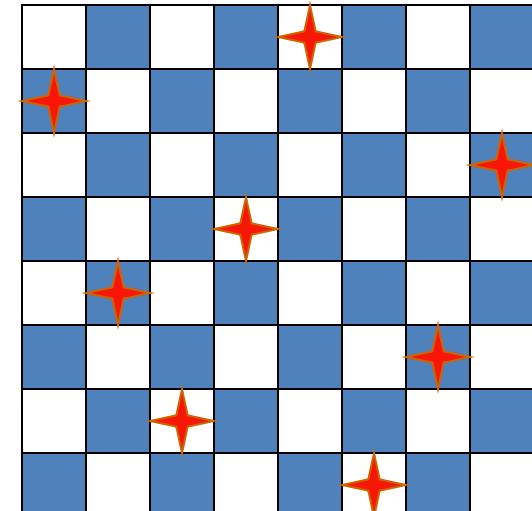
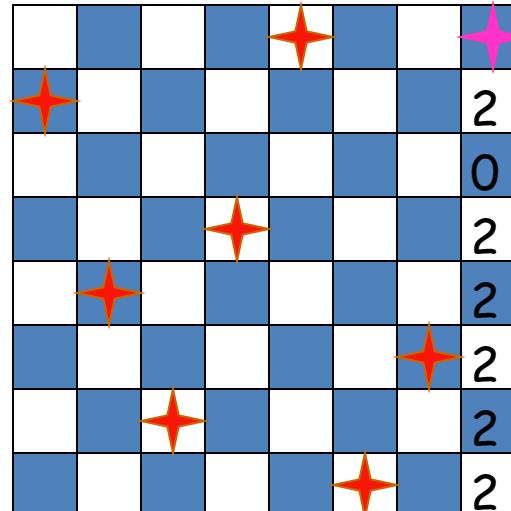
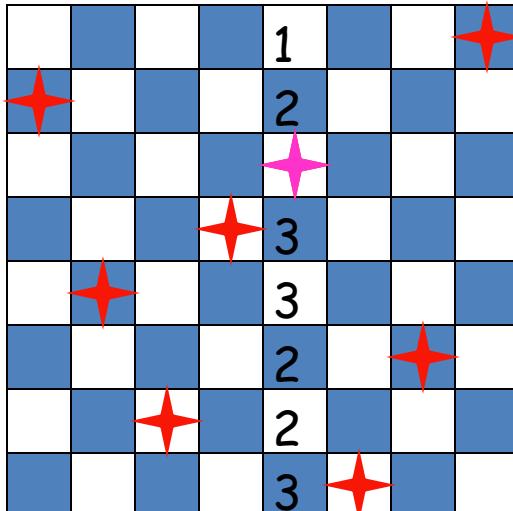
Steepest Descent

1. $S \leftarrow$ initial state
2. Repeat:
3. $S' \leftarrow \arg \min_{S' \in \text{succ}(S)} \{h(S')\}$
4. if GOAL?(S') return S'
5. if $h(S') < h(S)$ then $S \leftarrow S'$ else failure

Application: 8-Queen

Repeat n times:

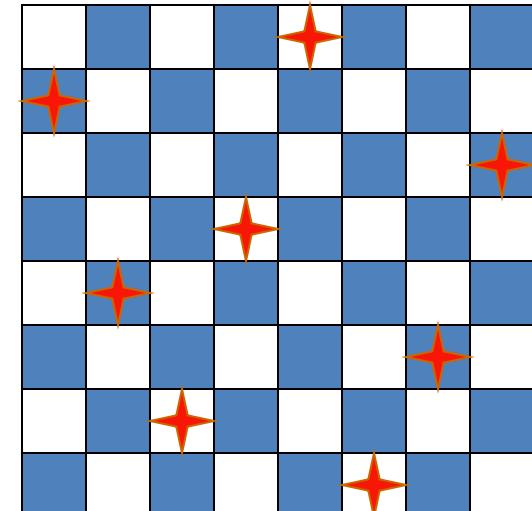
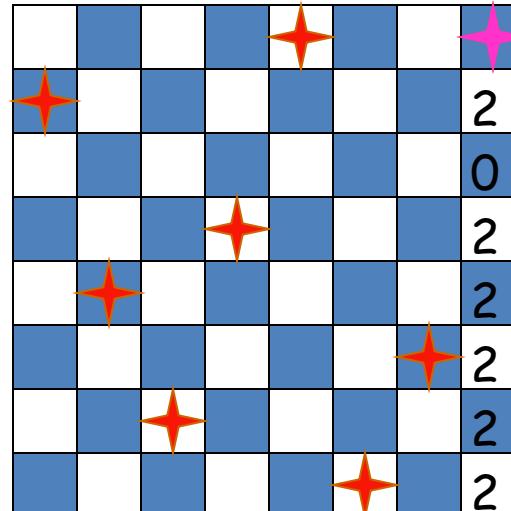
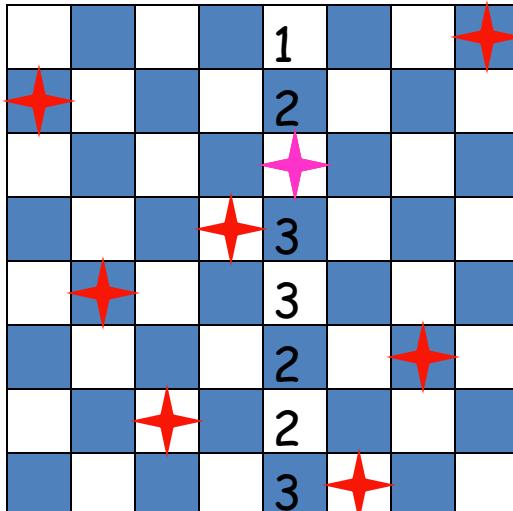
- Pick an initial state S at random with one queen in each column
- Repeat k times:
 - If GOAL?(S) then return S
 - Pick an attacked queen Q at random
 - Move Q in its column to minimize the number of attacking queens → new S [min-conflicts heuristic]
- Return failure



Application: 8-Queen

Repeat n times:

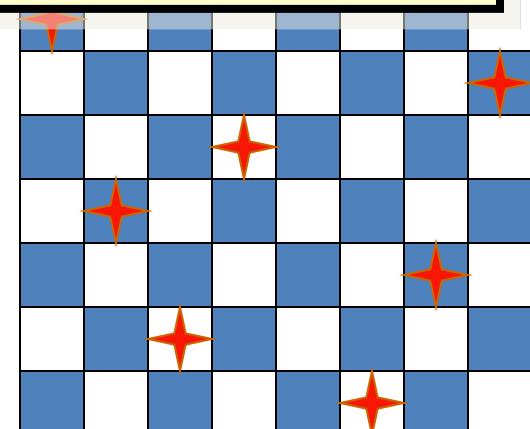
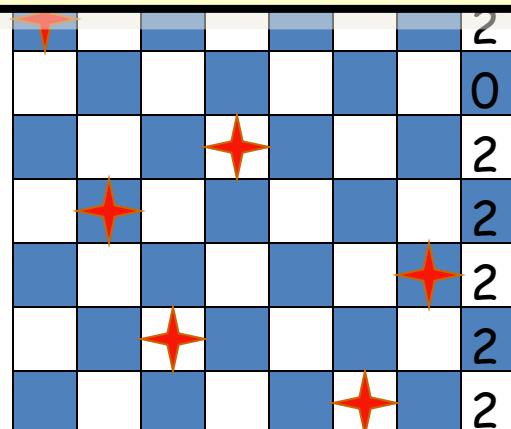
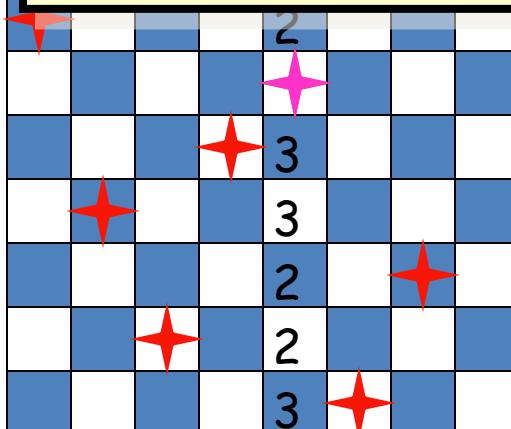
- Pick an initial state S at random with one queen in each column
- Repeat k times:
 - If GOAL?(S) then return S
 - Pick an attacked queen Q at random
 - Move Q in its column to minimize the number of attacking queens → new S [min-conflicts heuristic]
- Return failure



Application: 8-Queen

R Why/when does it work well?

- 1) There are **many goal states** that are well-distributed over the state space
- 2) If no solution has been found after a few steps, it's **better to start it all over again.**
- 3) Building a search tree would be much less efficient because of the **high branching factor**



Monte Carlo Descent

1. $S \leftarrow$ initial state

2. Repeat k times:

- If $\text{GOAL?}(S)$ then return S

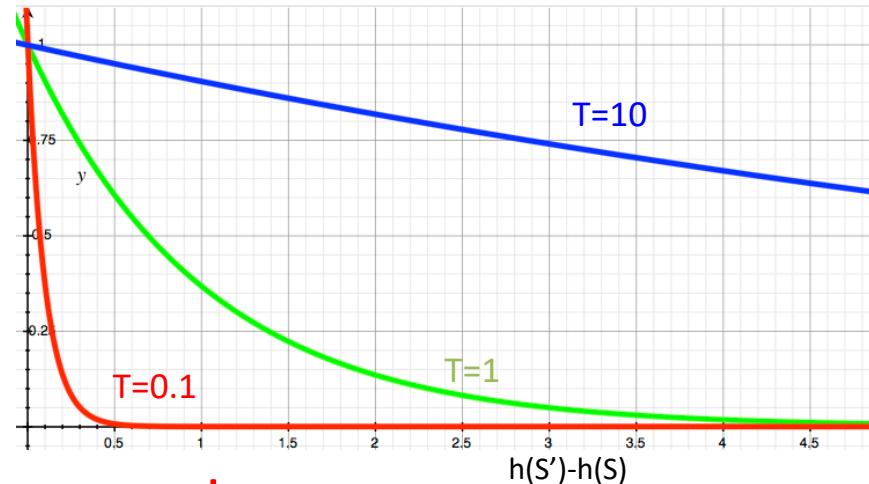
- $S' \leftarrow$ successor of S **picked at random**

- if $h(S') \leq h(S)$ then $S \leftarrow S'$

else with prob. $\exp(-(h(S')-h(S))/T)$ (for some T), $S \leftarrow S'$

3. Return failure

- **Simulated annealing** lowers T as k increases



Other local search techniques: Beam, genetic, branch-and-bound

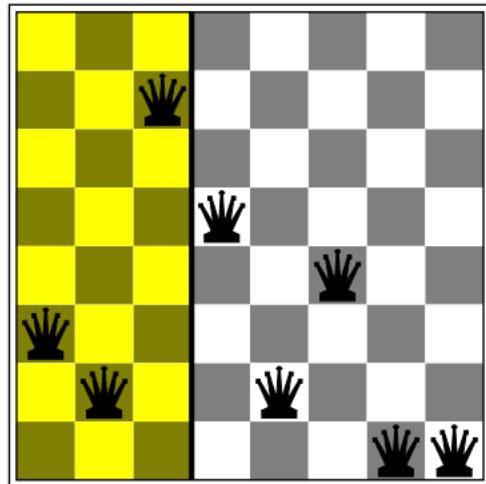
Beam search:

Explore promising states in “parallel”

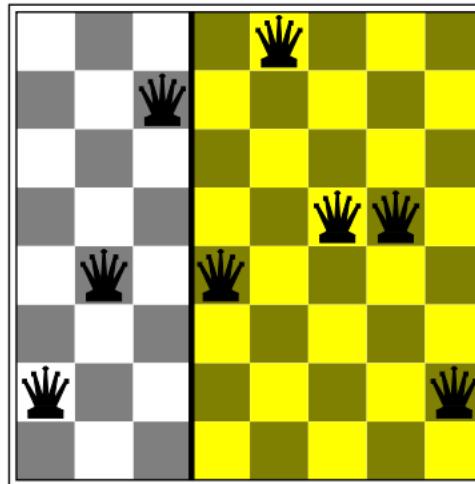
1. $A \leftarrow$ Set of n randomly-generated states
 2. Repeat k times:
 - $A' \leftarrow$ Successors of states in A
 - If there exists S in A' such that $\text{GOAL?}(S)$, then return S
 - $A \leftarrow$ Subset of n best states in A'
 3. Return failure
-
- Idea: Searches that find states will “recruit” other searches to join them

Genetic search: “Evolve” promising states

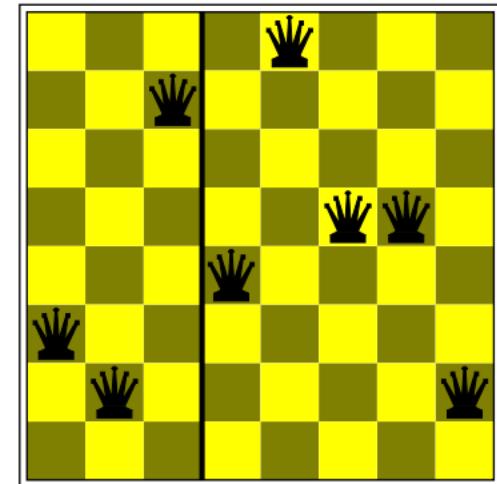
- Idea: Successors are generated by combining *pairs* of promising states. Most promising “offspring” states are kept.



+



=



672 47588

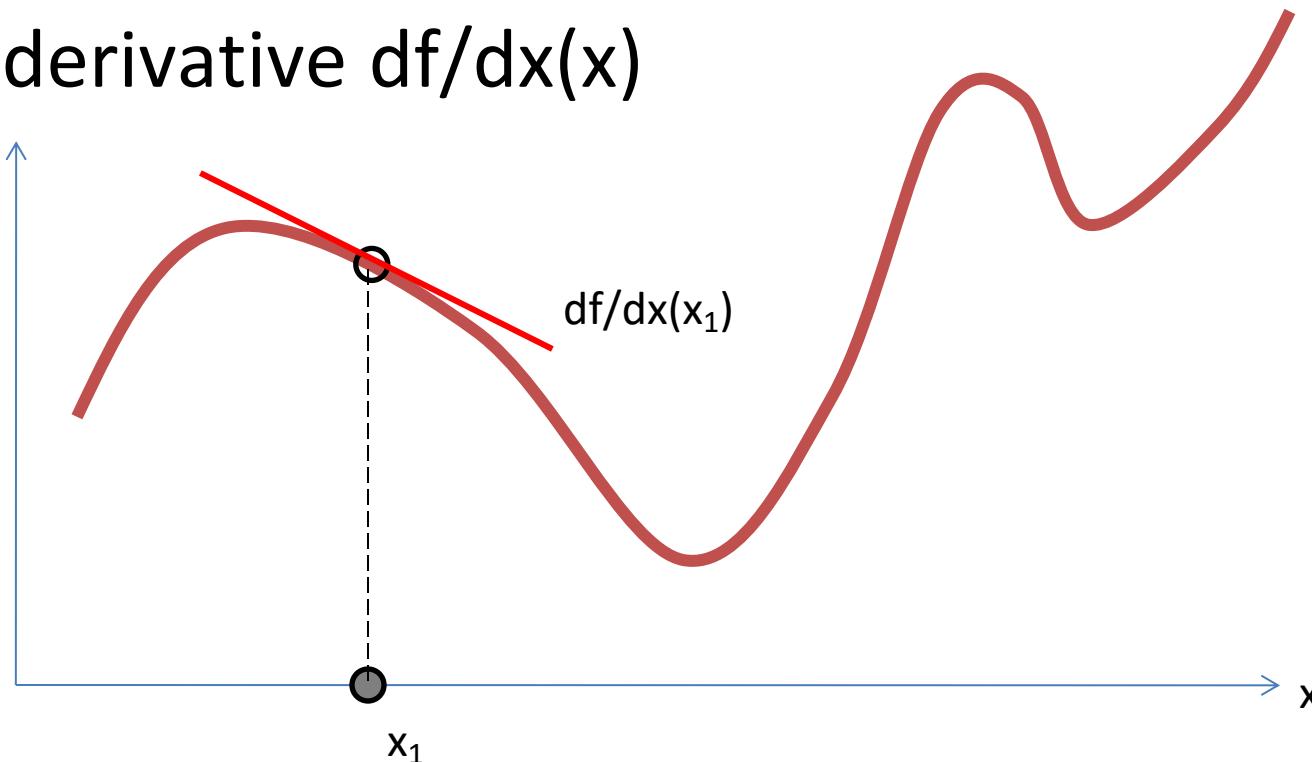
752 51447

672 51447

Local search in continuous spaces: Gradient descent

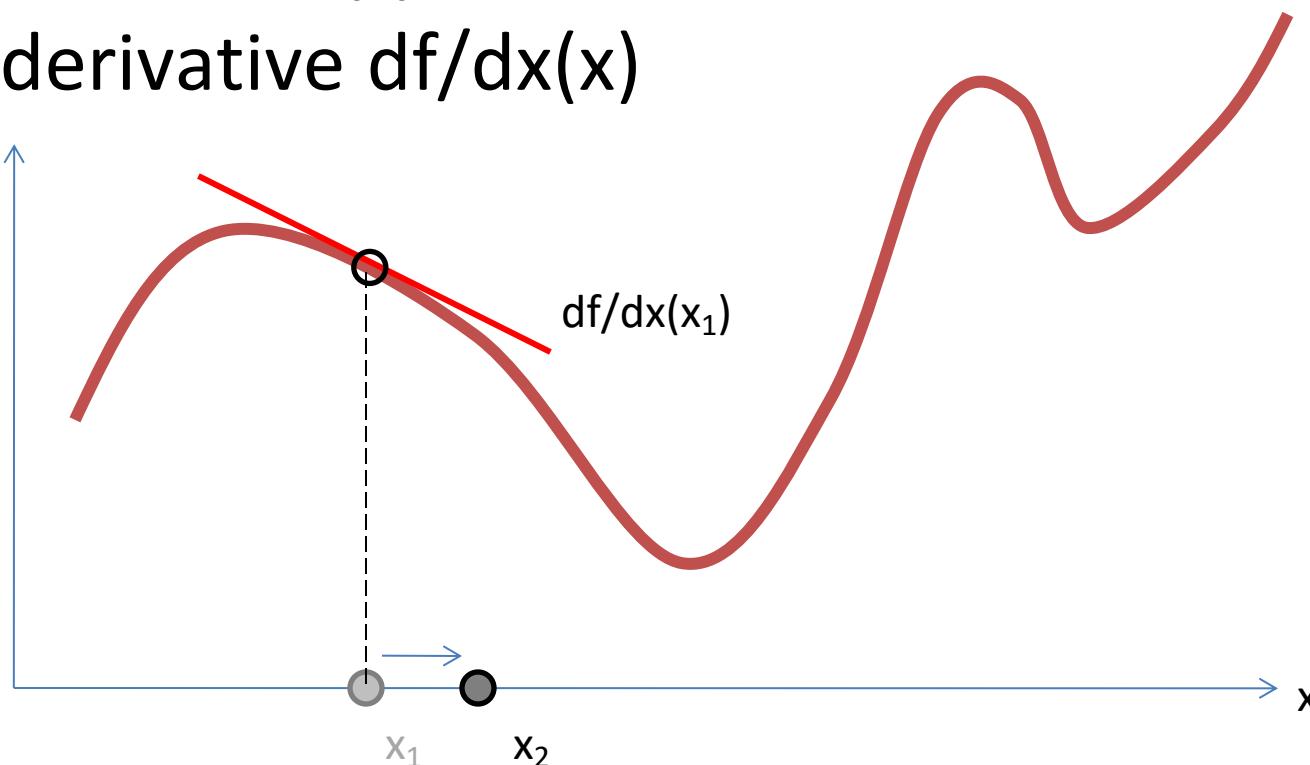
Gradient Descent in Continuous Space

- Minimize $y=f(x)$
- Move in opposite direction of derivative $df/dx(x)$



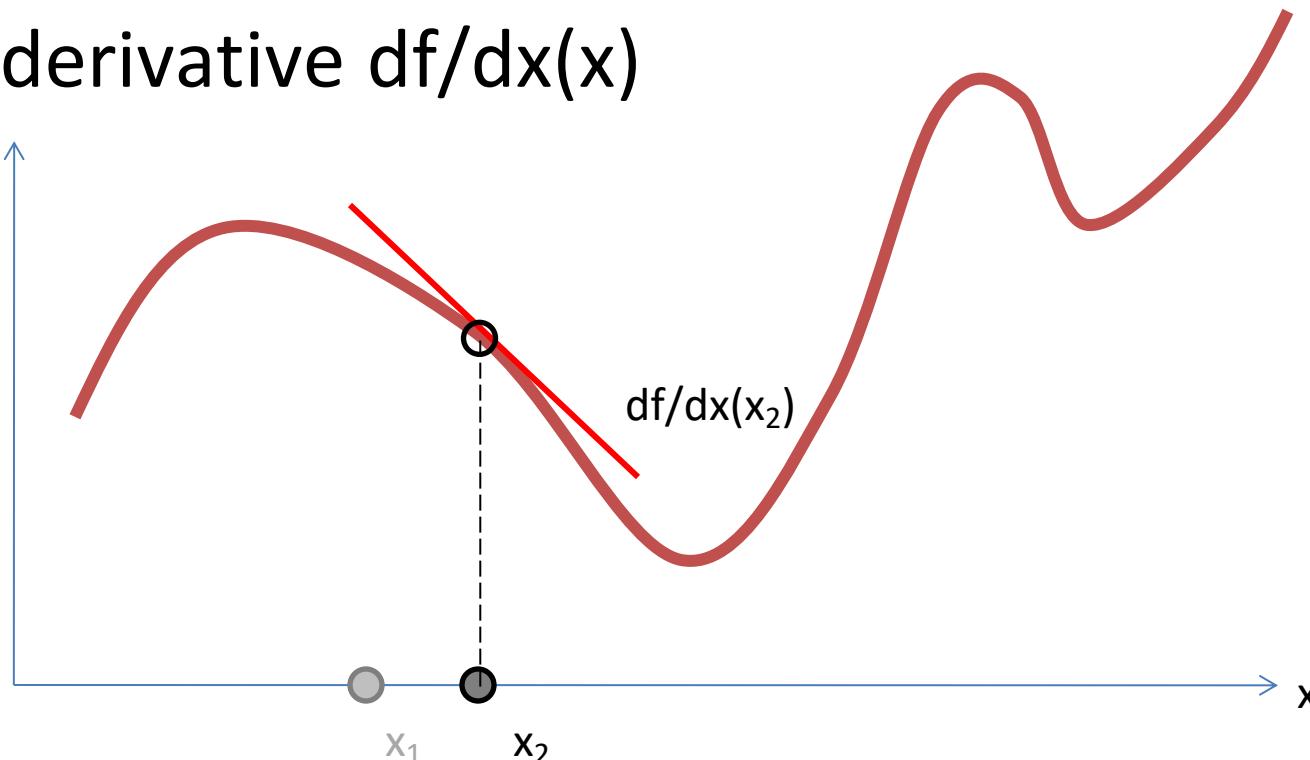
Gradient Descent in Continuous Space

- Minimize $y=f(x)$
- Move in opposite direction of derivative $df/dx(x)$



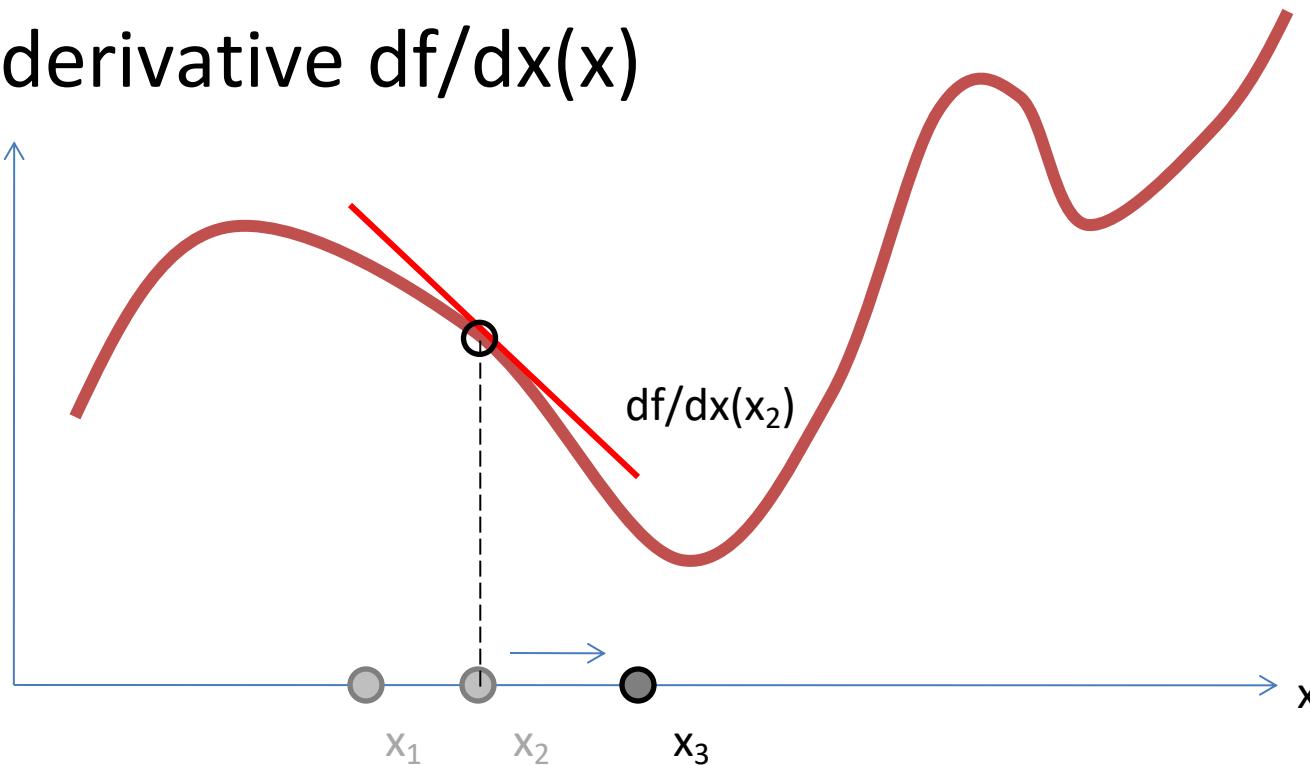
Gradient Descent in Continuous Space

- Minimize $y=f(x)$
- Move in opposite direction of derivative $df/dx(x)$



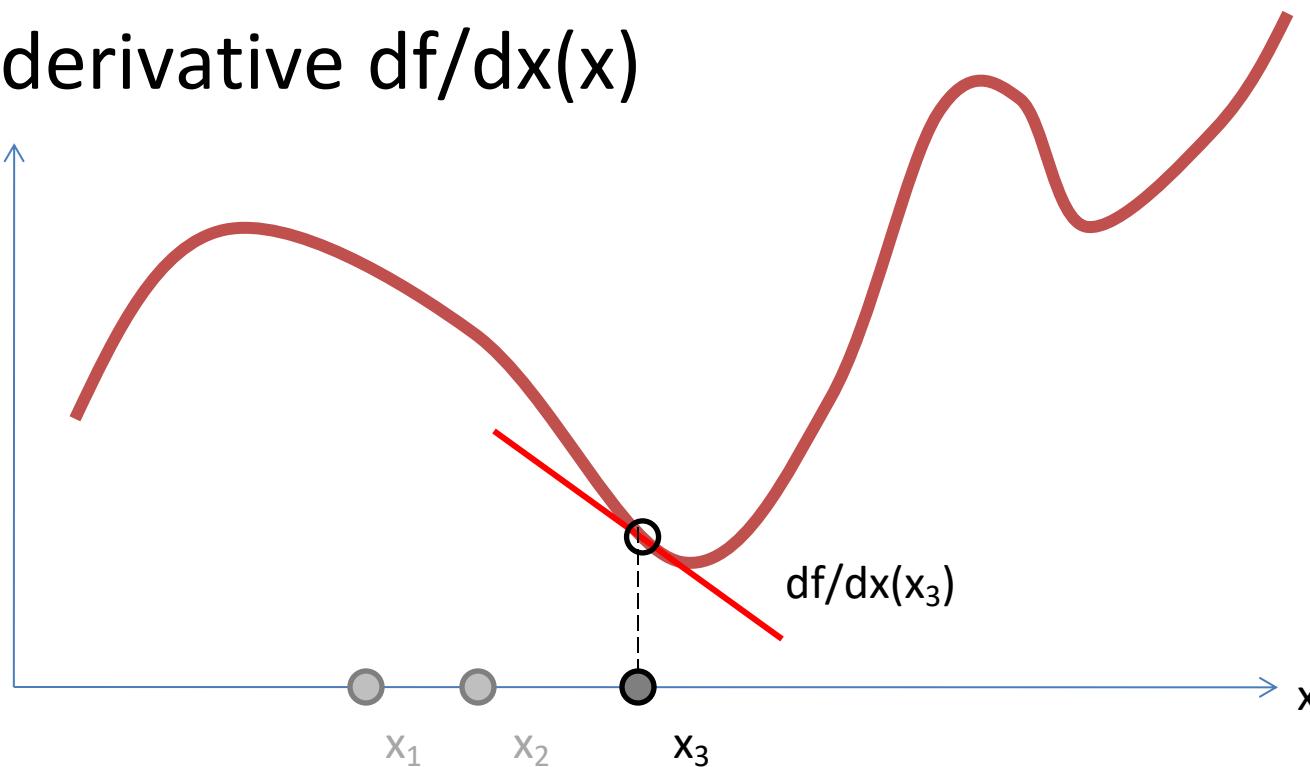
Gradient Descent in Continuous Space

- Minimize $y=f(x)$
- Move in opposite direction of derivative $df/dx(x)$



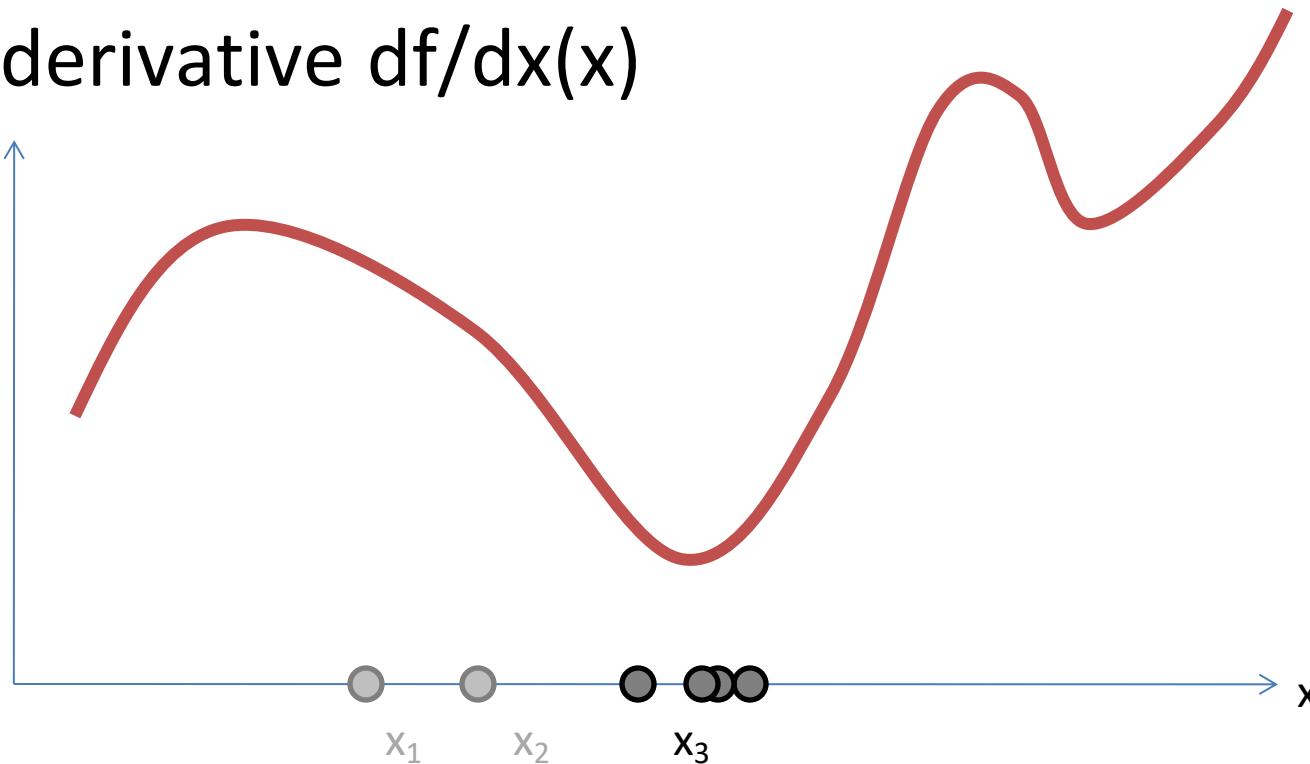
Gradient Descent in Continuous Space

- Minimize $y=f(x)$
- Move in opposite direction of derivative $df/dx(x)$



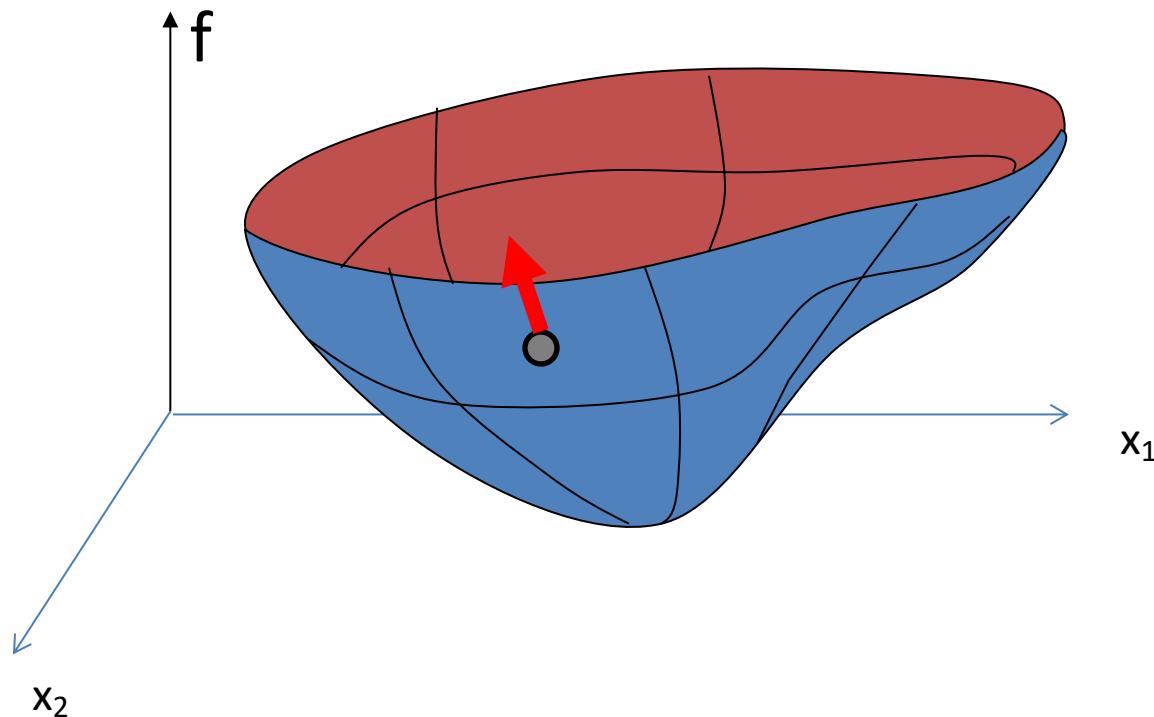
Gradient Descent in Continuous Space

- Minimize $y=f(x)$
- Move in opposite direction of derivative $df/dx(x)$

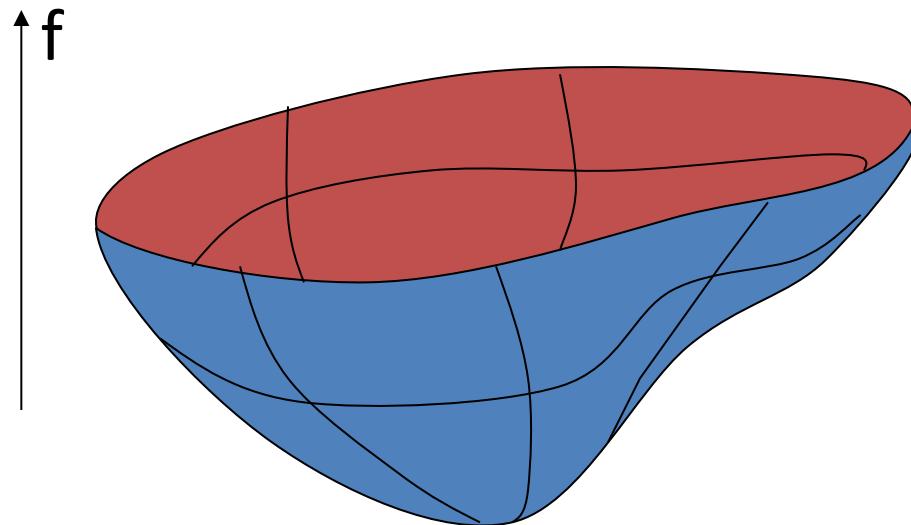


Gradient: analogue of derivative in multivariate functions $f(x_1, \dots, x_n)$

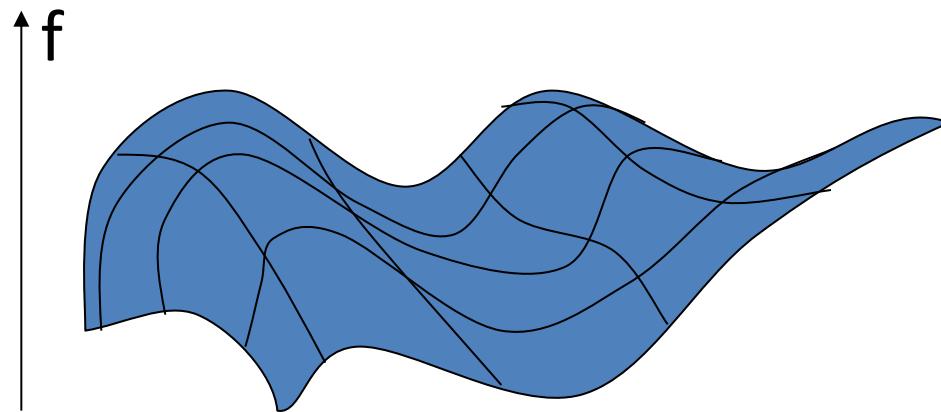
Direction that you would move x_1, \dots, x_n to make the steepest increase in f



GD works well



GD works poorly



Algorithm for Gradient Descent

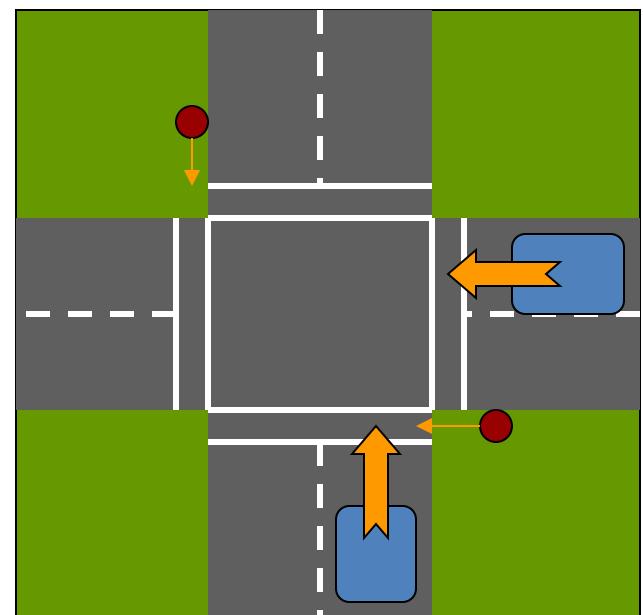
- Input: continuous *objective function* f , initial point $\mathbf{x}^0 = (x_1^0, \dots, x_n^0)$
 1. For $t=0, \dots, N-1$:
 2. Compute gradient $\mathbf{g}^t = (\partial f / \partial x_1(\mathbf{x}^t), \dots, \partial f / \partial x_n(\mathbf{x}^t))$
 3. If length of \mathbf{g}^t is small enough, return \mathbf{x}
 4. Pick a *step size* α^t
 5. Let $\mathbf{x}^{t+1} = \mathbf{x}^t - \alpha^t \mathbf{g}^t$
 6. Return failure

When will this work? What can go wrong?

Online search

How to handle imperfect observations?

- Classical search assumes that:
 - World states are perfectly observable,
→ the current state is exactly known
 - Action representations are perfect,
→ states are exactly predicted
- How an agent can cope with **adversaries**,
uncertainty, and **imperfect information**?



Distance, speed, acceleration?

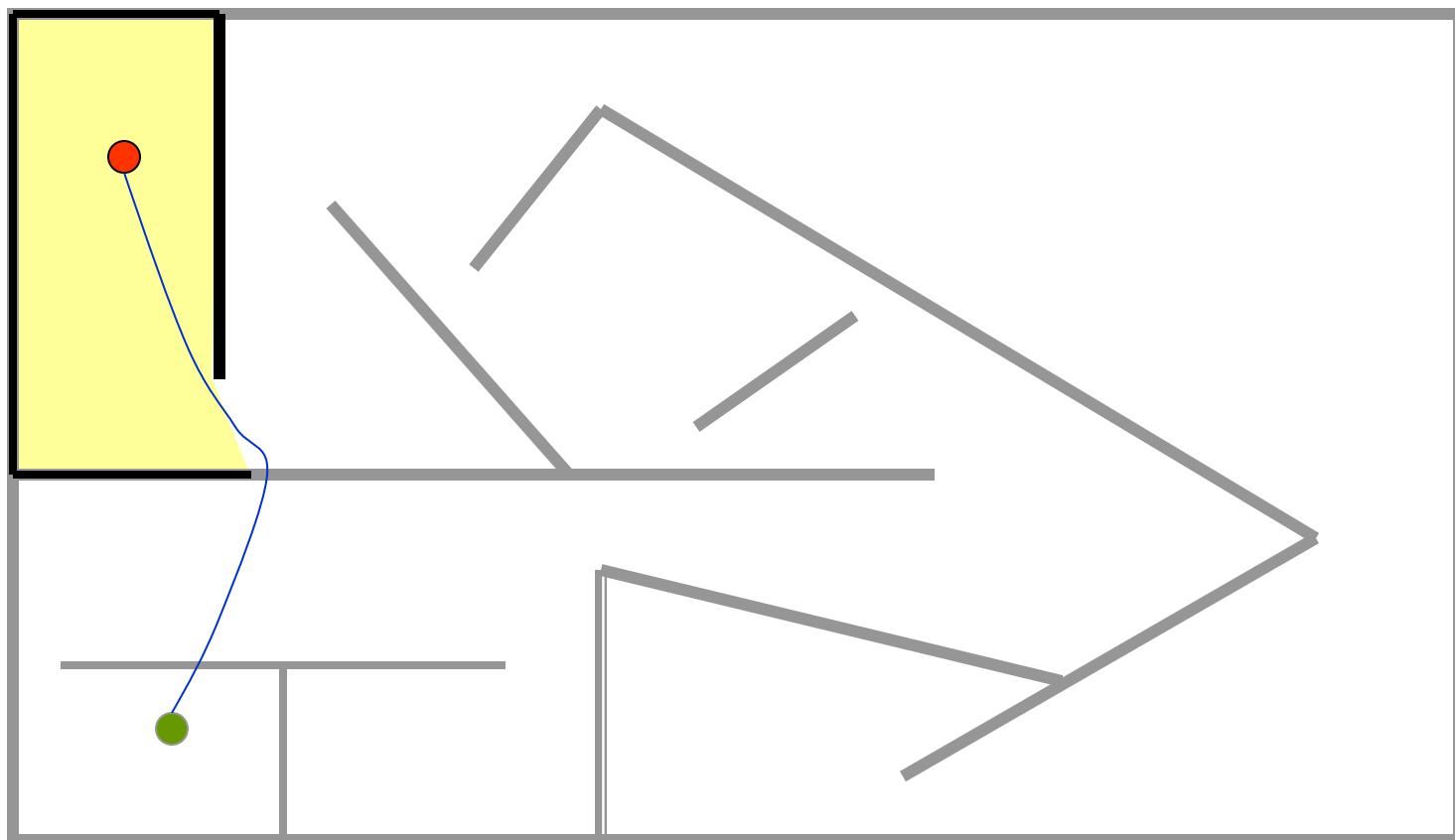
Intent?

Personality?

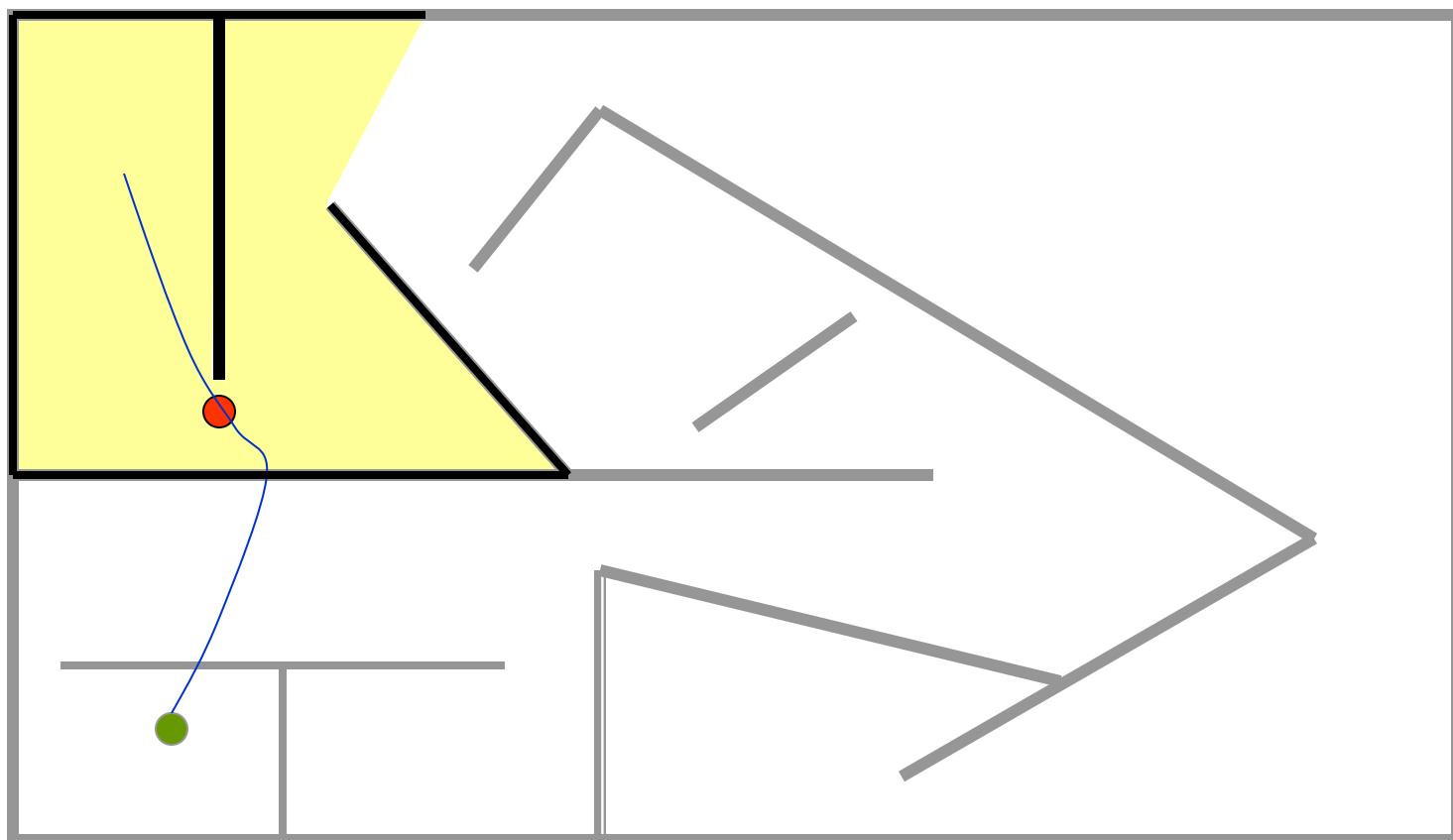
Online Search

- On-line search: repeatedly observe effects, and replan
 - A **proactive** approach for planning
 - A **reactive** approach to uncertainty
- Example: A robot must reach a goal position. It has no prior map of the obstacles, but its vision system can detect all the obstacles visible from the robot's current position

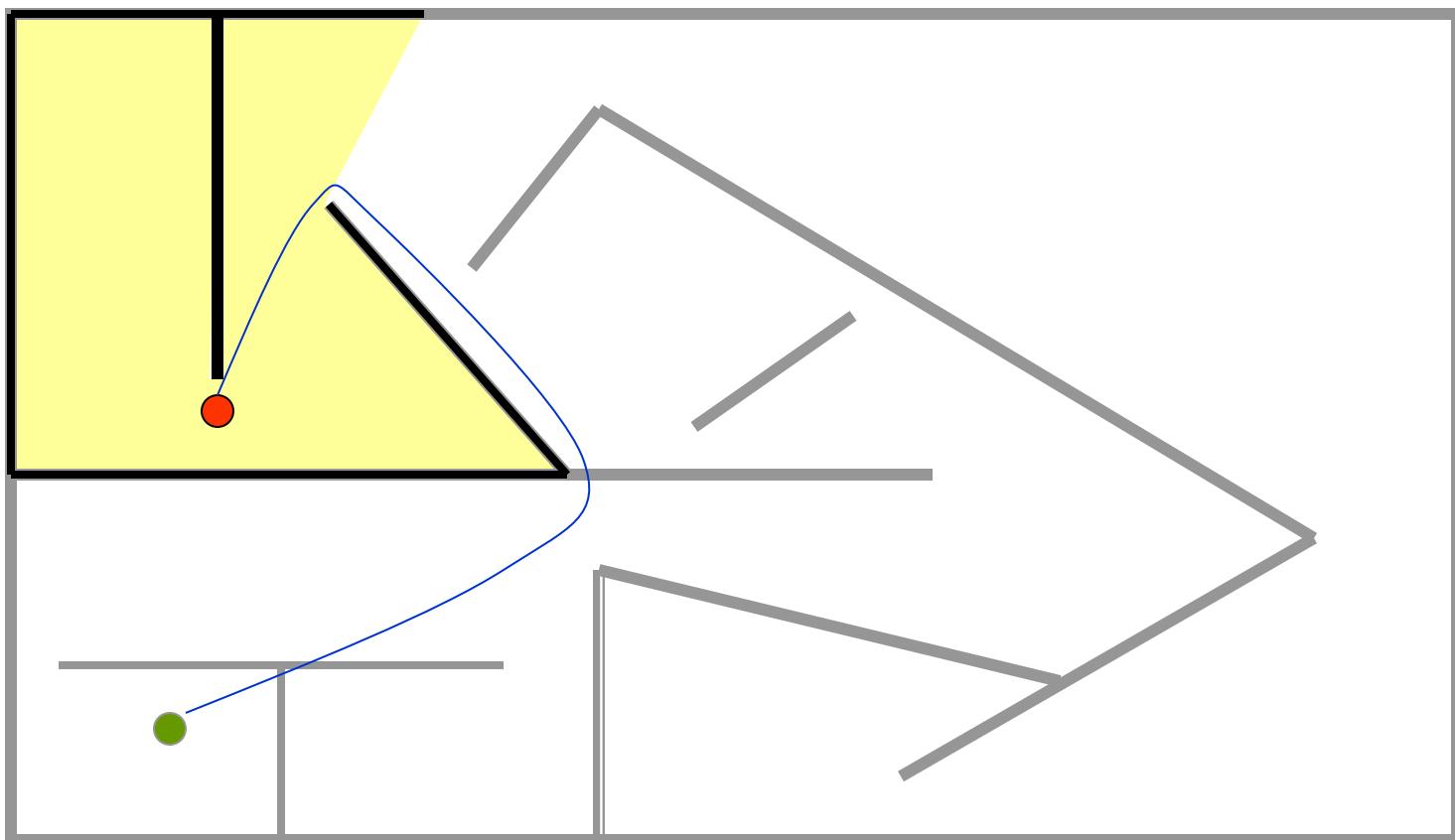
Assuming no obstacles in the unknown region and taking the shortest path to the goal **is similar to searching with an admissible (optimistic) heuristic**



Assuming no obstacles in the unknown region and taking the shortest path to the goal is similar to searching with an admissible (optimistic) heuristic



Assuming no obstacles in the unknown region and taking the shortest path to the goal is similar to searching with an admissible (optimistic) heuristic



Just as with classical search, on-line search may detect dead-ends and move to a more promising position

Next class

- Adversarial search and game playing

Adversarial search and game playing

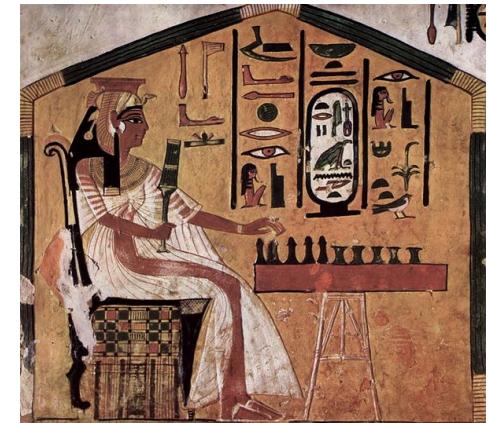
Adversarial search is search when there is an "enemy" or "opponent" changing the state of the problem every step in a direction you do not want

Announcements

- A0 due Friday
 - Remember: automatic 48 hour extension with 10% grade penalty
- A1 coming next week

Puzzles and games have long been considered a challenge for human intelligence:

- **Chess** in Persia and India ~4000 years ago
- **Checkers** in 3600-year-old Egyptian paintings
- **Go** in China over 3000 years ago



Computer chess: beginnings

Alex Bernstein, 1957: First computer chess program on an IBM 704
(vacuum tubes, ~12k FLOPS)



1966 Hubert Dreyfus writes
“computers cannot play chess”...



Seymour Papert arranged for a match
between Dreyfus and “Mac Hack”.

Dreyfus lost (1967)



In 1968, David Levy, International Master in Chess, had a friendly game against John McCarthy, which he won.

McCarthy then bet that within 10 years, a computer could beat Levy. They had a bet of 500 pounds



Levy won the bet in 1978. The first computer program that beat him was “Deep Thought”, in 1989.

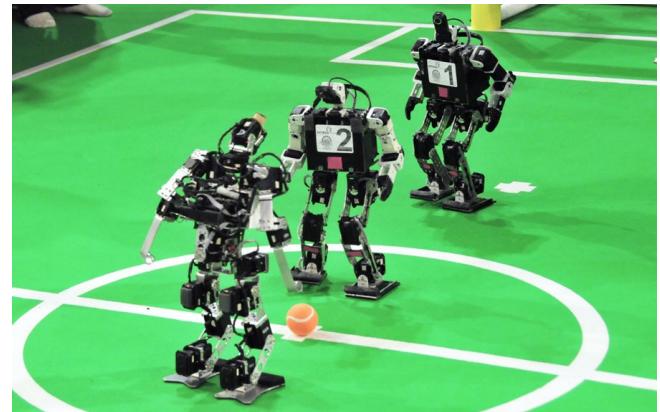
Game Playing vs. the “Real World”

- Games are compact settings that mimic the uncertainty of interacting with the natural world
- Games are (kind of) like the real world:
 - I have a goal (to win) that I want to work/plan towards
 - I have to decide among many possible actions
 - Other agents can try to keep me from achieving my goal
- Games are much “simpler” than the real world:
 - **Chess:** I need to know the rules of chess and the state of the chess board
 - **Walking across campus to get to class:** So many things to consider...

Specific Setting

This lecture considers:

- Two-player
- turn-taking
- fully observable
- zero-sum
- time-constrained games

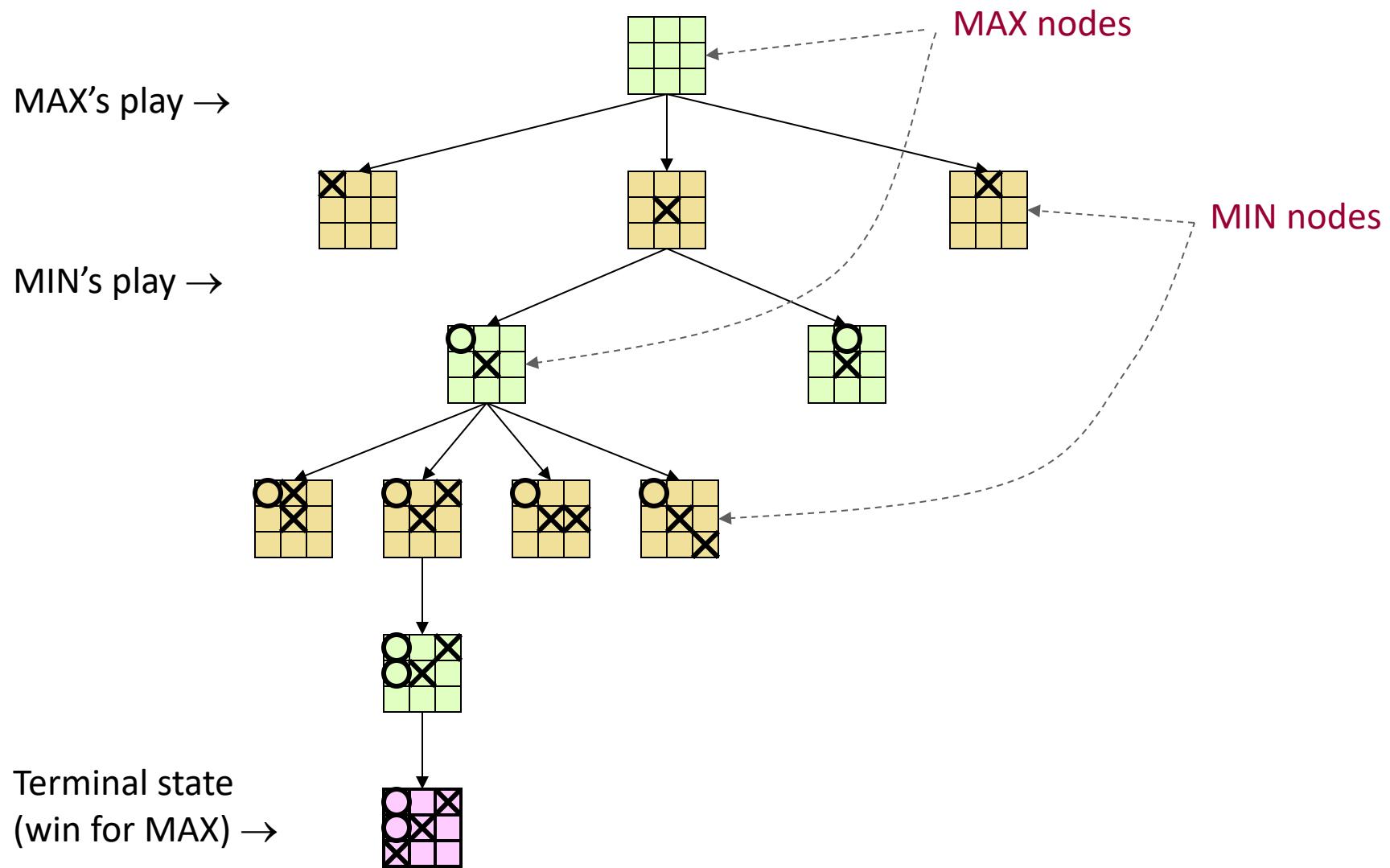


Specific Setting

Two-player, turn-taking, fully observable, zero-sum, time-constrained game

- State space?
- Initial state?
- Successor function?
- Goal state?
- Cost?

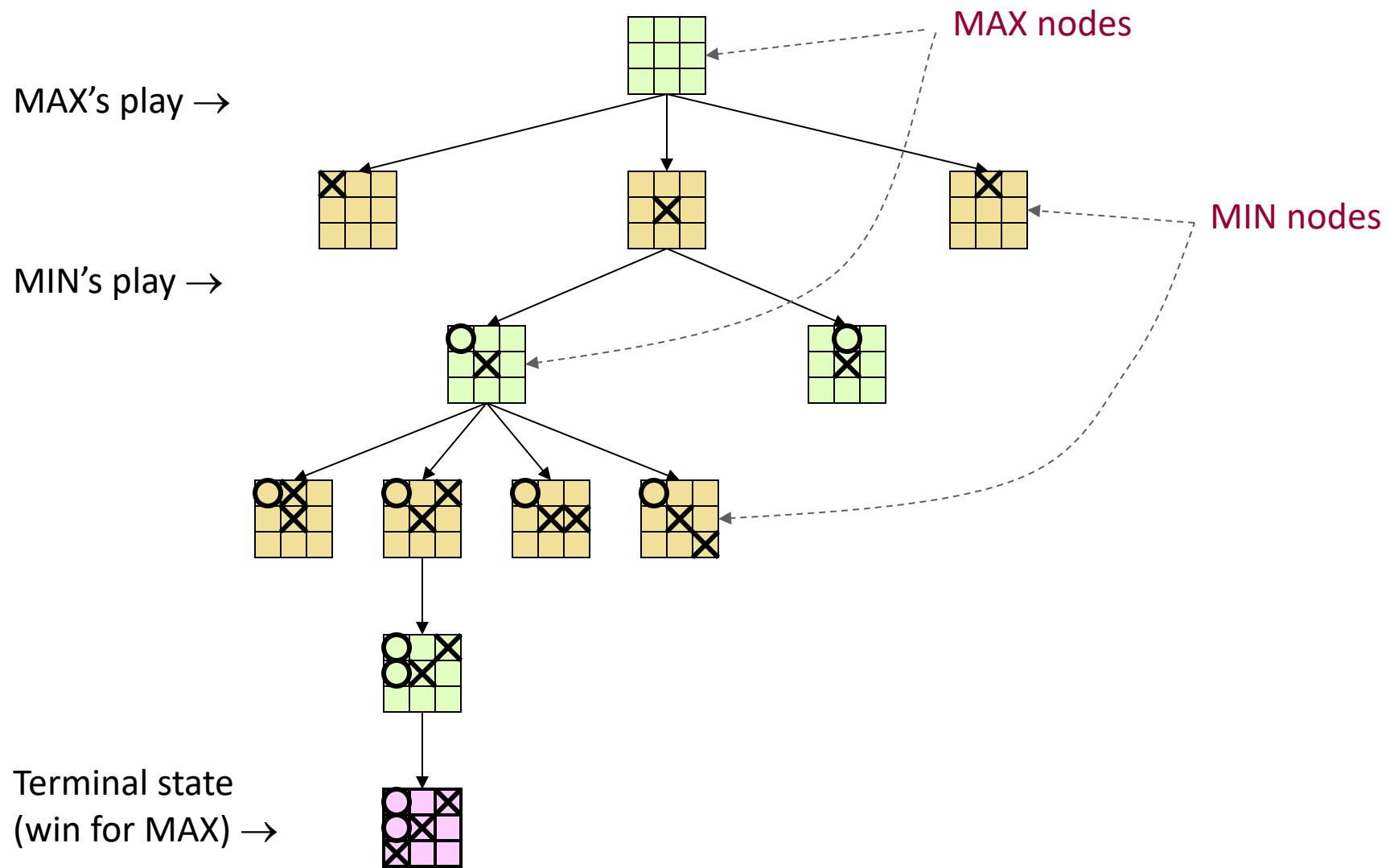
Game Tree



Formal Abstraction of Games

- State space
- Initial state
- Successor function: which actions can be executed in each state and the successor state for each action
- MAX's and MIN's actions alternate, with MAX playing first in the initial state
- Terminal test: tells if a state is terminal and, if yes, if it's a win or a loss for MAX, or a draw

Game Tree

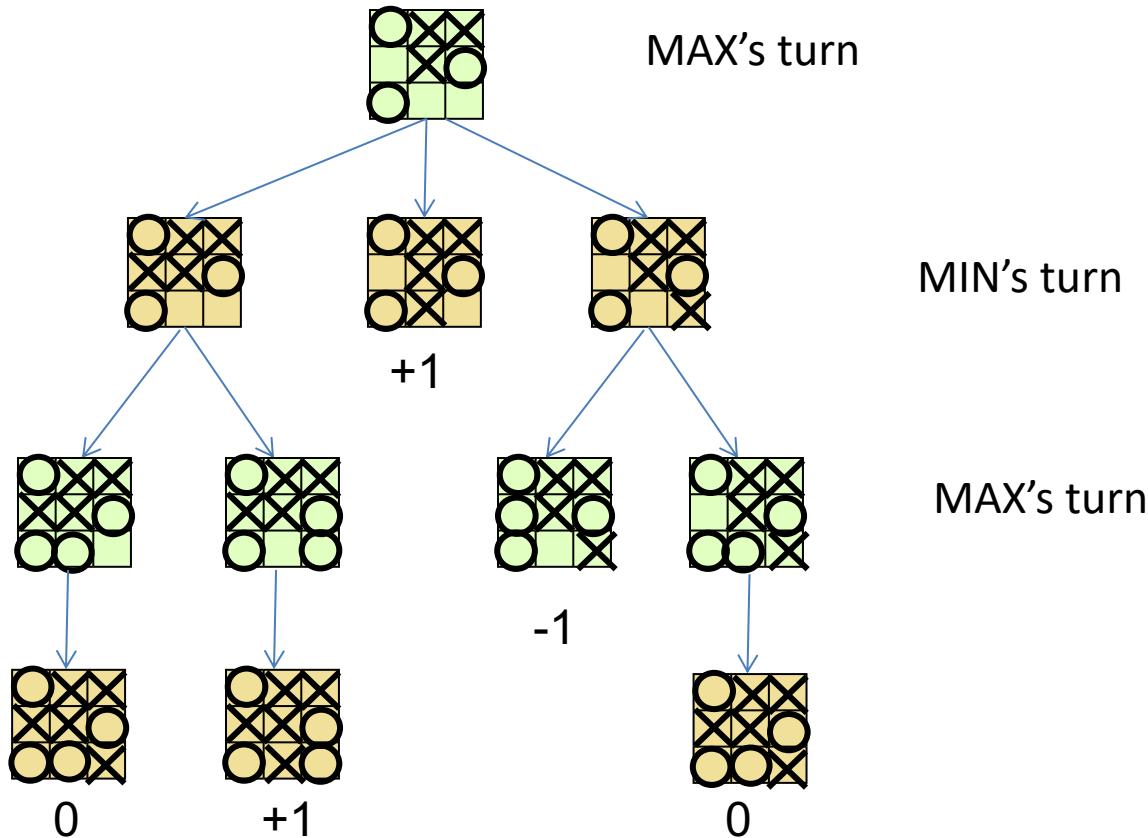


Choosing an Action: Basic Idea

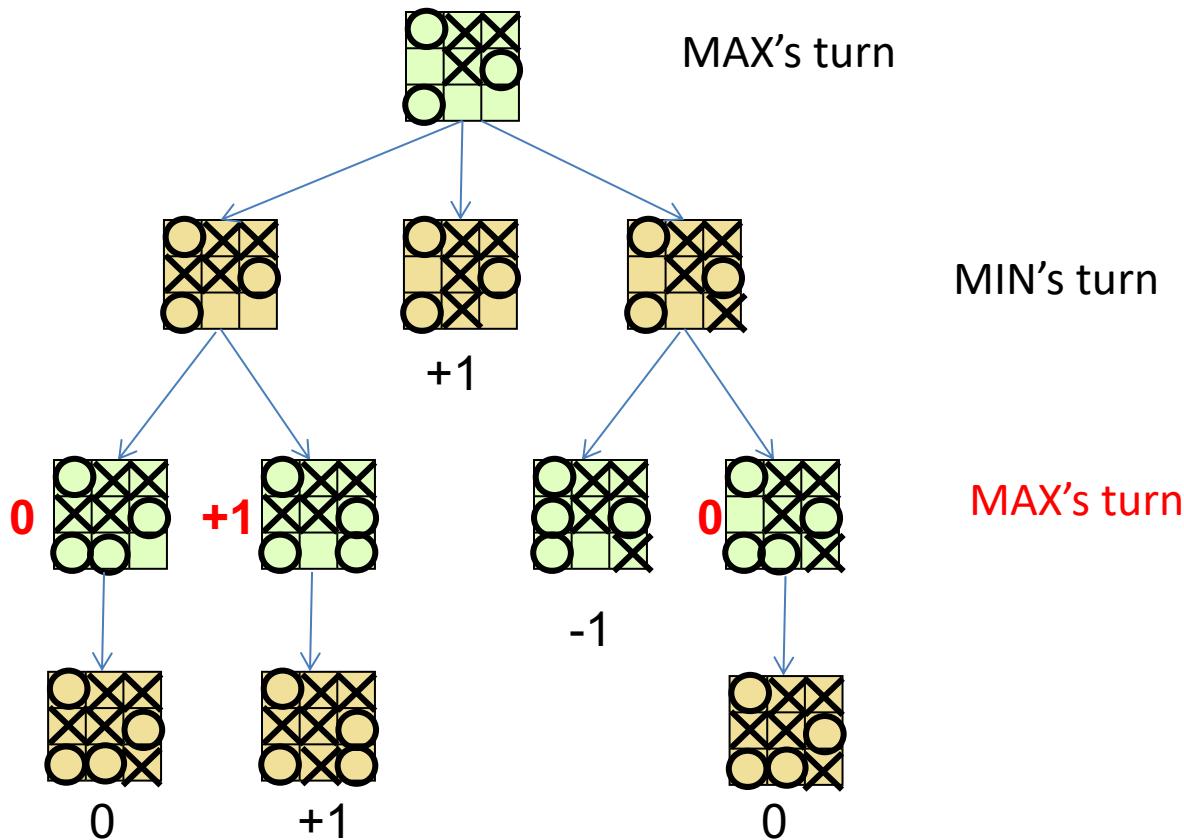
1. Using the current state as the initial state, build the entire game tree to the leaf nodes
2. Evaluate whether leaf nodes are wins (+1), losses (-1), or draws (0). Other payoffs are possible.
3. Back up the results from the leaves to the root and pick the best action assuming the “worst” from MIN (MIN plays optimally)

→ **Minimax algorithm**

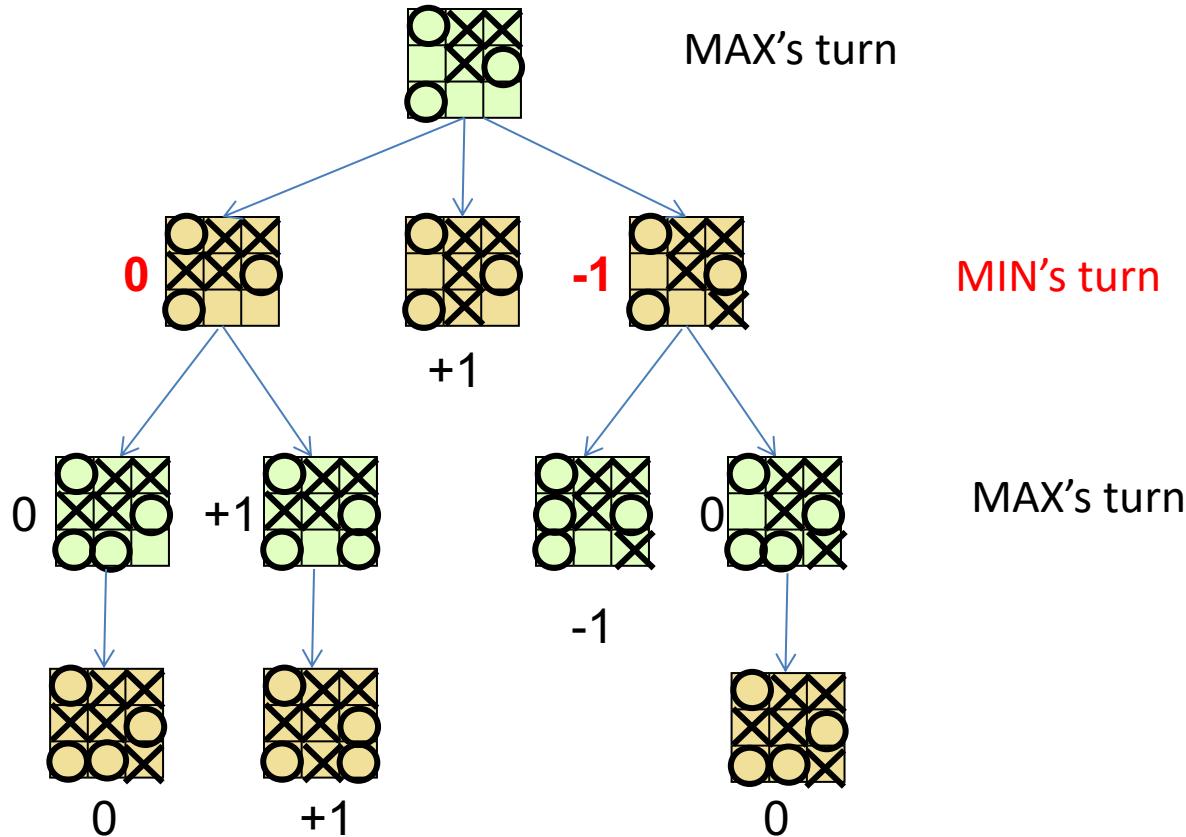
Minimax Backup



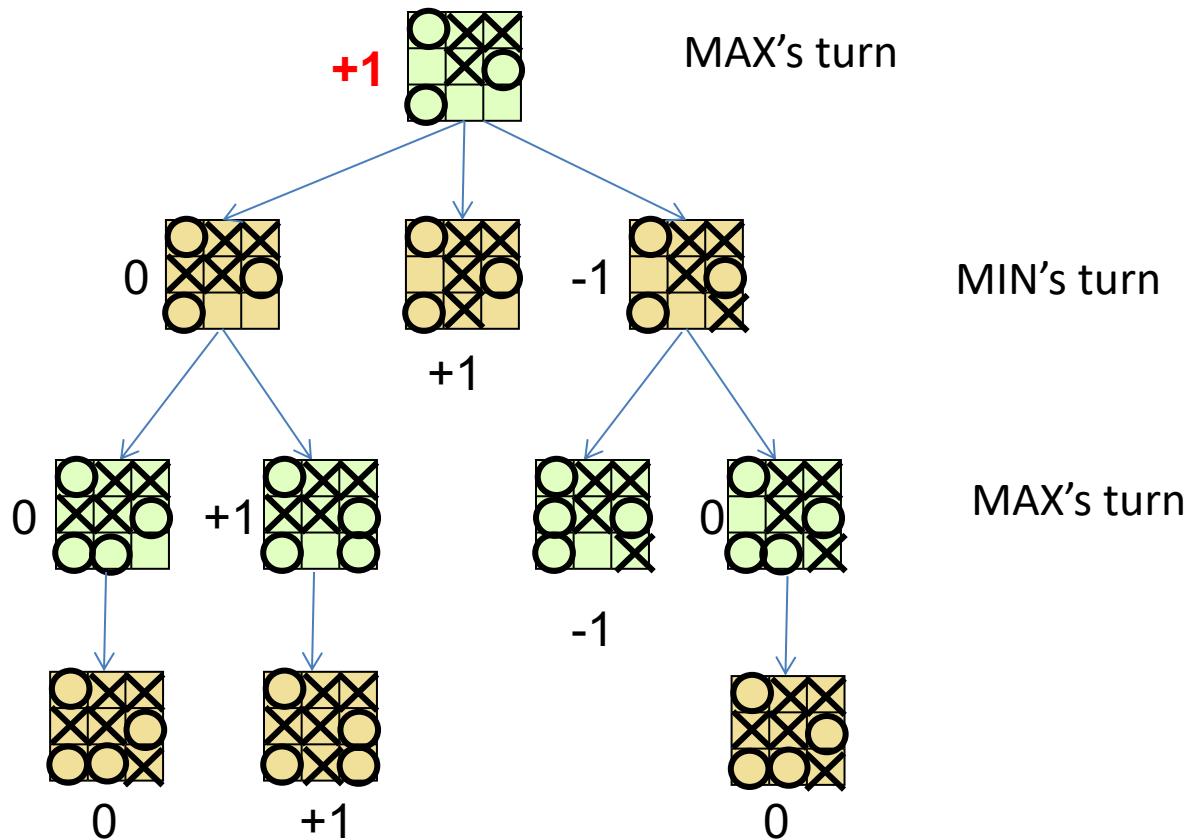
Minimax Backup



Minimax Backup



Minimax Backup



Minimax Algorithm

- Expand the game tree from current state (MAX's turn)
- Evaluate if each leaf is a win (+1), lose (-1), draw (0)
- Backup the values from leaves to root tree as follows:
 - MAX node gets maximum of the utilities of its successors
 - MIN node gets minimum of the utilities of its successors
- Choose the move that has the largest backed-up value

Recursive Minimax Algorithm

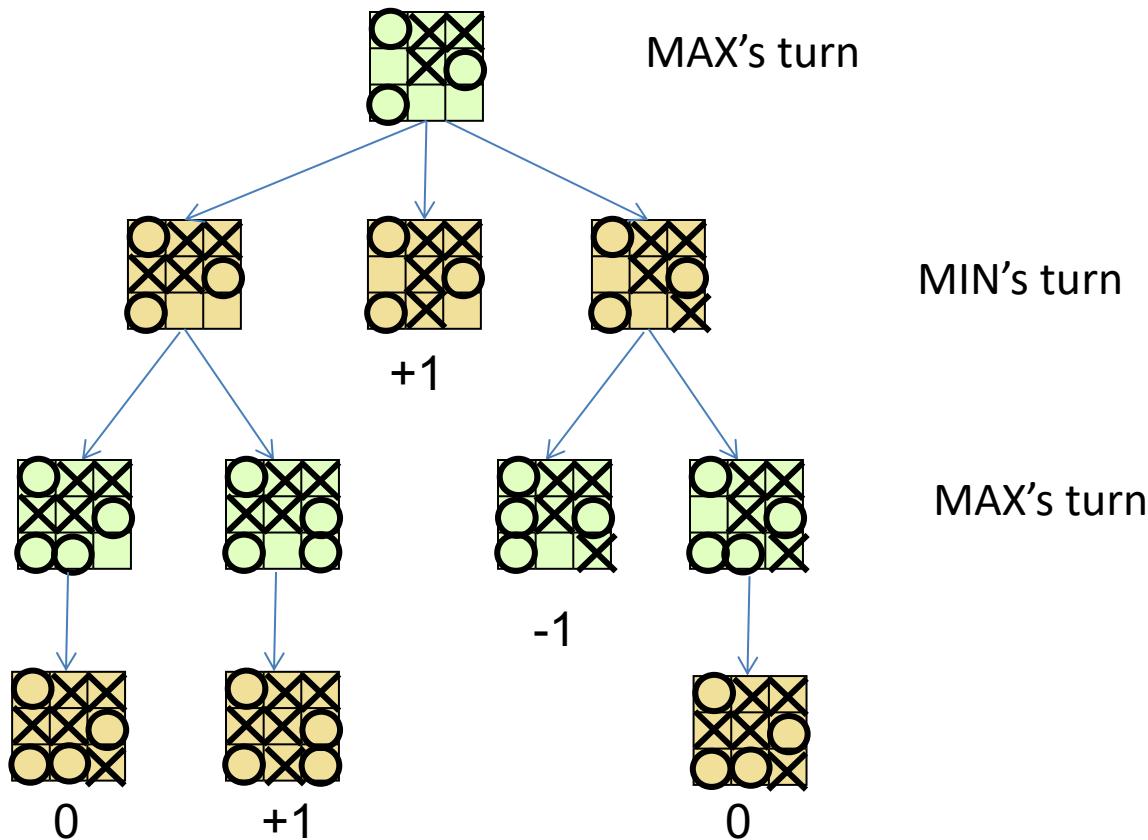
- **MINIMAX-Decision(S)**
 - Return action leading to state $S' \in \text{SUCC}(S)$ that maximizes **MIN-Value**(S')
- **MAX-Value(S)**
 - **If** Terminal?(S) return Result(S)
 - **Else** return $\max_{S' \in \text{SUCC}(S)} \text{MIN-Value}(S')$
- **MIN-Value(S)**
 - **If** Terminal?(S) return Result(S)
 - **Else** return $\min_{S' \in \text{SUCC}(S)} \text{MAX-Value}(S')$

- **MINIMAX-Decision(S)**
 - Return action leading to state $S' \in \text{SUCC}(S)$ that maximizes **MIN-Value**(S')
- **MAX-Value(S)**
 - **If** Terminal?(S) return Result(S)
 - **Else** return $\max_{S' \in \text{SUCC}(S)} \text{MIN-Value}(S')$
- **MIN-Value(S)**
 - **If** Terminal?(S) return Result(S)
 - **Else** return $\min_{S' \in \text{SUCC}(S)} \text{MAX-Value}(S')$

Questions

- Does Minimax perform a depth- or breadth-first exploration of the game tree?
- Assuming tree has depth d with b moves at each point,
 - What's the time complexity?
 - What's the space complexity?

Minimax Backup



What's the catch?

- The Minimax algorithm guarantees to make the optimal decision
- Is our AI already invincible?

In general, the branching factor and the depth of terminal states are large

Chess:

- Branching factor: ~35
- Number of total moves in a game: ~100

Real-Time decisions

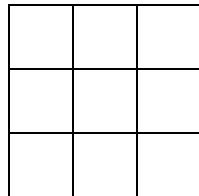
- In many games, the state space is *enormous*, and only a tiny fraction can be explored
- A heuristic:
 1. Using the current state as initial state, build game tree to **maximal depth h (the *horizon*) feasible within the time limit**
 2. Evaluate the states of the nodes at depth h
 3. Back up results from those nodes to the root and pick the best action, assuming the worst from MIN

Evaluation Function

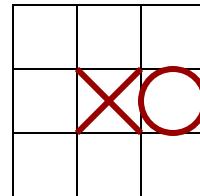
- Function e : state $s \rightarrow$ number $e(s)$
- $e(s)$: **heuristic** estimating favorability of s for MAX
 - $e(s) > 0$ means that s is favorable to MAX
(the larger the better)
 - $e(s) < 0$ means that s is favorable to MIN
 - $e(s) = 0$ means that s is neutral
- Other payoffs are possible, but still MAX wants to maximize and MIN to minimize

Example: Tic-tac-Toe

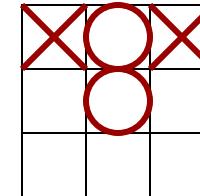
$e(s) = (\text{number of rows, columns, and diagonals open for MAX}) - (\text{number of rows, columns, and diagonals open for MIN})$



$$8 - 8 = 0$$



$$6 - 4 = 2$$



$$3 - 3 = 0$$

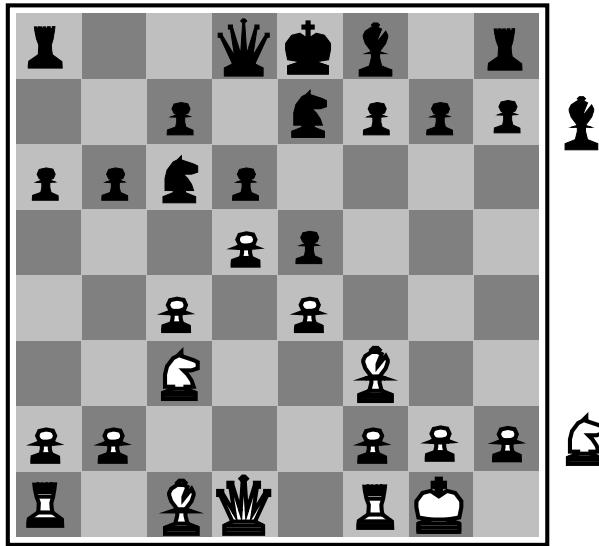
Construction of a Static Evaluation Function

- Usually a weighted sum of “features”:

$$e(s) = \sum_{i=1}^n w_i f_i(s)$$

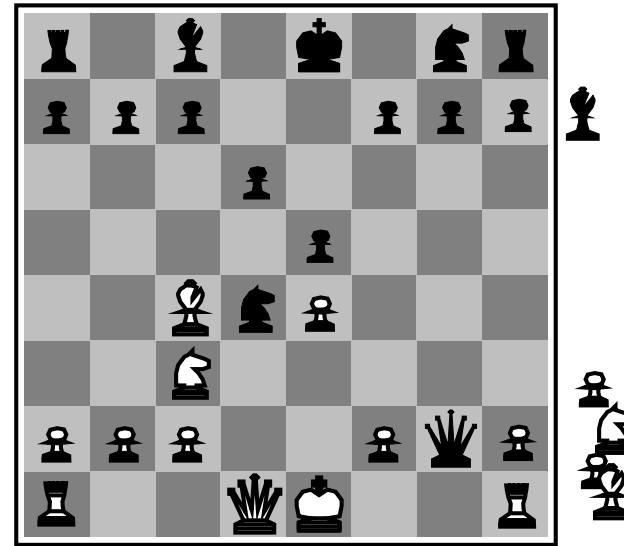
- Features may include
 - Number of pieces of each type
 - Number of possible moves
 - Number of squares controlled

Example from chess



Black to move

White slightly better



White to move

Black winning

$$e(s) = 1(\text{number of white pawns} - \text{number of black pawns}) + \\ + 3(\text{number of white knights} - \text{number of black knights}) + \\ + \dots$$

Minimax Algorithm

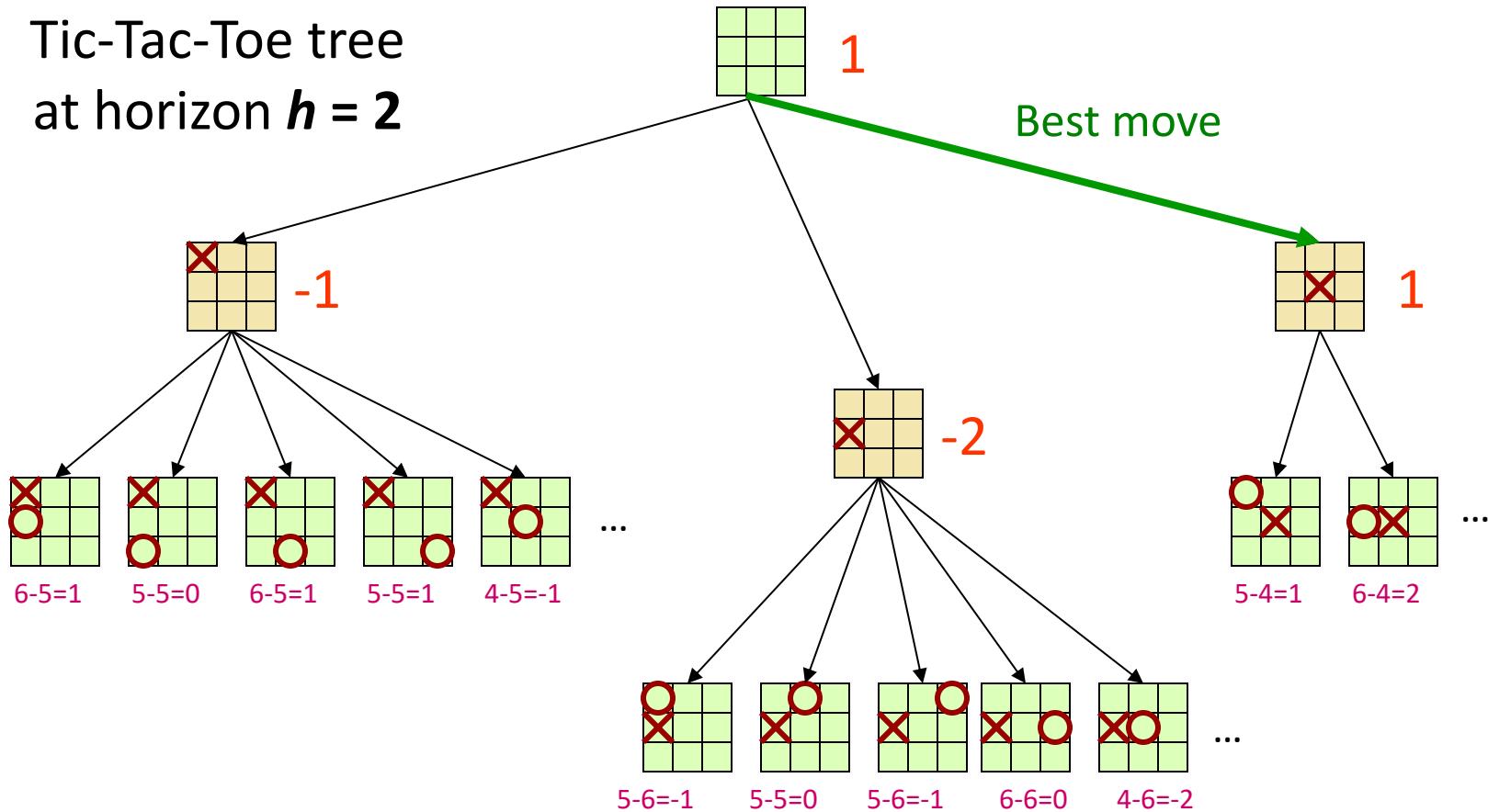
- Expand the game tree from current state (MAX's turn)
- Evaluate if each leaf is a win (+1), lose (-1), draw (0)
- Backup the values from leaves to root tree as follows:
 - MAX node gets maximum of the utilities of its successors
 - MIN node gets minimum of the utilities of its successors
- Choose the move that has the largest backed-up value

Minimax Algorithm

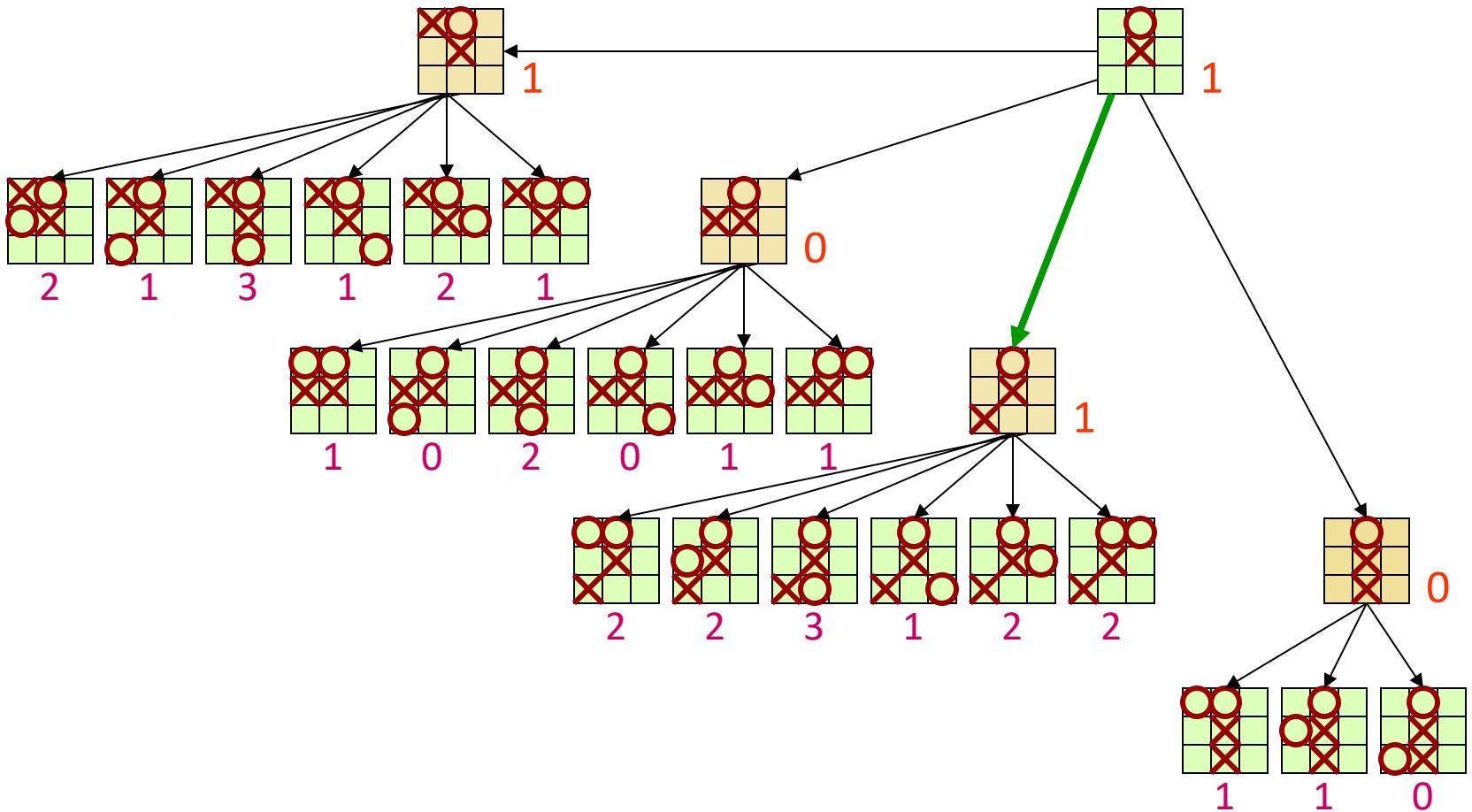
- Expand the game tree from current state (MAX's turn)
to depth h
- Evaluate ~~if heuristic function at each leaf is a win (+1), lose (-1), draw (0)~~
- Back-up the values from leaves to root tree as follows:
 - MAX node gets maximum of the utilities of its successors
 - MIN node gets minimum of the utilities of its successors
- Choose the move that has the largest backed-up value

Backing up Values

Tic-Tac-Toe tree
at horizon $h = 2$



Continuation



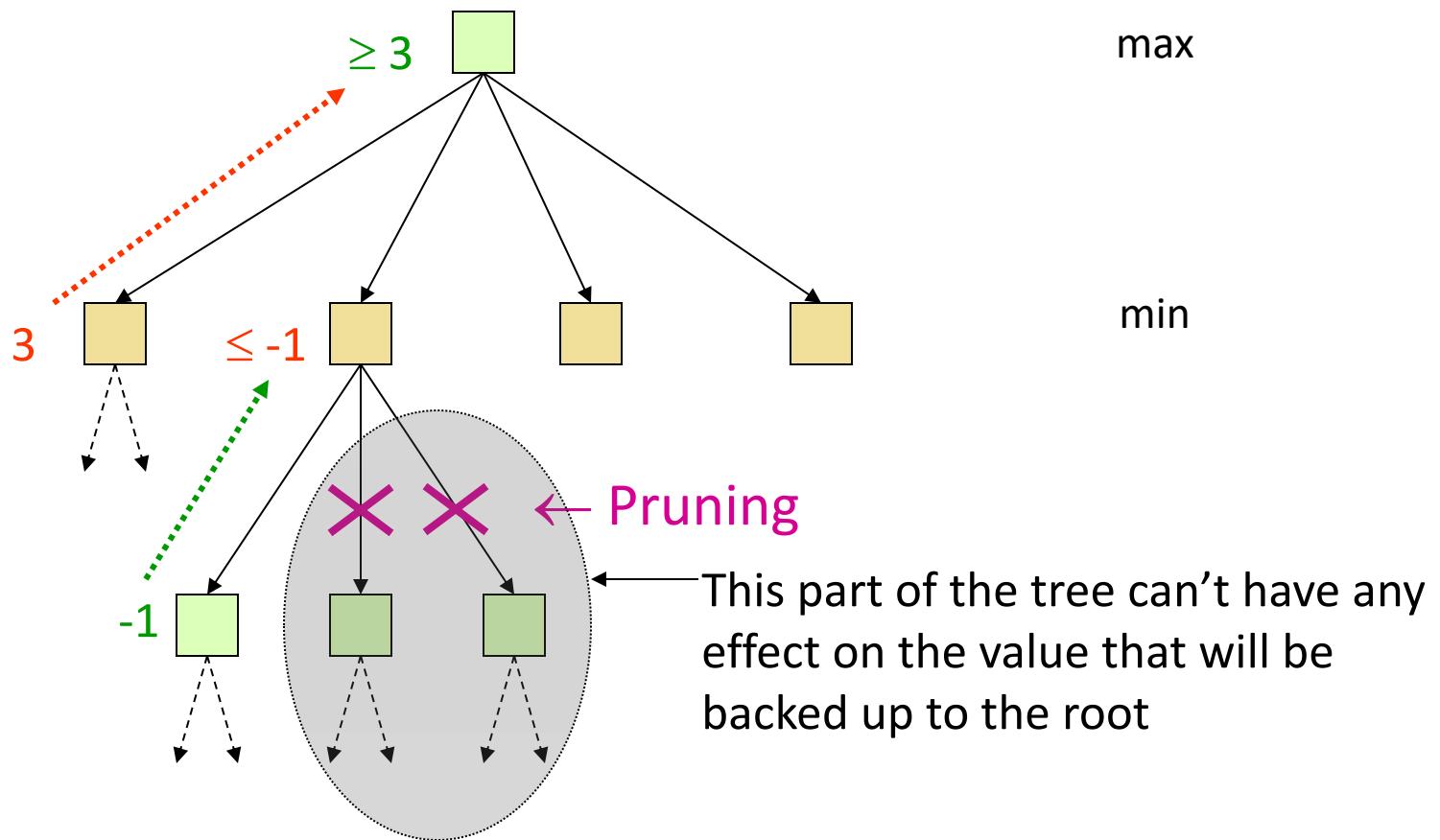
Chess Horizon

- Time complexity? $O(b^h)$
- Space complexity? $O(bh)$
- For chess, $b=35$:

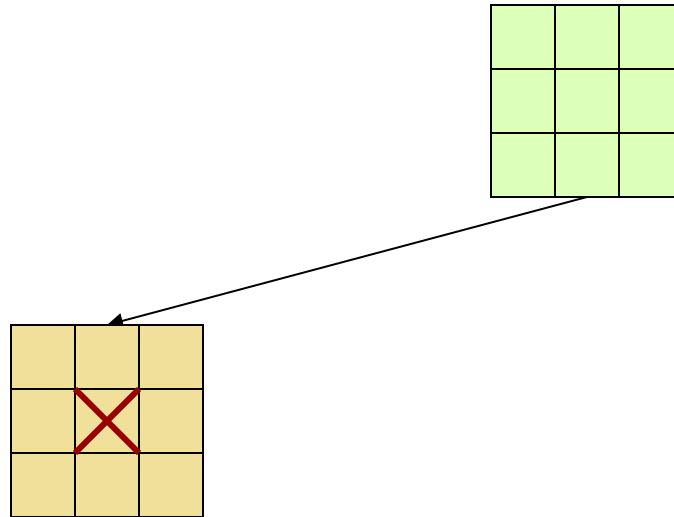
h	b^h
3	42875
5	5×10^7
Good	$\xrightarrow{} 10$
Master	$\xrightarrow{} 15$
	3×10^{15}
	1×10^{23}

Can we do better?

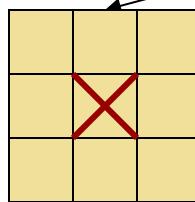
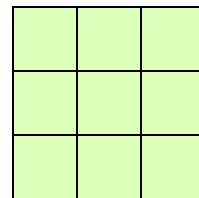
- Yes!



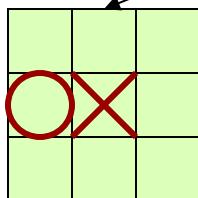
Example



Example



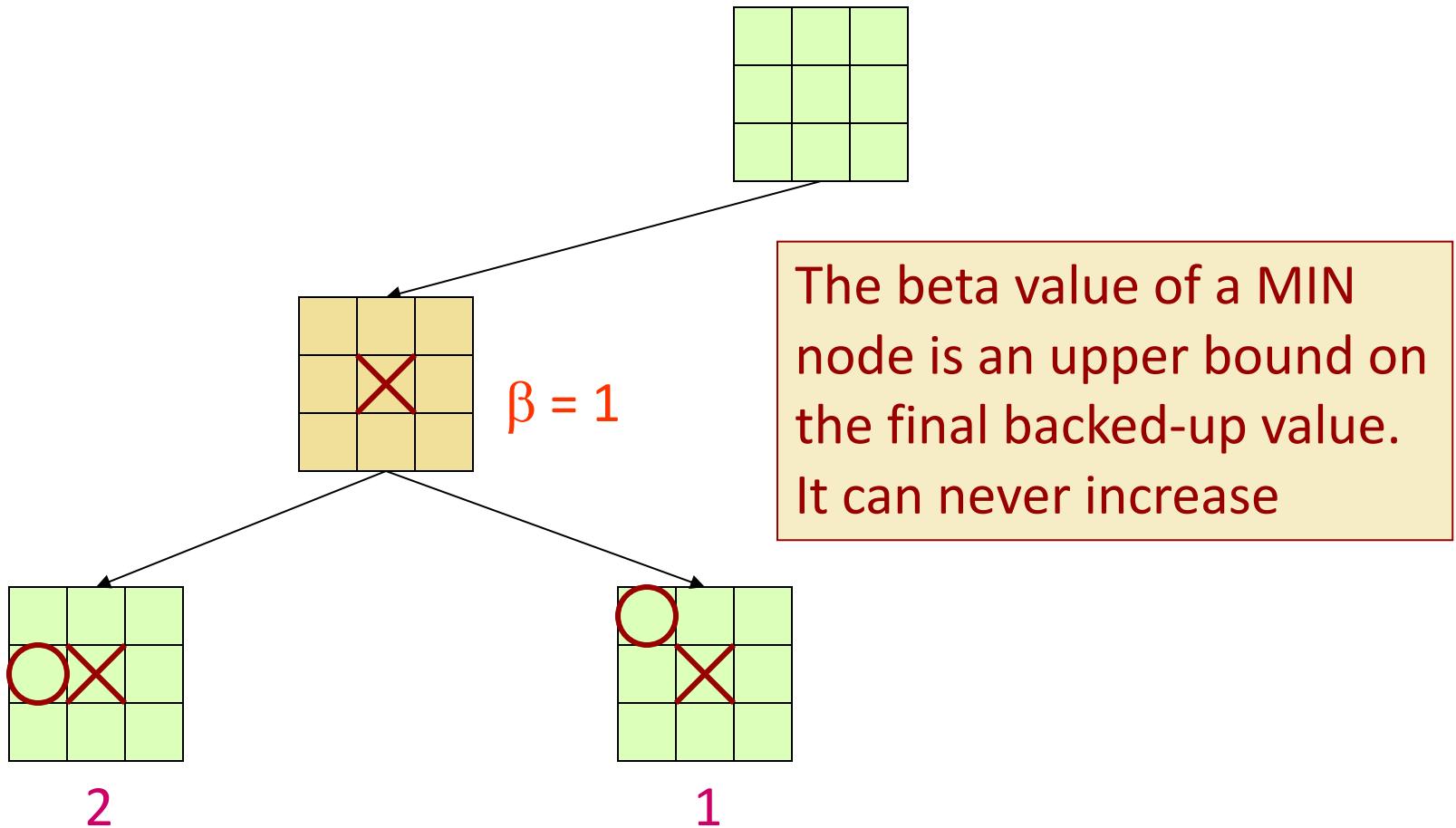
$\beta = 2$



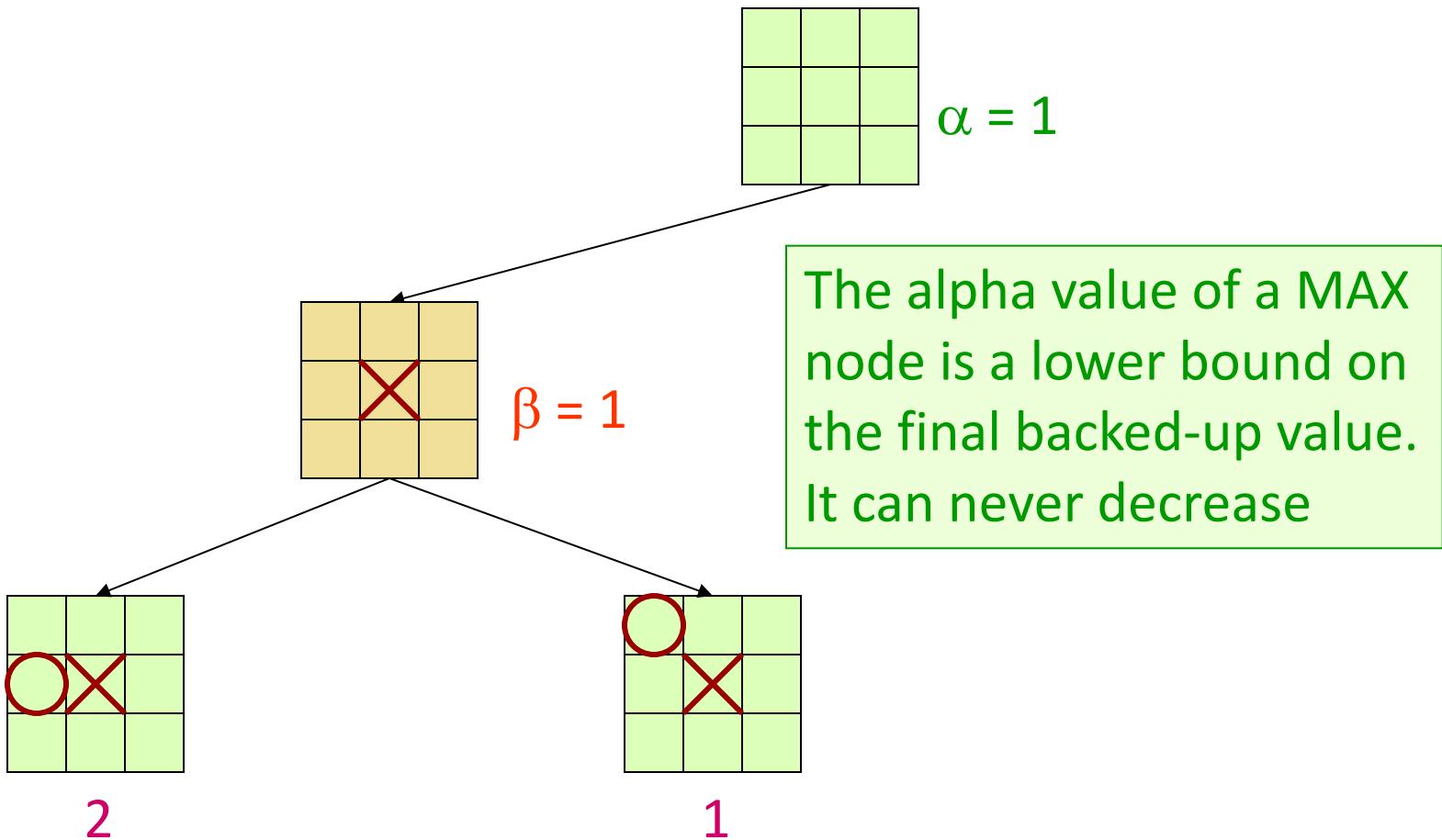
2

The beta value of a MIN node is an upper bound on the final backed-up value.
It can never increase

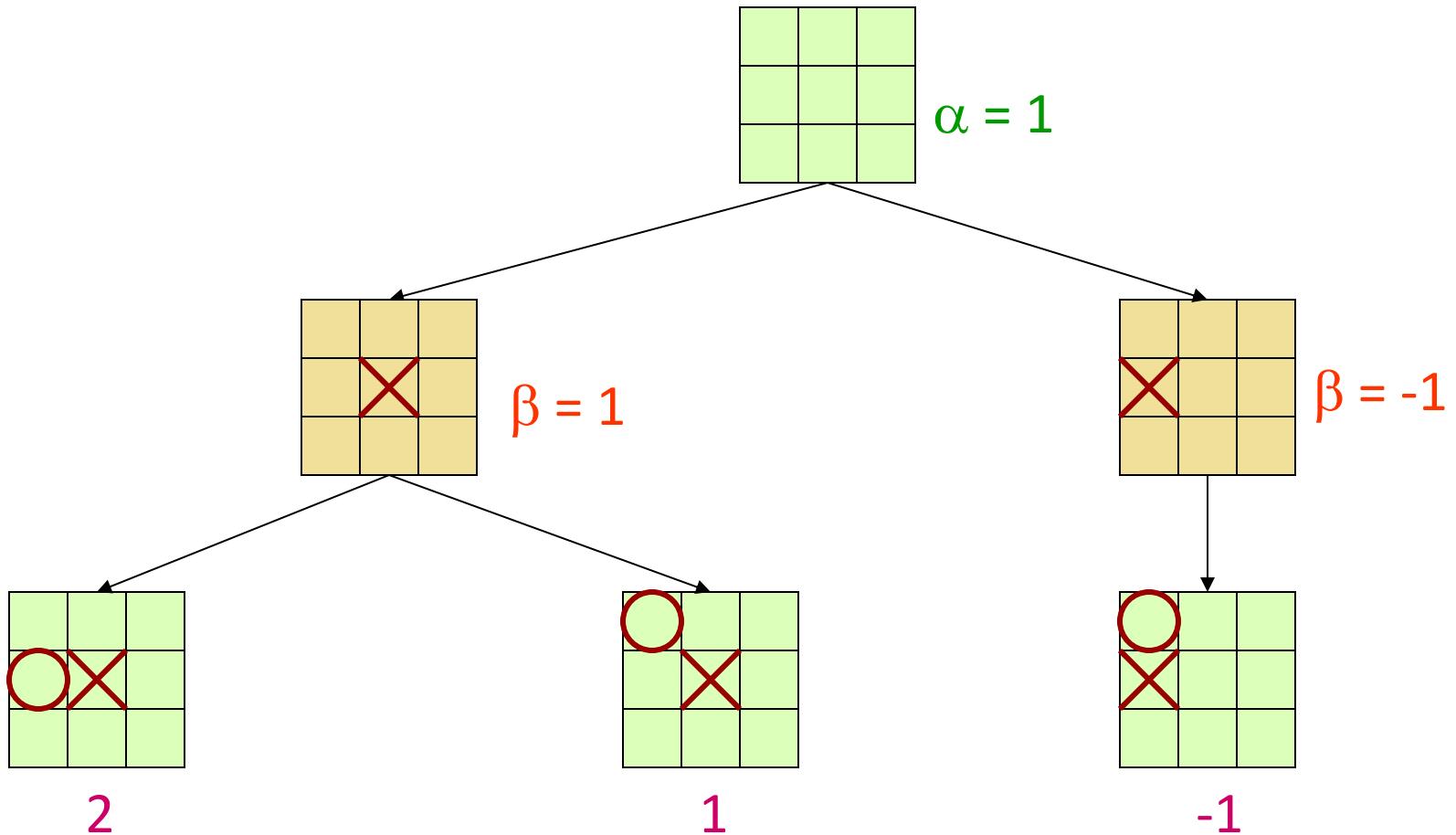
Example



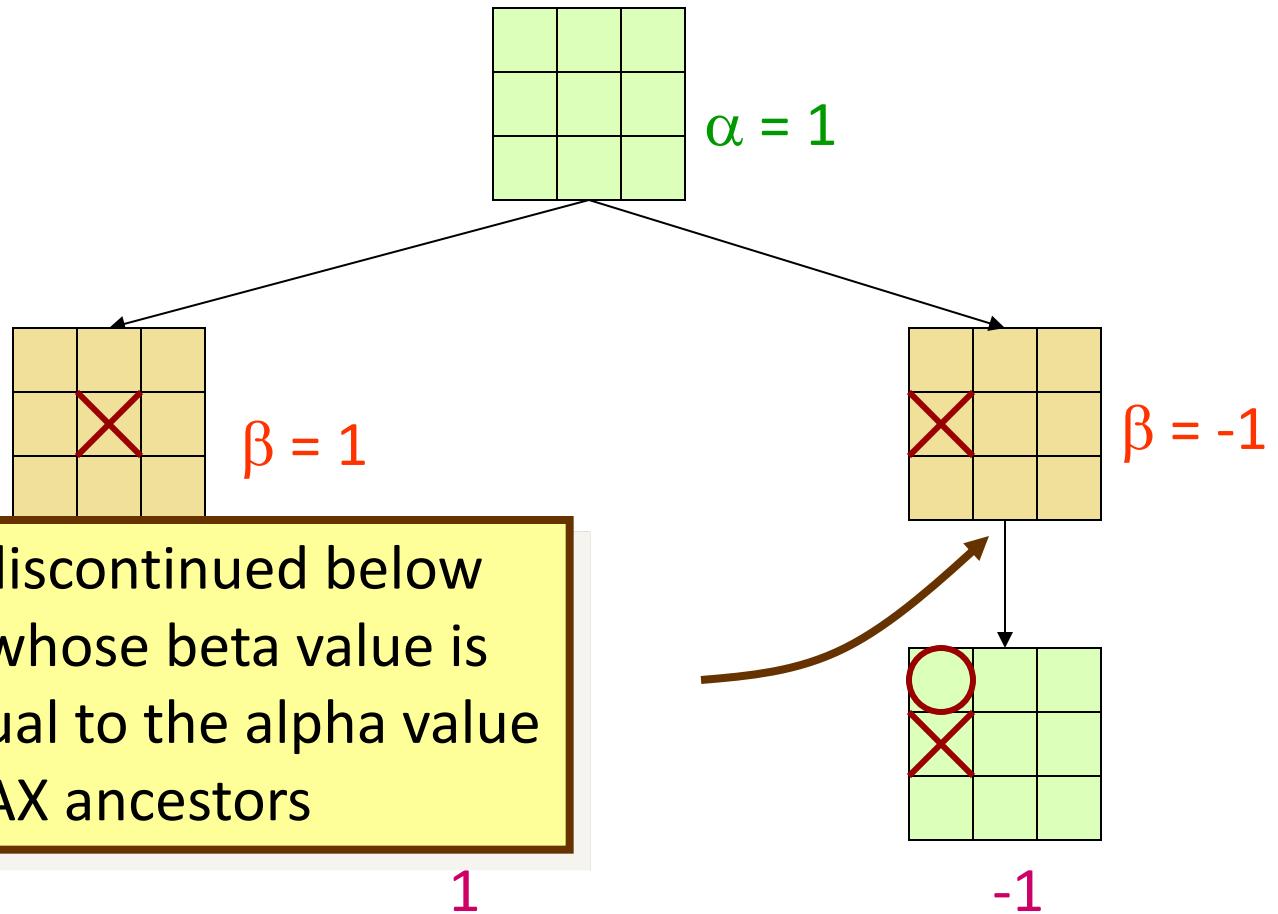
Example



Example



Example



Alpha-Beta Pruning

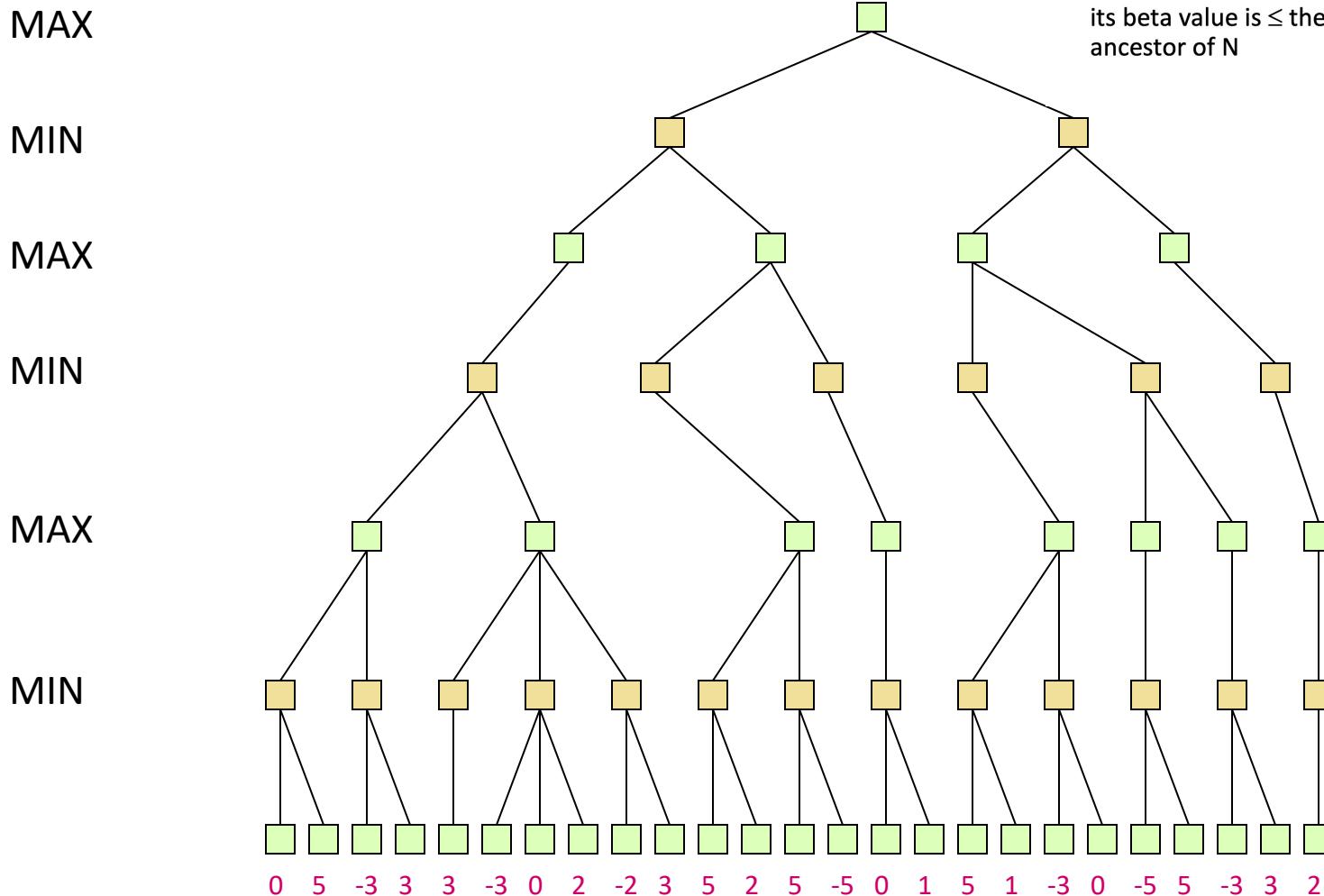
- Explore the game tree to depth h in **depth-first** manner
- Back up alpha and beta values whenever possible
- Prune branches that can't lead to changing the final decision

Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N

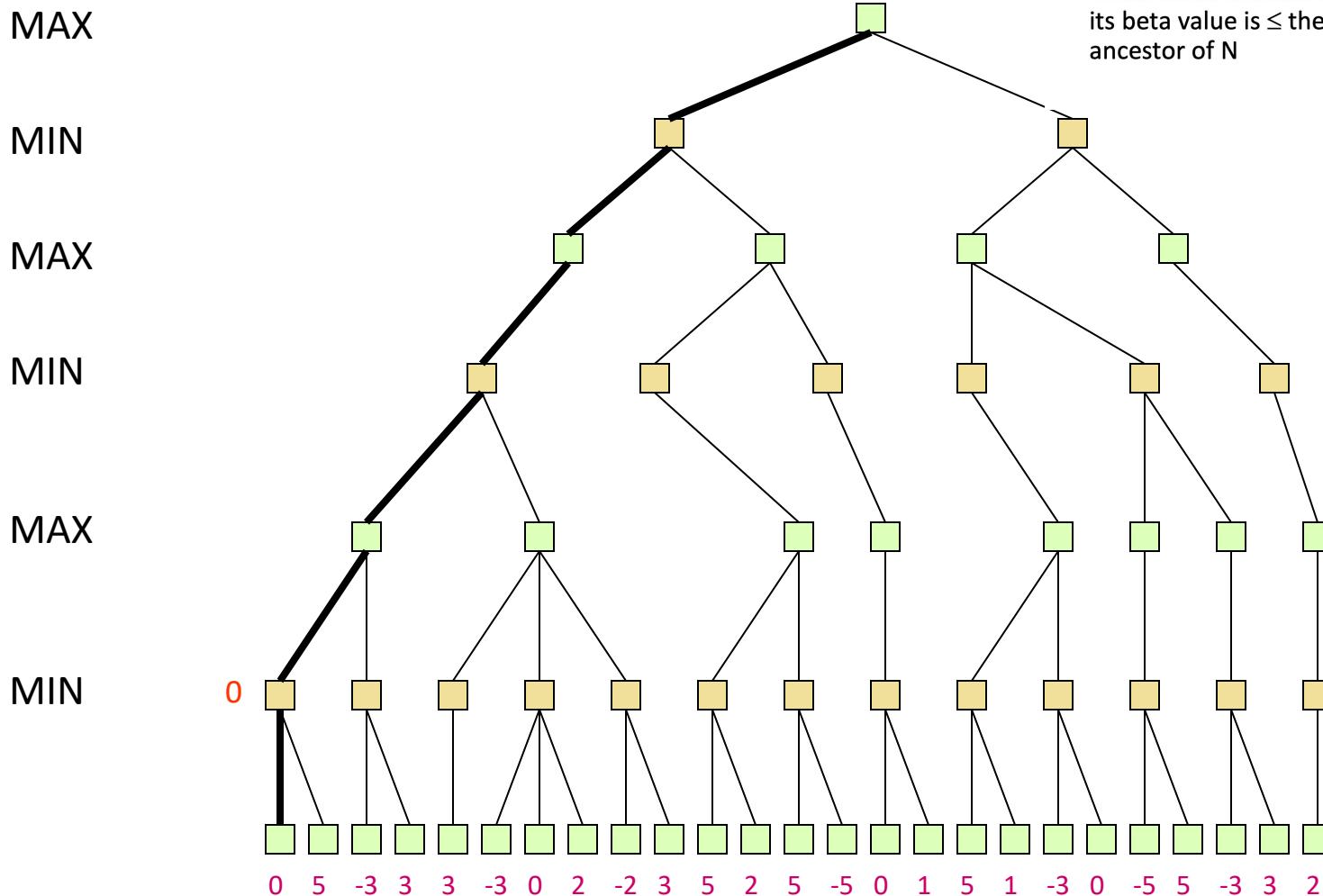
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



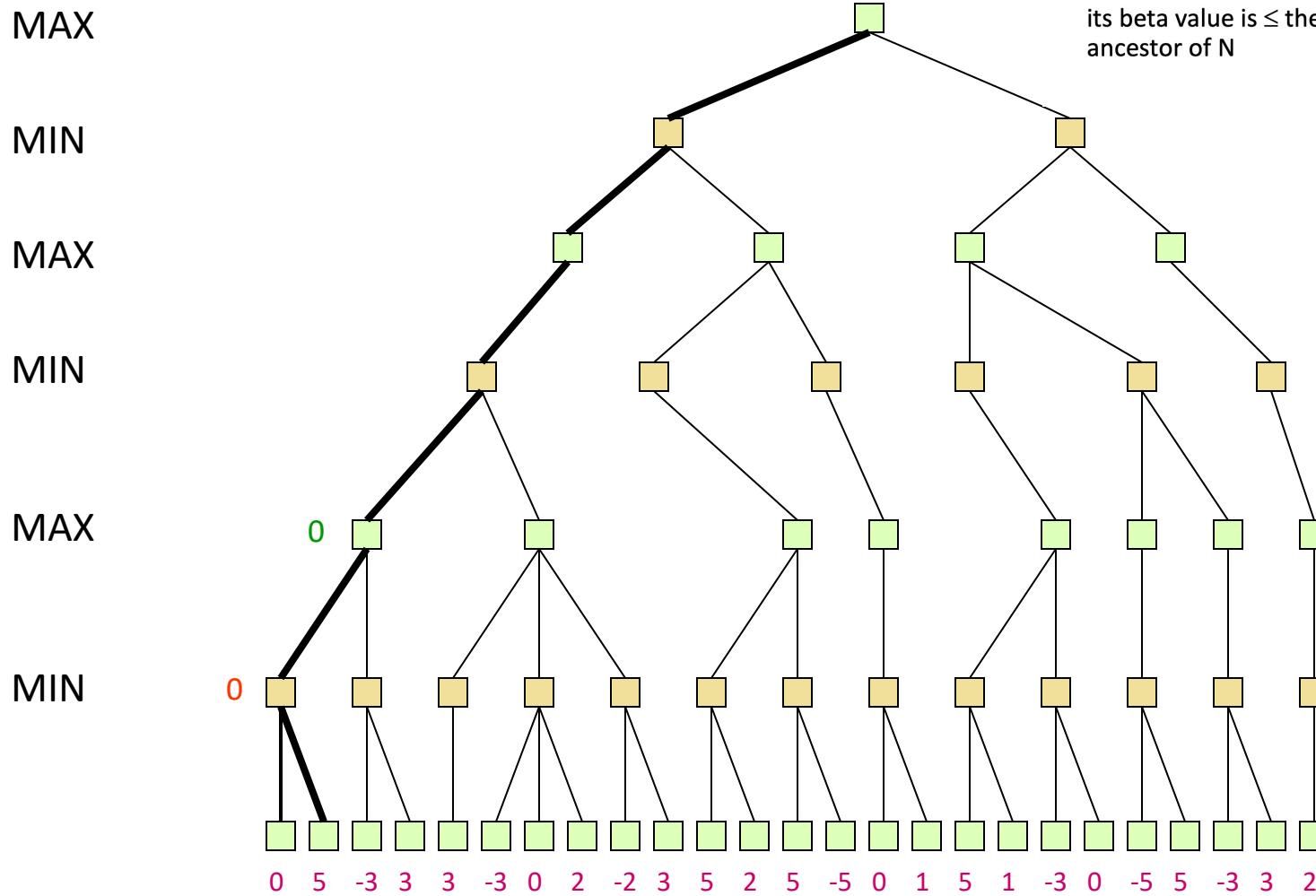
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



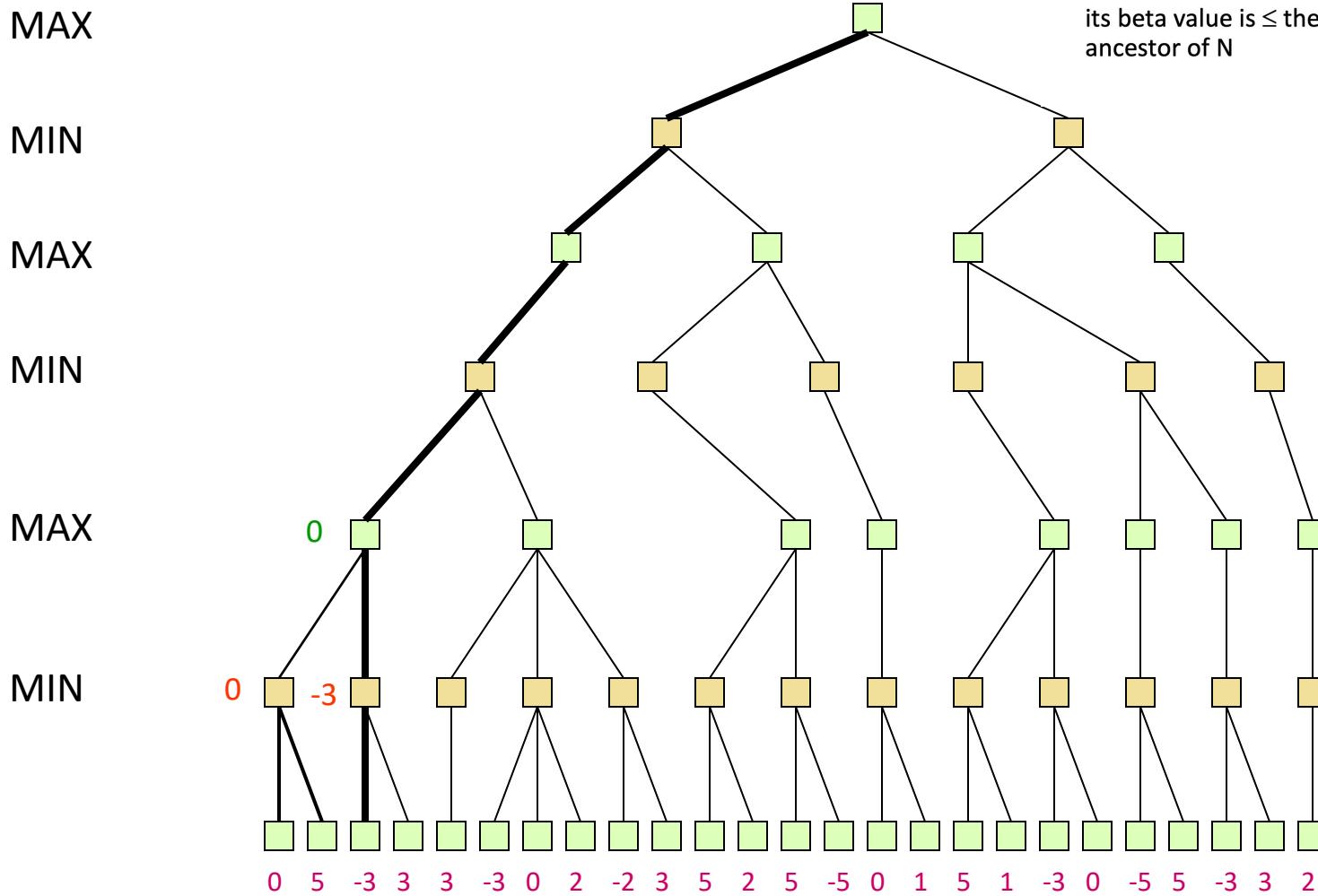
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



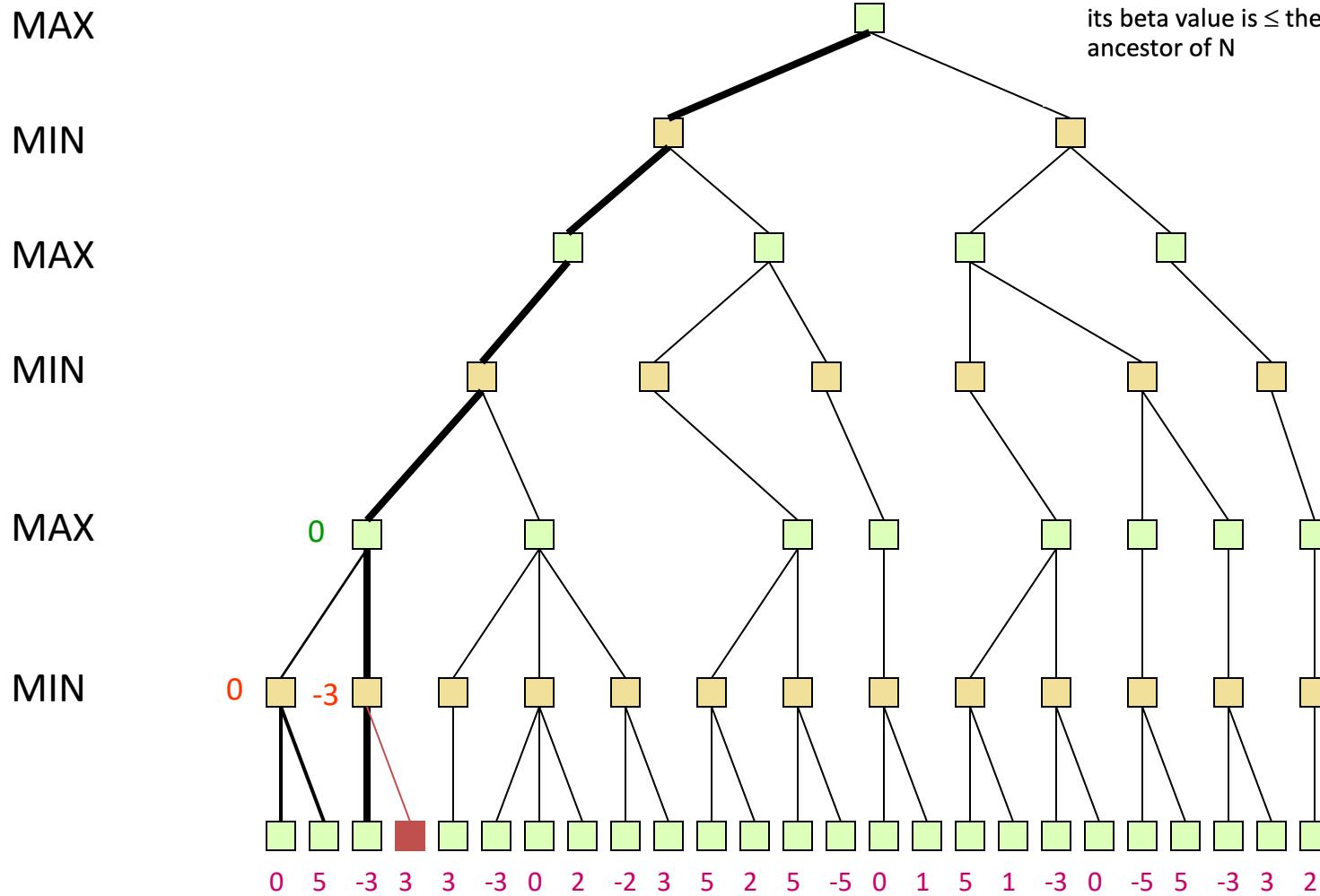
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



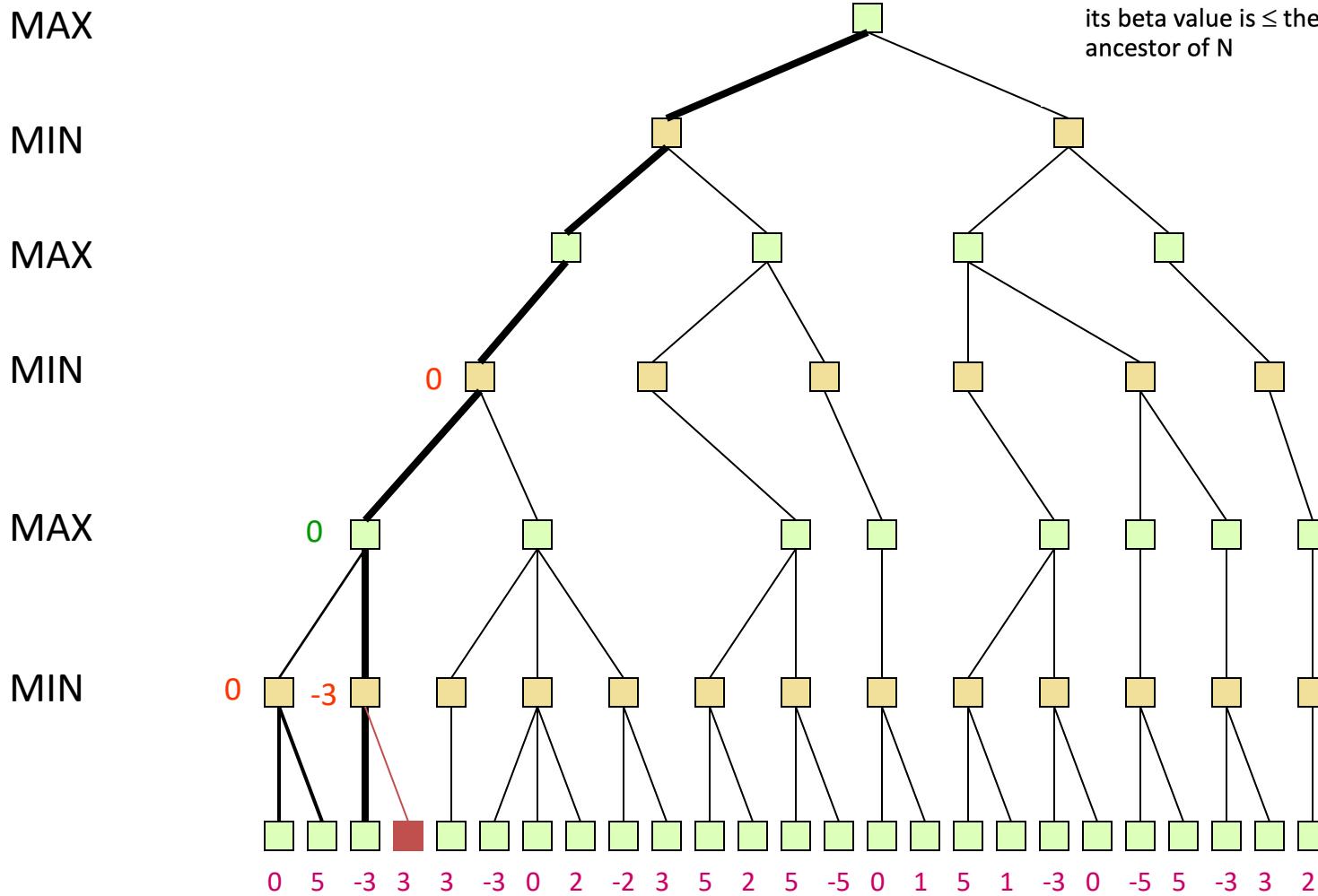
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



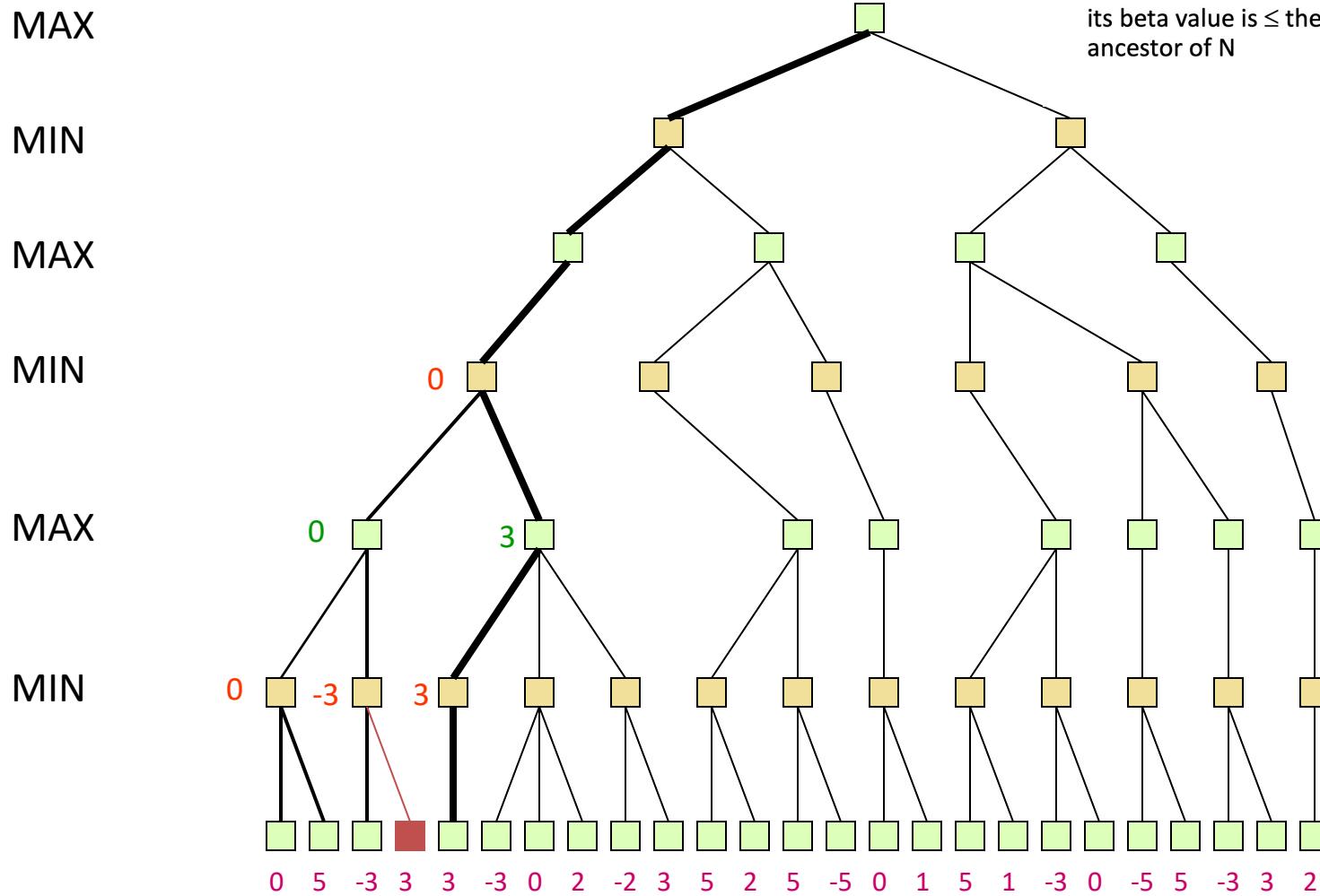
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



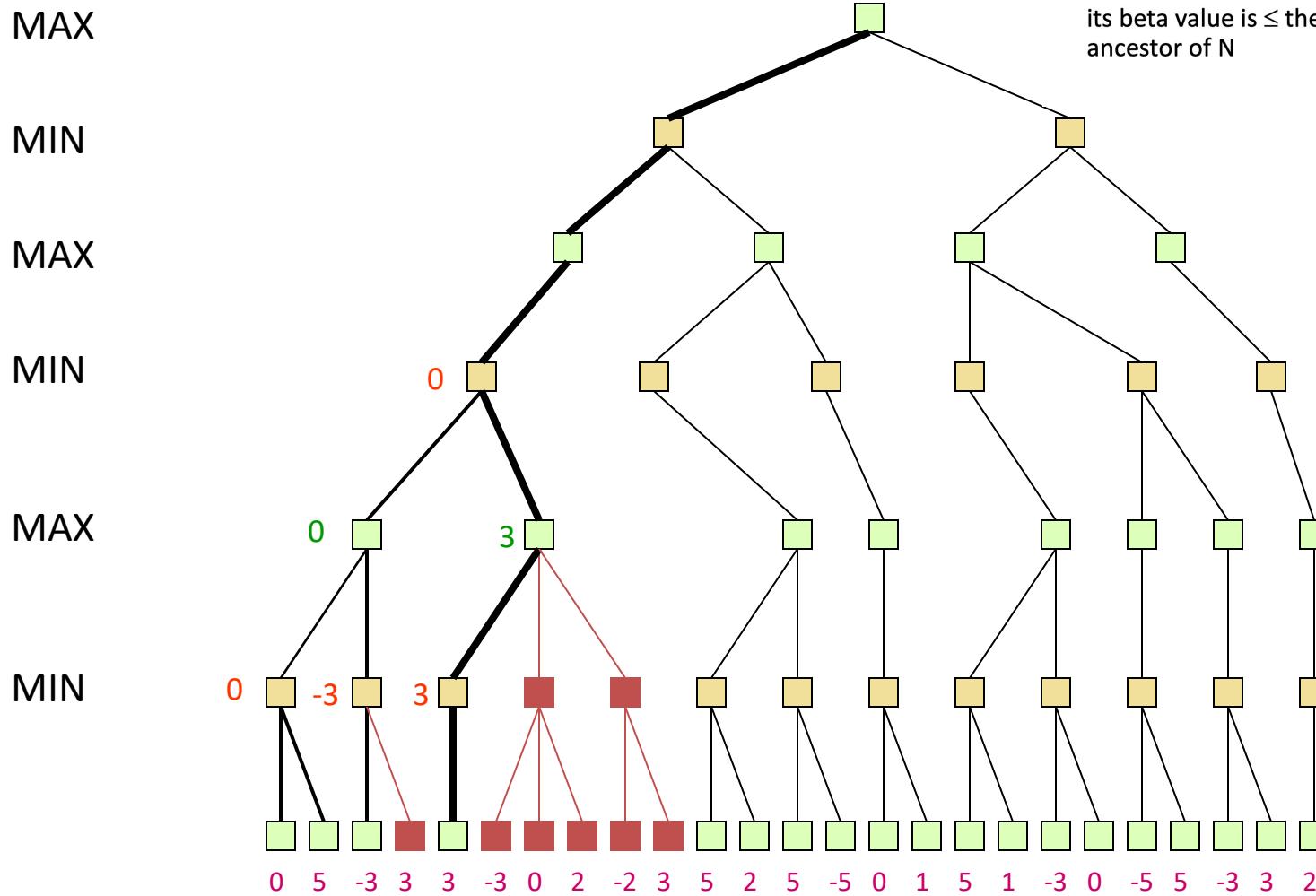
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



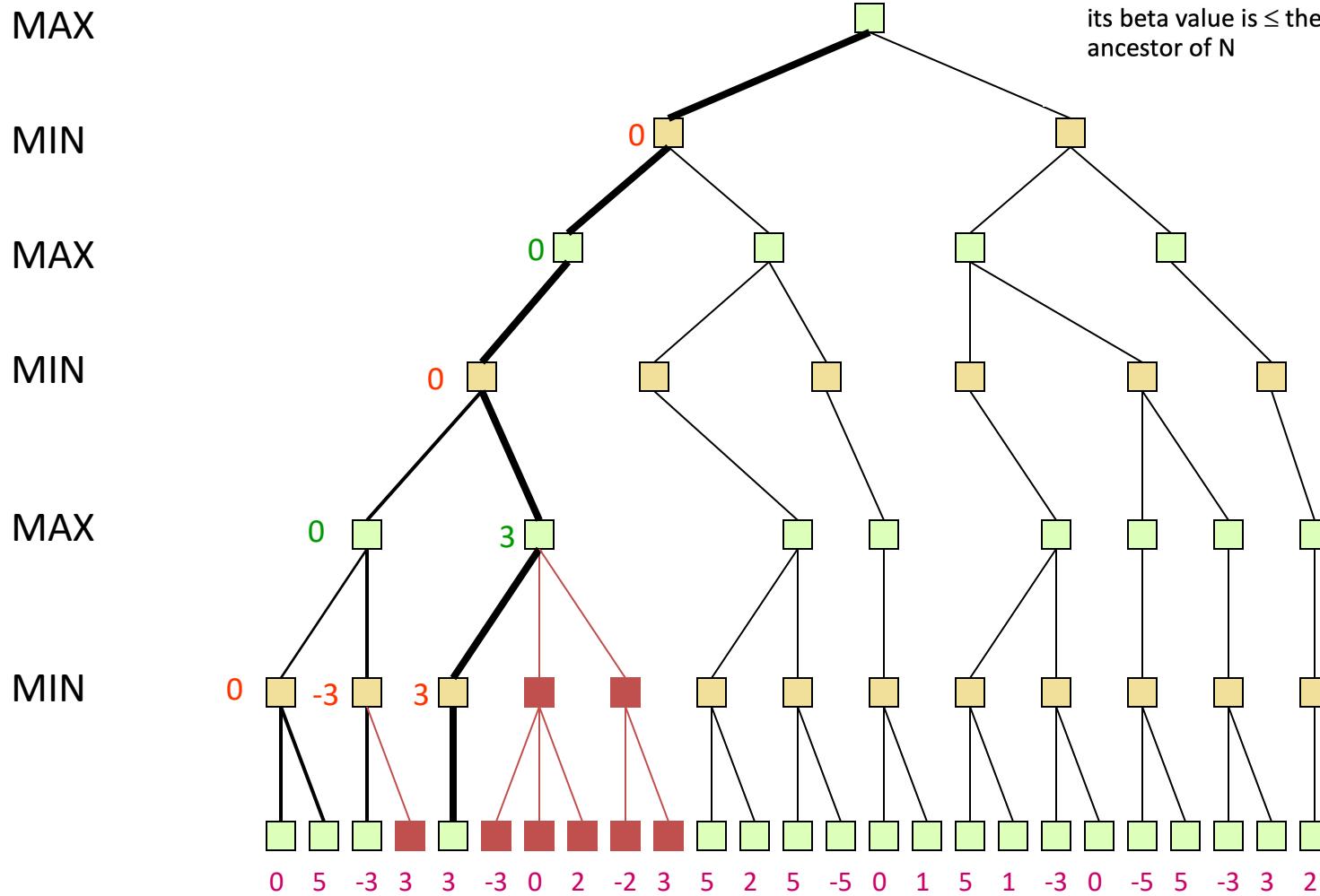
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



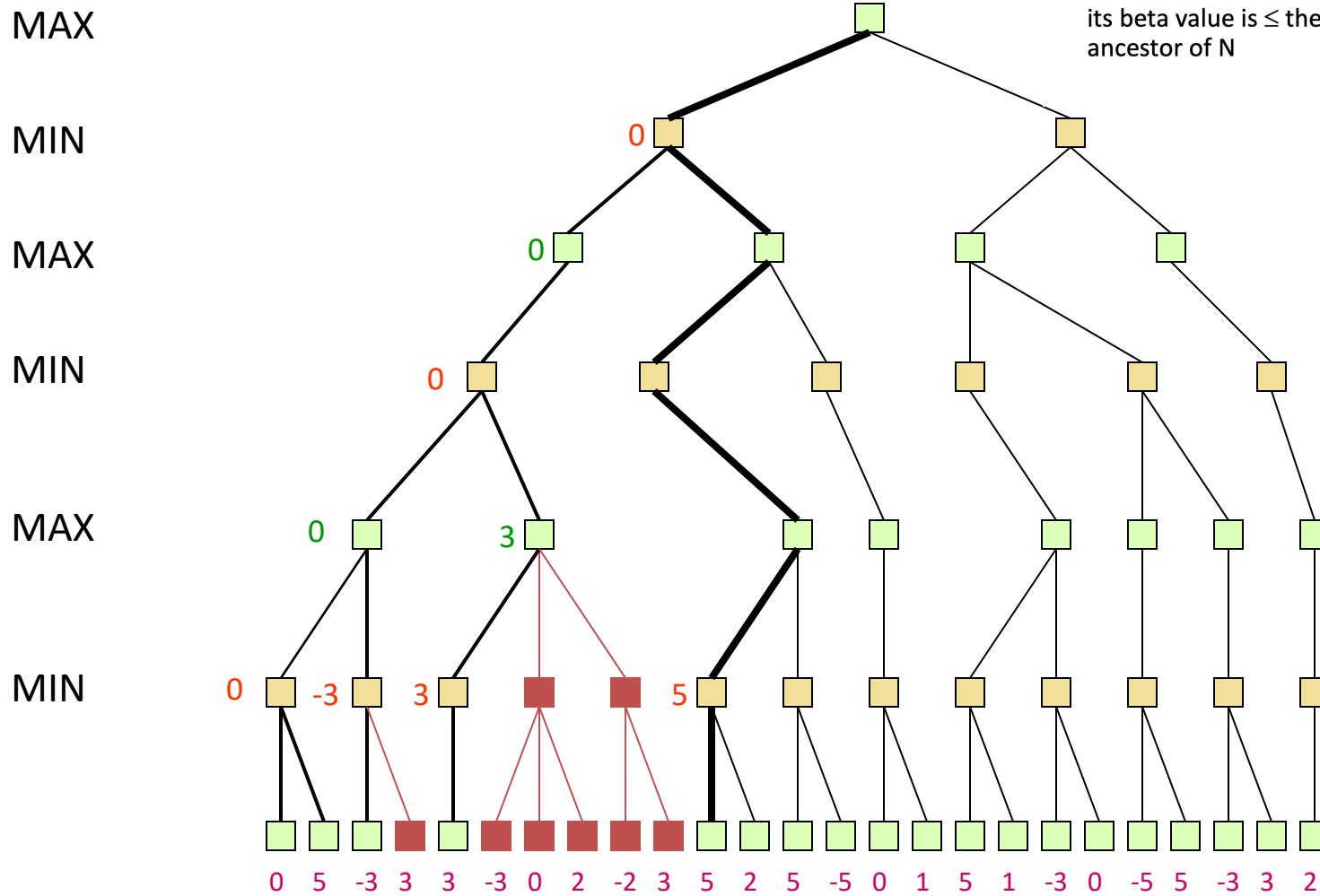
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



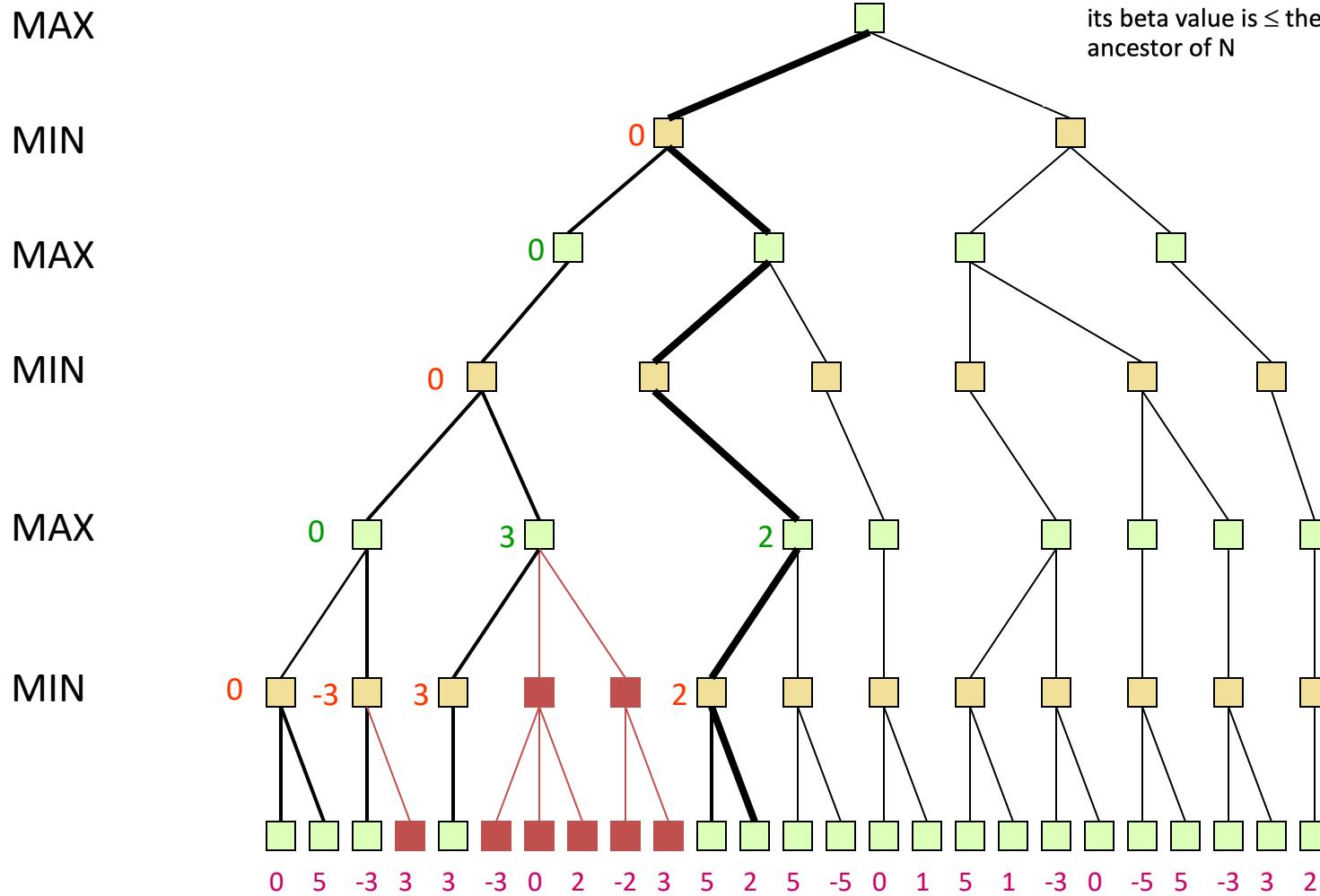
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



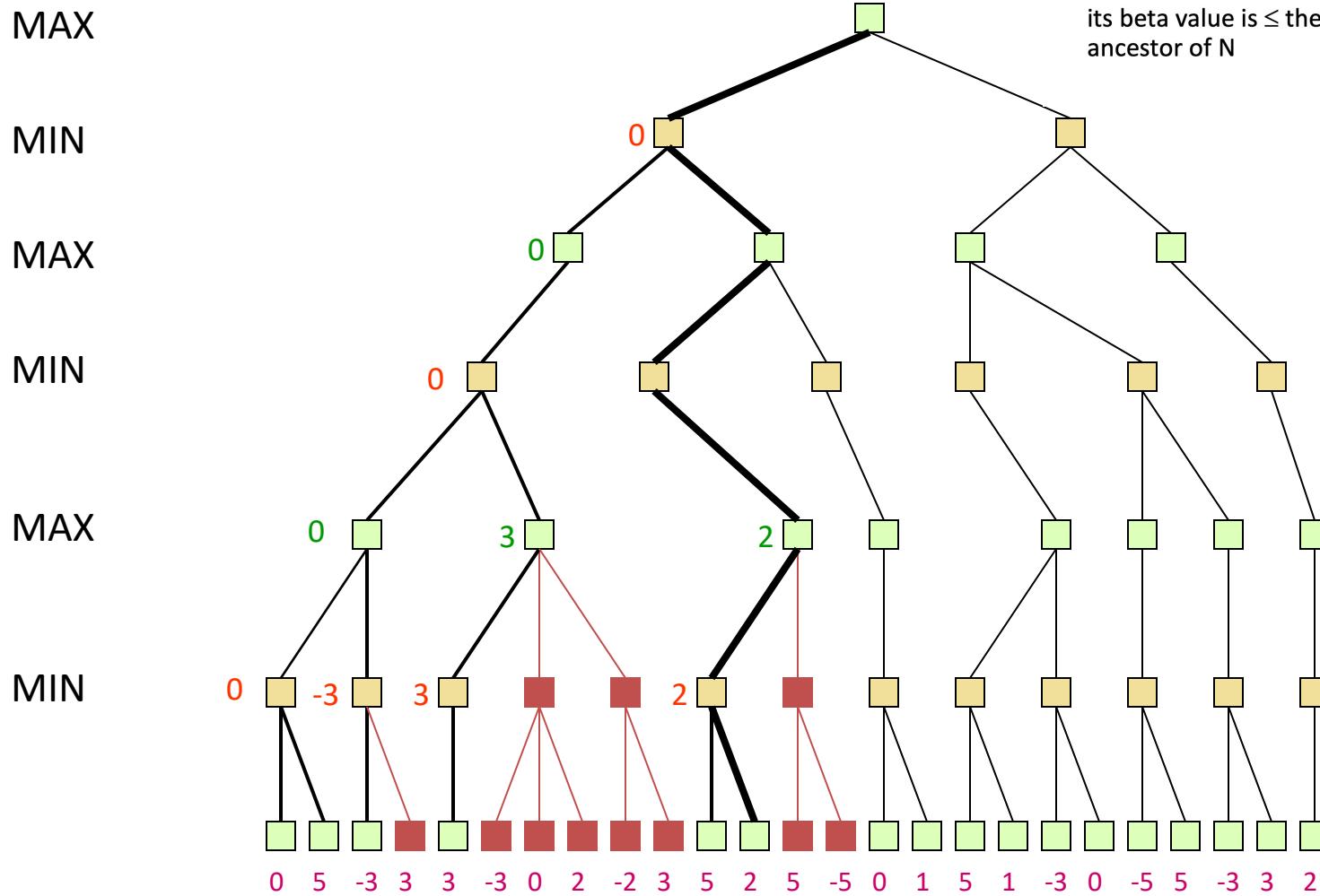
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



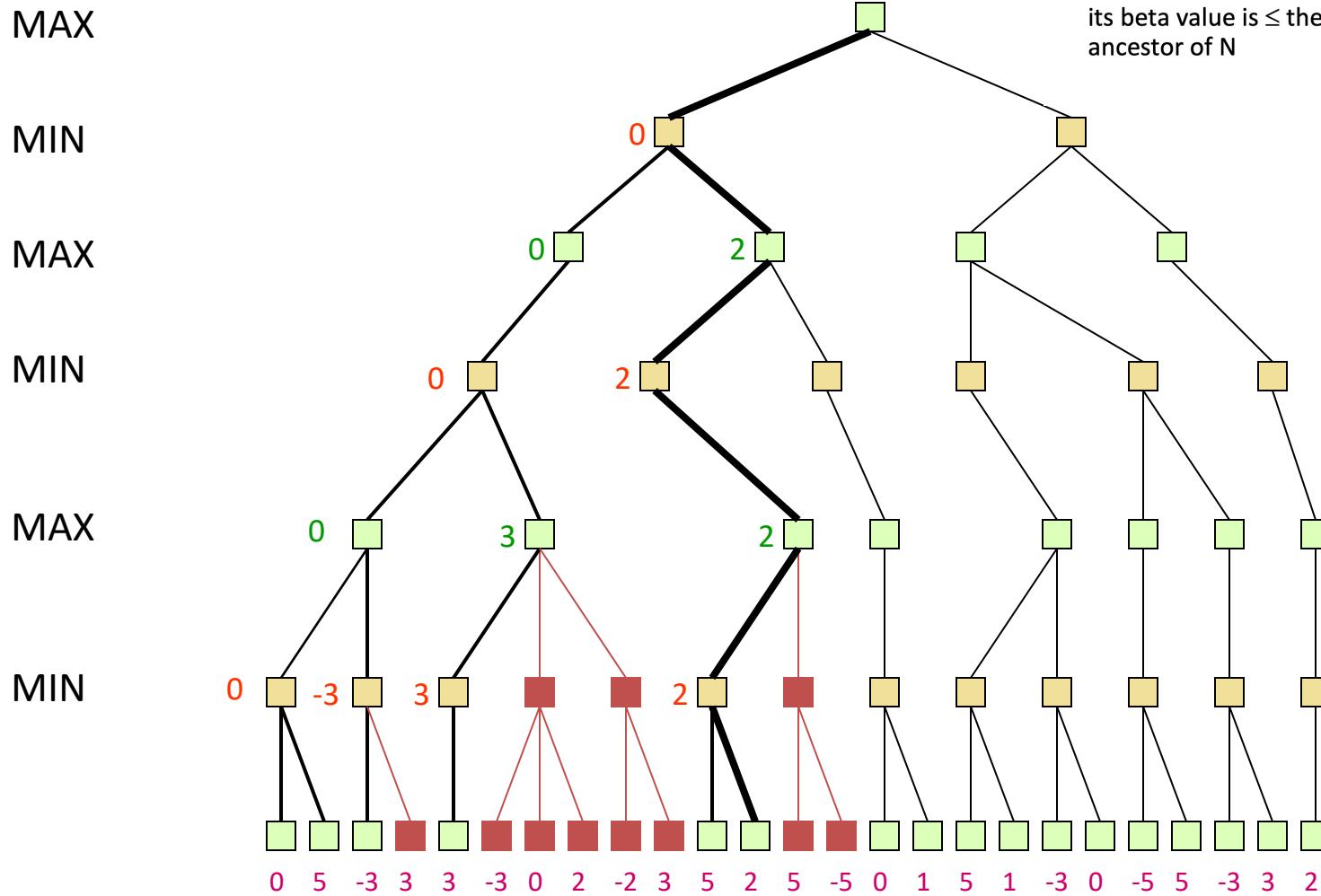
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



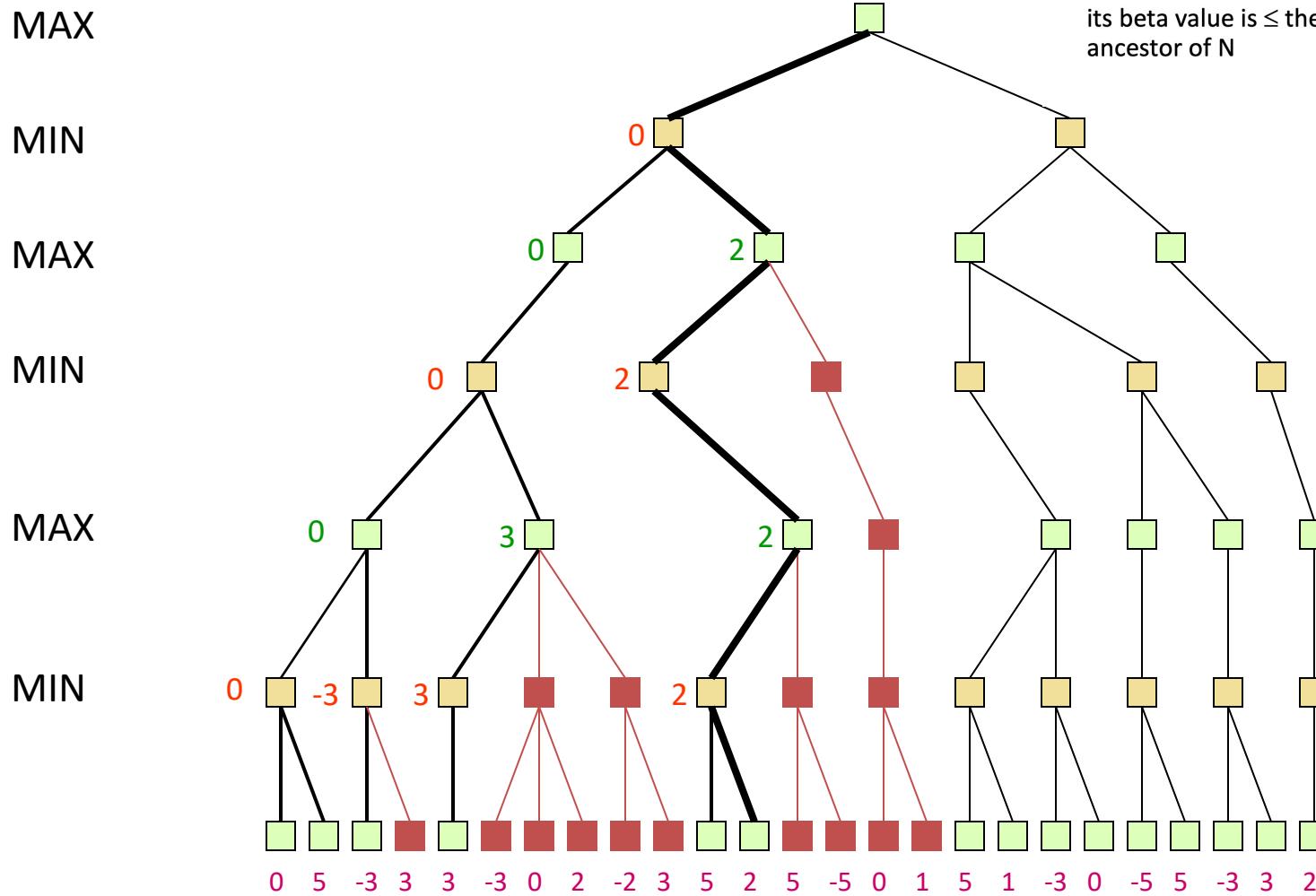
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



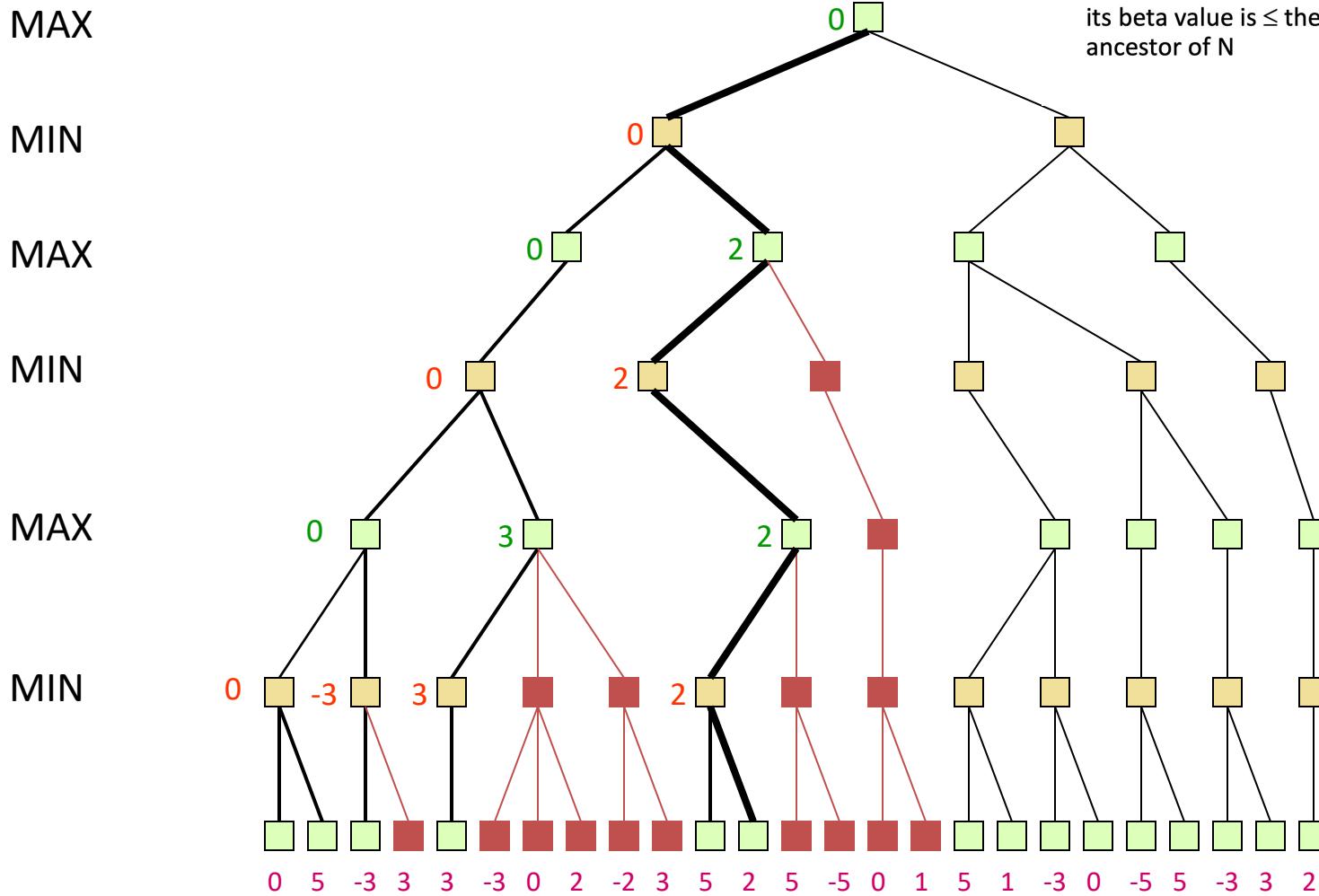
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



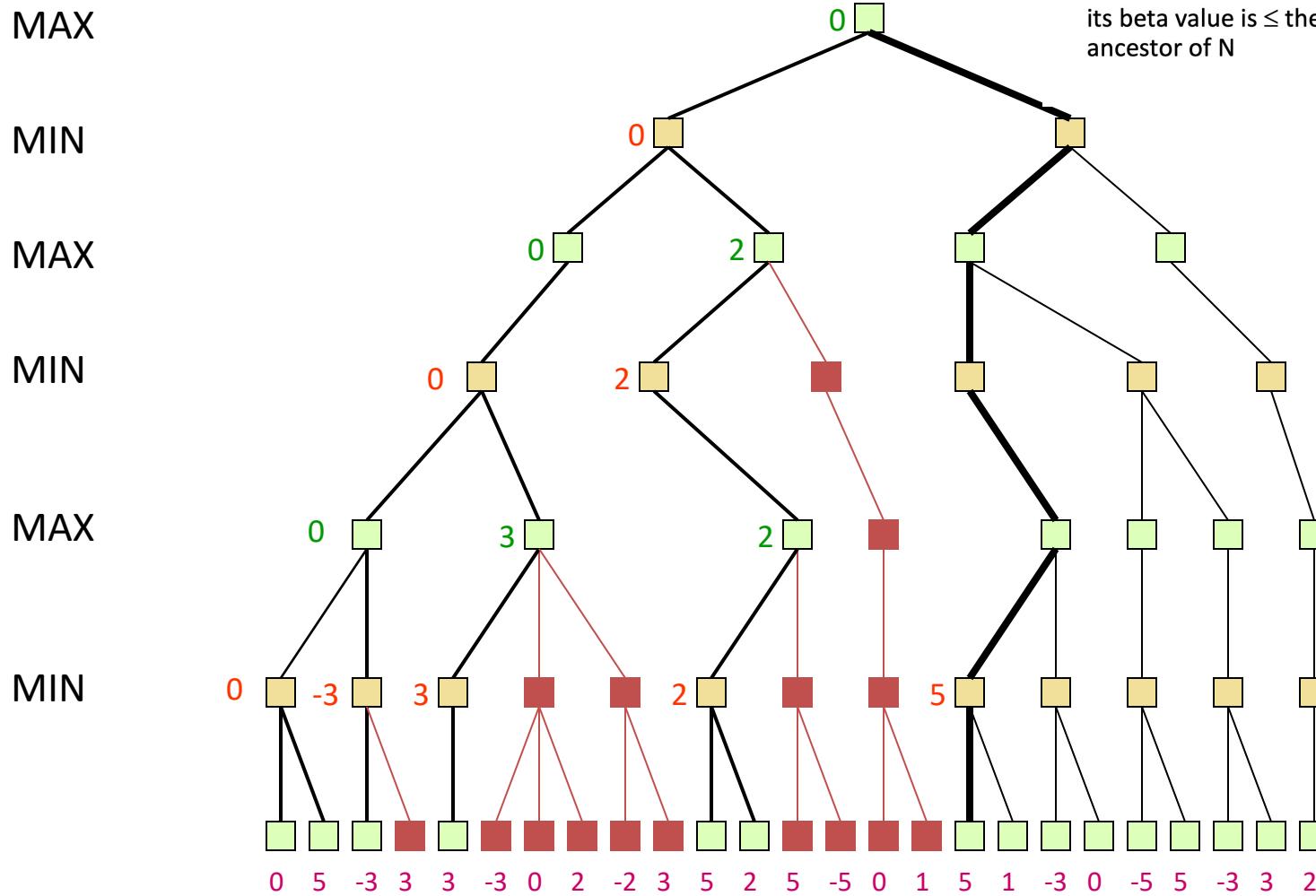
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



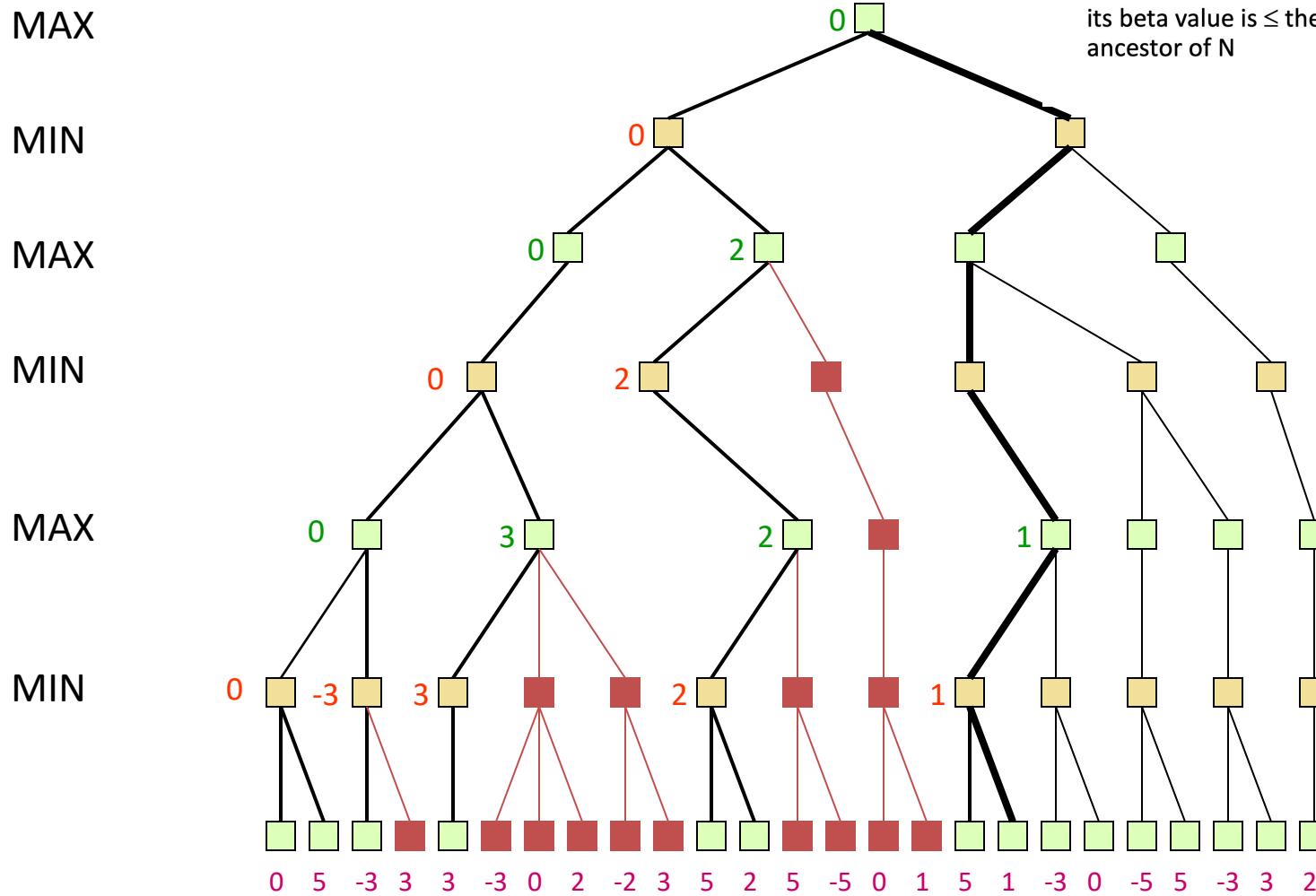
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



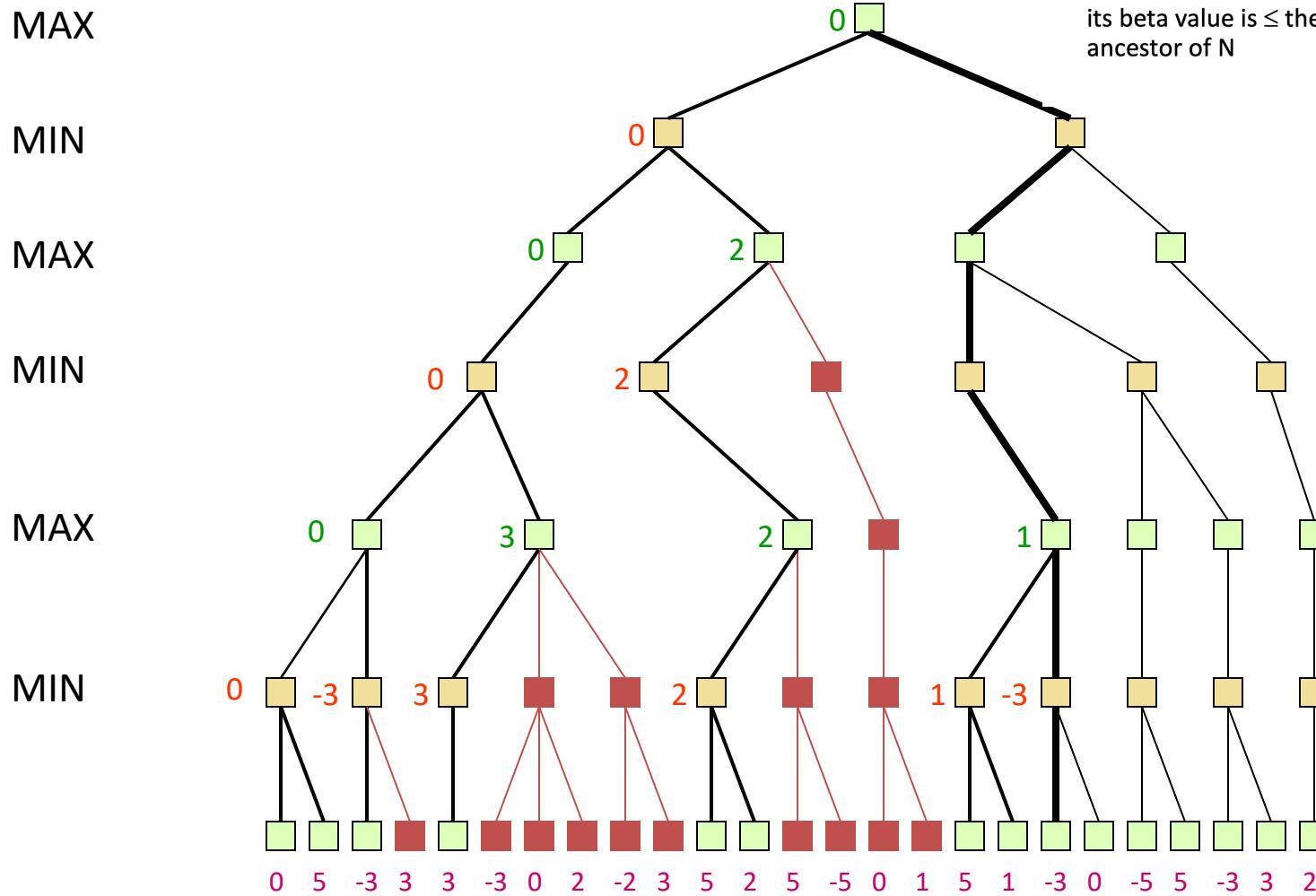
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



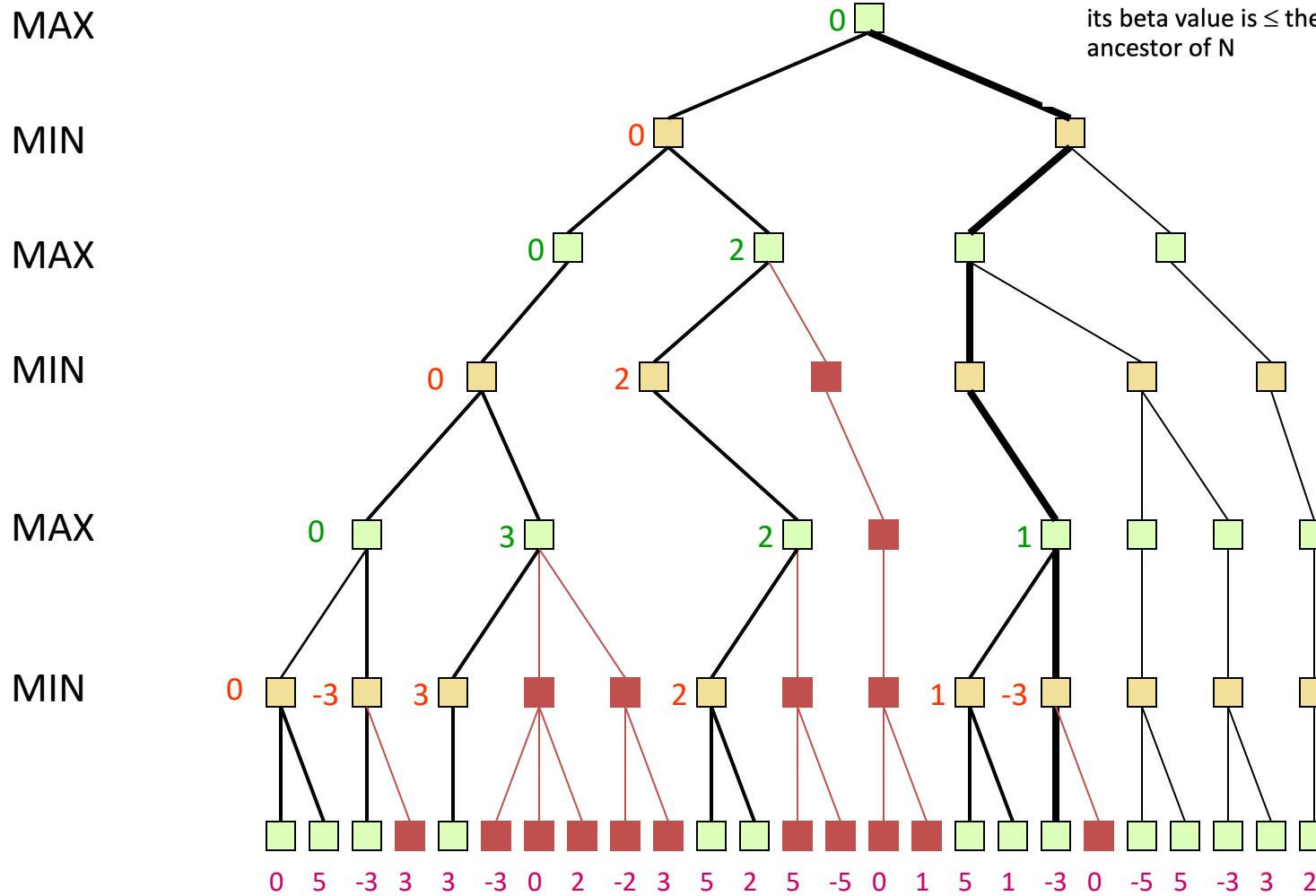
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



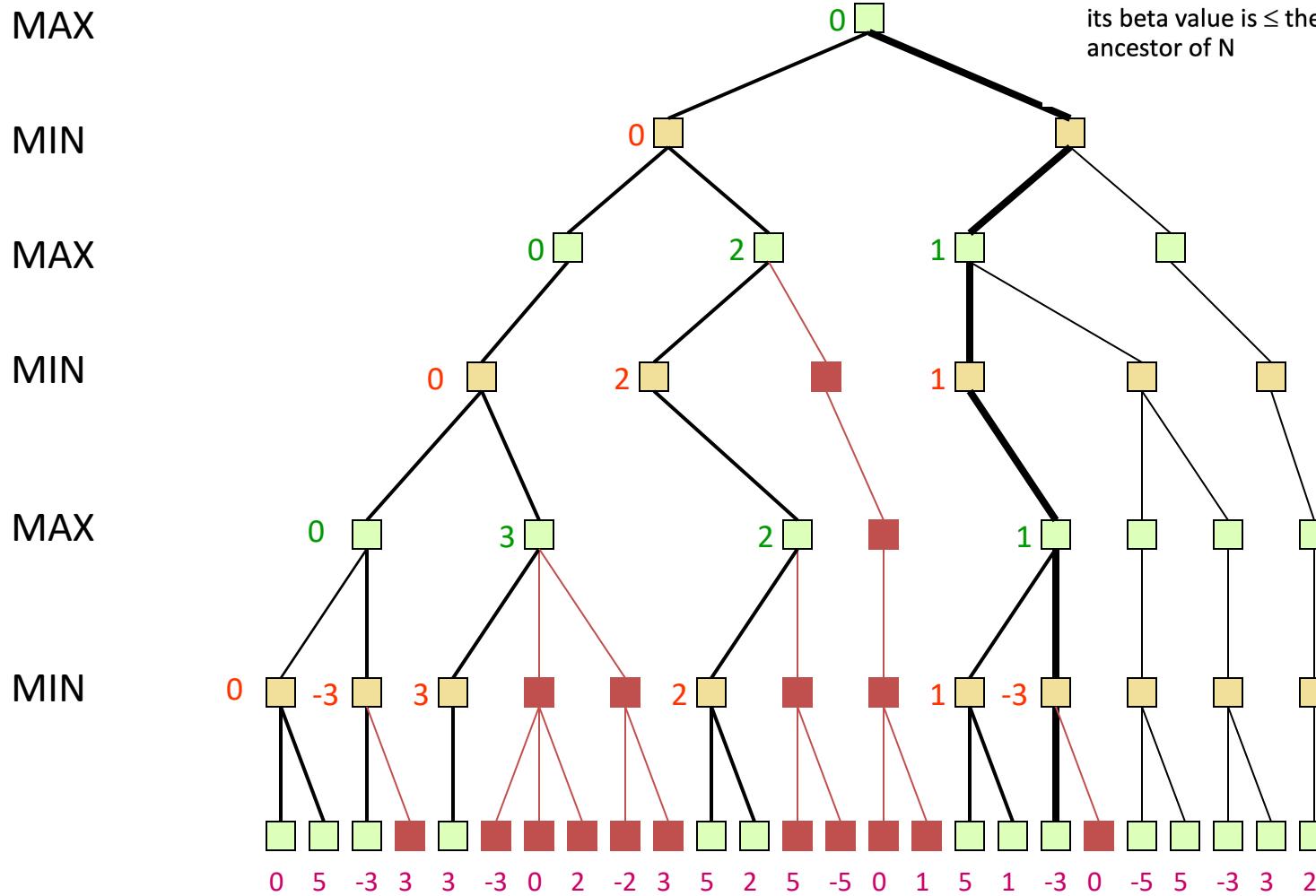
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



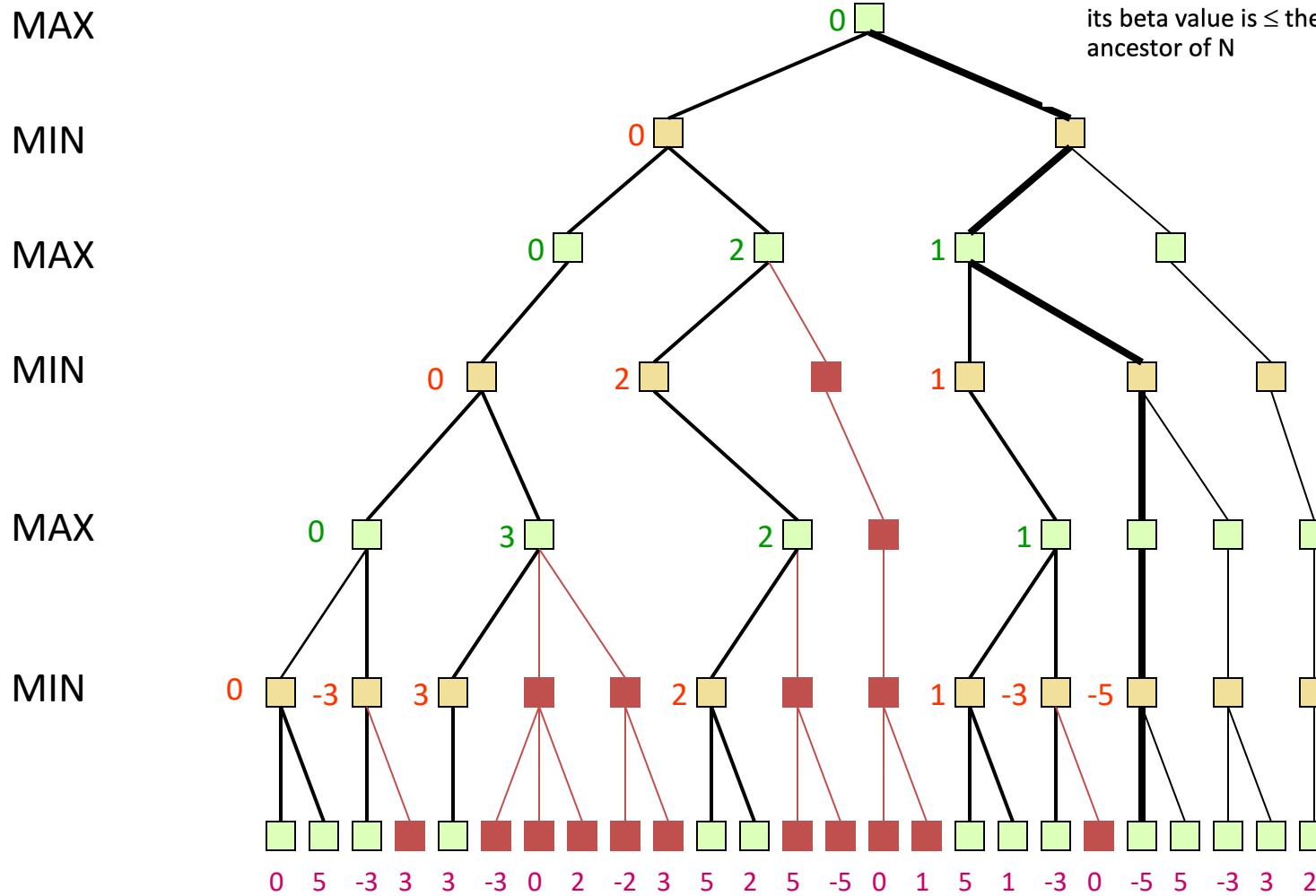
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



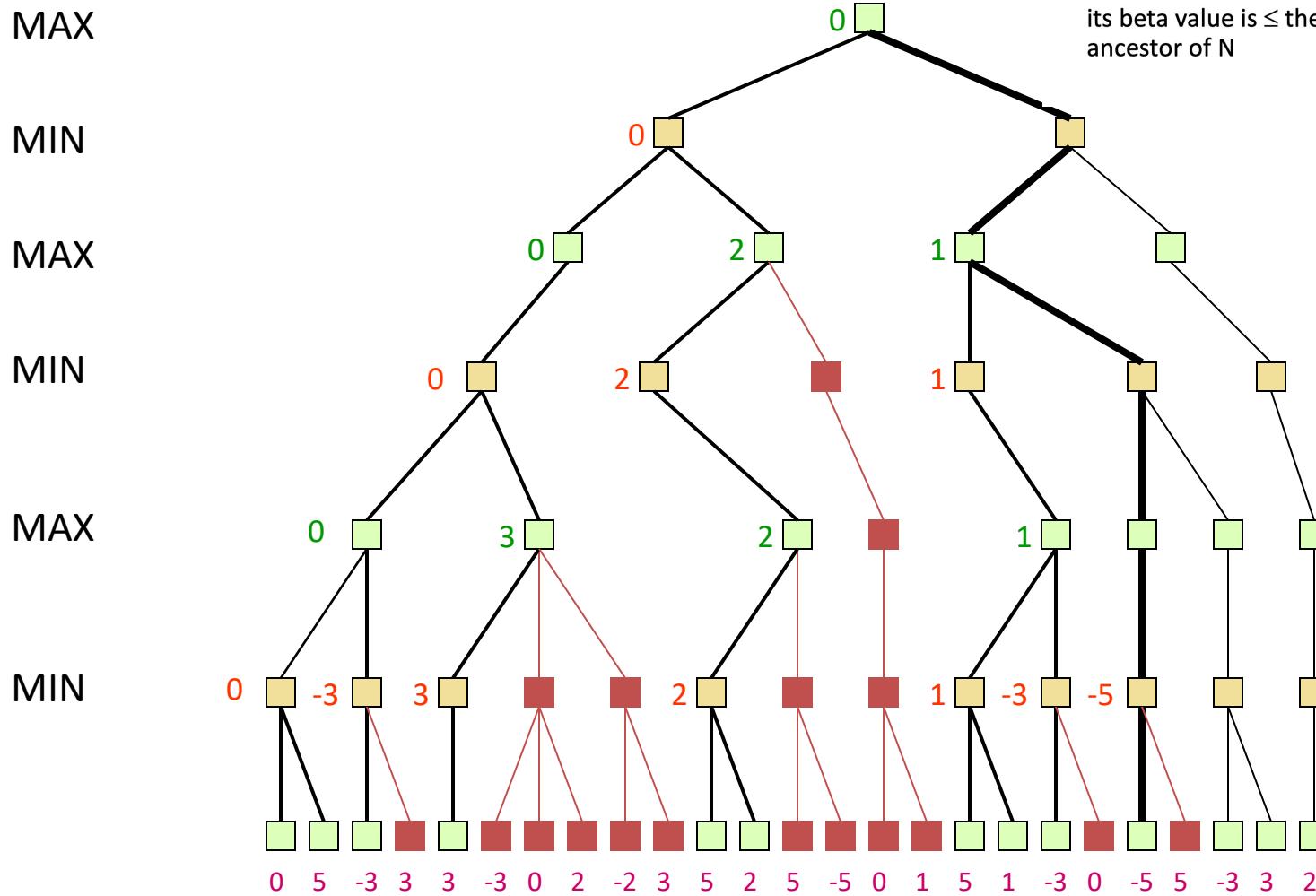
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



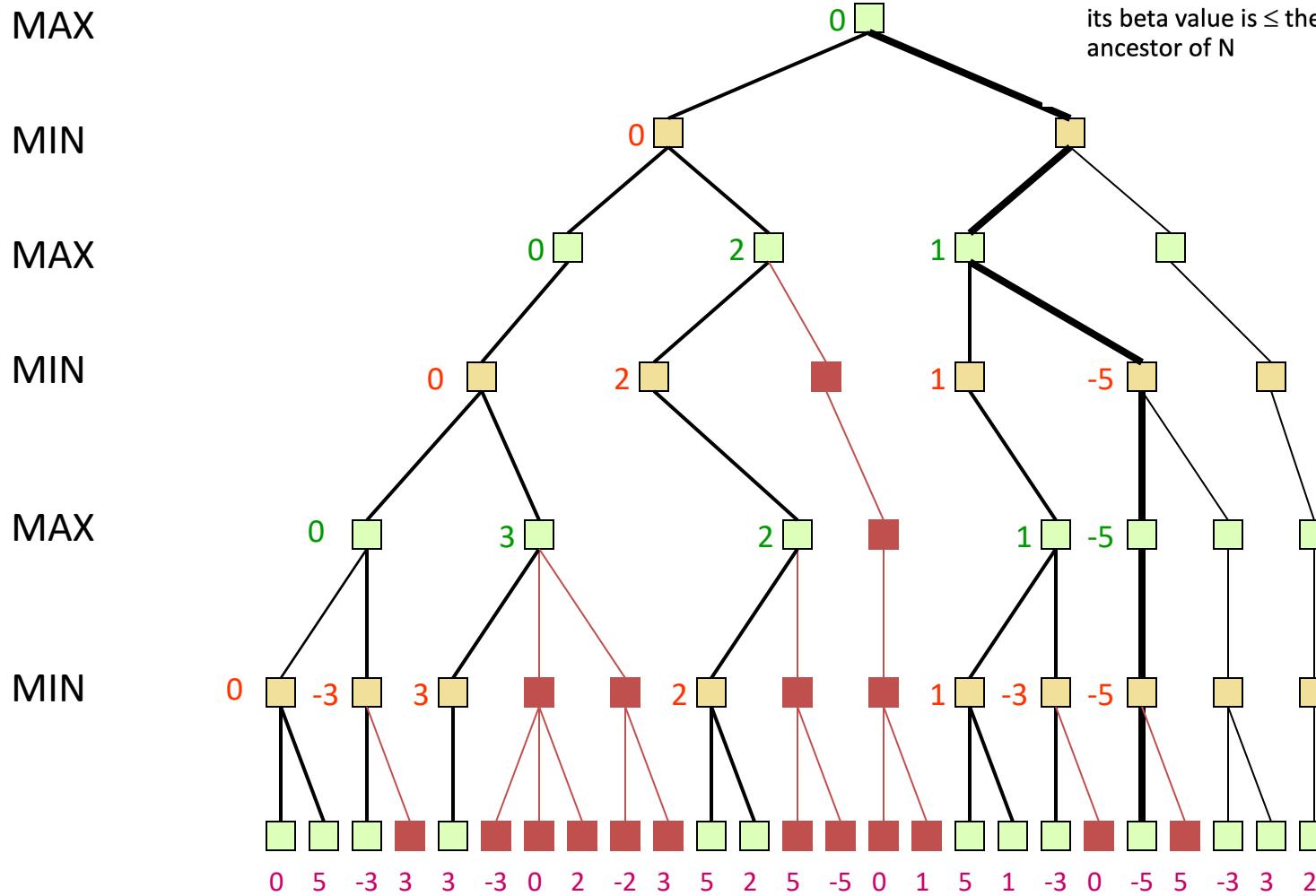
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



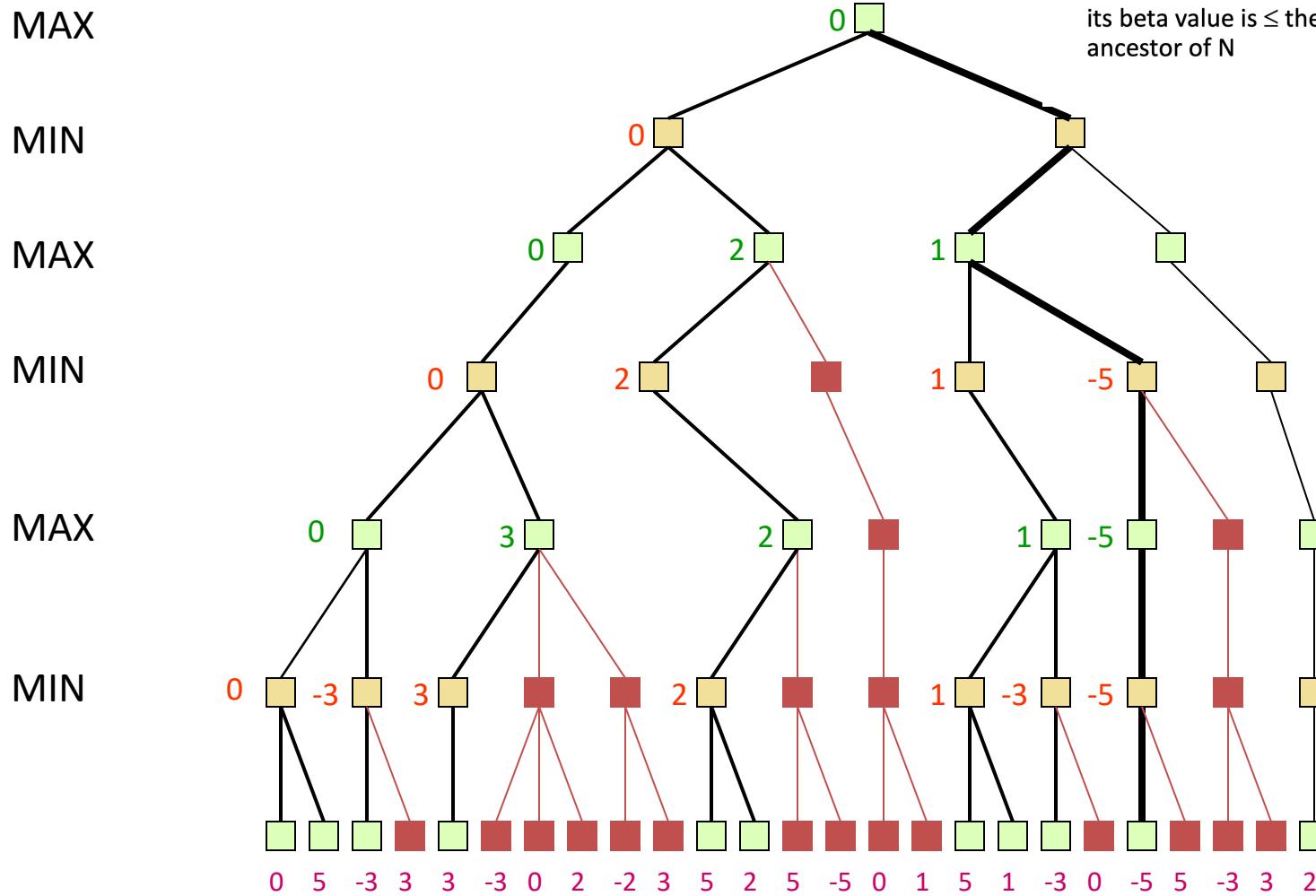
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



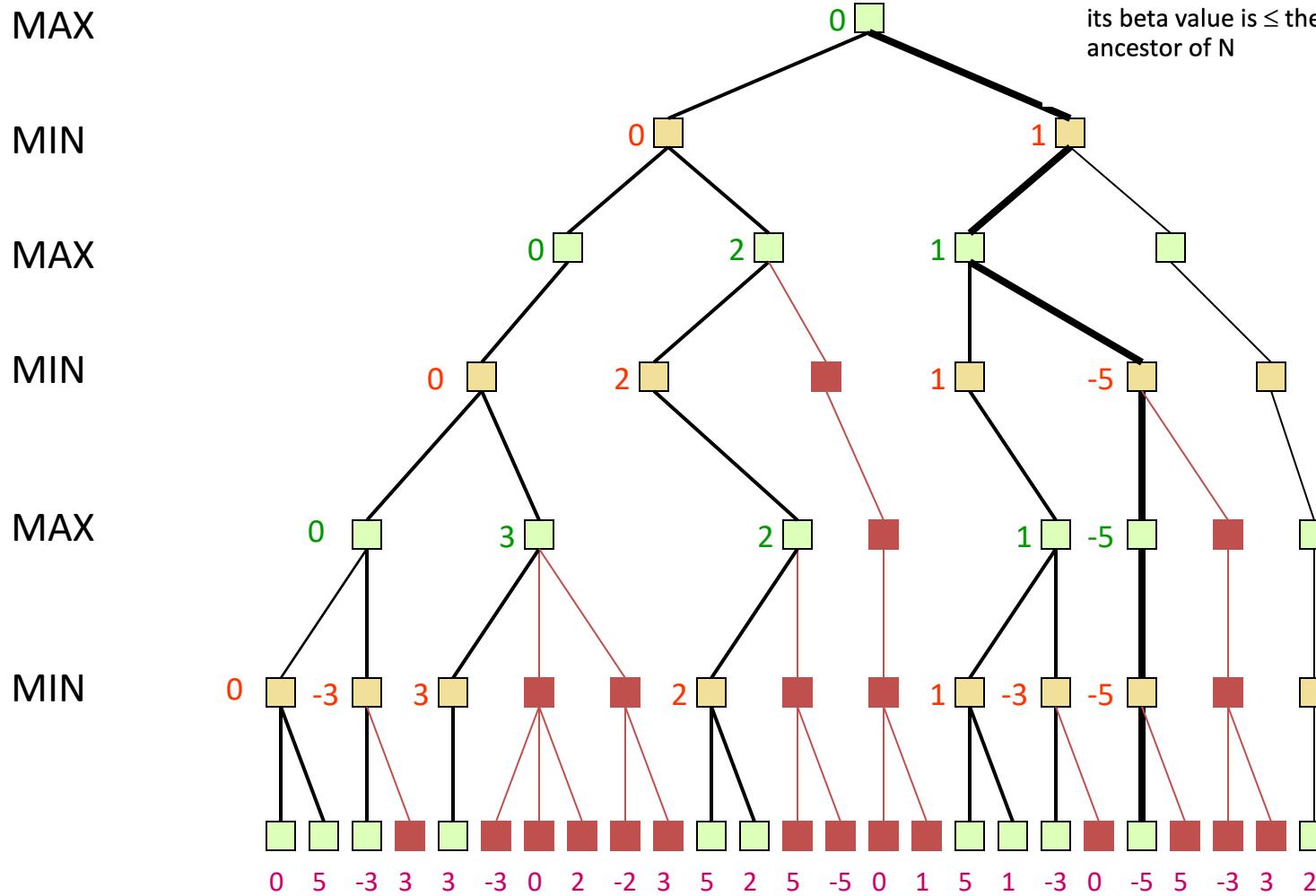
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



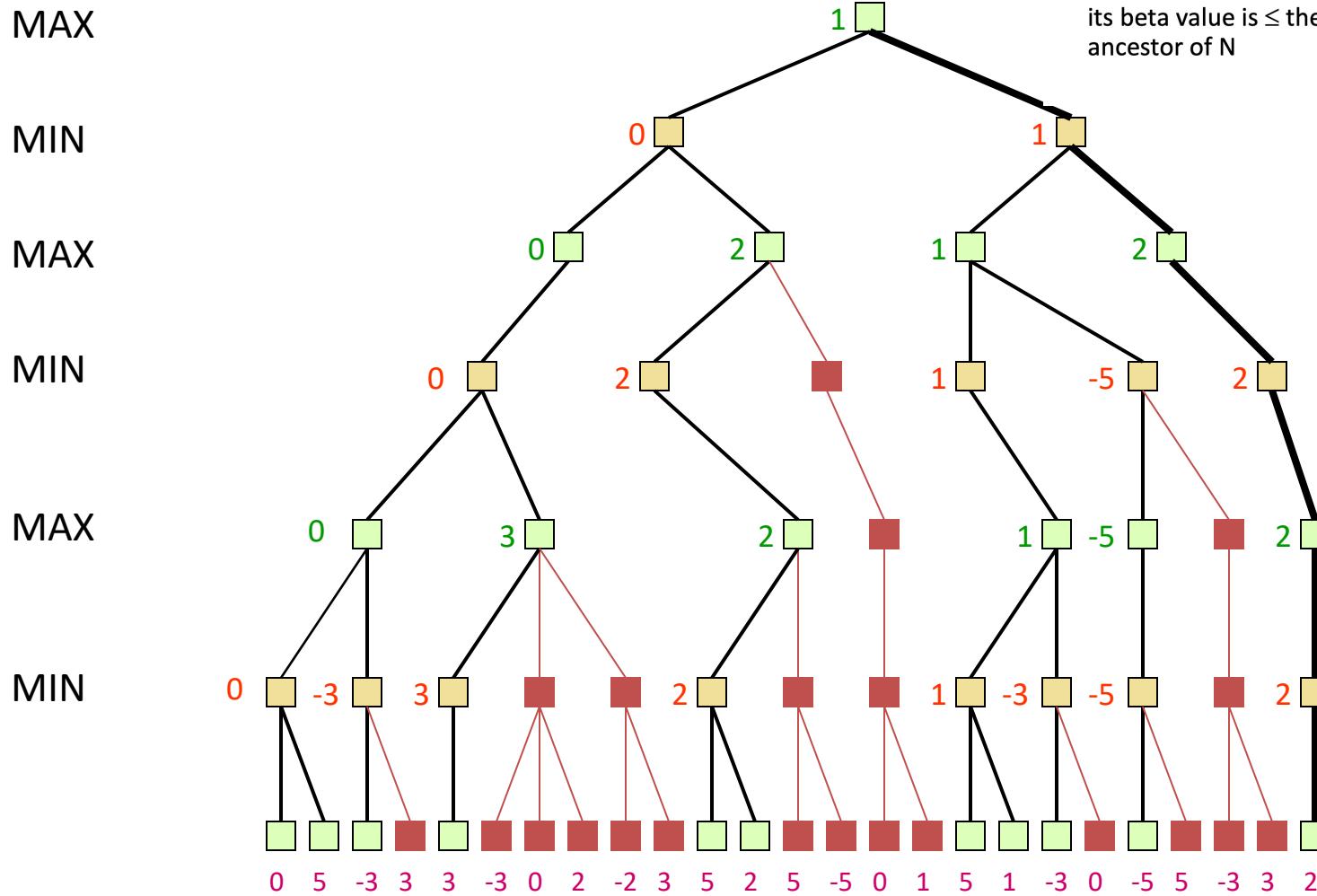
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



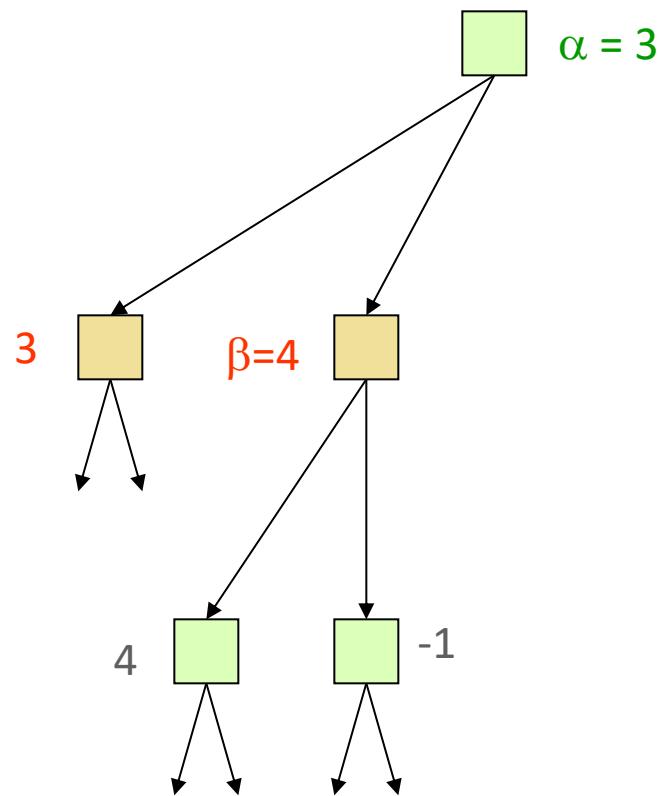
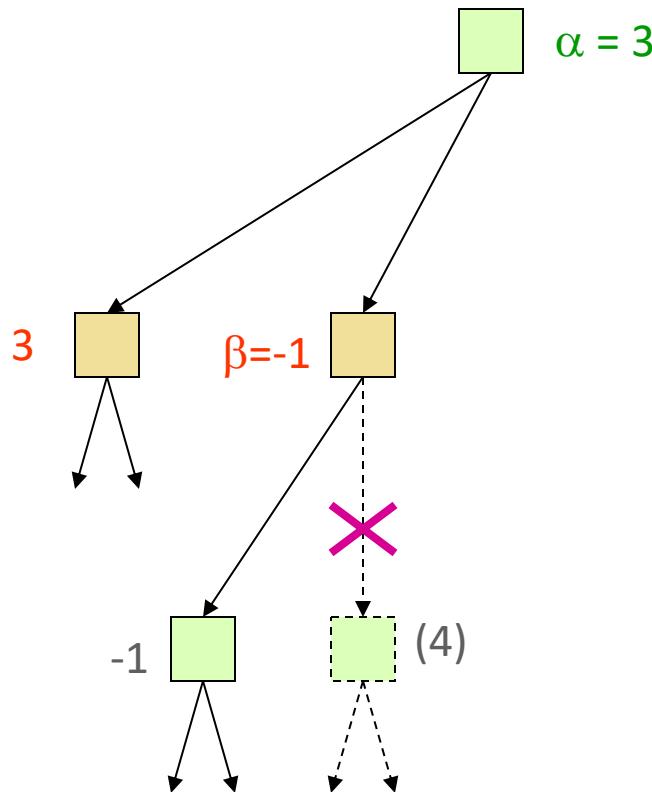
Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is \geq the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is \leq the alpha value of a MAX ancestor of N



How much do we gain?

- Consider these two cases:



How much do we gain?

- Assume a game tree of uniform branching factor b
- Minimax examines $O(b^h)$ nodes, as does alpha-beta in worst-case
- The gain for alpha-beta is **maximum** when:
 - children of a MAX node are ordered in decreasing backed up values
 - children of a MIN node are ordered in increasing backed up values
- Then alpha-beta examines $O(b^{h/2})$ nodes
[Knuth&Moore1975]
- But this requires an oracle
- If nodes are ordered at random, then the average number of nodes examined by alpha-beta is $\sim O(b^{3h/4})$

Alpha-Beta Implementation

- **Alpha-Beta-Decision(S)**
 - Return action leading to state $S' \in \text{SUCC}(S)$ that maximizes **MIN-Value**($S', -\infty, +\infty$)
- **MAX-Value(S, α, β)**
 - If Terminal?(S) return Result(S)
 - For all $S' \in \text{SUCC}(S)$
 - $\alpha \leftarrow \max(\alpha, \text{MIN-Value}(S', \alpha, \beta))$
 - ????????????????
 - Return α
- **MIN-Value(S, α, β)**
 - If Terminal?(S) return Result(S)
 - For all $S' \in \text{SUCC}(S)$
 - $\beta \leftarrow \min(\beta, \text{MAX-Value}(S', \alpha, \beta))$
 - If $\alpha \geq \beta$, then return β
 - Return β

Examples & Current Directions

Checkers: Tinsley vs. Chinook



Name: Marion Tinsley
Profession: Teach mathematics
Hobby: Checkers
Record: Over 42 years loses only 3 games of checkers
World champion for over 40 years

Checkers was solved in April 2007: from the standard starting position, both players can guarantee a draw with perfect play. This took 10^{14} calculations over 18 years. Checkers has a search space of size 5×10^{20}

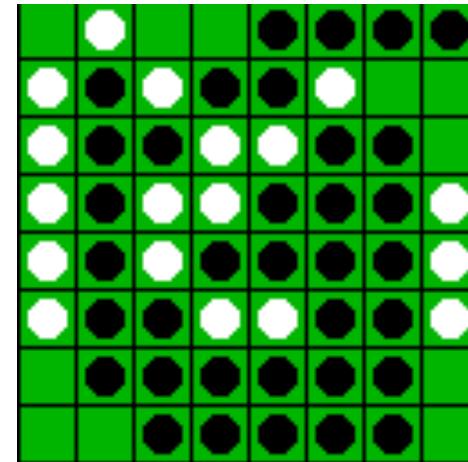
Mr. Tinsley suffered his 4th and 5th losses against Chinook (1990)

Chinook



- First computer to become official world champion of Checkers!

Othello (Reversi): *Murakami vs. Logistello*



Takeshi Murakami
World Othello Champion

1997: The Logistello software crushed Murakami
by 6 games to 0

Chess: Kasparov vs. Deep Blue

Kasparov		Deep Blue
5'10"	Height	6' 5"
176 lbs	Weight	2,400 lbs
34 years	Age	4 years
50 billion neurons	Computers	32 RISC processors + 256 VLSI chess engines
2 pos/sec	Speed	200,000,000 pos/sec
Extensive	Knowledge	Primitive
Electrical/chemical	Power Source	Electrical
Enormous	Ego	None

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Chess: Kasparov vs. Deep Junior



Deep Junior

8 CPU, 8 GB RAM, Win 2000

2,000,000 pos/sec

Available at \$100

August 2, 2003: Match ends in a 3/3 tie!

Milestone: Go (2016)



Google's AlphaGo beats Lee Sedol (9-dan player), March 2016

Secrets

- Alpha-beta pruning + iterative deepening + transposition tables (hash tables containing minimax values) + huge databases + Monte Carlo Tree Search + Deep Learning +
- E.g. Chinook searched all checkers configurations with 8 pieces or less and created an endgame database of 444 billion board configurations
- The methods are general, but their implementation is dramatically improved by many specifically tuned-up enhancements (e.g., the evaluation functions)

Perspective on Games: Con and Pro

Chess is the Drosophila of artificial intelligence. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophila. We would have some science, but mainly we would have very fast fruit flies.

John McCarthy



Saying Deep Blue doesn't really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings.

Drew McDermott

Next time

- **Uncertainty and Probability**

Introduction to uncertainty

Announcements

- Next class (Wednesday, September 21) will be online only (same zoom link).
 - We will do a coding exercise (about minmax)
- Assignment 1 (almost) released

Probabilistic techniques

- AI is full of uncertainty
 - Can't observe full state of system
 - Observations we can make are noisy
 - Our models of the world are imperfect



Martin-Shepard 2010

- Probabilistic frameworks give us a principled way of dealing with and reasoning about this uncertainty
 - Largely championed by Judea Pearl (2011 Turing Award)

But they're not a silver bullet!

- We'll still face challenges like...
 - Probability distributions that are impossibly complex, with intractably many dimensions
 - Parameter estimation problems that would require exponential amounts of data
- Much work is thus devoted to balancing between what we'd like to model and what we are able to model
 - *Probabilistic graphical models* are a popular framework

Probability 101

Probability definitions

- A *finite probability space* consists of:
 - A finite set S of mutually-exclusive *outcomes*
 - A function $P : S \rightarrow \mathbb{R}$ such that:

$$P(s) \geq 0, \forall s \in S \quad \sum_{s \in S} p(s) = 1 \quad P(\emptyset) = 0$$

- An *event* A is a subset of S , $A \subseteq S$.
 - The probability of an event is defined as

$$P(A) = \sum_{s \in A} P(s)$$

Basic identities

- For two events A and B ...
 - What's the probability that either A or B (or both) occur?

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

If A and B are *disjoint*, their intersection is the empty set, and the last term is 0.

Basic identities

- For two events A and B ...
 - What's the probability that either A or B (or both) occur?

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

If A and B are *disjoint*, their intersection is the empty set, and the last term is 0.

$$A = \text{even roll} = \{2, 4, 6\}$$

$$B = \text{roll} \geq 5 = \{5, 6\}$$

$$P(A \cup B) = ?$$

Basic identities

- For two events A and B ...
 - What's the probability that either A or B (or both) occur?

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

If A and B are *disjoint*, their intersection is the empty set, and the last term is 0.

$$A = \text{even roll} = \{2, 4, 6\}$$

$$B = \text{roll} \geq 5 = \{5, 6\}$$

$$P(A \cup B) = \frac{3}{6} + \frac{2}{6} - \frac{1}{6} = \frac{2}{3}$$

Super simple example #1

- Suppose you roll a six-sided die 5 times. What's the probability of rolling a “three” all 5 times?

Super simple example #1

- Suppose you roll a six-sided die 5 times. What's the probability of rolling a “three” all 5 times?

$$S = \{11111, 11112, 11113\dots\}$$

$$A = \{33333\}$$

$$P(A) = \frac{|A|}{|S|} = \frac{1}{6^5}$$

Super simple example #2

- Suppose you roll a six-sided die 5 times. What's the probability of rolling a “three” during the first roll?

Super simple example #2

- Suppose you roll a six-sided die 5 times. What's the probability of rolling a “three” during the first roll?

$$S = \{11111, 11112, 11113\dots\}$$

$$A = \{31111, 31112\dots\}$$

$$P(A) = \frac{|A|}{|S|} = \frac{6^4}{6^5} = \frac{1}{6}$$

Super simple example #3

- Suppose you roll a die 5 times. What's the probability of getting at least 1 six?

Example #3 (2nd try)

- Suppose you roll a die 5 times. What's the probability of getting at least 1 six?

- Answer 2: Sum probabilities of disjoint events

$$\begin{aligned} P(\text{at least 1 six}) &= P(1 \text{ six and 4 non-sixes}) + \\ &\quad P(2 \text{ sixes and 3 non-sixes}) + \\ &\quad P(3 \text{ sixes and 2 non-sixes}) + \\ &\quad P(4 \text{ sixes and 1 non-six}) + \\ &\quad P(5 \text{ sixes}) \\ &= \dots \end{aligned}$$

- Right, but a lot of work.

An example (3rd try)

- Suppose you roll a die 5 times. What's the probability of getting at least 1 six?

$$S = \{11111, 11112, 11113\dots\}$$

A = at least one six

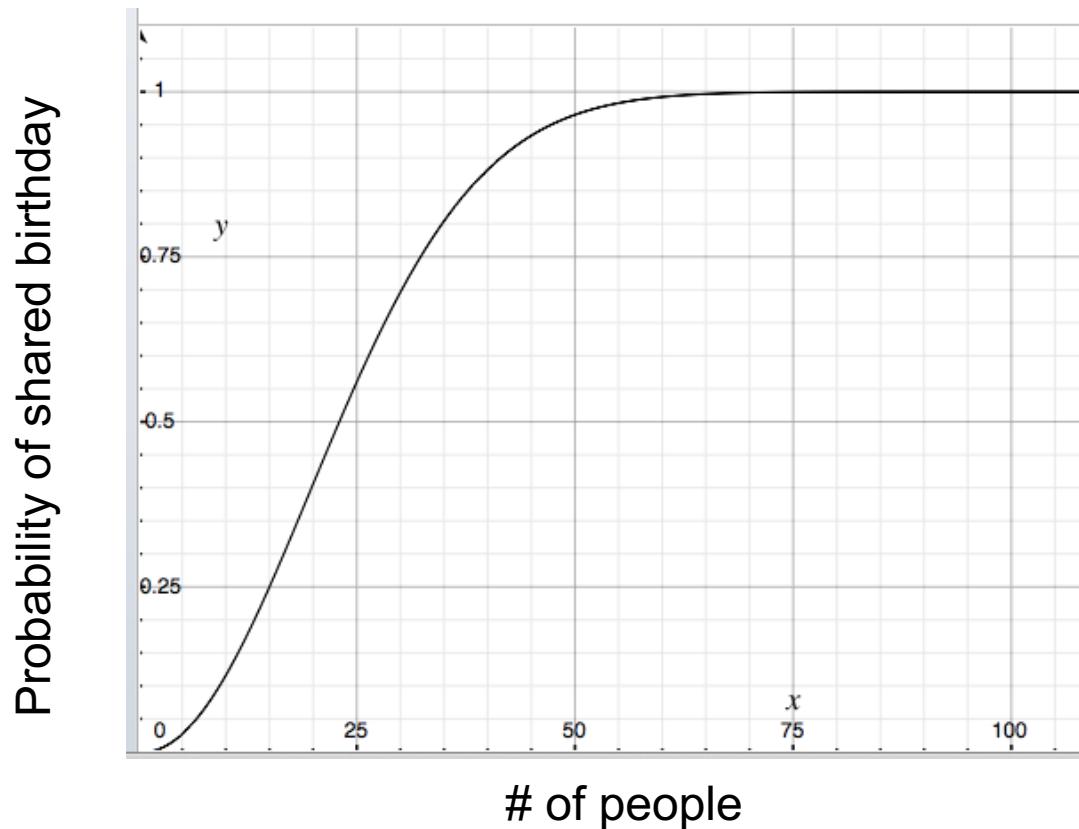
B = no six

$$A \cap B = \emptyset, A \cup B = S$$

$$P(A) = 1 - P(B) = 1 - \left(\frac{5}{6}\right)^5 \approx 0.6$$

The Birthday Problem

- Given a class of ~25 people, what's the probability that at least two of us share the same birthday?



Conditional probabilities and Bayes' Law

Conditional probabilities

- Probability that one event occurs, given that another event is known to have occurred
 - Denoted $P(A|B)$. “Probability of A given B”
 - Defined as:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Conditional probabilities

- Probability that one event occurs, given that another event is known to have occurred
 - Denoted $P(A|B)$. “Probability of A given B”
 - Defined as:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

A = role an even die

B = rolling 6

$$P(B|A) = \frac{P(\{6\})}{P(\{2, 4, 6\})} = \frac{1/6}{3/6} = \frac{1}{3}$$

Conditional probabilities

- Probability that one event occurs, given that another event is known to have occurred
 - Denoted $P(B|A)$. “Probability of B given A”
 - Defined as:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

- Leads directly to the *chain rule*:

$$P(A \cap B) = P(B|A)P(A)$$

- More generally:

$$P(A_1 \cap A_2 \cap \dots \cap A_n) = P(A_1)P(A_2|A_1)\dots P(A_n|A_1 \cap \dots \cap A_{n-1})$$

Independence of events

- Two events are *independent* if $P(A|B) = P(A)$
 - Or, equivalently, if $P(B|A) = P(B)$
 - Independence denoted $A \perp B$
- The joint probability of independent events A and B both occurring is then simply:
$$P(A \cap B) = P(A)P(B)$$
 - This idea of *factoring* a distribution into a product of two simpler distributions will be a recurring theme!

Conditional independence

- Sometimes events are both conditioned on the same event, but otherwise are independent
 - Mary and Bob live in same city but independently
 - A denotes event that it's raining, B denotes event that Mary has an umbrella, C denotes event that Bob has an umbrella
 - Events B and C are **not** independent
 - But B and C are **conditionally independent** given A,

$$P(B|A, C) = P(B|A)$$

$$P(C|A, B) = P(C|A)$$

- Denoted

$$B \perp C | A$$

Bayes' Law

- For two events A and B ,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Posterior

Likelihood

Priors

- Useful when you want to know something about A, but all you can directly observe is B
 - This process is called *Bayesian inference*

Bayes' Law Example #1

- I have two coins, one fair and one with heads on both sides. I choose a coin at random, flip it, and get heads. What is the probability that it is the fair coin?

Bayes' Law Example #1

- I have two coins, one fair and one with heads on both sides. I choose a coin at random, flip it, and get heads. What is the probability that it is the fair coin?

Bayes' Law Example #1

- I have two coins, one fair and one with heads on both sides. I choose a coin at random, flip it, and get heads. What is the probability that it is the fair coin?

F = Fair

H = Heads

$$P(F|H) = \frac{P(H|F)P(F)}{P(H)} = \frac{\frac{1}{2} \frac{1}{2}}{\frac{3}{4}} = \frac{1}{3}$$

$$P(H|F) = \frac{1}{2}$$

$$P(F) = \frac{1}{2}$$

$$\begin{aligned} P(H) &= P(H \cap F) + P(H \cap \bar{F}) \\ &= P(H|F)P(F) + P(H|\bar{F})P(\bar{F}) \\ &= \frac{1}{2} \frac{1}{2} + 1 \frac{1}{2} = \frac{3}{4} \end{aligned}$$

Bayes' Law Example #2

- A doctor says you have an illness that afflicts 0.01% of the population. Her diagnoses are right 99% of the time. What's the probability that you have the illness?

Bayes' Law Example #2

- A doctor says you have an illness that afflicts 0.01% of the population. Her diagnoses are right 99% of the time. What's the probability that you have the illness?

i = Ill

p = Positive

$$P(i|p) = \frac{P(p|i)P(i)}{P(p)} = \frac{(0.99)(0.0001)}{0.010098} \approx 0.0098$$

$$P(p|i) = 0.99$$

$$P(i) = 0.0001$$

$$\begin{aligned} P(p) &= P(p \cap i) + P(p \cap \bar{i}) \\ &= P(p|i)P(i) + P(p|\bar{i})P(\bar{i}) \end{aligned}$$

$$= (0.99)(0.0001) + (0.01)(0.9999) \approx 0.010098$$

Next class

- Inference on Bayes Nets

Bayes' Law and Bayesian inference

Announcements

- Don't forget to reach out to your team members for A1.
- Walkthrough A0 posted on Canvas

Conditional probabilities

- Probability that one event occurs, given that another event is known to have occurred
 - Denoted $P(B|A)$. “Probability of B given A”
 - Defined as:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

- Leads directly to the *chain rule*:

$$P(A \cap B) = P(B|A)P(A)$$

- More generally:

$$P(A_1 \cap A_2 \cap \dots \cap A_n) = P(A_1)P(A_2|A_1)\dots P(A_n|A_1 \cap \dots \cap A_{n-1})$$

Independence of events

- Two events are *independent* if $P(A|B) = P(A)$
 - Or, equivalently, if $P(B|A) = P(B)$
 - Independence denoted $A \perp B$
- The joint probability of independent events A and B both occurring is then simply:
$$P(A \cap B) = P(A)P(B)$$
 - This idea of *factoring* a distribution into a product of two simpler distributions will be a recurring theme!

Conditional independence

- Sometimes events are both conditioned on the same event, but otherwise are independent
 - Mary and Bob live in same city but independently
 - A denotes event that it's raining, B denotes event that Mary has an umbrella, C denotes event that Bob has an umbrella
 - Events B and C are **not** independent
 - But B and C are **conditionally independent** given A,

$$P(B|A, C) = P(B|A)$$

$$P(C|A, B) = P(C|A)$$

- Denoted

$$B \perp C | A$$

Bayes' Law

- For two events A and B ,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Posterior

Likelihood

Priors

- Useful when you want to know something about A, but all you can directly observe is B
 - This process is called *Bayesian inference*

Bayes' Law Example #1

- I have two coins, one fair and one with heads on both sides. I choose a coin at random, flip it, and get heads. What is the probability that it is the fair coin?

Bayes' Law Example #1

- I have two coins, one fair and one with heads on both sides. I choose a coin at random, flip it, and get heads. What is the probability that it is the fair coin?

Bayes' Law Example #1

- I have two coins, one fair and one with heads on both sides. I choose a coin at random, flip it, and get heads. What is the probability that it is the fair coin?

F = Fair

H = Heads

$$P(F|H) = \frac{P(H|F)P(F)}{P(H)} = \frac{\frac{1}{2} \frac{1}{2}}{\frac{3}{4}} = \frac{1}{3}$$

$$P(H|F) = \frac{1}{2}$$

$$P(F) = \frac{1}{2}$$

$$\begin{aligned} P(H) &= P(H \cap F) + P(H \cap \bar{F}) \\ &= P(H|F)P(F) + P(H|\bar{F})P(\bar{F}) \\ &= \frac{1}{2} \frac{1}{2} + 1 \frac{1}{2} = \frac{3}{4} \end{aligned}$$

Bayes' Law Example #2

- A doctor says you have an illness that afflicts 0.01% of the population. Her diagnoses are right 99% of the time. What's the probability that you have the illness?

Bayes' Law Example #2

- A doctor says you have an illness that afflicts 0.01% of the population. Her diagnoses are right 99% of the time. What's the probability that you have the illness?

i = Ill

p = Positive

$$P(i|p) = \frac{P(p|i)P(i)}{P(p)} = \frac{(0.99)(0.0001)}{0.010098} \approx 0.0098$$

$$P(p|i) = 0.99$$

$$P(i) = 0.0001$$

$$\begin{aligned} P(p) &= P(p \cap i) + P(p \cap \bar{i}) \\ &= P(p|i)P(i) + P(p|\bar{i})P(\bar{i}) \end{aligned}$$

$$= (0.99)(0.0001) + (0.01)(0.9999) \approx 0.010098$$

Bayes' Law Example #3a

- Suppose that you have a friend, Mary, who lives in Seattle. She tells the truth 80% of the time and lies the other 20% of the time. The weather in Seattle on any given day is either sunny or cloudy; 30% of days are sunny and 70% are cloudy.
- Mary calls you one day and says that the weather in Seattle is cloudy. What is the probability that the weather is actually cloudy?

Solution #3a

C: cloudy

S: sunny

Mc: Marry says cloudy

Ms: Marry says sunny

We are interested in finding the probability that it is actually cloudy given that Mary says it is cloudy, $P(C|Mc)$. By Bayes' Law:

$$P(C|Mc) = \frac{P(Mc|C)P(C)}{P(Mc)}$$

We know that $P(C) = 0.7$ from the problem statement.

The probability that Mary says it is cloudy given that it actually is cloudy is just the probability that she tells the truth, $P(Mc|C) = 0.8$.

The probability that Mary says cloudy is given by,

$$P(Mc) = P(Mc|C)P(C) + P(Mc|S)P(S) = (0.8)(0.7) + (0.2)(0.3) = 0.62$$

$$P(C|Mc) = \frac{(0.8)(0.7)}{0.62} = 0.903$$

Making decisions with Bayes' Law

Applications to AI

- Perhaps the simplest application of probabilistic methods to AI is the Bayesian Classifier
 - Want to make a decision based on some evidence
 - “Train” the classifier by computing prior probability distributions and likelihood functions from training data
 - Then apply Bayes law to compute posterior probabilities

Bayes' Law: An example

- You're a juror in a murder case
 - You need to decide between guilt (G) and innocence (\bar{G})
 - You have heard some evidence (E)
 - Bayesian approach: Compute $P(G|E)$, and vote to convict if

$$P(G|E) > \tau$$

where τ is a threshold

- Or, compute an odds ratio and convict if:

$$\frac{P(G|E)}{P(\bar{G}|E)} > \tau_2$$

Bayes' Law: An example

- Say you have to decide before hearing any evidence
 - what is the *prior* probability, $P(G)$?
- How to estimate $P(G)$?
 - Based on population constraints
 - 1 person in Bloomington ($\sim 20,000$ people) did it
 - $P(G) \approx 1/20000 = 0.00005$
 - Based on historical data
 - U.S. murder conviction rate: $0.06/1000$ [BJS96]
 - $P(G) \approx 0.00006$

Bayesian inference example

- Eyewitness testimony (T) identifies the suspect

- Now we want to compute $P(G|T) = \frac{P(T|G)P(G)}{P(T)}$

- $P(G) =$

$$\begin{aligned} P(T) &= P(T|G)P(G) + P(T|\bar{G})P(\bar{G}) \\ &= (0.55)(0.00005) + (0.32)(0.99995) \\ &\approx 0.32001 \end{aligned}$$

- $P(T|G) =$

$$P(G|T) \approx \frac{(0.55)(0.00005)}{0.32001} \approx 0.0000859$$

- $P(T|\bar{G}) =$

- Alternatively, we can compute an *odds ratio*:

$$\frac{P(G|T)}{P(\bar{G}|T)} = \frac{P(T|G)P(G)P(T)}{P(T)P(T|\bar{G})P(\bar{G})} = \frac{P(T|G)P(G)}{P(T|\bar{G})P(\bar{G})} = \frac{(0.55)(0.00005)}{(0.32)(0.99995)} \approx 1:11635$$

Bayesian inference example (2)

- Now you hear that the murderer had a red car, and that the suspect owns a red car (R)
 - We want to compute $P(G|T,R) = \frac{P(T,R|G)P(G)}{P(T,R)}$. How?
 - Assuming that T and R are independent conditioned on G ,

$$P(G|T,R) = \frac{P(R|G)P(T|G)P(G)}{P(R|G)P(T|G)P(G) + P(R|\bar{G})P(T|\bar{G})P(\bar{G})}$$

- Computing odds:

$$\frac{P(G|T,R)}{P(\bar{G}|T,R)} = \left(\frac{P(G)}{P(\bar{G})} \right) \left(\frac{P(T|G)}{P(T|\bar{G})} \right) \left(\frac{P(R|G)}{P(R|\bar{G})} \right)$$

The posterior odds

Prior odds

New evidence

Bayesian inference example (3)

- Given the testimony (T) and red car evidence (R),

$$\frac{P(G|T,R)}{P(\bar{G}|T,R)} = \left(\frac{P(G)}{P(\bar{G})} \right) \left(\frac{P(T|G)}{P(T|\bar{G})} \right) \left(\frac{P(R|G)}{P(R|\bar{G})} \right)$$

– $P(R|G) = 1$, $P(R|\bar{G}) \approx 0.13$ [DuPont07]

$$\frac{P(G|T,R)}{P(\bar{G}|T,R)} = \left(\frac{1}{11635} \right) \left(\frac{1}{0.13} \right) \approx 1:1513$$

Bayesian inference example (4)

- Now suppose a partial fingerprint (F) matches the suspect
 - $P(F|G) = 1, P(F|\bar{G})=0.001$

$$\frac{P(G|T,R,F)}{P(\bar{G}|T,R,F)} = \left(\frac{P(G|T,R)}{P(\bar{G}|T,R)} \right) \frac{P(F|G)}{P(F|\bar{G})} \approx \left(\frac{1}{1513} \right) \frac{1}{0.001} = \frac{1}{1.513}$$

Probabilistic inference

- Notice that we avoided making hard classification decisions until all evidence had been considered
 - Very useful when evidence is weak, noisy, imprecise

An actual application: **Spam Filtering**

Spam

- Spam = junk e-mail
- A big problem!
 - ~50% of all email traffic on the Internet
 - ~320 billion junk emails per day
 - >2 petabytes (= 2,000 terabytes = 2,000,000 gigabytes) daily
 - Spreads malware, worms, phishing schemes, etc.
- Possible solutions
 - Block e-mails from blacklisted users and servers
 - Accept e-mails only from whitelisted addresses
 - Cost-based solutions (e.g. micropayments)
 - Filtering rules (ignore mail with “debt”, “viagra”, “stock”)
 - Content-based statistical filtering

Reduce Debt by up to 60 Percent

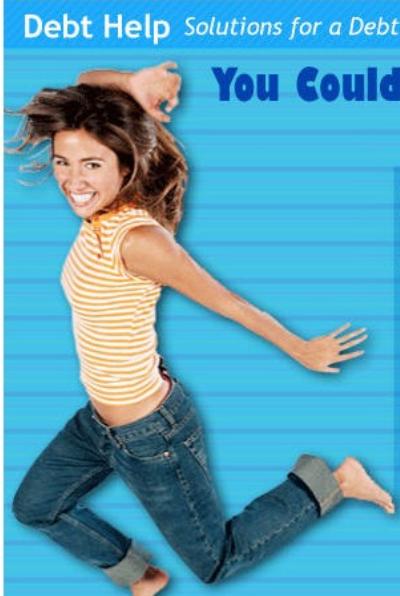
Spam | X

from Financial Assistance <debtrelief@airving.com>
to david.crandall@gmail.com
date Wed, Sep 10, 2008 at 8:30 PM
subject Reduce Debt by up to 60 Percent

[hide details](#) 8:30 PM (3)

Debt Help Solutions for a Debt Free Lifestyle

You Could be DEBT FREE in Minutes!



We can help you reduce these debts:

- All Major Credit Cards
- Medical Bills
- Phone Bills
- Retail Store Charge Cards
- Unsecured Loans or Liens
- High Credit Card Rates & Fees

GET HELP NOW

This is a service of Debt Help. If you no longer wish to receive these announcements,
[Click Here.](#)

3409 Executive Center Dr., Suite 110, Austin, TX 78731

You could be debt free in minutes!

We can help you reduce these debts:

- All major credit cards
- Medical bills
- Phone Bills
- Retail Store Charge Cards
- Unsecured loans or liens
- High credit card rates and fees

Get help from one of our professional consultants today, and get your life back on track:

<http://kperduc.airving.com/debt/>

This is a service of Debt Help. If you no longer wish to receive these announcements, go here
<http://kperduc.airving.com/debt/> or write to 3409 Executive Center Dr, Suite 110 Austin TX 78731

Quality watches at 25% discount

Spam | X

from Exceptional Watches <info@fliesen-becht.de>
to Great watch Service <david.crandall@gmail.com>
date Wed, Sep 10, 2008 at 5:58 AM
subject Quality watches at 25% discount

[hide details](#) 5:58 AM (17 hours ago) [Reply](#) | [▼](#)

Why would you want to purchase a replica watch from King-relicas?

There may be many reasons:

1. You want a genuine Rolex / Breitling watch, but the price is too ridiculous
2. You want to impress your friends or business clients
3. You want to keep your original safe, while using the replica for daily wear and tear on it.

[Browse our King-relica watches shop!](#)

Modeling a document

- Represent a document as an unordered collection of words
(a *bag of words* model)

There may be many reasons:

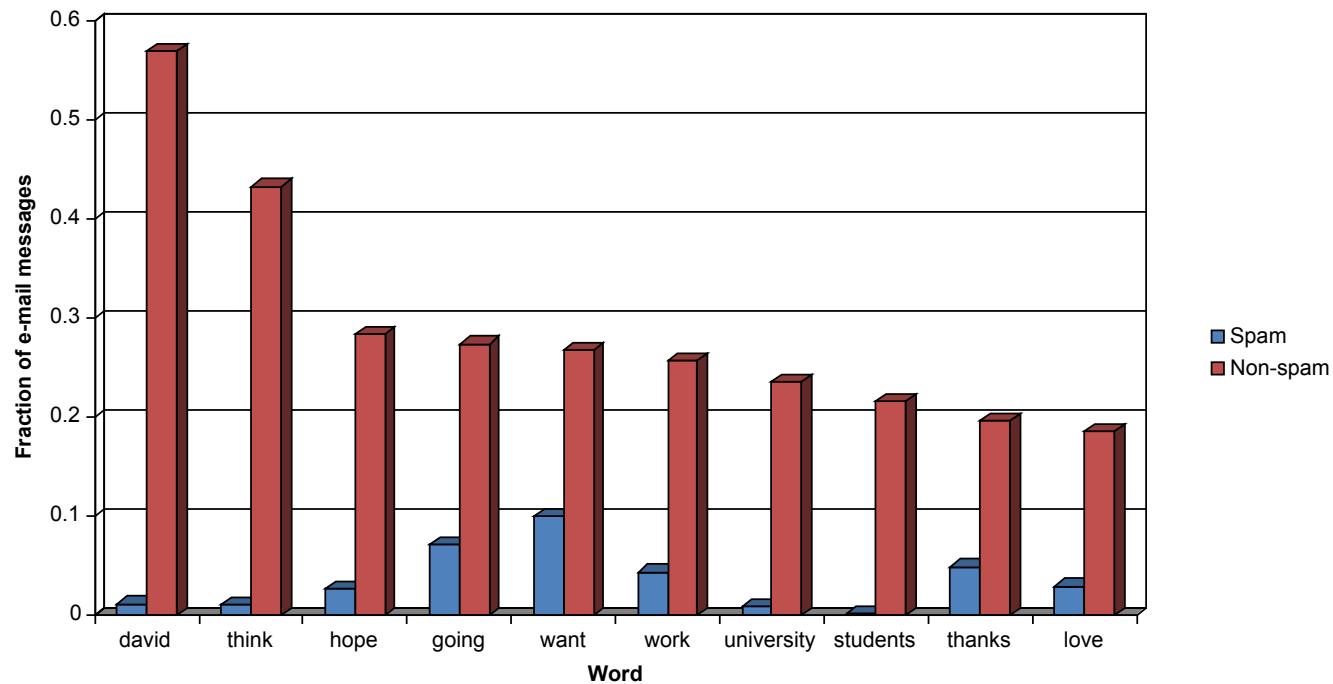
1. You want a genuine Rolex / Breitling
2. You want to impress your friends or
3. You want to keep your original safe, and tear on it.



A word cloud diagram enclosed in a curved line. The words and their approximate frequencies are: want (1), reasons (1), Rolex (1), original (1), be (1), tear (1), safe (1), many (1), friends (1), You (1), may (1), to (1), genuine (1), impress (1).

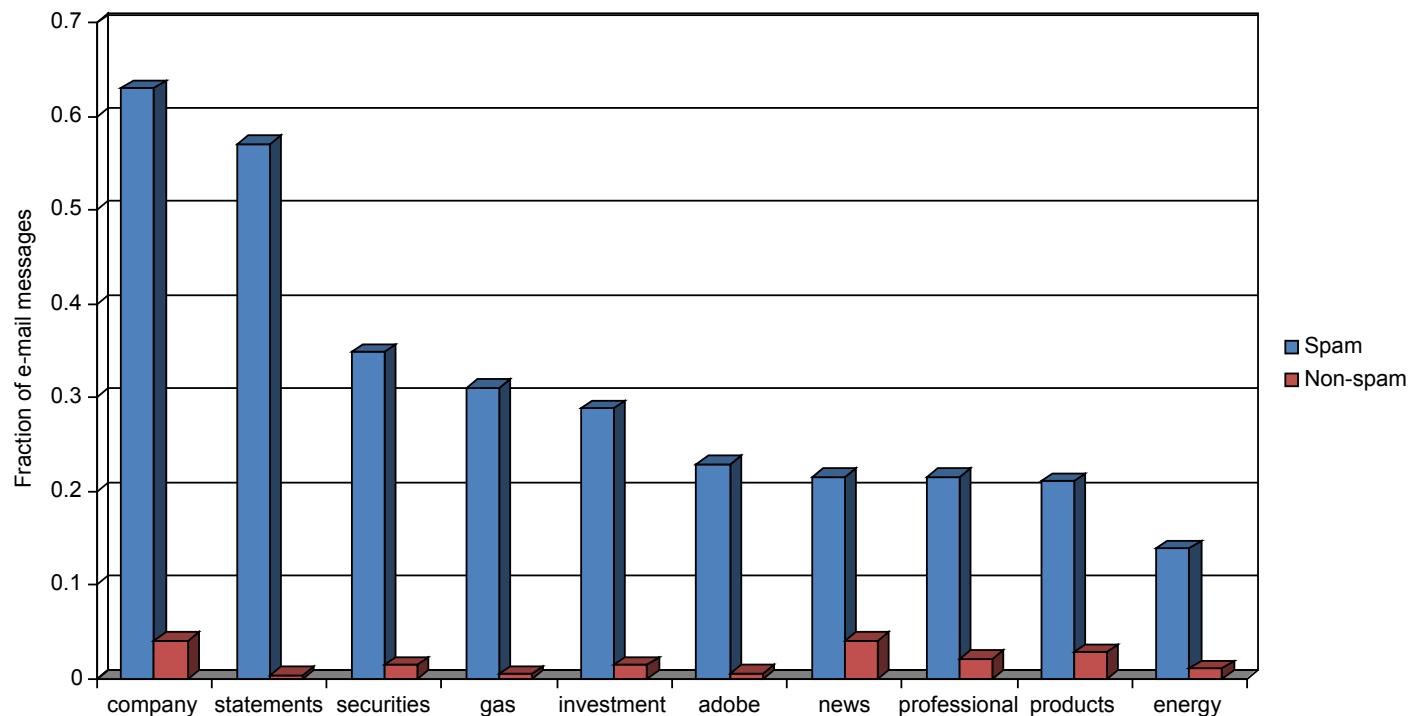
Statistical motivation

- Spam and (my) non-spam are statistically very different



Statistical motivation

- Spam and (my) non-spam are statistically very different



Bayesian spam filtering

- Suppose we get an email containing the word “debt”
 - What is the probability it is spam (S), $P(S | \text{debt})$?

$$P(S | \text{debt}) = \frac{P(\text{debt} | S)P(S)}{P(\text{debt})}$$

- $P(\text{debt} | S) = 0.309$, $P(\text{debt} | \bar{S}) = 0.00447$
- $P(S) = 0.5$
- $P(\text{debt}) = 0.157$

$$P(S | \text{debt}) = \frac{P(\text{debt} | S)P(S)}{P(\text{debt})} \approx 0.986$$

$$P(\bar{S} | \text{debt}) = 1 - P(S | \text{debt}) \approx 0.014$$

$0.986/0.014 \approx 70:1$ odds that message is spam

More examples

- Assuming a uniform prior, $P(S)=0.5$

Word	$P(\text{word} \text{spam})$	$P(\text{word} \text{not spam})$	$P(\text{word})$	$P(\text{spam} \text{word})$
debt	0.309	0.00447	0.157	0.986
news	0.215	0.0395	0.127	0.845
investment	0.288	0.0137	0.151	0.955
david	0.012	0.575	0.294	0.020
want	0.101	0.268	0.185	0.274
thanks	0.0491	0.196	0.123	0.200



Computed from Bayes' Law

Bayesian spam filtering

- A new email has the words “debt” and “price”
 - What is the probability it is spam (S), $P(S \mid \text{debt, price})$?

$$P(S \mid \text{debt, price}) \propto P(\text{debt, price} \mid S)P(S)$$

- If we assume that the occurrence of the words “debt” and “price” are independent events *conditioned on S*,

$$P(S \mid \text{debt, price}) \propto P(\text{debt} \mid S)P(\text{price} \mid S)P(S)$$

- This is called the *naïve Bayes* assumption.

Bayesian spam filtering

- Generalize to an arbitrary number of words,

$$P(S | W_1, W_2, W_3, \dots, W_n) \propto P(W_1 | S)P(W_2 | S)\dots P(W_n | S)P(S)$$

which is equivalent to,

$$P(S | \bigcap_{i=1}^n W_i) \propto P(S) \prod_{i=1}^n P(W_i | S)$$

- For example,

$$P(S | \text{debt, free, credit}) \propto P(S)P(\text{debt} | S)P(\text{free} | S)P(\text{credit} | S)$$

A practical spam filter [Graham02]

- Break a message into *tokens* of words, numbers, etc.
- Look for the 15 “most interesting words”
 - i.e. words for which $P(S|W)$ is farthest from 0.5
 - Then compute $P(S|W_1, W_2, \dots, W_{15})$

Dear Sir or Madam:

Please reply to

Receiver: China Enterprise Management Co., Ltd. (CMC)

E-mail: unido@chinatop.net

As one technical organization supported by China Investment and Technical Promotion Office of United Nation Industry Development Organization (UNIDO), we cooperate closely with the relevant Chinese Quality Supervision and Standardization Information Organization. We provide the most valuable consulting services to help you to open Chinese market within the shortest time:

1. Consulting Service on Mandatory National Standards of The People's Republic of China.

2. Consulting Service on Inspection and Quarantine Standards of The People's Republic of China.

3. Consulting Service for Permission to Enter Chinese Market

We are very sorry to disturb you!

More information, please check our World Wide Web:
<http://www.chinatop.net>

Sincerely yours

madam	0.99
promotion	0.99
republic	0.99
shortest	0.047225013
mandatory	0.047225013
standardization	0.07347802
sorry	0.08221981
supported	0.09019077
people's	0.09019077
enter	0.9075001
quality	0.8921298
organization	0.12454646
investment	0.8568143
very	0.14758544
valuable	0.82347786

$$P(S|W_1, W_2, \dots, W_{15})=0.9$$

A true negative

Hi,

Do you have any examples online of that continuation style web programming that you describe?

For example, you mention that you needed the user to go to a color picker screen and then return to the same spot. I'm interested on what was required to achieve that. Did you have to use real continuations to achieve that?

Dru Nelson
San Carlos, California

continuation	0.01
describe	0.01
continuations	0.01
example	0.033600237
programming	0.05214485
i'm	0.055427782
examples	0.07972858
color	0.9189189
localhost	0.09883721
hi	0.116539136
california	0.84421706
same	0.15981844
spot	0.1654587
us-ascii	0.16804294
what	0.19212411

A false negative

Dear Paul Graham,

As a person involved in website development you recognize that finding a good web host offering reasonably priced quality services can be quite difficult. We believe that offering a combination of quality services, timely and responsive customer support and good prices is where our company excels.

HOSTEX (<http://www.hostex.com>) is a flexible company providing quality and cost effective web hosting solutions with focus on small and medium businesses.

Recently we introduced new service plans offering flexibility, performance and value:

Our plans are:

Basic - \$6.95 / month
- 40 Mb of diskspace
- 10 POP3 mailboxes, unlimited aliases, autoresponders
- unlimited ftp accounts, URL protection, shell access and more
- no setup fee

Advanced - \$14.95 / month
- 80 Mb of diskspace
- 15 POP3 mailboxes, unlimited aliases, autoresponders
- unlimited ftp accounts, URL protection, shell access
- CGI/SSI scripting (Perl, Python, Tcl)
- PHP4 scripting
- Frontpage 2002 extensions
- Graphical web statistics
- various optional services and more
- no setup fee

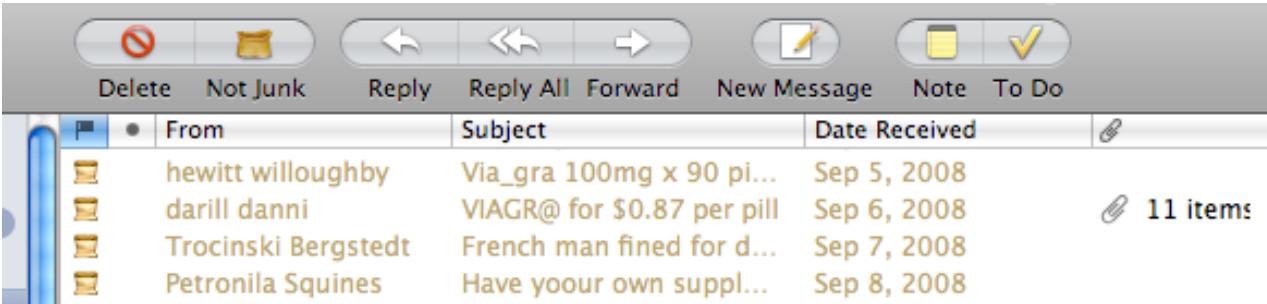
Professional - customizable, make-your-own plan
- select only those services that you need

All plans include full 30-day money back guarantee and there is no setup fee.

perl	0.01
python	0.01
tcl	0.01
scripting	0.01
morris	0.01
graham	0.01491078
guarantee	0.9762507
cgi	0.9734398
paul	0.027040077
quite	0.030676773
pop3	0.042199217
various	0.06080265
prices	0.9359873
managed	0.06451222
difficult	0.071706355

Learning

- The advantage of a Bayesian classifier is that it can learn optimal values for its parameters
 - Given a set of training data
 - No need for hand-crafted rules. More accurate, less work.
 - But a good set of training data is critical
- The classifier can continue to learn with time
 - User corrects the classifier's errors, classifier adjusts probabilities accordingly



The screenshot shows a window of a mail application. At the top is a toolbar with icons for Delete, Not Junk, Reply, Reply All, Forward, New Message, Note, and To Do. Below the toolbar is a header row with columns for From, Subject, Date Received, and a small icon. Underneath is a list of four messages. A scroll bar is visible on the left side of the message list. In the bottom right corner of the message list area, there is a small icon followed by the text "11 items".

From	Subject	Date Received	
hewitt willoughby	Via_gra 100mg x 90 pi...	Sep 5, 2008	
darill danni	VIAGR@ for \$0.87 per pill	Sep 6, 2008	
Trocinski Bergstedt	French man fined for d...	Sep 7, 2008	
Petronila Squines	Have yoour own suppl...	Sep 8, 2008	

Naïve Bayes

- Our framework assumes that evidence is independent, given a class label, e.g.

$$P(\text{debt}, \text{loan}, \text{stock} | S) = P(S)P(\text{debt}|S)P(\text{loan}|S)P(\text{stock}|S)$$

- This is not generally true
- It is very popular anyway and often works surprisingly well

Next class

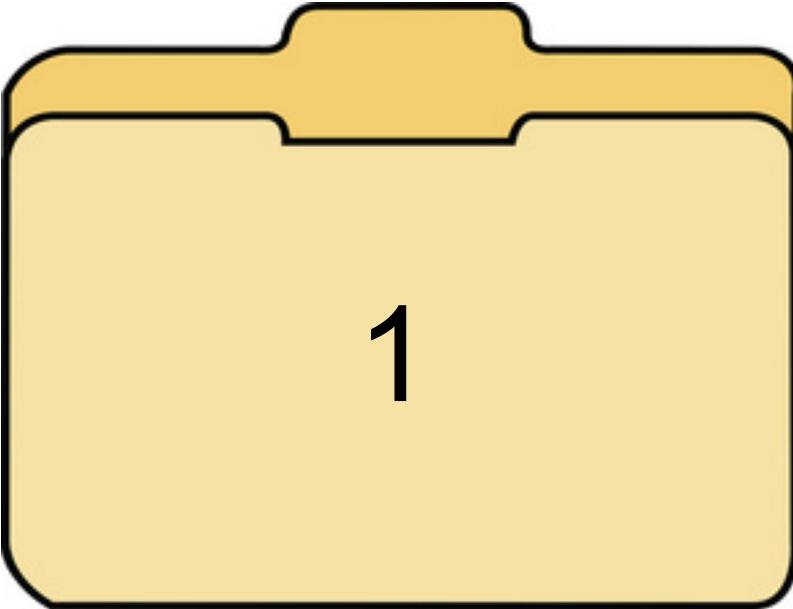
- **More on Inference and Bayes Nets**



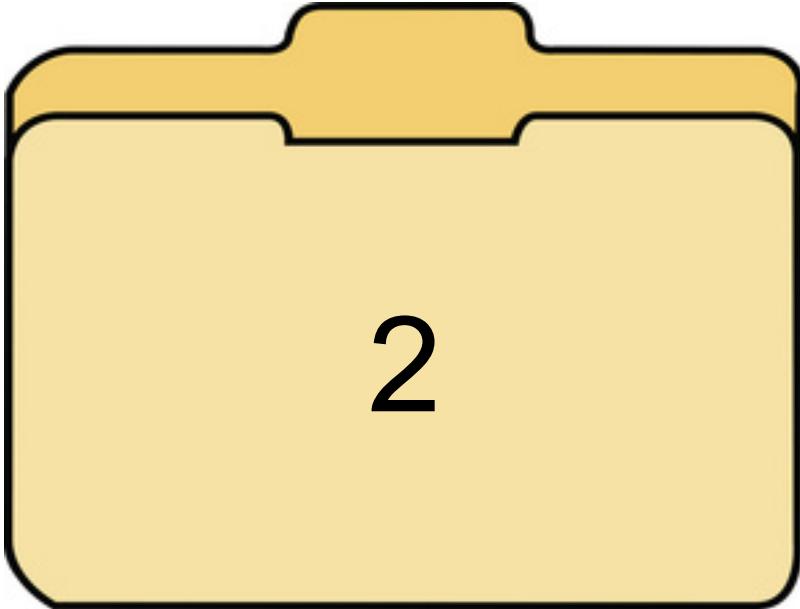
Bayes' Law and Bayes Nets

Announcements

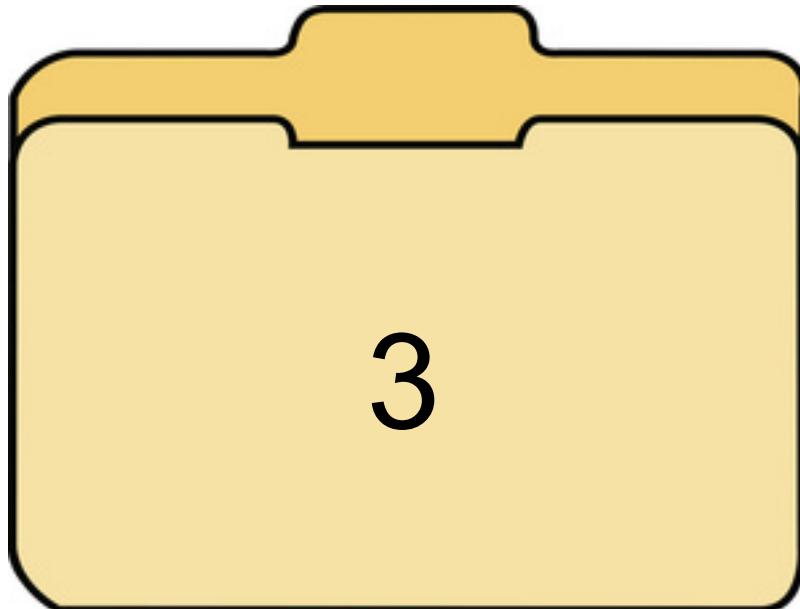
- Don't forget to reach out to your team members for A1.



1



2



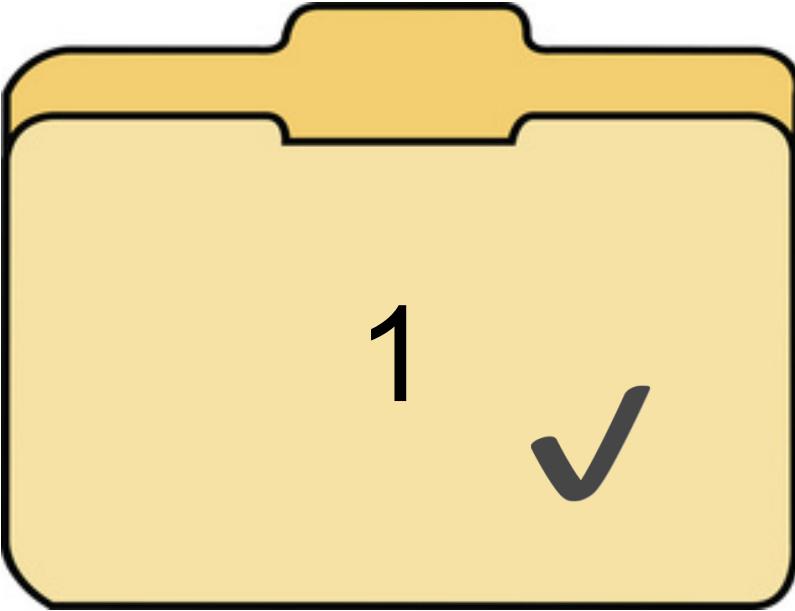
3

1

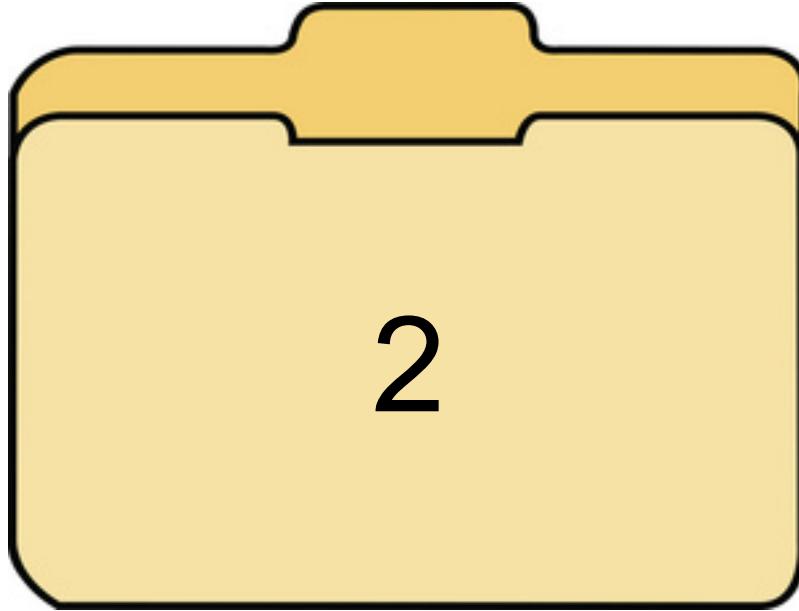


2

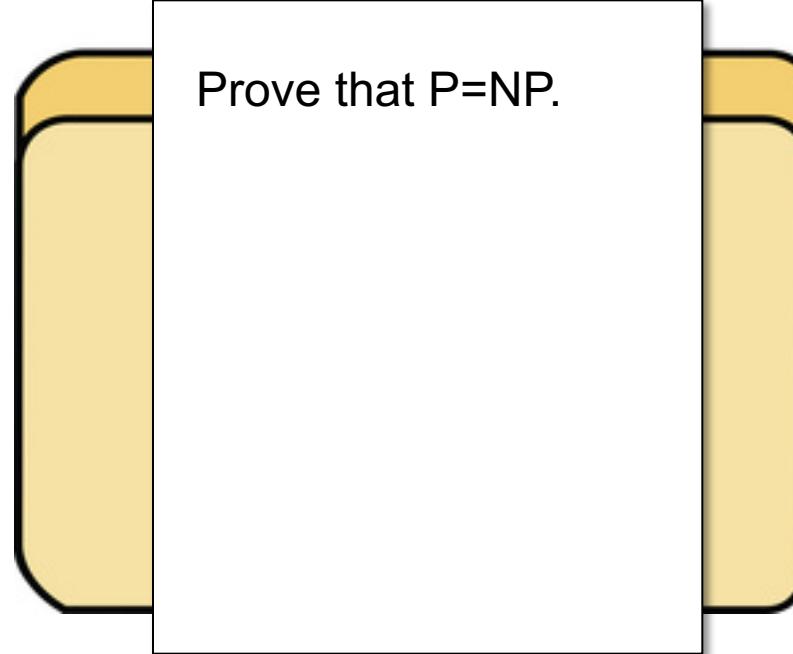
3



1



2



Prove that $P=NP$.

Assumptions

- Easy exam randomly placed in one of the 3 folders
- The teacher always reveals a hard exam
 - If the student chooses a hard exam, the teacher reveals the other hard exam
 - If the student chooses an easy exam, the teacher reveals one of the hard exams, chosen at random

Using Bayes' law...

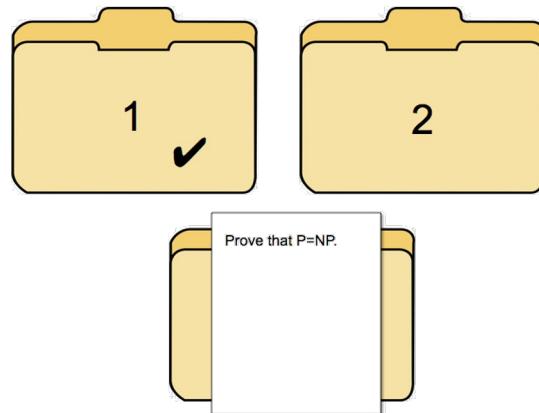
Given that #1 was chosen by the student,

$$P(2 \text{ easy} \mid 3 \text{ shown}) = P(3 \text{ shown} \mid 2 \text{ easy}) P(2 \text{ easy}) / P(3 \text{ shown})$$

$$P(2 \text{ easy}) = ?$$

$$P(3 \text{ shown} \mid 2 \text{ easy}) = ?$$

$$P(3 \text{ shown}) = ?$$



Using Bayes' law...

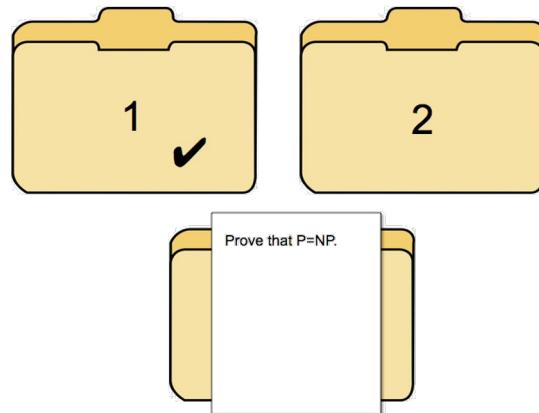
Given that #1 was chosen by the student,

$$P(2 \text{ easy} \mid 3 \text{ shown}) = P(3 \text{ shown} \mid 2 \text{ easy}) P(2 \text{ easy}) / P(3 \text{ shown})$$

$$P(2 \text{ easy}) = 1/3$$

$$P(3 \text{ shown} \mid 2 \text{ easy}) = ?$$

$$P(3 \text{ shown}) = ?$$



Using Bayes' law...

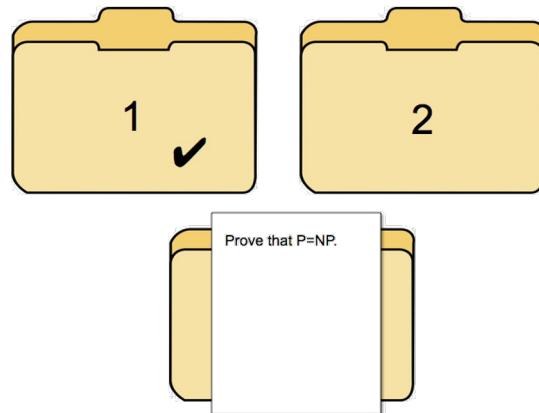
Given that #1 was chosen by the student,

$$P(2 \text{ easy} \mid 3 \text{ shown}) = P(3 \text{ shown} \mid 2 \text{ easy}) P(2 \text{ easy}) / P(3 \text{ shown})$$

$$P(2 \text{ easy}) = 1/3$$

$$P(3 \text{ shown} \mid 2 \text{ easy}) = 1$$

$$P(3 \text{ shown}) = ?$$



Using Bayes' law...

Given that #1 was chosen by the student,

$$P(2 \text{ easy} \mid 3 \text{ shown}) = P(3 \text{ shown} \mid 2 \text{ easy}) P(2 \text{ easy}) / P(3 \text{ shown})$$

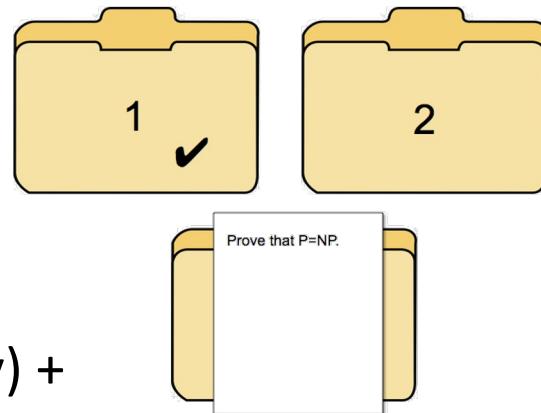
$$P(2 \text{ easy}) = 1/3$$

$$P(3 \text{ shown} \mid 2 \text{ easy}) = 1$$

$$P(3 \text{ shown}) = P(1 \text{ easy}) P(3 \text{ shown} \mid 1 \text{ easy}) +$$

$$P(2 \text{ easy}) P(3 \text{ shown} \mid 2 \text{ easy}) +$$

$$P(3 \text{ easy}) P(3 \text{ shown} \mid 3 \text{ easy})$$



Using Bayes' law...

Given that #1 was chosen by the student,

$$P(2 \text{ easy} \mid 3 \text{ shown}) = P(3 \text{ shown} \mid 2 \text{ easy}) P(2 \text{ easy}) / P(3 \text{ shown})$$

$$P(2 \text{ easy}) = 1/3$$

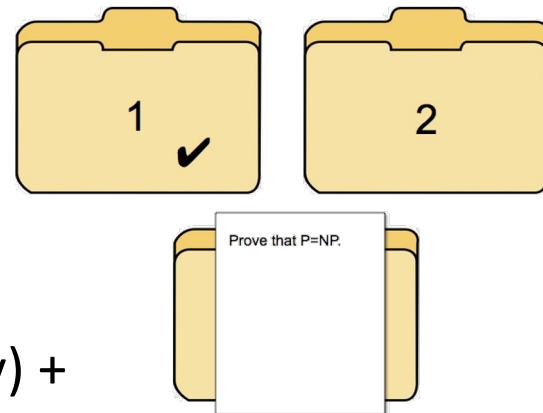
$$P(3 \text{ shown} \mid 2 \text{ easy}) = 1$$

$$P(3 \text{ shown}) = P(1 \text{ easy}) P(3 \text{ shown} \mid 1 \text{ easy}) +$$

$$P(2 \text{ easy}) P(3 \text{ shown} \mid 2 \text{ easy}) +$$

$$P(3 \text{ easy}) P(3 \text{ shown} \mid 3 \text{ easy})$$

$$= (1/3) (1/2) + (1/3) (1) + (1/3) (0) = 1/2$$



Using Bayes' law...

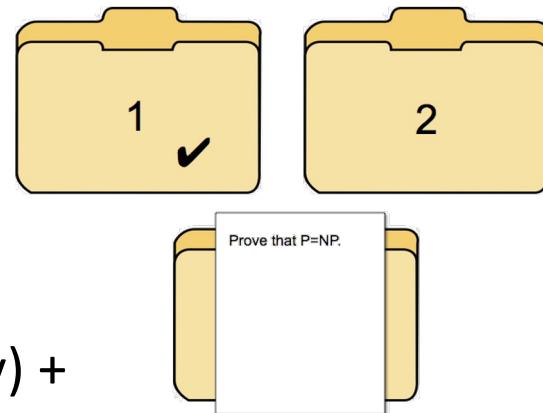
Given that #1 was chosen by the student,

$$P(2 \text{ easy} \mid 3 \text{ shown}) = P(3 \text{ shown} \mid 2 \text{ easy}) P(2 \text{ easy}) / P(3 \text{ shown})$$

$$P(2 \text{ easy}) = 1/3$$

$$P(3 \text{ shown} \mid 2 \text{ easy}) = 1$$

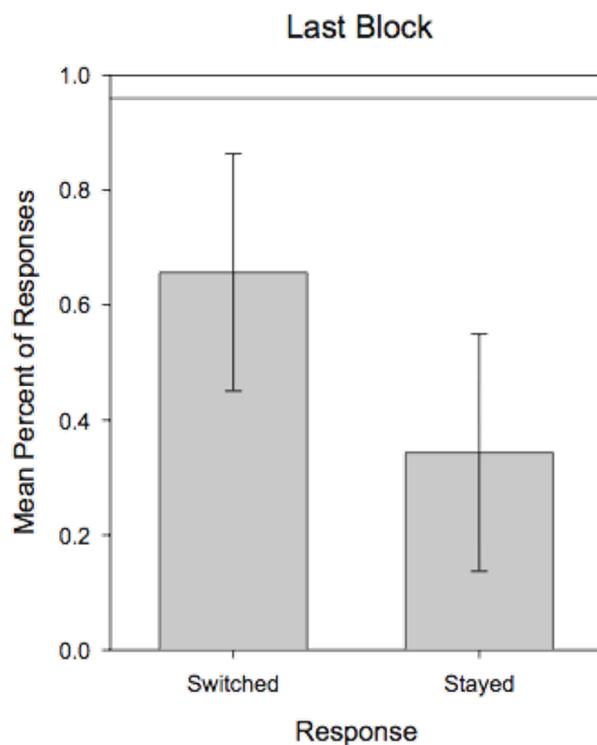
$$\begin{aligned} P(3 \text{ shown}) &= P(1 \text{ easy}) P(3 \text{ shown} \mid 1 \text{ easy}) + \\ &\quad P(2 \text{ easy}) P(3 \text{ shown} \mid 2 \text{ easy}) + \\ &\quad P(3 \text{ easy}) P(3 \text{ shown} \mid 3 \text{ easy}) \\ &= (1/3) (1/2) + (1/3) (1) + (1/3) (0) = 1/2 \end{aligned}$$



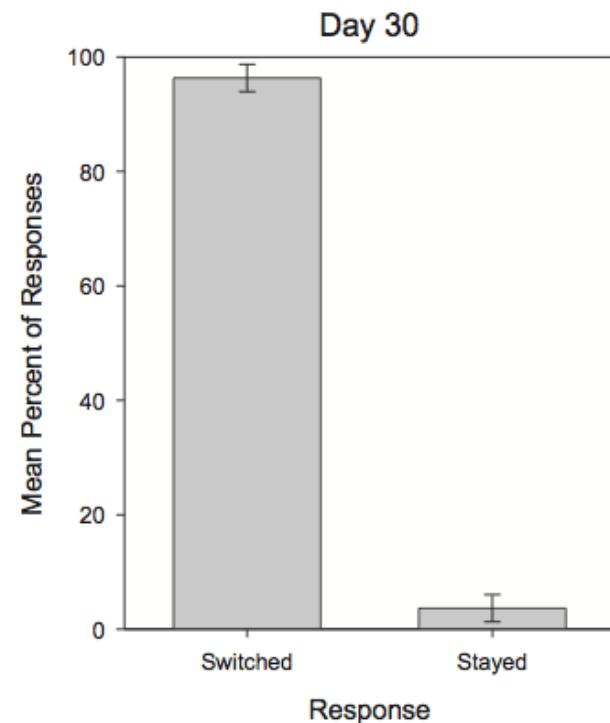
$$P(2 \text{ easy} \mid 3 \text{ shown}) = (1)(1/3) / (1/2) = 2/3$$

Monty Hall Problem

Behavior of undergrads
after 200 trials:



Behavior of pigeons:



Bayes' Law Example #3b

- Suppose that you have a friend, Mary, who lives in Seattle. She tells the truth 80% of the time and lies the other 20% of the time. The weather in Seattle on any given day is either sunny or cloudy; 30% of days are sunny and 70% are cloudy.
- Suppose that you have another friend in Seattle, Sue, who tells the truth 90% of the time. Sue tells you that the weather is cloudy, while on the same day Mary says that it is sunny. What is the probability that the weather is actually sunny? You may assume that whether or not Sue lies is independent of whether or not Mary lies.

Solution #3b

In addition to events C, S, Mc, and Ms, defined as in the last problem, let Sc and Ss denote the events that Sue says cloudy and sunny, respectively. From Bayes' Law we have:

$$P(S|M_s, S_c) = \frac{P(M_s, S_c|S)P(S)}{P(M_s, S_c)}$$

The assumption that Mary and Sue decide whether or not to lie independently lets us factor P (Ms, Sc|S) as,

$$P(M_s, S_c|S) = P(M_s|S)P(S_c|S) = (0.8)(0.1) = 0.08$$

$$\begin{aligned} P(M_s, S_c) &= P(M_s, S_c|S)P(S) + P(M_s, S_c|C)P(C) = \\ &= P(M_s|S)P(S_c|S)P(S) + P(M_s|C)P(S_c|C)P(C) = \\ &= (0.8)(0.1)(0.3) + (0.2)(0.9)(0.7) = 0.15 \end{aligned}$$

$$P(S|M_s, S_c) = \frac{(0.08)(0.3)}{0.15} = 0.16$$

Random variables

- A *random variable* is like a multi-valued “attribute” that can be used to specify sets of outcomes
 - Formally, it’s a function that associates an attribute with each outcome, typically $f : S \rightarrow \mathbb{N}$ or $f : S \rightarrow \mathbb{R}$
- E.g. Roll a die 5 times. Let random variable X be the number of 1’s rolled.
 - Now $P(X=2)$ denotes probability of rolling two 1’s, i.e.

$$P(X = 2) = P(\{a \in S | f(a) = 2\})$$

where $f(a)$ counts the number of 1’s in a given outcome, i.e. $f(11111)=5$, $f(11114)=4$, ...

Marginal distributions

- A probability distribution over the values of a random variable is called a *marginal distribution*
- E.g. Roll a die 5 times. Let random variable X be the number of 1's rolled.
 - $P(X)$ is the marginal distribution over X

Joint distributions

- A probability distribution over multiple random variables is called a *joint distribution*
- E.g. Roll a die 5 times. Let random variable X be the number of 1's rolled, and Y be the *sum* of the rolls.
 - $P(X,Y)$ is the joint distribution, i.e. $P : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{R}$
 - E.g. $P(X=3, Y=6)$ refers to the probability that a given roll has 3 1's and sums to 6

An example

- Given set of students, define random variables X for Intelligence and Y for grade in class
 - Joint distribution $P(X,Y)$ given by:

		<i>Intelligence</i>	
		<i>low</i>	<i>high</i>
		A	B
<i>Grade</i>	A	0.07	0.18
	B	0.28	0.09
	C	0.35	0.03

- What are the marginal distributions $P(X)$ and $P(Y)$?

Conditional probabilities

- Conditional probabilities can also be written using random variables
 - Denoted $P(X|Y)$ for random variables X and Y
 - For any value of Y, this is a distribution over X
- Example: What's $P(X=\text{low} \mid Y=B)$?

		<i>Intelligence</i>	
		<i>low</i>	<i>high</i>
<i>Grade</i>	<i>A</i>	0.07	0.18
	<i>B</i>	0.28	0.09
	<i>C</i>	0.35	0.03

Expectation

- The expected value of a (discrete) random variable X ,

$$E[X] = \sum_{x \in Val(X)} xP(x)$$

- E.g. Let X be the amount of money you win in the \$50 million Powerball Jackpot (Sept 2019). The probability of winning is about 1 in 300 million. What is $E[X]$?

Statistical Inference

Inference: Using distributions to answer questions

- Several main types of queries
- Type 1: Probability queries
 - Calculate distribution over some random variables given observed values for others
 - e.g. calculate $P(\mathbf{Y} \mid \mathbf{X} = \mathbf{x})$, where \mathbf{X} and \mathbf{Y} are sets of random variables
 - There might be a set of other random variables \mathbf{Z} that are neither the query variables nor the observations; these can be “marginalized out”:

$$P(\mathbf{Y}|\mathbf{X} = \mathbf{x}) = \sum_{\mathbf{Z}} P(\mathbf{Y}, \mathbf{Z}|\mathbf{X} = \mathbf{x})$$

Using distributions to answer questions

- Type 2: Expectation queries
 - Calculate the expected value “on average” for some random variables, given observed values for others
 - i.e. $E[Y | E=e]$
- Type 3: Maximum A Priori (MAP) queries
 - Calculate the most likely values for some random variables, given observed values for others,

$$\arg \max_{\mathbf{y}} P(\mathbf{Y} = \mathbf{y} | \mathbf{E} = \mathbf{e})$$

Back to AI...

- In AI we often want to predict an unknown answer given known answers to past problems
 - E.g., Given current weather observations, will it rain later?
- Whether it will rain (R) may depend on hundreds or thousands of observations, $V_1, V_2, \dots, V_{1000}$
 - Temperatures across U.S., moisture in atmosphere, etc...
- Given enough days of data, we could directly estimate a probability function $P(R | V_1, V_2, \dots, V_{1000})$
 - Then problem would be solved!
 - How many days of data would you need?

A huge problem

- Say all variables of $(R, V_1, V_2, \dots, V_{1000})$ are binary
 - Need at least 2^{1000} days of data just to observe all possible combinations of the variables
 - Need to observe multiple days for each combination of variables to estimate conditional probability robustly
 - Simply impossible from a computational, representational, or intuitive point of view
- This seemed fatal for the first ~30 years of AI research
 - Graphical models are a framework for avoiding this problem by making assumptions about the structure of a model

Next class

- More Bayes Nets Inference

Bayes net inference

Announcements

- Assignment 1
 - See Canvas

Back to AI...

- In AI we often want to predict an unknown answer given known answers to past problems
 - E.g., Given current weather observations, will it rain later?
- Whether it will rain (R) may depend on hundreds or thousands of observations, $V_1, V_2, \dots, V_{1000}$
 - Temperatures across U.S., moisture in atmosphere, etc...
- Given enough days of data, we could directly estimate a probability function $P(R | V_1, V_2, \dots, V_{1000})$
 - Then problem would be solved!
 - How many days of data would you need?

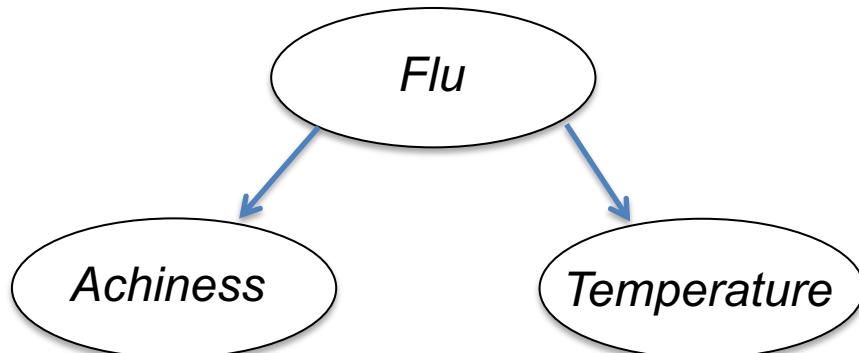
A huge problem

- Say all variables of $(R, V_1, V_2, \dots, V_{1000})$ are binary
 - Need at least 2^{1000} days of data just to observe all possible combinations of the variables
 - Need to observe multiple days for each combination of variables to estimate conditional probability robustly
 - Simply impossible from a computational, representational, or intuitive point of view
- This seemed fatal for the first ~30 years of AI research
 - Graphical models are a framework for avoiding this problem by making assumptions about the structure of a model

Bayesian Networks

Another example

- Say we want to decide whether someone has the flu (F) based on their temperature (T) and achiness (A)
- A, T, and F are clearly **not** independent
- But a weaker assumption of conditional independence may be appropriate, $A \perp T|F$
 - Says that A and T are independent *for a given value of F*
 - We can represent this assumption with a *Bayesian network*:

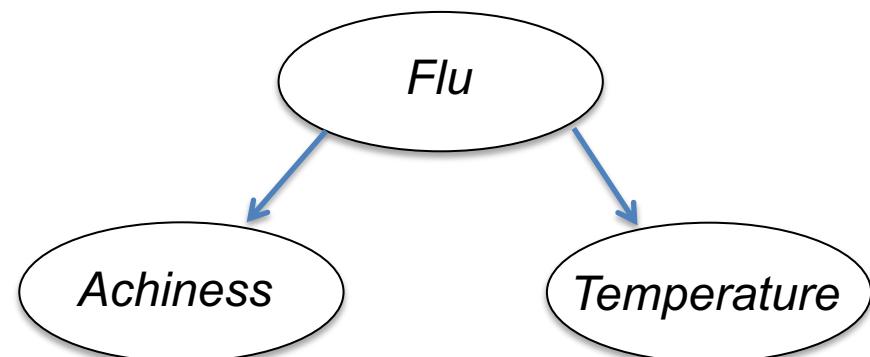


Another example

- Now we can factor $P(A,T,F)$ as:

$$P(A, T, F) = P(A|F)P(T|F)P(F)$$

- To decide whether someone has the flu given observed symptoms, we'll want to compute $P(F | A, T)$
 - How to compute this?



Back to the weather...

- We want to compute probability of rain (R) given observed variables $V_1, V_2, \dots, V_{1000}$. Using Bayes' law,

$$P(R|V_1, V_2, \dots, V_{1000}) = \frac{P(V_1, V_2, \dots, V_{1000}|R)P(R)}{P(V_1, V_2, \dots, V_{1000})}$$

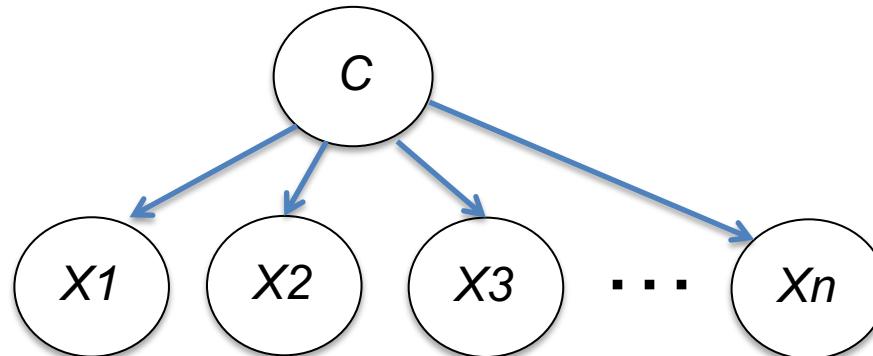
- Now, assuming that $V_1 \dots V_{1000}$ are conditionally independent given R :

$$P(V_1, V_2, \dots, V_{1000}|R) = \prod_1^{1000} P(V_i|R)$$

- How many parameters do we need to estimate in this factored model?

Naïve Bayes model

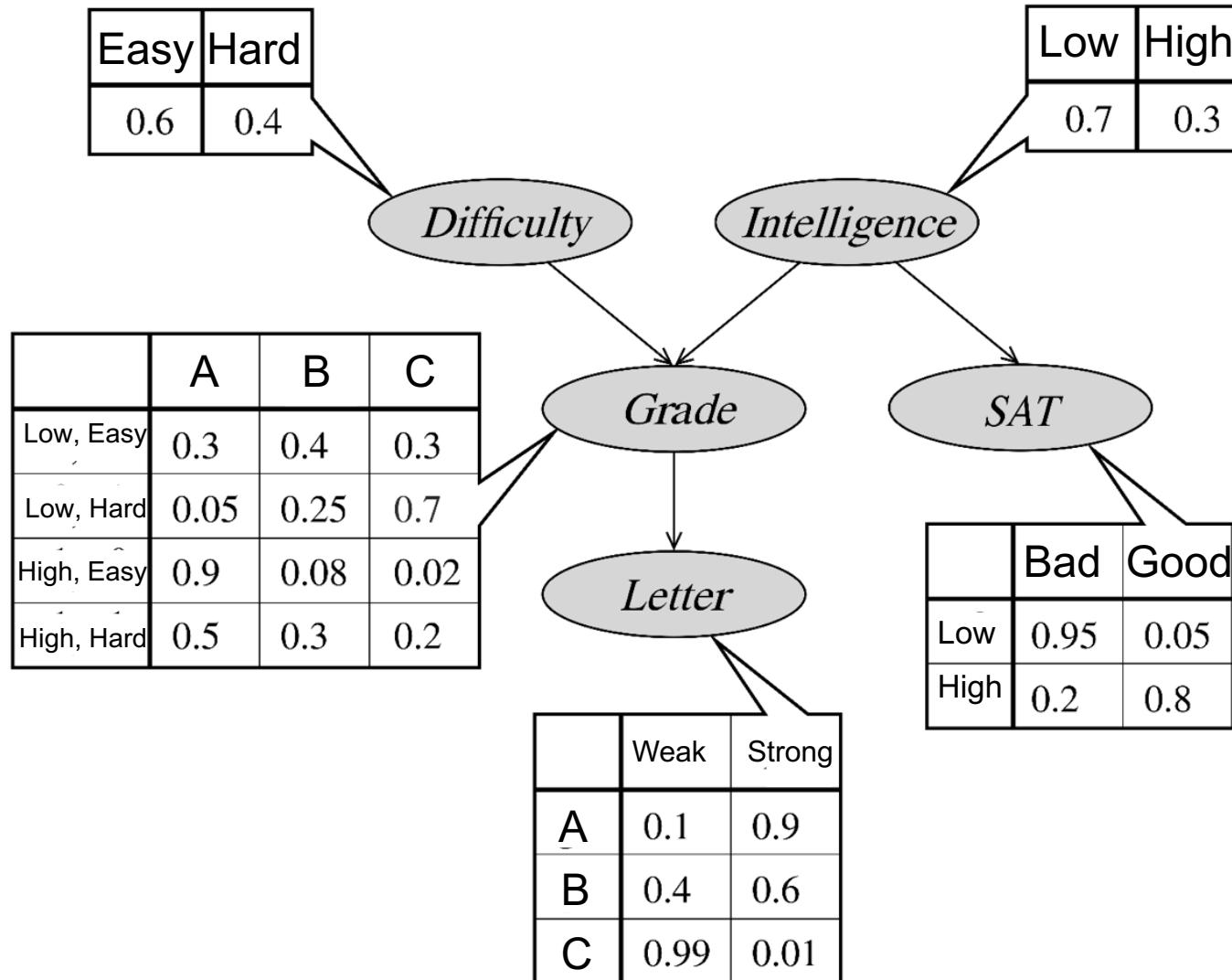
- Assuming conditional independence among observed variables is called *naïve Bayes*
 - Class label C we want to infer
 - Set of observable variables X_1, X_2, \dots, X_n
 - Assume that observable variables are independent conditioned on the class label C
 - Estimate prior distribution $P(C)$ and conditional distributions $P(X_1 | C), \dots, P(X_n | C)$ from training data
 - Use Bayes' Law to calculate $P(C | X_1 \dots X_n)$



Another example

- Suppose we want to model students in B551, using several random variables:
 - Intelligence (I)
 - GPA (G)
 - SAT score (S)
 - Difficulty of courses taken (D)
 - Strength of letter of recommendation (L)
- Intuitively, arrows in the BN represent direct dependencies between variables
 - Assuming these dependences, how does the joint distribution $P(I,G,S,D,L)$ factor?

Conditional probability distributions



Bayesian networks

- A Bayesian network is defined by a pair (G, P) , where:
 - G is a dag (directed acyclic graph), with nodes corresponding to variables $\{X_1, X_2, \dots, X_n\}$ and edges to direct dependencies
 - P is a probability distribution that satisfies independence assumptions induced by G
- The dag G encodes the conditional independence assumptions:

$$X_i \perp \text{Nd}(X_i) \mid \text{Pa}(X_i)$$

- where $\text{Nd}(X_i)$ is the set of non-descendants of X_i ,
and $\text{Pa}(X_i)$ is the set of parents of X_i

Factorization of Bayes nets

- Given a Bayes net (G, P) over variables $\{X_1, X_2 \dots X_n\}$,
the joint probability distribution factors as,

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Pa}(X_i))$$

Independencies in Bayes nets

- We already have a set of some conditional independence relationships, given by:

$$X_i \perp \text{Nd}(X_i) \mid \text{Pa}(X_i)$$

- These are just the relationships directly defined by G; there are often others

For three nodes, Four cases

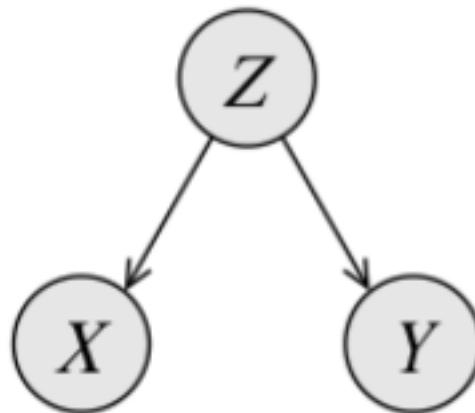
- Is $X \perp Y \mid Z$ in each case? **Is $X \perp Y$ in each case?**



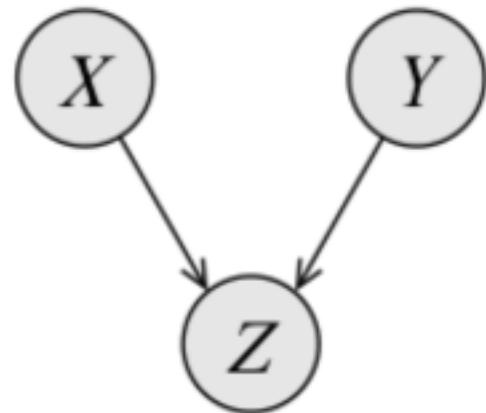
(a)



(b)



(c)



(d)

Solving problems with Bayes Nets

Solving problems with Bayes nets

- We'd like to use Bayes nets to estimate (distributions of) variables, given observed values for some variables
 - aka *Conditional Probability Queries*: Given a set of variables \mathbf{E} and corresponding values \mathbf{e} , estimate distributions over unobservable values \mathbf{Y} , i.e. $P(\mathbf{Y} \mid \mathbf{E}=\mathbf{e})$

Marginal inference example

- Alice flips a fair coin, and tells the result to Bob.
- Bob tells Charlie the true result 80% of the time, but lies 20% of the time.
- If Charlie hears heads, he tells Donna heads with probability 90% and tails with probability 10%. If he hears tails, he tells Donna heads 40% of the time and tails 60% of the time.
- What is the probability Donna hears heads?

Marginal inference example

- We want to compute $P(D)$

$$P(D) = \sum_C \sum_B \sum_A P(A, B, C, D)$$

$$P(D) = \sum_C \sum_B \sum_A P(A)P(B|A)P(C|B)P(D|C)$$

$$P(D) = \sum_C \sum_B P(C|B)P(D|C) \left(\sum_A P(A)P(B|A) \right)$$

$$P(D) = \sum_C \sum_B P(C|B)P(D|C)\tau_1(B)$$

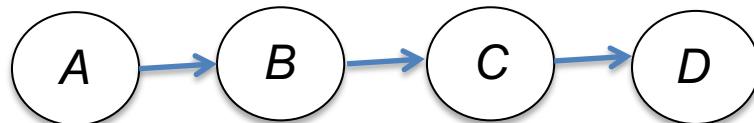
$$P(D) = \sum_C P(D|C) \left(\sum_B P(C|B)\tau_1(B) \right)$$

$$P(D) = \sum_C P(D|C)\tau_2(C)$$

Dynamic programming

- This idea of caching intermediate results (in the form of tables τ_1 and τ_2) is called *dynamic programming*
 - General algorithmic concept
 - Other examples: Dijkstra's algorithm, string algorithms (e.g. for bioinformatics), Tower of Hanoi puzzle, ...

How did we avoid exponential time?



- Two important ingredients:
 - 1. The independence assumptions of the Bayes net allowed us to factor the joint distribution into simpler terms, each of which involved only a few variables.
 - 2. Dynamic programming let us “cache” intermediate results, avoiding re-computing them repeatedly.

More generally...

- More generally, notice that for any **sets** of random variables \mathbf{U} , \mathbf{V} , \mathbf{W} , and \mathbf{X} , and random variable $Z \notin \mathbf{U} \cup \mathbf{V}$,

$$\sum_Z P(\mathbf{U}|\mathbf{V})P(\mathbf{W}|\mathbf{X}) = P(\mathbf{U}|\mathbf{V}) \sum_Z P(\mathbf{W}|\mathbf{X})$$

- So, in the chain example above, this lets us do:

$$\begin{aligned} P(D) &= \sum_C \sum_B \sum_A P(A)P(B|A)P(C|B)P(D|C) \\ &= \sum_C \sum_B P(C|B)P(D|C) \left(\sum_A P(A)P(B|A) \right) \\ &= \sum_C P(D|C) \left(\sum_B P(C|B) \left(\sum_A P(A)P(B|A) \right) \right) \\ &= \sum_C P(D|C) \sum_B P(C|B) \sum_A P(A)P(B|A) \end{aligned}$$

Next class

- Hidden Markov Models

Hidden Markov Models

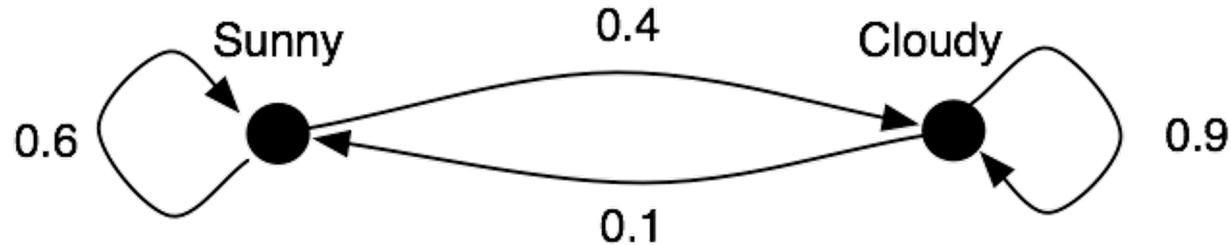
Sequence models

- A recurring theme in AI is modeling variables that change over time (or another single dimension)
 - E.g. weather across time, words across a sentence, etc.

Markov chains



- Stochastic process model
 - Due to Andrey Markov (1906)
 - e.g.,

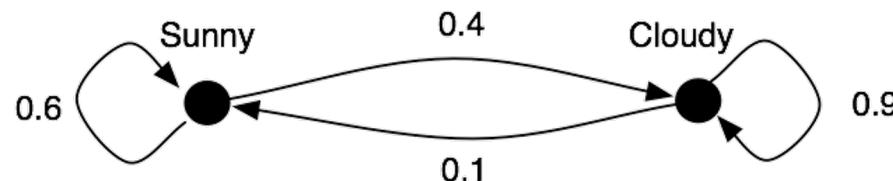


Markov chain

- Models a system which is in exactly one state at any time t , denoted by random variable Q_t
- A Markov chain model consists of:
 - A discrete set of states $S=\{s_1, \dots s_N\}$
 - An *initial probability distribution* $P(Q_0)$
 - Transition probability distribution, given by a conditional distribution $P(Q_{t+1} | Q_t)$
- The Markov assumption:
 - The probability of transitioning to each new state depends *only* on the current state (and not on the previous states)
 - More formally,

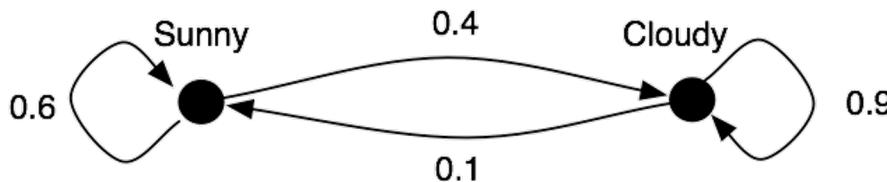
$$P(Q_{t+1} = q_{t+1} | Q_t = q_t, Q_{t-1} = q_{t-1}, \dots, Q_0 = q_0) = P(Q_{t+1} = q_{t+1} | Q_t = q_t)$$

Markov chains



- Suppose there's a 90% chance of sun on day 0.
What is the probability of sun on day 3?

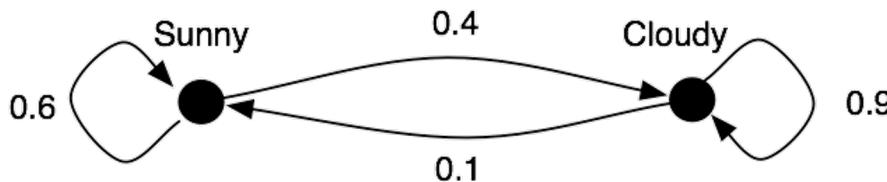
Markov chains



- Suppose there's a 90% chance of sun on day 0.
What is the probability of sun on day 3?

$$P(Q_3 = \text{☀}) = P(Q_3 = \text{☀} | Q_2 = \text{☀})P(Q_2 = \text{☀}) + P(Q_3 = \text{☀} | Q_2 = \text{☁})P(Q_2 = \text{☁})$$

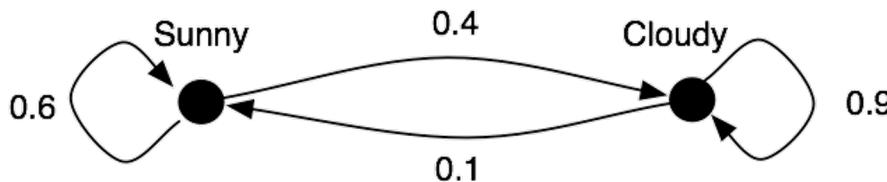
Markov chains



- Suppose there's a 90% chance of sun on day 0.
What is the probability of sun on day 3?

$$\begin{aligned} P(Q_3 = \text{☀}) &= P(Q_3 = \text{☀}|Q_2 = \text{☀})P(Q_2 = \text{☀}) + P(Q_3 = \text{☀}|Q_2 = \text{☁})P(Q_2 = \text{☁}) \\ &= 0.6P(Q_2 = \text{☀}) + 0.1P(Q_2 = \text{☁}) \end{aligned}$$

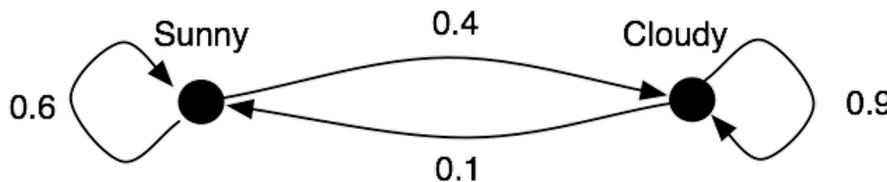
Markov chains



- Suppose there's a 90% chance of sun on day 0.
What is the probability of sun on day 3?

$$\begin{aligned}P(Q_3 = \text{☀}) &= P(Q_3 = \text{☀}|Q_2 = \text{☀})P(Q_2 = \text{☀}) + P(Q_3 = \text{☀}|Q_2 = \text{☁})P(Q_2 = \text{☁}) \\&= 0.6P(Q_2 = \text{☀}) + 0.1P(Q_2 = \text{☁}) \\&= 0.6(0.6P(Q_1 = \text{☀})) + 0.1(0.4P(Q_1 = \text{☀})) + 0.1(0.4P(Q_1 = \text{☁})) + 0.9P(Q_1 = \text{☁})\end{aligned}$$

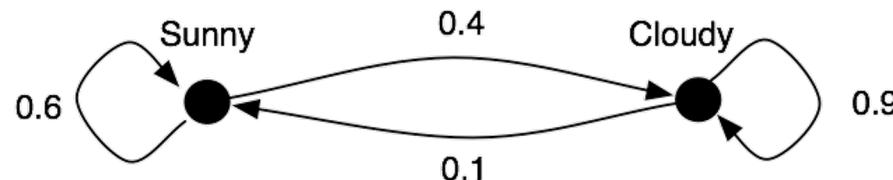
Markov chains



- Suppose there's a 90% chance of sun on day 0.
What is the probability of sun on day 3?

$$\begin{aligned} P(Q_3 = \text{Sunny}) &= P(Q_3 = \text{Sunny} | Q_2 = \text{Sunny})P(Q_2 = \text{Sunny}) + P(Q_3 = \text{Sunny} | Q_2 = \text{Cloudy})P(Q_2 = \text{Cloudy}) \\ &= 0.6P(Q_2 = \text{Sunny}) + 0.1P(Q_2 = \text{Cloudy}) \\ &= 0.6(0.6P(Q_1 = \text{Sunny}) + 0.1P(Q_1 = \text{Cloudy})) + 0.1(0.4P(Q_1 = \text{Sunny}) + 0.9P(Q_1 = \text{Cloudy})) \\ &= 0.6(0.6(0.6P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.1(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \\ &\quad + 0.1(0.4(0.4P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.9(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \end{aligned}$$

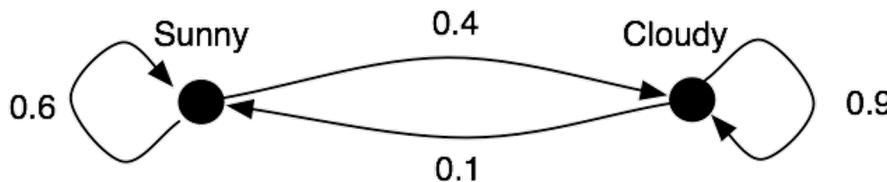
Markov chains



- Suppose there's a 90% chance of sun on day 0.
What is the probability of sun on day 3?

$$\begin{aligned} P(Q_3 = \text{Sunny}) &= P(Q_3 = \text{Sunny} | Q_2 = \text{Sunny})P(Q_2 = \text{Sunny}) + P(Q_3 = \text{Sunny} | Q_2 = \text{Cloudy})P(Q_2 = \text{Cloudy}) \\ &= 0.6P(Q_2 = \text{Sunny}) + 0.1P(Q_2 = \text{Cloudy}) \\ &= 0.6(0.6P(Q_1 = \text{Sunny}) + 0.1P(Q_1 = \text{Cloudy})) + 0.1(0.4P(Q_1 = \text{Sunny}) + 0.9P(Q_1 = \text{Cloudy})) \\ &= 0.6(0.6(0.6P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.1(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \\ &\quad + 0.1(0.4(0.4P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.9(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \\ &= 0.6(0.6(0.6(0.8) + 0.1(0.2)) + 0.1(0.4(0.8) + 0.9(0.2))) \\ &\quad + 0.1(0.4(0.6(0.8) + 0.1(0.2)) + 0.9(0.4(0.8) + 0.9(0.2))) \end{aligned}$$

Markov chains

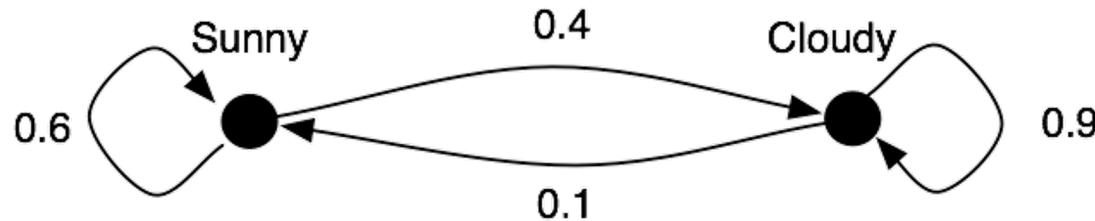


- Suppose there's a 80% chance of sun on day 0.
What is the probability of sun on day 3?

$$\begin{aligned} P(Q_3 = \text{Sunny}) &= P(Q_3 = \text{Sunny} | Q_2 = \text{Sunny})P(Q_2 = \text{Sunny}) + P(Q_3 = \text{Sunny} | Q_2 = \text{Cloudy})P(Q_2 = \text{Cloudy}) \\ &= 0.6P(Q_2 = \text{Sunny}) + 0.1P(Q_2 = \text{Cloudy}) \\ &= 0.6(0.6P(Q_1 = \text{Sunny}) + 0.1P(Q_1 = \text{Cloudy})) + 0.1(0.4P(Q_1 = \text{Sunny}) + 0.9P(Q_1 = \text{Cloudy})) \\ &= 0.6(0.6(0.6P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.1(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \\ &\quad + 0.1(0.4(0.4P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.9(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \\ &= 0.6(0.6(0.6(0.8) + 0.1(0.2)) + 0.1(0.4(0.8) + 0.9(0.2))) \\ &\quad + 0.1(0.4(0.6(0.8) + 0.1(0.2)) + 0.9(0.4(0.8) + 0.9(0.2))) \\ &= 0.275 \end{aligned}$$

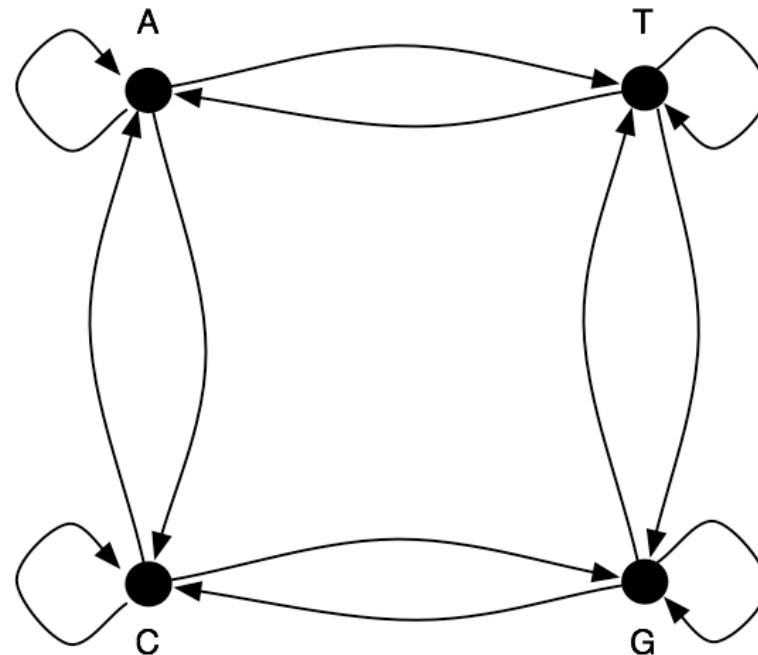
Learning a Markov chain

- Suppose the transition probabilities weren't given.
How would you estimate them?



Application: bioinformatics

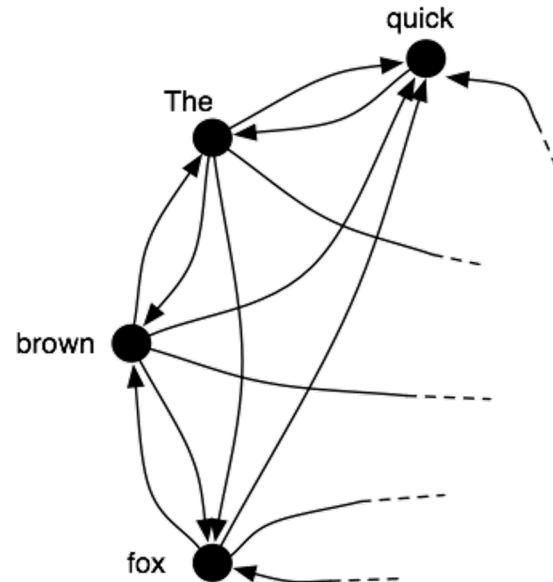
- Markov chains are used to model biological sequences
 - e.g. peptide



Application: language modeling

- A sentence is just a sequence of words
 - which we can model as a sequence of states.
- Sentence generation can be modeled as a Markov chain

The quick brown fox jumps
over the lazy dog.



Automatic sentence generation

- Random walks on the Markov chain produce sentences!
 - e.g. using a model trained on an essay of Jean Baudrillard, “The Precession of Simulacra”

If we were to revive the fable is useless. Perhaps only the allegory of simulation is unendurable--more cruel than Artaud's Theatre of Cruelty, which was the first to practice deterrence, abstraction, disconnection, deterritorialisation, etc.; and if it were our own past. We are witnessing the end of the negative form. But nothing separates one pole from the very swing of voting "rights" to electoral "duties" that the disinvestment of the revolutionary and total strike collapses at the real and its object, as Castaneda does, etc., and to escape the spectre raised by simulation--namely that truth, reference and objective causes have ceased to exist.

By “Mark V. Shaney” and Rob Pike, 1989

Automatic sentence generation

- Random walks on the Markov chain produce sentences!
 - e.g. using a model trained on poetry

He was a light, slow, and there is a small Saturn -- away from a high flame lying in the life within it, a new dune, we are formations of caterpillars, we are formations of craziness to innocent, and as it moves it is complete different than the rising face, the cold water, even we can't see infinity is an ocean of downy treasure the welldeep pleasure of caterpillars, we are formations of the world, and what it with the ecstasy of the day is an iceberg we find ourselves on a caress mingled with sleep kill me its lights bands of subjective experience, and wonder why I had dirt a star-crystal-flower plants, made the dragon. Its neck was a novel entitled "Kaleidoscope Vision," which is hat crinkle were like fresh glass domain key - you become someone mentioned them and build in. We see the white my own rising and thunder clapping in the singularity of it, evaporating into a tree, like a long before shade.

Automatic sentence generation

- Random walks on the Markov chain produce sentences!
 - e.g. using a model trained on postings from alt.singles

When I meet someone on a professional basis, I want them to shave their arms. While at a conference a few weeks back, I spent an interesting evening with a grain of salt. I wouldn't dare take them seriously! This brings me back to the brash people who dare others to do so or not. I love a good flame argument, probably more than anyone....

By “Mark V. Shaney” and Rob Pike et al, 1984

Practical uses?

- Generating spam
- SCigen: Generating scientific papers?!
 - “Rooter: A Methodology for the Typical Unification of Access Points and Redundancy,” *11th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI)*, 2005.
 - "I/O Automata No Longer Considered Harmful," *3rd International Symposium of Interactive Media Design*, 2005.
 - Cooperative, Compact Algorithms for Randomized Algorithms, *Applied Mathematics and Computation* (accepted but eventually rescinded)

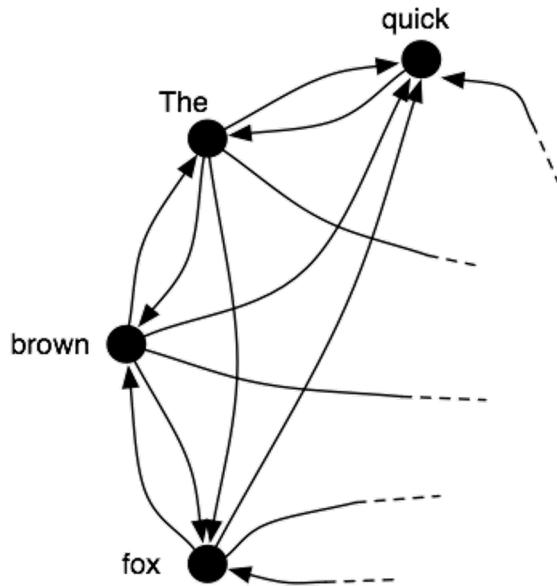
See <http://pdos.csail.mit.edu/scigen/>

Grammar checking

- Using a Markov model, we can compute the probability of sentences, marking low-probability ones
 - e.g.

Very low probability: Their is the quick brown fox.

Relatively high probability: There is the quick brown fox.



Hidden Markov Models (HMMs)

- A Markov Chain, but the system state is *not observable*
 - Instead there is an observable random variable, O , whose value probabilistically depends on the current state
- More formally, an HMM consists of:
 - Transition probabilities

$$p_{ij} = P(Q_{t+1} = j | Q_t = i)$$

- Initial state distribution

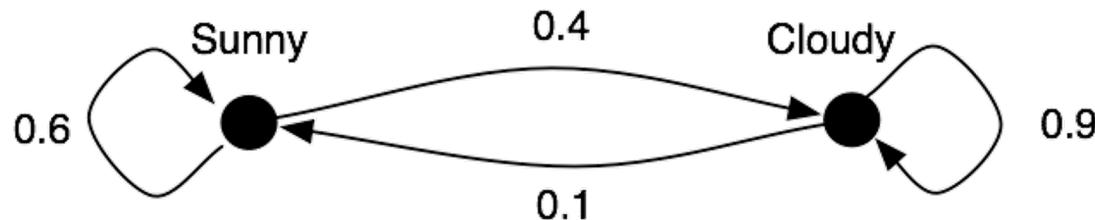
$$w_i = P(Q_0 = i)$$

- Emission probabilities

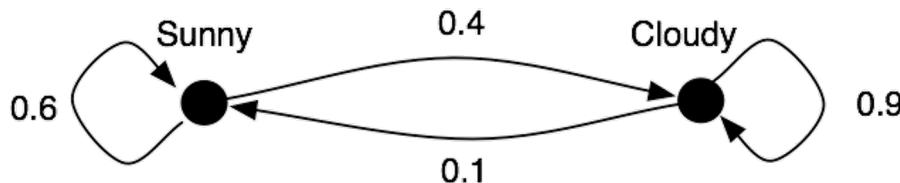
$$e_i(a) = P(O_t = a | Q_t = i)$$

Example

- Mary lives in Seattle, and tells the truth 80% of the time. Every day, she calls you to report the weather in Seattle.
 - It's either Sunny (S) or Cloudy (C)
- You know (based on historical data) that the weather in Seattle follows a Markov chain,



- Also, the probability of sun on any given day is 0.2
- Mary reports that the following sequence over a 5 day period: SCSCC



- Transition probabilities $p_{SS} = P(Q_{t+1} = S | Q_t = S) = 0.6$

$$p_{CS} = 0.1 \quad p_{CC} = 0.9 \quad p_{SC} = 0.4$$

- Emission probabilities

$$e_C(S) = P(O_t = S | Q_t = C) = 0.2 \quad e_S(C) = P(O_t = C | Q_t = S) = 0.2$$

$$e_C(C) = P(O_t = C | Q_t = C) = 0.8 \quad e_S(S) = P(O_t = S | Q_t = S) = 0.8$$

- Initial state distribution

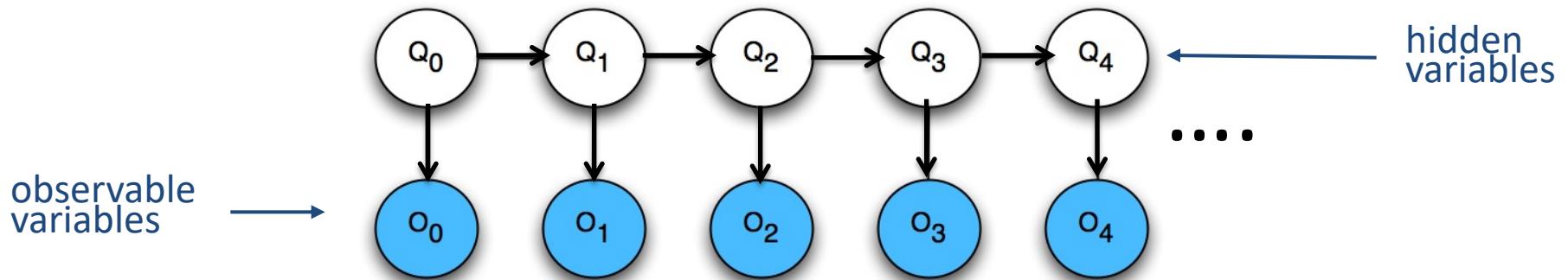
$$w_S = P(Q_0 = S) = 0.2 \quad w_C = P(Q_0 = C) = 0.8$$

- Observation sequence

$$O_0 = S, O_1 = C, O_2 = S, O_3 = C, O_4 = C$$

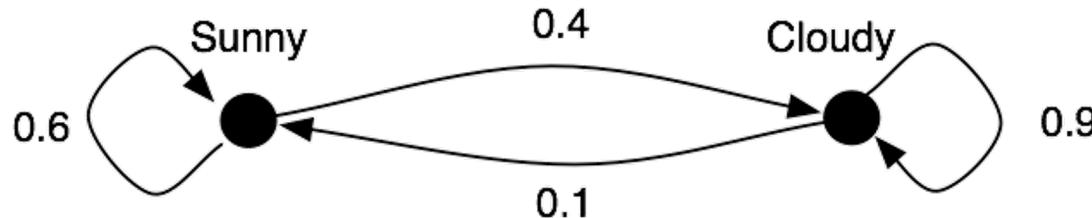
Inference on HMMs

- HMMs are just special cases of Bayes Nets!



- Intuitively, the HMM is balancing two goals:
 - maximizing emission probabilities -- finding a state sequence that agrees with the observations
 - maximizing transition probabilities -- finding a state sequence that has high likelihood according to the Markov chain

HMM inference



- Two important types of questions:
 - Given a particular observation (e.g. SCS), what is the distribution over the weather *on a particular day*? (Marginal inference)
 - Given a particular observation (e.g. SCS), what is the *most likely sequence of weather across all days*? (Maximum a posterior (MAP) inference)

HMM inference

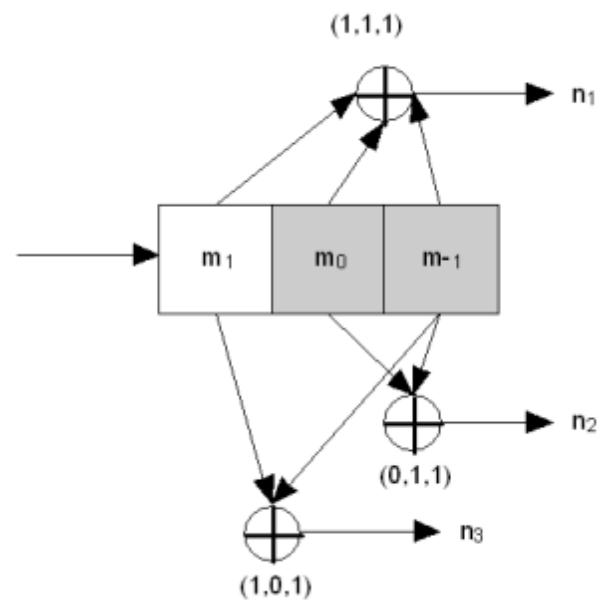
- How do we find the most likely state sequence, given a sequence of observations?
 - Brute force approach: Try all possible state sequences. Find the one that maximizes $P(Q|O)$.
 - Viterbi decoding: Efficient algorithm based on dynamic programming.

Convolution and turbo codes

- Used extensively in wireless communications
 - Adds redundancy to a signal, so that transmission errors can be detected and corrected
 - Transmitted bits are a combination of the last k input bits
 - Transmitted bits are possibly corrupted by interference
 - The decoder uses Viterbi to infer the (hidden) state of the transmitter, from the (noisy) received bits



Andrew Viterbi



Natural Language Processing

- Statistical techniques like HMMs are very popular

“Every time I fire a linguist, my performance goes up.”

--- attributed to Fred Jelinek, 1980's, IBM Watson

- For example: Part-of-speech (POS) tagging

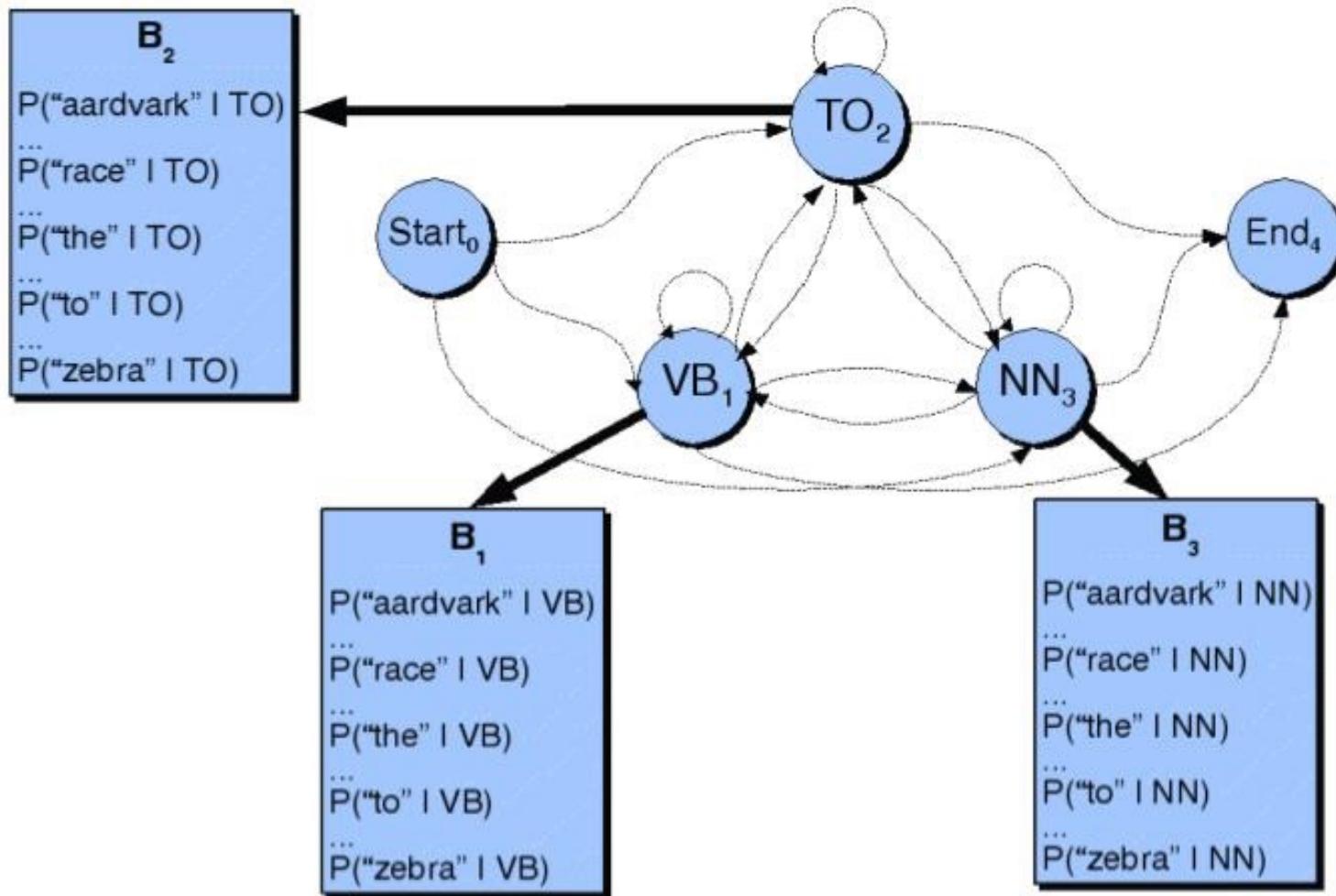
She promised to back the bill

POS tagging as an HMM problem

- Observation sequence
 - sequence of words
- States
 - parts of speech (noun, verb, adjective, etc.)
- Emission probabilities
 - probability that word w is produced by state s
 - related (by Baye's law) to the probability that w has POS s

$$P(V | \text{race}) = \frac{\text{Count}(\text{race is verb})}{\text{total Count}(\text{race})} \sim 0.95$$

HMM-based POS tagging



HMM-based POS tagging

- Transition probabilities

	VB	TO	NN	PPSS
<S>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

Figure 4.15 Tag transition probabilities (the α array, $p(t_i|t_{i-1})$) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus $P(PPSS|VB)$ is .0070. The symbol $<\text{s}>$ is the start-of-sentence symbol.

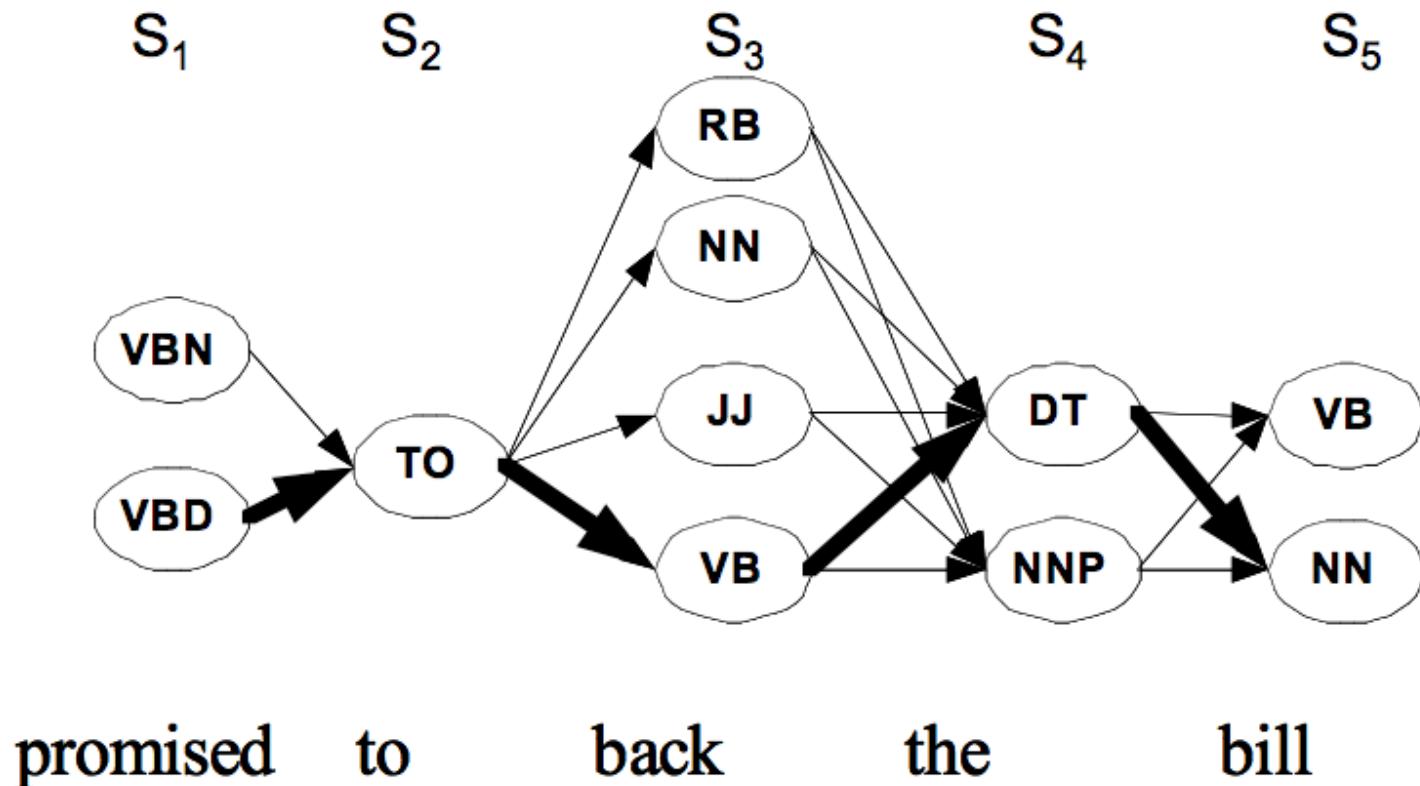
HMM-based POS tagging

- Emission probabilities

	I	want	to	race
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

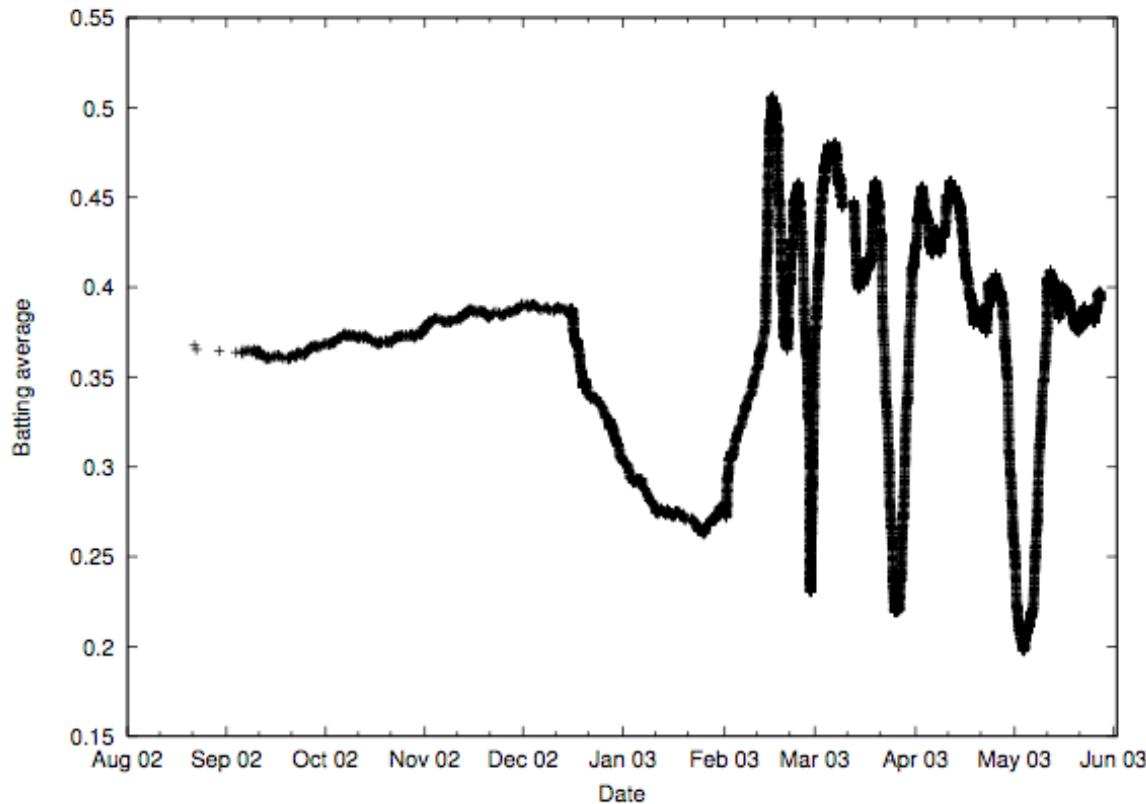
Figure 4.16 Observation likelihoods (the b array) computed from the 87-tag Brown corpus without smoothing.

POS decoding



Application: Analyzing noisy data

- Click-through rate for a particular webpage:

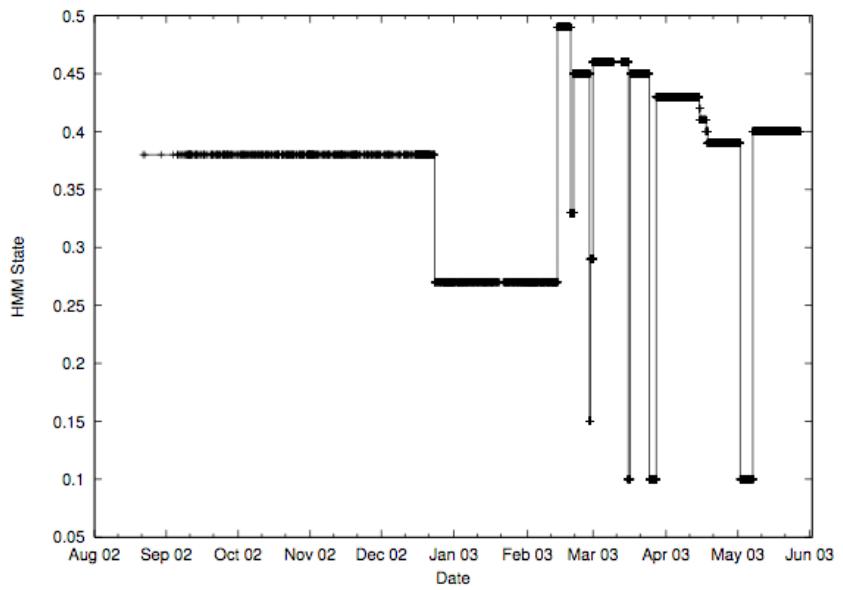
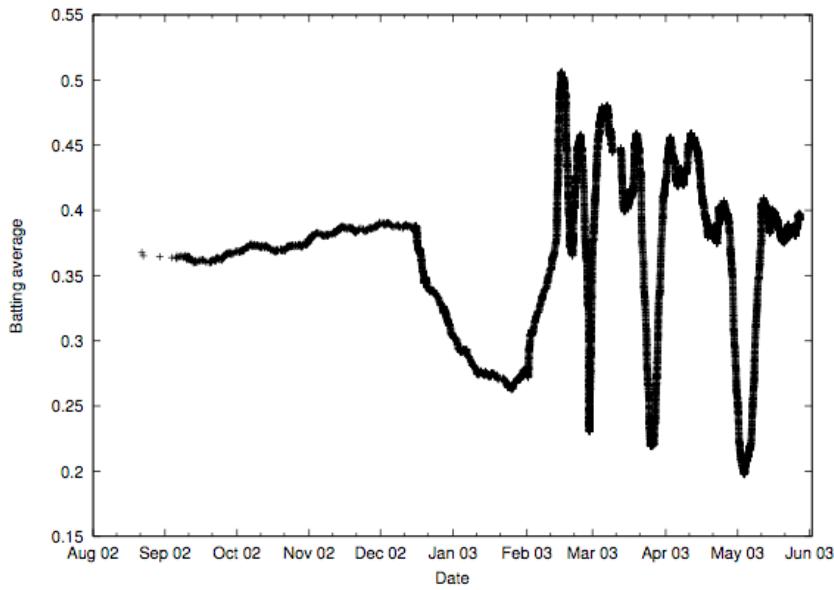


[Aizen04]

Application: Analyzing noisy data

- Model the world's interest in page as Markov chain
 - Changes with news events, etc.
 - But it is hidden -- we can't observe it directly
 - Instead we can observe raw click-through rates
- Use HMM inference to denoise the data
 - The states are the different interest levels
 - Observable variable is the empirical click-through rate
 - The Markov chain is learned from actual data
 - Viterbi gives the most likely interest level sequence

Application: Analyzing noisy data



Classifying photo streams



3:35pm

Alcatraz, SF bay?
Ellis Island, NYC?



8:03pm

Piazza San Marco, Venice?
Sather Tower, Berkeley?



9:27pm

Bay Bridge, SF bay?
Geo Wash Bridge, NYC?

Classifying photo streams



3:35pm

Alcatraz, SF bay?
~~Ellis Island, NYC?~~



8:03pm

~~Piazza San Marco, Venice?~~
Sather Tower, Berkeley?



9:27pm

Bay Bridge, SF bay?
~~Geo Wash Bridge, NYC?~~

Classifying photo streams



3:35pm

Alcatraz, SF bay?
~~Ellis Island, NYC?~~



8:03pm

~~Piazza San Marco, Venice?~~
Sather Tower, Berkeley?

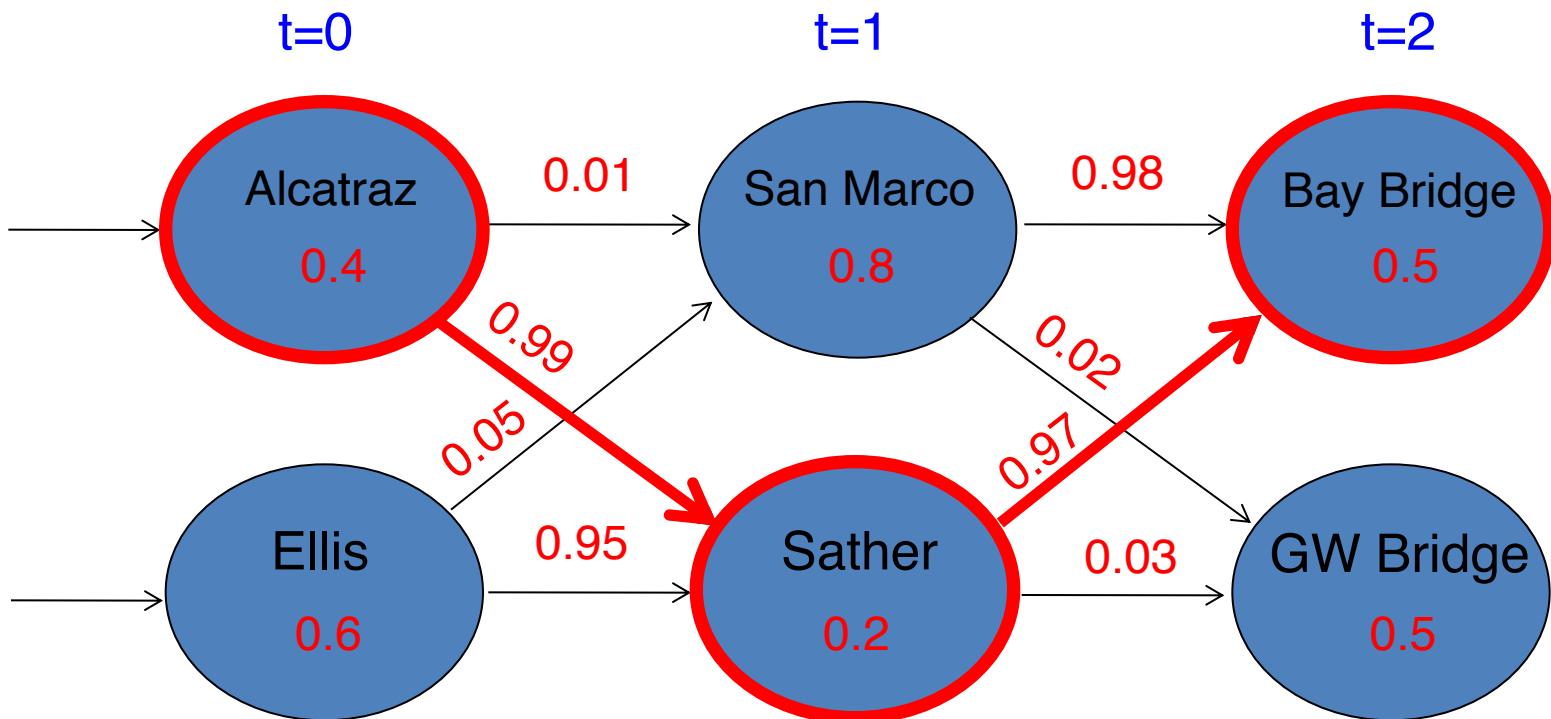


9:27pm

Bay Bridge, SF bay?
~~Geo Wash Bridge, NYC?~~

- Model as a Hidden Markov Model, do fast inference using the Viterbi algorithm

HMM decoding



Classifying photo streams with HMMs



Probabilities with individual photo classifiers:

Bathroom:	0.931	0.023	0.002	0.007	0.009	0.018	0.0160	0.073
Bedroom:	0.006	0.734	0.461	0.120	0.082	0.002	.885	0.018
Garage:	0.006	0.192	0.117	0.744	0.746	0.168	0.059	0.003
Living:	0.014	0.020	0.420	0.127	0.162	0.811	0.023	0.005
Office:	0.042	0.031	0.000	0.001	0.001	0.001	0.018	0.901
	✓	✗	✗	✗	✗	✓	✗	✓

Probabilities after applying HMM:

Bathroom:	0.896	0.436	0.060	0.015	0.010	0.006	0.002	0.000
Bedroom:	0.010	0.052	0.026	0.004	0.002	0.002	0.002	0.000
Garage:	✓	✓	✓	✓	✓	✓	✓	✓
Living:	0.079	0.441	0.881	0.968	0.975	0.873	0.125	0.005
Office:	0.006	0.027	0.009	0.009	0.012	0.116	0.865	0.994

HMM inference

- How do we find the most likely state sequence, given a sequence of observations?
 - Brute force approach: Try all possible state sequences. Find the one that maximizes $P(Q|O)$.
 - Viterbi decoding: Efficient algorithm based on dynamic programming.

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

$$\text{(Bayes' Law)} \quad = \quad \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)}$$

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

(Bayes' Law) $= \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)}$

(denom depends only on O) $\propto P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)$

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

$$\begin{aligned} \text{(Bayes' Law)} &= \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)} \\ \text{(denom depends only on O)} &\propto P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T) \\ \text{(O}_t \text{ depends only on Q}_t\text{)} &= P(Q_0 = q_0 \dots Q_T = q_T) \prod_{t=0}^T P(O_t | Q_t = q_t) \end{aligned}$$

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

(Bayes' Law) $= \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)}$

(denom depends only on O) $\propto P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)$

(O_t depends only on Q_t) $= P(Q_0 = q_0 \dots Q_T = q_T) \prod_{t=0}^T P(O_t | Q_t = q_t)$

(Markov property:
 Q_{t+1} depends only on Q_t) $= P(Q_0 = q_0) \prod_{t=0}^{T-1} P(Q_{t+1} = q_{t+1} | Q_t = q_t) \prod_{t=0}^T P(O_t | Q_t = q_t)$

Viterbi decoding

- Based on dynamic programming
 - Let $v_i(t)$ be the probability of the most probable path ending at state i at time t ,

$$v_i(t) = \max_{q_0 \dots q_{t-1}} P(Q_0 = q_0, \dots, Q_{t-1} = q_{t-1}, Q_t = i | O_0, O_1, \dots, O_t)$$

- Then we can recursively find the probability of the most probable path ending at state j at time $t+1$,

$$v_j(t+1) = e_j(O_{t+1}) \max_{1 \leq i \leq N} v_i(t) p_{ij}$$

Probability that system is in state j at time $t+1$ ($Q_{t+1}=j$)

Probability of observing O_{t+1} given that system is in state j at time $t+1$

Max over all possible states at time t

Probability of transition from state i to j

Probability that system in state i at time t

Next time

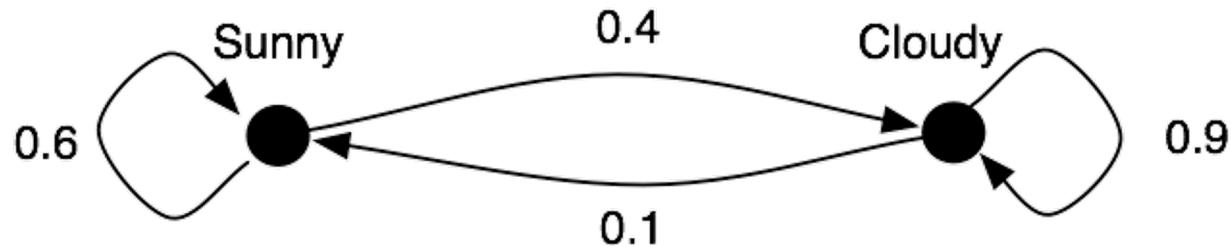
- Finish Viterbi, talk about Constraint Satisfaction Problems

Hidden Markov Models: Viterbi

Markov chains



- Stochastic process model
 - Due to Andrey Markov (1906)
 - e.g.,



Markov chain

- Models a system which is in exactly one state at any time t , denoted by random variable Q_t
- A Markov chain model consists of:
 - A discrete set of states $S=\{s_1, \dots s_N\}$
 - An *initial probability distribution* $P(Q_0)$
 - Transition probability distribution, given by a conditional distribution $P(Q_{t+1} | Q_t)$
- The Markov assumption:
 - The probability of transitioning to each new state depends *only* on the current state (and not on the previous states)
 - More formally,

$$P(Q_{t+1} = q_{t+1} | Q_t = q_t, Q_{t-1} = q_{t-1}, \dots, Q_0 = q_0) = P(Q_{t+1} = q_{t+1} | Q_t = q_t)$$

Hidden Markov Models (HMMs)

- A Markov Chain, but the system state is *not observable*
 - Instead there is an observable random variable, O , whose value probabilistically depends on the current state
- More formally, an HMM consists of:
 - Transition probabilities

$$p_{ij} = P(Q_{t+1} = j | Q_t = i)$$

- Initial state distribution

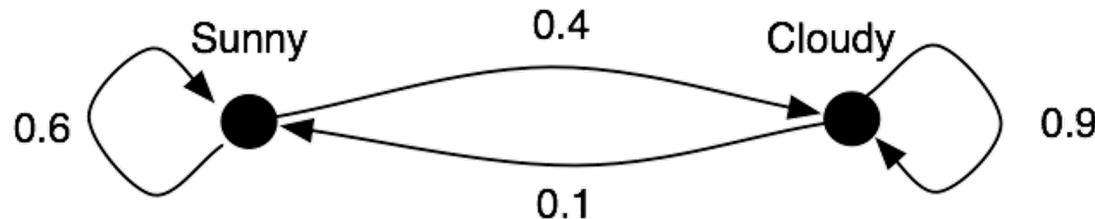
$$w_i = P(Q_0 = i)$$

- Emission probabilities

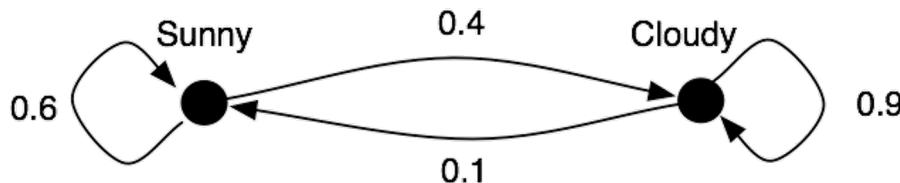
$$e_i(a) = P(O_t = a | Q_t = i)$$

Example

- Mary lives in Seattle, and tells the truth 80% of the time. Every day, she calls you to report the weather in Seattle.
 - It's either Sunny (S) or Cloudy (C)
- You know (based on historical data) that the weather in Seattle follows a Markov chain,



- Also, the probability of sun on any given day is 0.2
- Mary reports that the following sequence over a 5 day period: SCSCC



- Transition probabilities $p_{SS} = P(Q_{t+1} = S | Q_t = S) = 0.6$

$$p_{CS} = 0.1 \quad p_{CC} = 0.9 \quad p_{SC} = 0.4$$

- Emission probabilities

$$e_C(S) = P(O_t = S | Q_t = C) = 0.2 \quad e_S(C) = P(O_t = C | Q_t = S) = 0.2$$

$$e_C(C) = P(O_t = C | Q_t = C) = 0.8 \quad e_S(S) = P(O_t = S | Q_t = S) = 0.8$$

- Initial state distribution

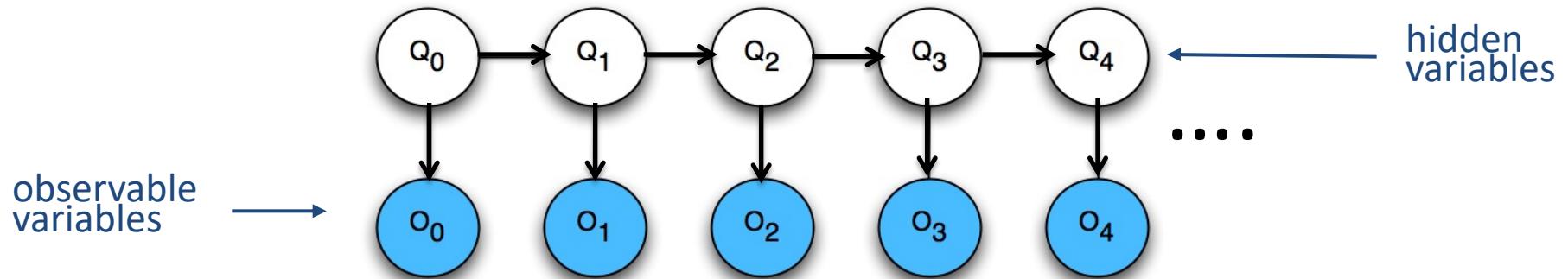
$$w_S = P(Q_0 = S) = 0.2 \quad w_C = P(Q_0 = C) = 0.8$$

- Observation sequence

$$O_0 = S, O_1 = C, O_2 = S, O_3 = C, O_4 = C$$

Inference on HMMs

- HMMs are just special cases of Bayes Nets!



- Intuitively, the HMM is balancing two goals:
 - maximizing emission probabilities -- finding a state sequence that agrees with the observations
 - maximizing transition probabilities -- finding a state sequence that has high likelihood according to the Markov chain

Classifying photo streams



3:35pm

Alcatraz, SF bay?
Ellis Island, NYC?



8:03pm

Piazza San Marco, Venice?
Sather Tower, Berkeley?



9:27pm

Bay Bridge, SF bay?
Geo Wash Bridge, NYC?

Classifying photo streams



3:35pm

Alcatraz, SF bay?
~~Ellis Island, NYC?~~



8:03pm

~~Piazza San Marco, Venice?~~
Sather Tower, Berkeley?



9:27pm

Bay Bridge, SF bay?
~~Geo Wash Bridge, NYC?~~

Classifying photo streams



3:35pm

Alcatraz, SF bay?
~~Ellis Island, NYC?~~



8:03pm

~~Piazza San Marco, Venice?~~
Sather Tower, Berkeley?



9:27pm

Bay Bridge, SF bay?
~~Geo Wash Bridge, NYC?~~

- Model as a Hidden Markov Model, do fast inference using the Viterbi algorithm

HMM inference

- How do we find the most likely state sequence, given a sequence of observations?
 - Brute force approach: Try all possible state sequences. Find the one that maximizes $P(Q|O)$.
 - Viterbi decoding: Efficient algorithm based on dynamic programming.

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

$$\text{(Bayes' Law)} \quad = \quad \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)}$$

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

(Bayes' Law) $= \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)}$

(denom depends only on O) $\propto P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)$

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

$$\begin{aligned} \text{(Bayes' Law)} &= \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)} \\ \text{(denom depends only on O)} &\propto P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T) \\ \text{(O}_t \text{ depends only on Q}_t\text{)} &= P(Q_0 = q_0 \dots Q_T = q_T) \prod_{t=0}^T P(O_t | Q_t = q_t) \end{aligned}$$

Viterbi decoding

- Key idea: the posterior probability of a state sequence, $P(Q|O)$, factors nicely

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

(Bayes' Law) $= \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)}$

(denom depends only on O) $\propto P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)$

(O_t depends only on Q_t) $= P(Q_0 = q_0 \dots Q_T = q_T) \prod_{t=0}^T P(O_t | Q_t = q_t)$

(Markov property:
 Q_{t+1} depends only on Q_t) $= P(Q_0 = q_0) \prod_{t=0}^{T-1} P(Q_{t+1} = q_{t+1} | Q_t = q_t) \prod_{t=0}^T P(O_t | Q_t = q_t)$

Viterbi decoding

- Based on dynamic programming
 - Let $v_i(t)$ be the probability of the most probable path ending at state i at time t ,

$$v_i(t) = \max_{q_0 \dots q_{t-1}} P(Q_0 = q_0, \dots, Q_{t-1} = q_{t-1}, Q_t = i | O_0, O_1, \dots, O_t)$$

- Then we can recursively find the probability of the most probable path ending at state j at time $t+1$,

$$v_j(t+1) = e_j(O_{t+1}) \max_{1 \leq i \leq N} v_i(t) p_{ij}$$

Probability that system is in state j at time $t+1$ ($Q_{t+1}=j$)

Probability of observing O_{t+1} given that system is in state j at time $t+1$

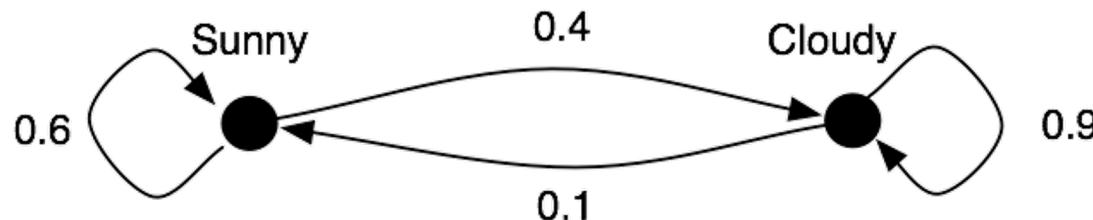
Max over all possible states at time t

Probability of transition from state i to j

Probability that system in state i at time t

Example

- Mary lives in Seattle, and tells the truth 80% of the time. Every day, she calls you to report the weather in Seattle.
 - It's either Sunny (S) or Cloudy (C)
- You know (based on historical data) that the weather in Seattle follows a Markov chain,



- Also, the probability of sun on any given day is 0.2
- Mary reports that the following sequence over a 5 day period: SCSCC

$$v_j(t+1) = e_j(O_{t+1}) \max_{1 \leq i \leq N} v_i(t) p_{ij}$$

Viterbi decoding

- Takes time $O(N^2T)$
 - N is the number of states
 - T is the length of the sequence
- For many useful state transition probability functions, it's possible to do this faster

Max probability vs Min cost

- Maximizing the probability $P(Q|O)$,

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T) = P(Q_0 = q_0) \prod_{t=0}^{T-1} P(Q_{t+1} = q_{t+1} | Q_t = q_t) \prod_{t=0}^T P(O_t | Q_t = q_t)$$

is equivalent to minimizing a negative log,

$$-\ln P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T)$$

$$= -\ln \left[P(Q_0 = q_0) \prod_{t=0}^{T-1} P(Q_{t+1} = q_{t+1} | Q_t = q_t) \prod_{t=0}^T P(O_t | Q_t = q_t) \right]$$

$$= -\ln P(Q_0 = q_0) + \sum_{t=0}^{T-1} -\ln P(Q_{t+1} = q_{t+1} | Q_t = q_t) + \sum_{t=0}^T -\ln P(O_t | Q_t = q_t)$$

Cost minimization view

- Viterbi finds a state sequence $q_0 \dots q_T$ that maximizes the posterior probability,

$$P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T) = \frac{P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) P(Q_0 = q_0 \dots Q_T = q_T)}{P(O_0 \dots O_T)}$$

- Equivalently, we can find a sequence that minimizes,

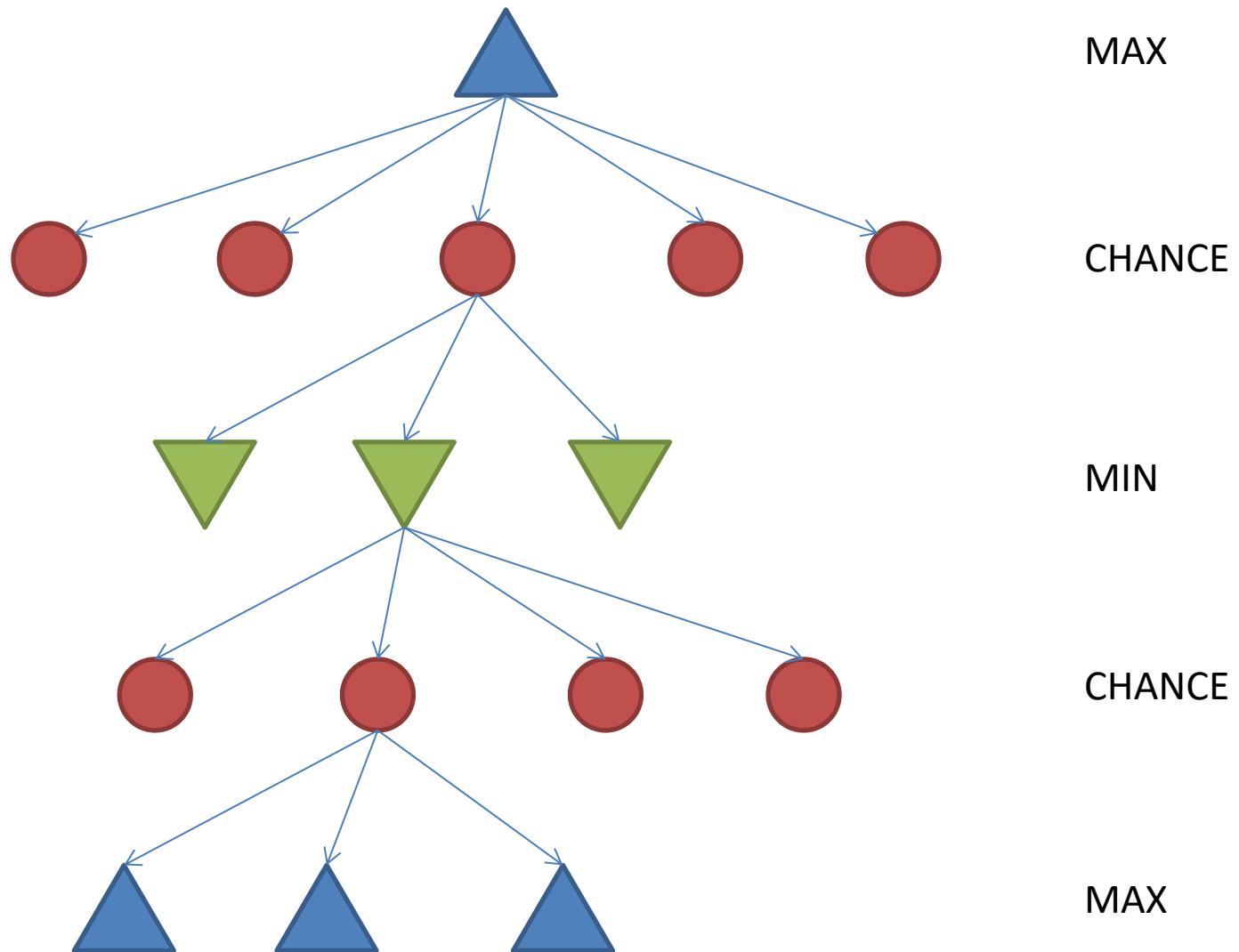
$$\begin{aligned} -\ln P(Q_0 = q_0, \dots, Q_T = q_T | O_0 \dots O_T) &= -\ln P(O_0 \dots O_T | Q_0 = q_0 \dots Q_T = q_T) \\ &\quad -\ln P(Q_0 = q_0 \dots Q_T = q_T) + \ln P(O_0 \dots O_T) \end{aligned}$$

- View the negative log probabilities as “costs”
- More convenient computationally (avoids multiplying very small probabilities)

Games of chance, and
more Variable Elimination

Games of chance

- Many games have non-determinism
 - E.g. Coin flips, dice, cards drawn from a pile, ...
- Other games have state that a player cannot observe
 - E.g., which cards other player holds, how much money has been bet, whether a monster is behind me, ...



Expected Values

- The **utility** of a MAX/MIN node in the game tree is the **max/min** of the utility values of its successors
- The **expected utility** of a CHANCE node is the **expected value** of the utility values of its successors

$$\text{ExpectedValue}(s) = \sum_{s' \in \text{SUCC}(s)} \text{ExpectedValue}(s') P(s')$$

CHANCE nodes

Compare to

$$\text{MinimaxValue}(s) = \max_{s' \in \text{SUCC}(s)} \text{MinimaxValue}(s')$$

MAX nodes

$$\text{MinimaxValue}(s) = \min_{s' \in \text{SUCC}(s)} \text{MinimaxValue}(s')$$

MIN nodes

Generalizing Minimax Values

- *Utilities* can be continuous numerical values, rather than +1, 0, -1
 - Allows maximizing the amount of “points” (e.g., \$) rewarded instead of just achieving a win
- **Rewards** associated with terminal states
- **Costs** can be associated with certain decisions at non-terminal states (e.g., placing a bet)

Roulette

- 18 red, 18 black, 2 green spots (American version)
- Bet on color: bet \$1, get \$2 back.
- Bet on a number: bet \$1, get \$35 back



Roulette

- “Game tree” only has depth 2
 - Place a bet
 - Observe the roulette wheel

No bet

Bet: Red, \$5

Chance node



Probabilities



$18/38$

Red

+10

$20/38$

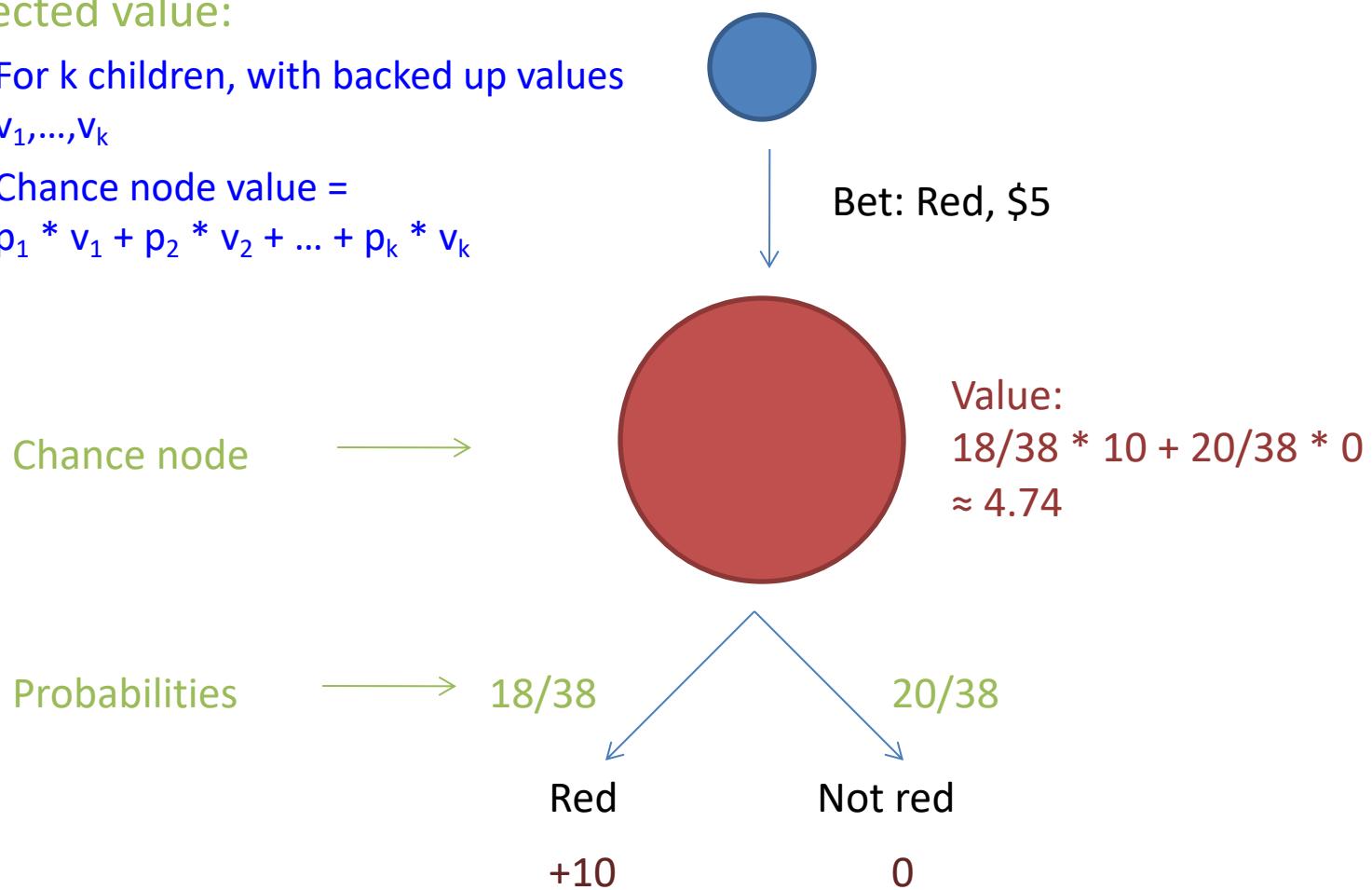
Not red

0

Chance Node Backup

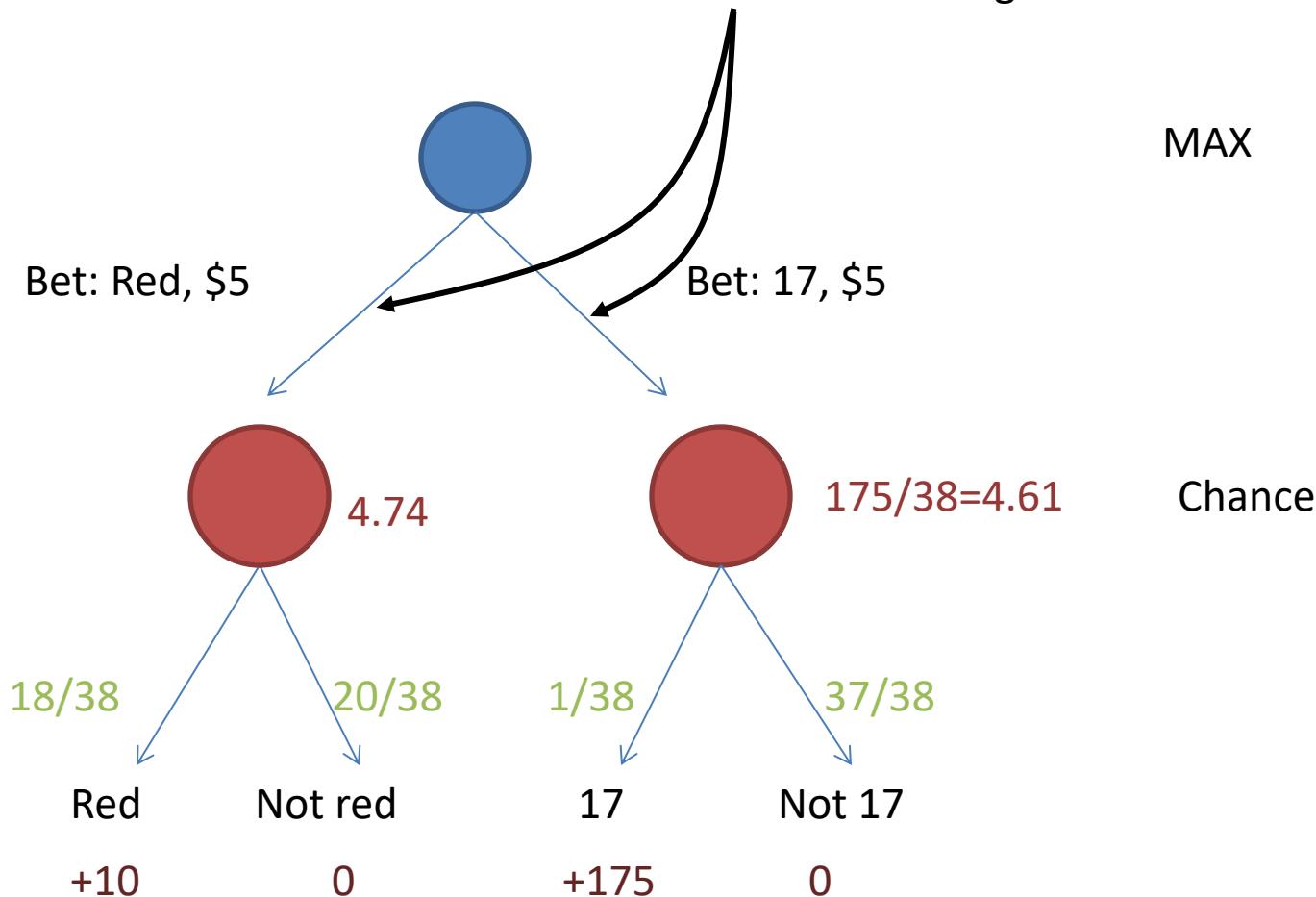
- Expected value:

- For k children, with backed up values v_1, \dots, v_k
- Chance node value = $p_1 * v_1 + p_2 * v_2 + \dots + p_k * v_k$



MAX/Chance Nodes

Max should pick the action leading to the node with the highest value



Adversarial Games of Chance

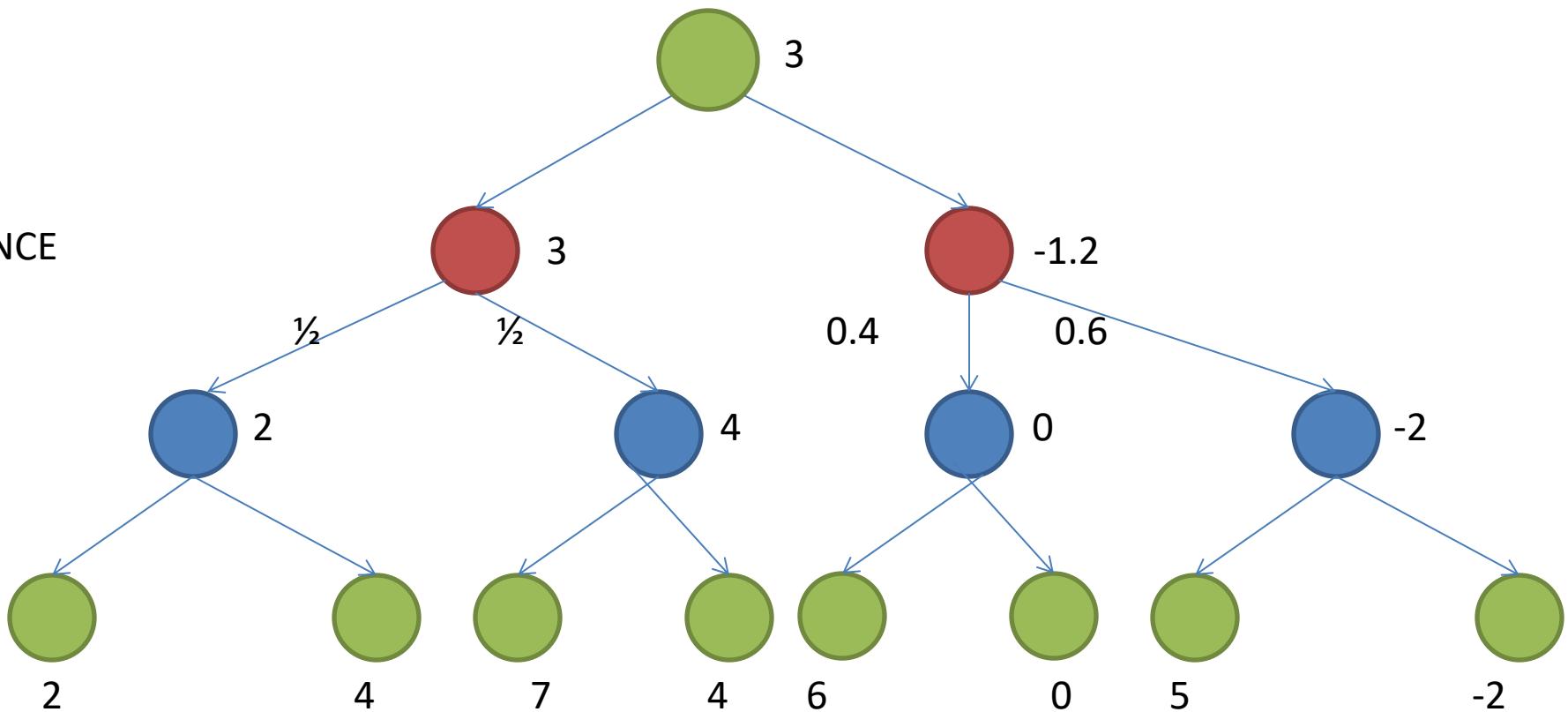
- E.g., Backgammon
- MAX nodes, MIN nodes, CHANCE nodes
- Expectiminimax search
- Backup step:
 - MAX = maximum of children
 - CHANCE = average of children
 - MIN = minimum of children
 - CHANCE = average of children
- 4 levels of the game tree separate each of MAX's turns!
- Evaluation function? Pruning?

Another example

MAX

CHANCE

MIN



Card Games

- Blackjack (6-deck), video poker: similar to coin-flipping game
- But in many card games, need to keep track of history of dealt cards in state because it affects future probabilities
 - One-deck blackjack
 - Bridge
 - Poker

Partially Observable Games

- Partial observability
 - Don't see entire state (e.g., other players' hands)
 - “Fog of war”
- Examples:
 - Kriegspiel (see R&N)
 - Battleship



Next time

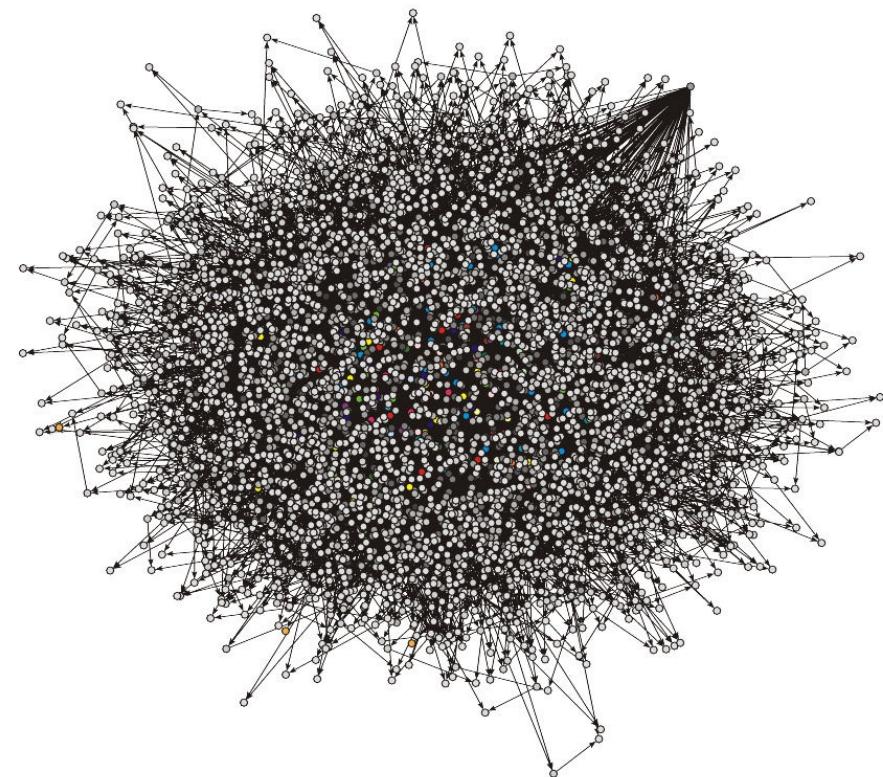
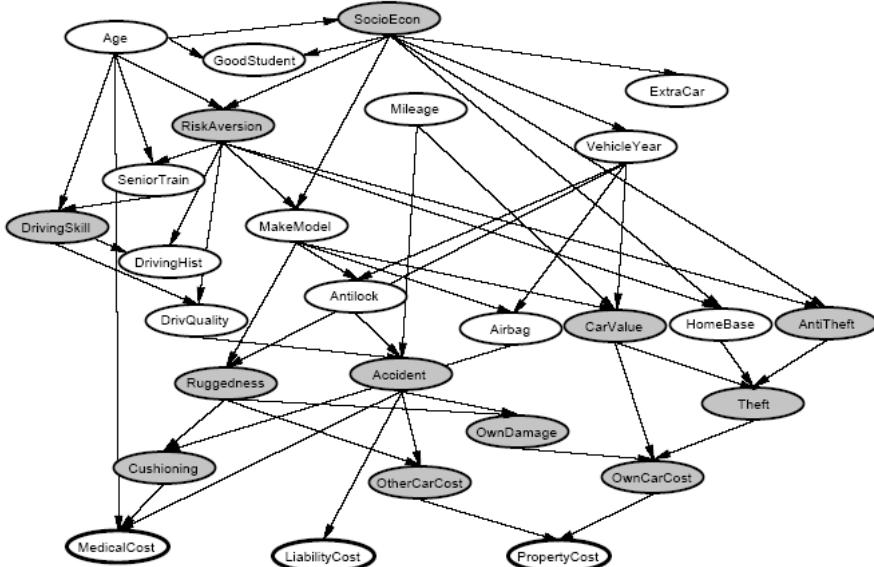
- Sampling and MCMC

MCMC

Announcements

- A2 coming soon (sign up for teams)
- Midterm exam 10/26 6:30pm-7:45pm
 - Mostly multiple choice
 - Review questions posted
 - Online using Canvas
- Final exam
 - Friday 12/16 7:40pm-9:40pm
 - Online using Canvas

What about complicated Bayes Nets?



Making inference tractable

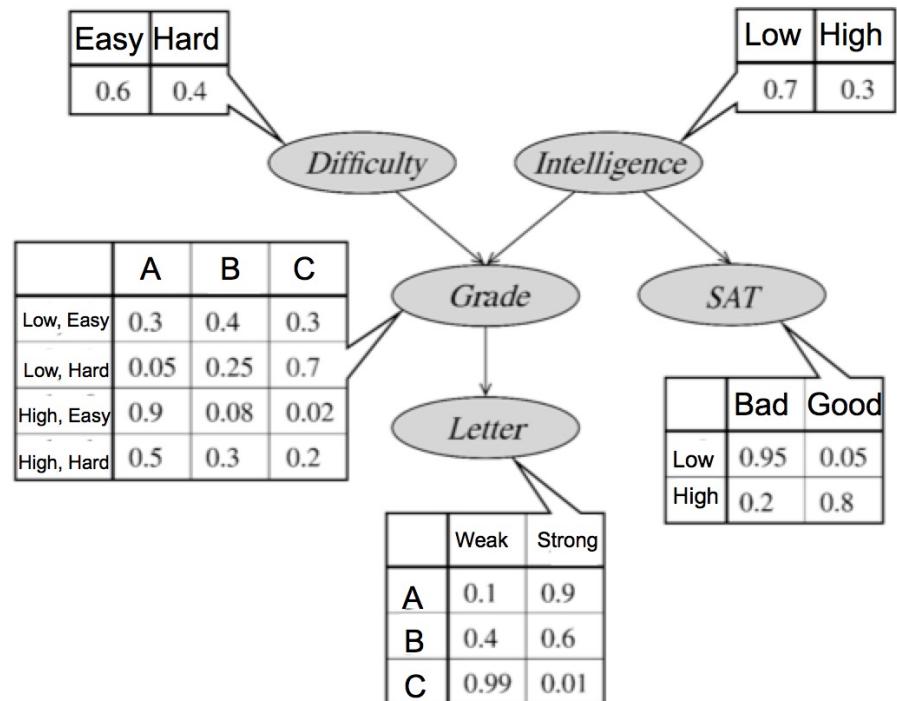
- In practice, making inference tractable is a key challenge in applying graphical models to applications
- Typically, the options are:
 - Exact inference with arbitrary conditional probability distributions, but with a simplified graphical structure
 - Exact inference with arbitrary graphical structure, but restricted conditional probability distributions
 - Arbitrary graphical structure and arbitrary conditional probability distributions, but approximate inference

Particle-based techniques

- A *particle* is an assignment of values to (some) variables of a graphical model
- Basic idea: Sets of particles can be used to approximate a distribution
 - E.g. Many samples from a distribution can be a good representation of original distribution

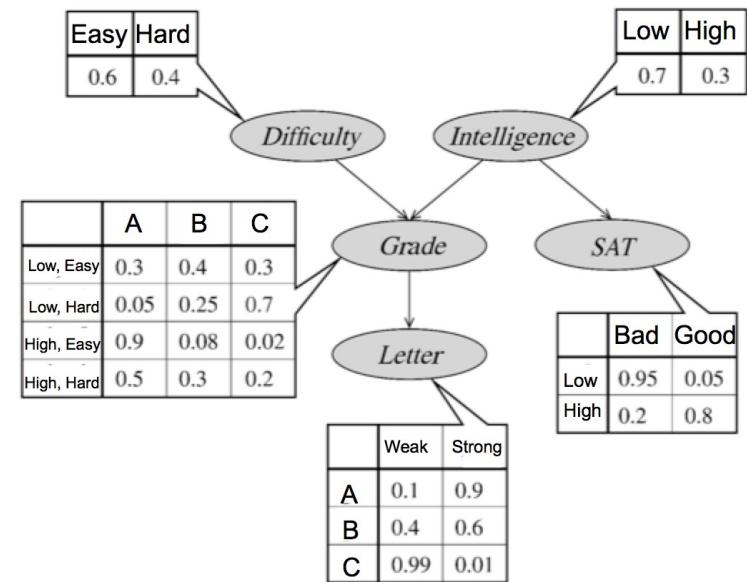
Forward sampling

- We can sample particles using the simple *Forward sampling* algorithm
 - Sample values from priors at root nodes
 - For a node X for which values have been sampled for all parents, sample from $P(X | \text{Parents}(X))$



Some sampled particles

Hard	Low	C	Bad	Strong
Hard	High	A	Good	Strong
Easy	High	A	Good	Strong
Hard	Low	C	Bad	Strong
Easy	Low	C	Bad	Strong
Easy	Low	C	Bad	Strong
Easy	Low	B	Bad	Strong
Hard	High	A	Bad	Strong
Easy	Low	B	Bad	Weak
Easy	Low	A	Bad	Strong



- What is $P(G=A)$?
- What is $P(L=Weak)$?
- What is $P(S=Good \mid Grade = A)$?
- What is $P(L=Strong \mid Grade = C)$?

Markov Chain Monte Carlo (MCMC)

- General class of techniques that produce a *sequence* of samples
- Main idea: Save effort by using information from *past samples* in producing *future samples*
 - Initial samples are from a *proposal (approximate) distribution* Q
 - Subsequent sampling is biased towards P
 - Eventually the samples are drawn from a distribution that is closer and closer to P

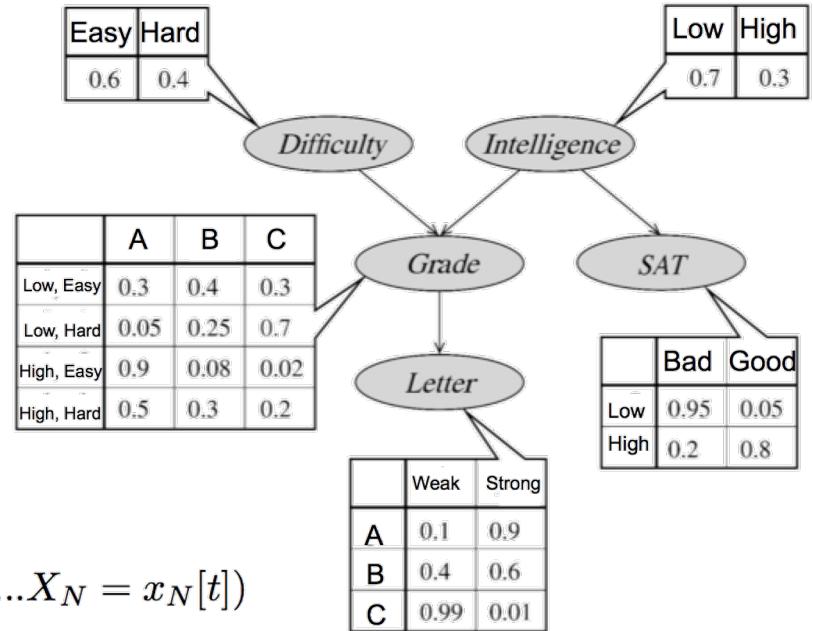
Special case of MCMC: Gibbs sampling

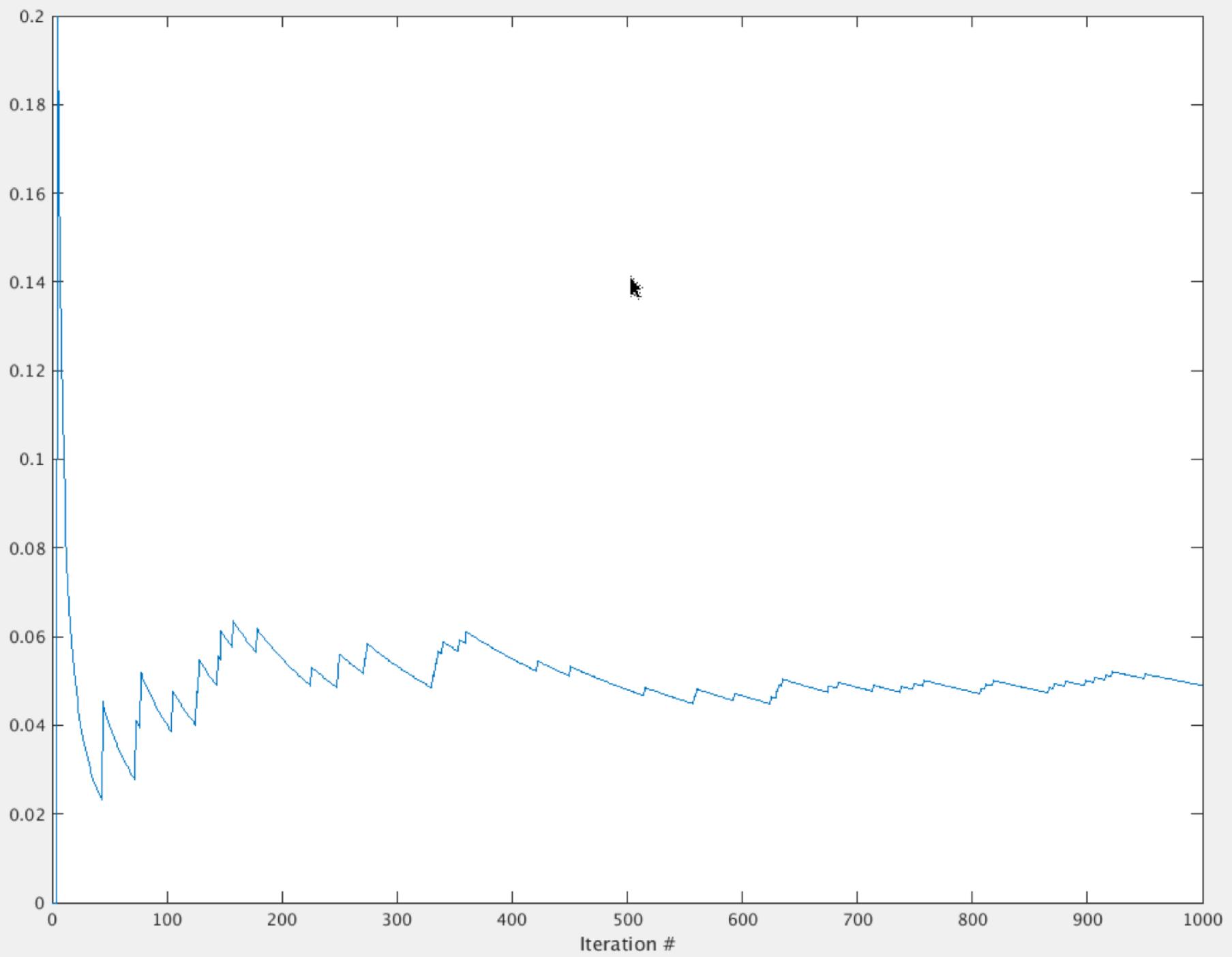
- Generate initial sample $x[0]$
- For each sample $t=1\dots T$
 - Let $x[t] = x[t-1]$
 - For each unobserved variable X_i ,
 - Sample a value for X_i given values for all other variables in $x[t]$; i.e. sample from:

$$P(X_i | X_1 = x_1[t], \dots, X_{i-1} = x_{i-1}[t], X_{i+1} = x_{i+1}[t], \dots, X_N = x_N[t]) \\ = P(X_i | \mathbf{X}_{-i} = \mathbf{x}_{-i}[t])$$

where $\mathbf{X}_{-i} = \mathbf{X} - \{X_i\}$

- Put this sampled value in $x_i[t]$

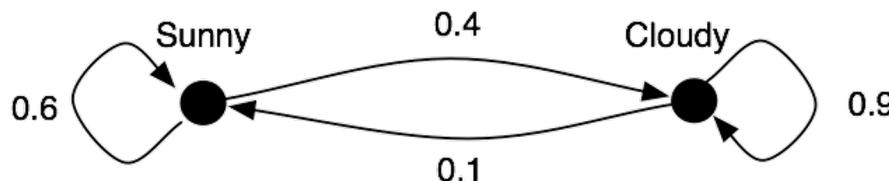




Properties of Gibbs sampling

- Gibbs can be applied to any Bayes networks
- Gibbs sampling will converge to sampling from the correct distribution, *eventually*
 - But may require a long time to converge
 - Why does this happen?

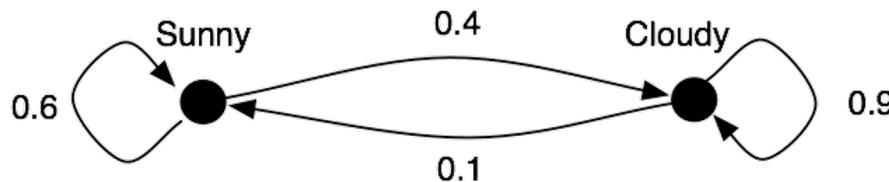
Markov chains



- Suppose there's an 80% chance of sun on day 0.
What is the probability of sun on day 3?

$$\begin{aligned} P(Q_3 = \text{Sunny}) &= P(Q_3 = \text{Sunny} | Q_2 = \text{Sunny})P(Q_2 = \text{Sunny}) + P(Q_3 = \text{Sunny} | Q_2 = \text{Cloudy})P(Q_2 = \text{Cloudy}) \\ &= 0.6P(Q_2 = \text{Sunny}) + 0.1P(Q_2 = \text{Cloudy}) \\ &= 0.6(0.6P(Q_1 = \text{Sunny}) + 0.1P(Q_1 = \text{Cloudy})) + 0.1(0.4P(Q_1 = \text{Sunny}) + 0.9P(Q_1 = \text{Cloudy})) \\ &= 0.6(0.6(0.6P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.1(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \\ &\quad + 0.1(0.4(0.4P(Q_0 = \text{Sunny}) + 0.1P(Q_0 = \text{Cloudy})) + 0.9(0.4P(Q_0 = \text{Sunny}) + 0.9P(Q_0 = \text{Cloudy}))) \\ &= 0.6(0.6(0.6(0.8) + 0.1(0.2)) + 0.1(0.4(0.8) + 0.9(0.2))) \\ &\quad + 0.1(0.4(0.6(0.8) + 0.1(0.2)) + 0.9(0.4(0.8) + 0.9(0.2))) \\ &= 0.275 \end{aligned}$$

Markov chains



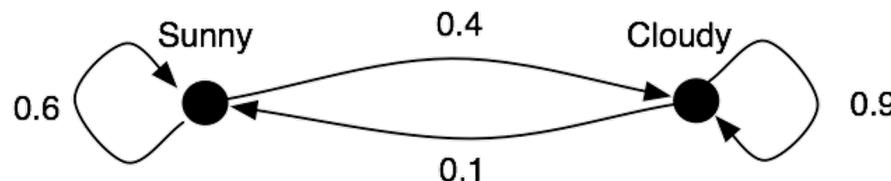
- Suppose there's an 80% chance of sun on day 0.
What is the probability of sun on day 3?

$$B = \begin{bmatrix} 0.6 & 0.4 \\ 0.1 & 0.9 \end{bmatrix} \quad w = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}$$

$$(B^T)^3 w = \begin{bmatrix} 0.275 \\ 0.725 \end{bmatrix}$$

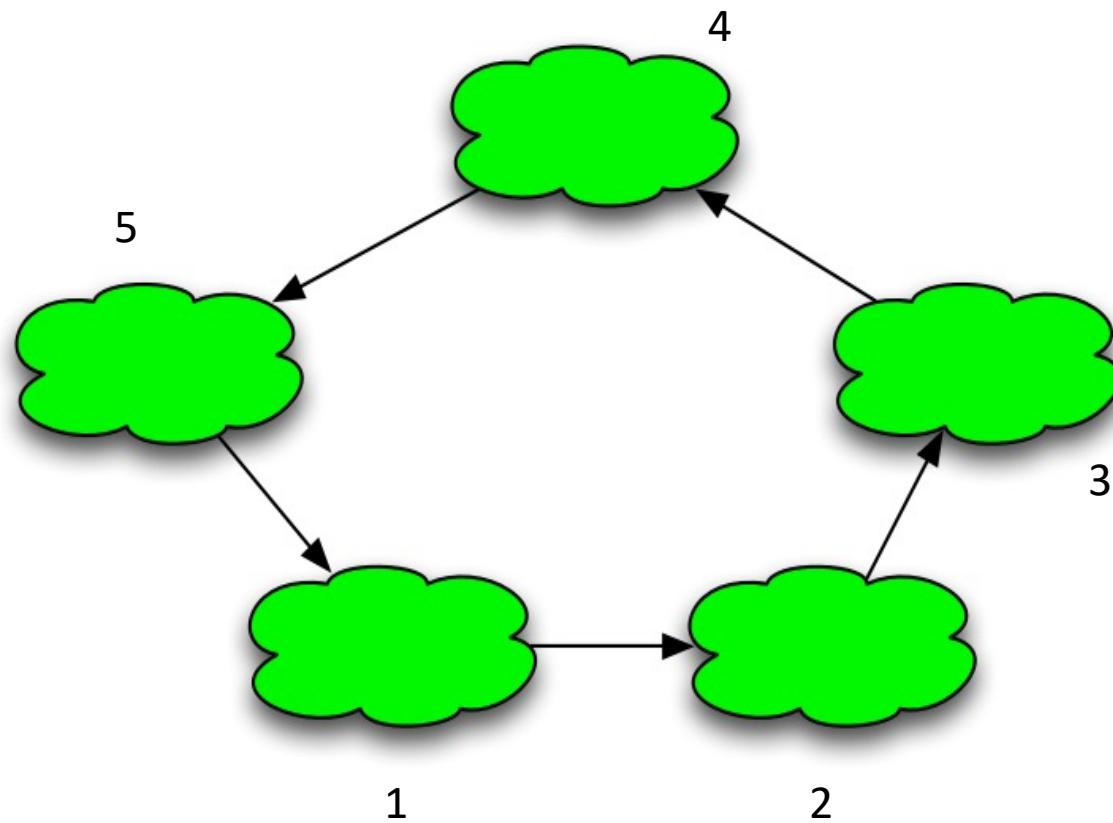
$\xrightarrow{\hspace{1cm}} P(X_3 = \text{sun})$
 $\xrightarrow{\hspace{1cm}} P(X_3 = \text{cloudy})$

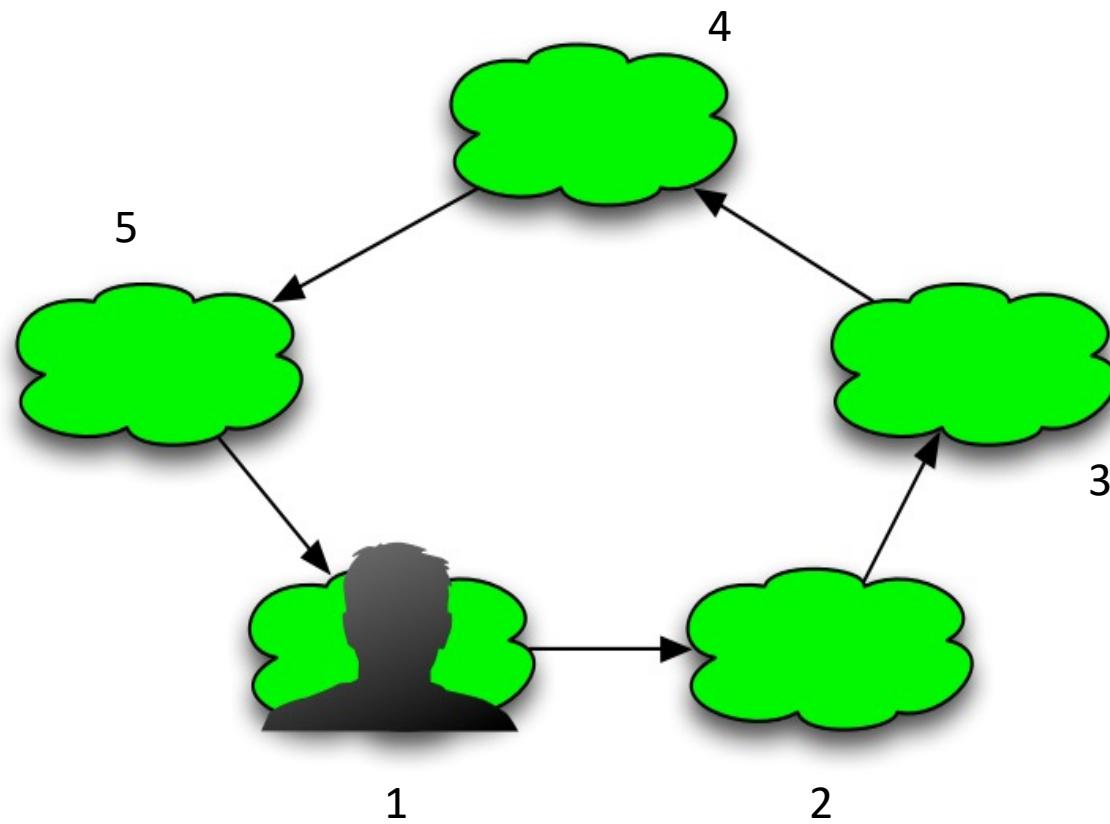
Stationary distributions



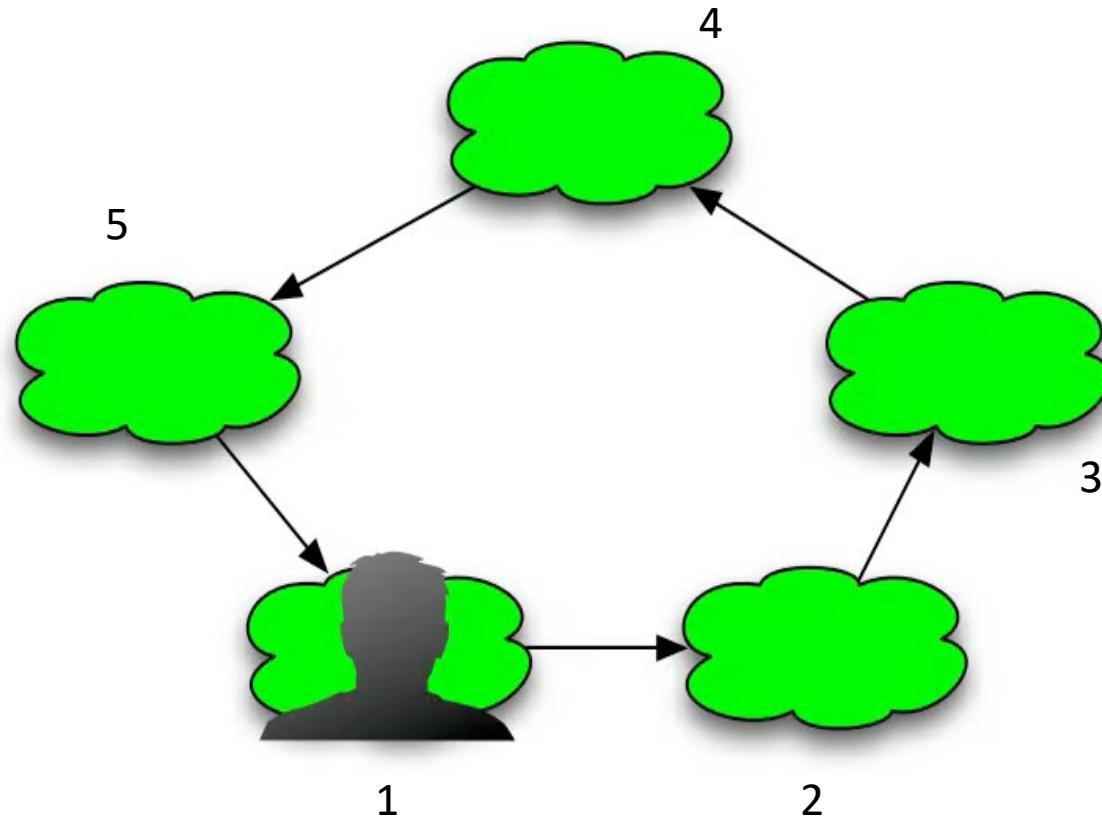
- For an *ergodic* chain, a *stationary distribution* exists
 - ergodic: all states are recurrent and aperiodic
 - stationary distribution: for large t , the probability of being in state i at time t depends *only* on the transition probabilities
 - the stationary distribution π is the vector satisfying

$$B^T \pi = \pi$$



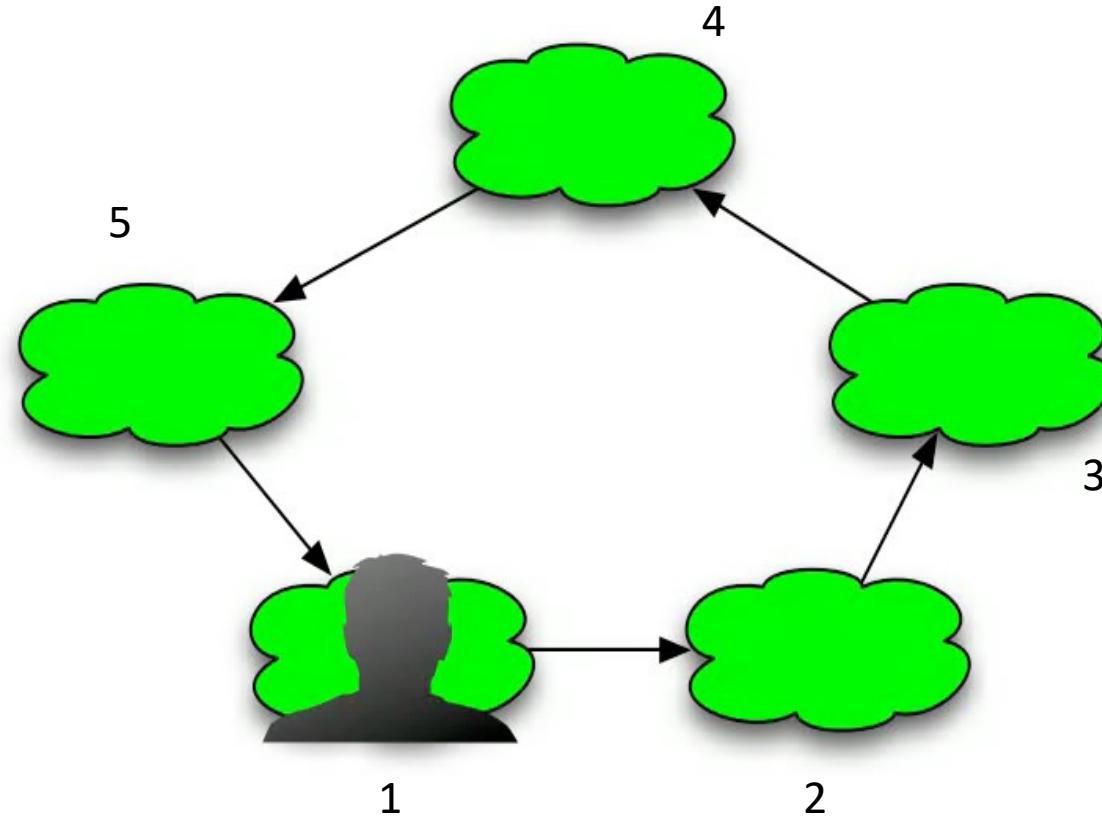


Q: At any moment in time, what's the probability that the frog is on pad 1?



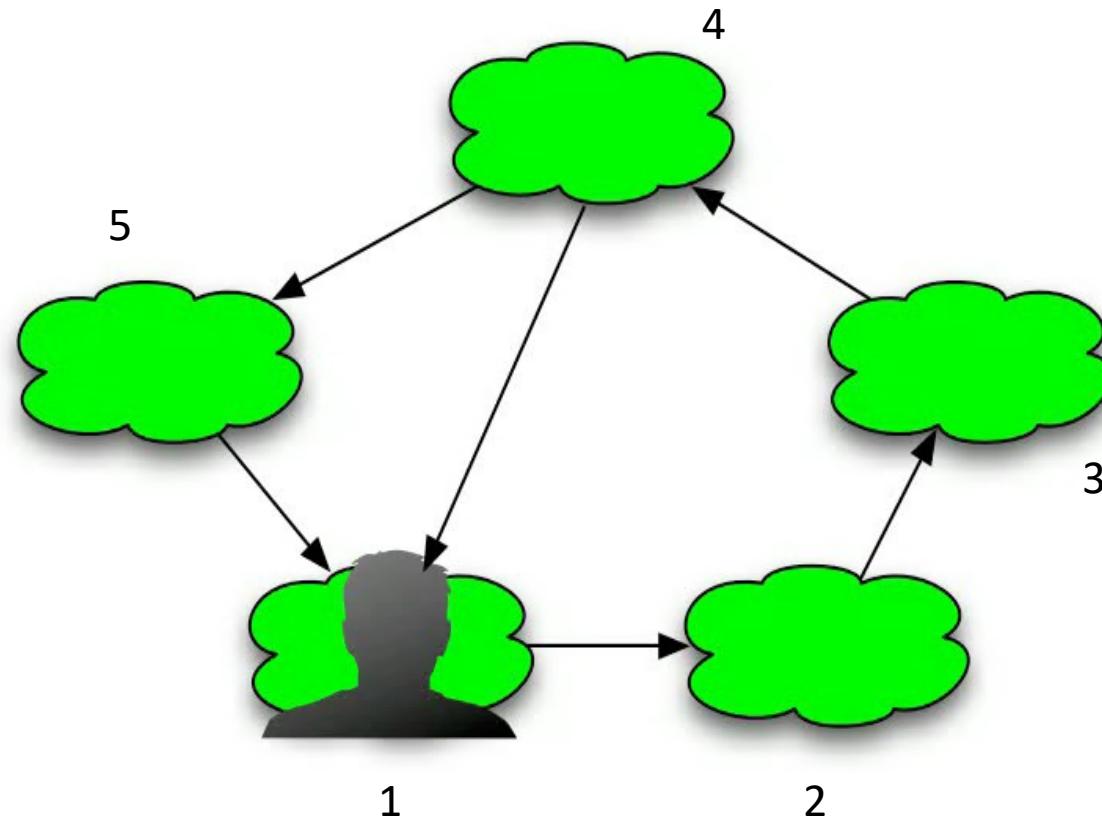
Q: At any moment in time, what's the probability that the frog is on pad 1?

A: $P(\text{Pad}=1) = 1/5$. Same for 2, 3, 4, 5.



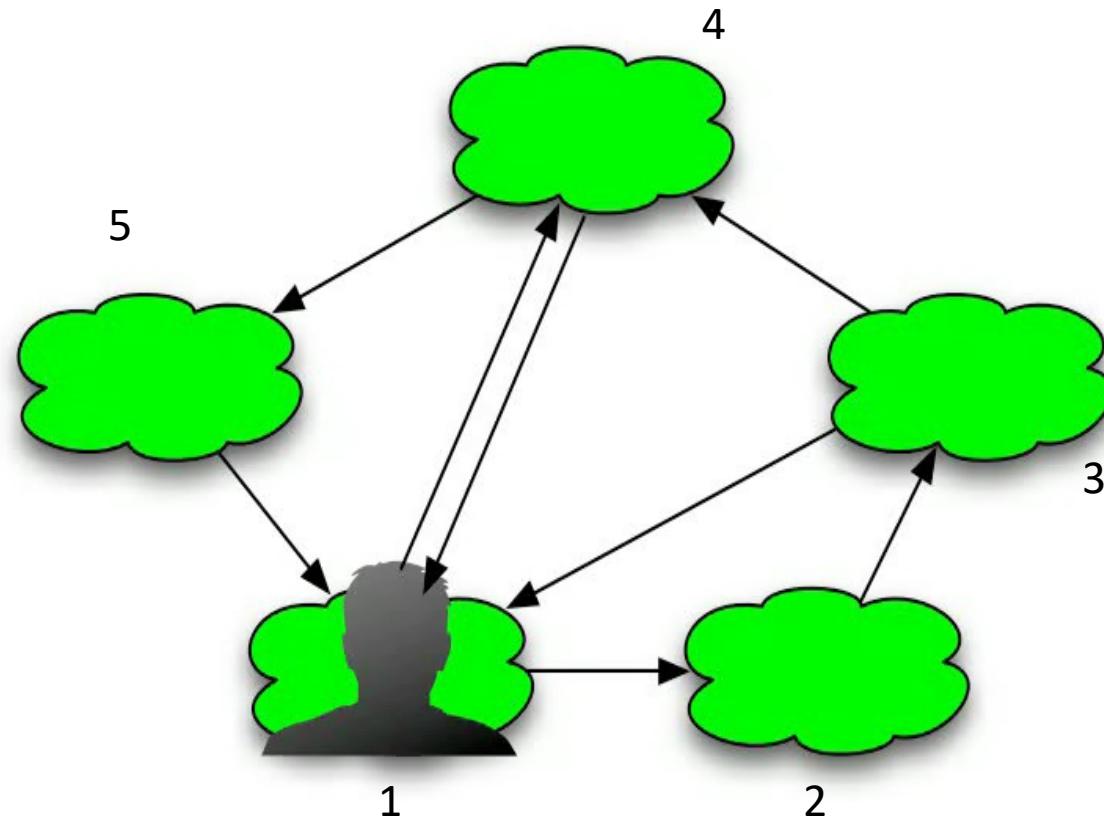
Q: At any moment in time, what's the probability that the frog is on pad 1?

A: $P(\text{Pad}=1) = ?$

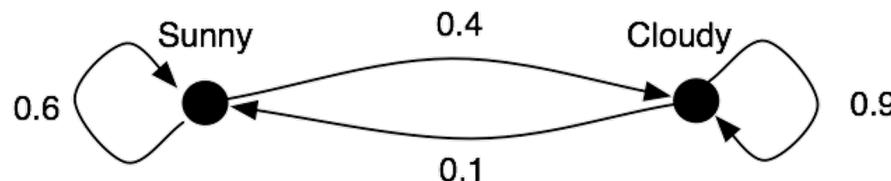


Q: At any moment in time, what's the probability that the frog is on pad 1?

A: $P(\text{Pad}=1) = ?$



Stationary distributions

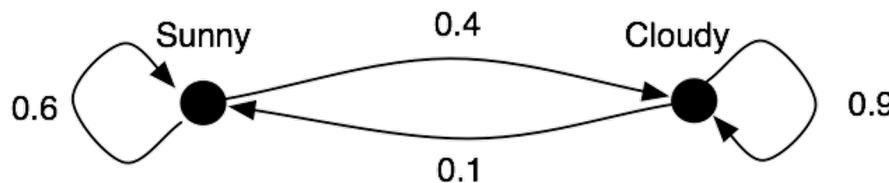


- For an *ergodic* chain, a *stationary distribution* exists
 - ergodic: all states are recurrent and aperiodic
 - stationary distribution: for large t , the probability of being in state i at time t depends *only* on the transition probabilities
 - the stationary distribution π is the vector satisfying

$$B^T \pi = \pi$$

How do we compute π ?

Stationary distribution of Markov chain



- What is the stationary distribution of this chain?

```
>> % e.g. in Matlab:
```

```
>> [v d]=eigs([0.6 0.4; 0.1 0.9]',1)
```

```
v =  
-0.24253562503633  
-0.97014250014533
```

```
d =  
1
```

```
>> v/sum(v)
```

```
ans =
```

```
0.200000000000000  
0.800000000000000
```

$$\pi = \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix}$$



Monte Carlo Methods

- Monte Carlo techniques involve random sampling for applications like numerical integration and optimization
 - Originally invented for nuclear physics, now widely used across a wide range of domains
- Useful when it's difficult to measure some quantity but it's easy to generate samples from a distribution related to that quantity

Back to Markov Chain Monte Carlo (MCMC)

- Recall we want to estimate some distribution $P(X)$
 - But inference is too hard to compute it directly
- Basic idea: Construct a Markov Chain whose *stationary distribution* is exactly $P(X)$
 - Then take random walks on the Markov Chain
 - If we walk long enough, sampling from the Markov Chain is exactly equivalent to sampling from $P(X)$

Next class

- More MCMC and statistical learning

MCMC

- Each state in the Markov Chain is one possible assignment of values to all variables
- In Gibb's Sampling, we chose the transition probabilities such that:
 - a stationary distribution exists and,
 - the stationary distribution is exactly the posterior probability distribution we want to sample from

Why does Gibbs work?

$$\mathcal{T}_i((\mathbf{x}_{-i}, x_i) \rightarrow (\mathbf{x}_{-i}, x'_i)) = P(X_i = x'_i | \mathbf{X}_{-i} = \mathbf{x}_{-i})$$

- To prove that Gibbs sampling works and is practical, we need to show that:
 1. A stationary distribution for this Markov Chain exists (under some assumptions)
 2. The stationary distribution of the Markov Chain is the posterior distribution of the Markov network
 3. It's possible to sample from $P(X_i = x'_i | \mathbf{X}_{-i} = \mathbf{x}_{-i})$ efficiently

MCMC in practice

- Might take a long time for samples to be close to the stationary distribution – to “mix”
 - The “burn-in” or “warm-up” time
- The burn-in time depends on the structure of the chain and the transition probabilities
 - When might the burn-in time be particularly long?
- It’s possible to compute bounds on the burn-in time, by spectral analysis of the transition matrix
 - I.e. computing eigenvalues and eigenvectors of the Markov Chains’ transition matrix
 - But this is completely unhelpful in practice – why?

Practical solutions

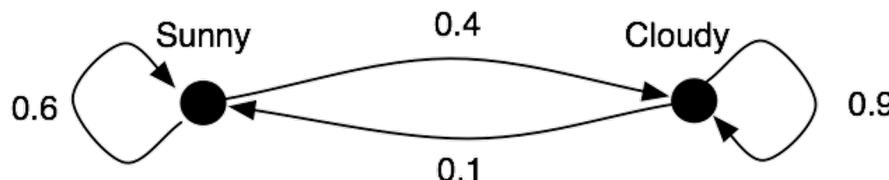
- Construct a small number of identical Markov chains
 - Take random walks on each of the chains for a large number of time steps, starting from different initial states
 - Run the chains until the samples seem to be coming from the same distribution across all (or most) of the chains
 - Now use each of the chains to generate (estimated) samples from the posterior distribution

MCMC and Statistical learning

Announcements

- A2 posted, sign up and create your teams
- Midterm exam 10/26 6:30pm-7:45pm
 - Mostly multiple choice
 - Review questions posted
 - Online using Canvas
- Final exam
 - Friday 12/16 7:40pm-9:40pm
 - Online using Canvas
- Don't forget the quiz

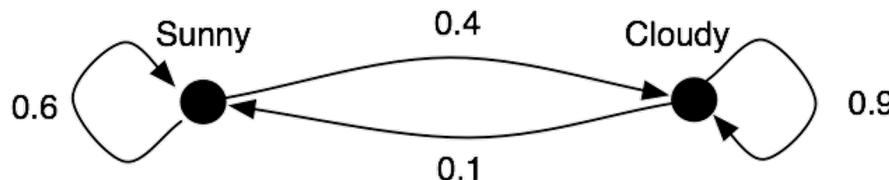
Markov chains



- Suppose there's an 80% chance of sun on day 0.
What is the probability of sun on day 3?

$$\begin{aligned} P(Q_3 = \text{☀}) &= P(Q_3 = \text{☀}|Q_2 = \text{☀})P(Q_2 = \text{☀}) + P(Q_3 = \text{☀}|Q_2 = \text{☁})P(Q_2 = \text{☁}) \\ &= 0.6P(Q_2 = \text{☀}) + 0.1P(Q_2 = \text{☁}) \\ &= 0.6(0.6P(Q_1 = \text{☀}) + 0.1P(Q_1 = \text{☁})) + 0.1(0.4P(Q_1 = \text{☀}) + 0.9P(Q_1 = \text{☁})) \\ &= 0.6(0.6(0.6P(Q_0 = \text{☀}) + 0.1P(Q_0 = \text{☁})) + 0.1(0.4P(Q_0 = \text{☀}) + 0.9P(Q_0 = \text{☁}))) \\ &\quad + 0.1(0.4(0.4P(Q_0 = \text{☀}) + 0.1P(Q_0 = \text{☁})) + 0.9(0.4P(Q_0 = \text{☀}) + 0.9P(Q_0 = \text{☁}))) \\ &= 0.6(0.6(0.6(0.8) + 0.1(0.2)) + 0.1(0.4(0.8) + 0.9(0.2))) \\ &\quad + 0.1(0.4(0.6(0.8) + 0.1(0.2)) + 0.9(0.4(0.8) + 0.9(0.2))) \\ &= 0.275 \end{aligned}$$

Markov chains



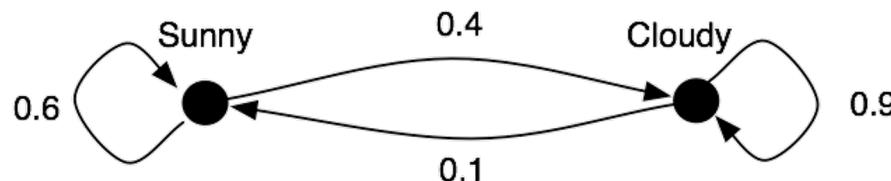
- Suppose there's an 80% chance of sun on day 0.
What is the probability of sun on day 3?

$$B = \begin{bmatrix} 0.6 & 0.4 \\ 0.1 & 0.9 \end{bmatrix} \quad w = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}$$

$$(B^T)^3 w = \begin{bmatrix} 0.275 \\ 0.725 \end{bmatrix}$$

$\xrightarrow{\hspace{1cm}}$ $P(X_3 = \text{sun})$
 $\xrightarrow{\hspace{1cm}}$ $P(X_3 = \text{cloudy})$

Stationary distributions

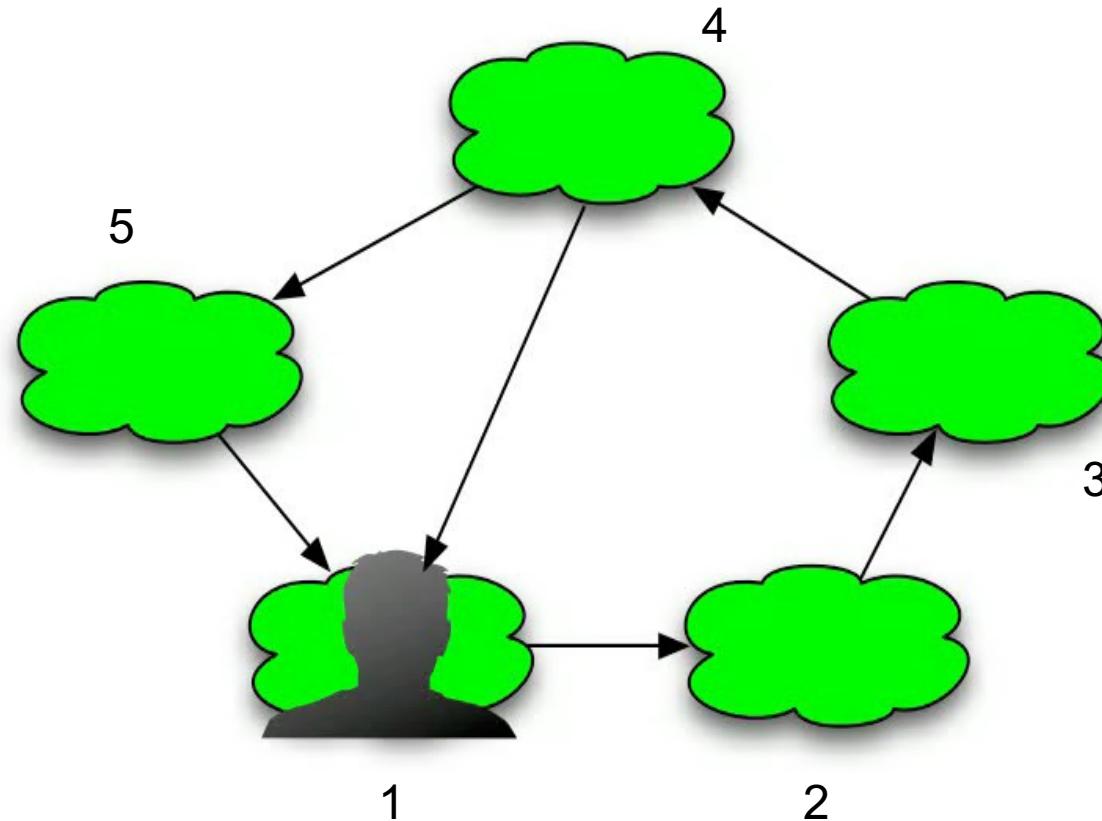


- For an *ergodic* chain, a *stationary distribution* exists
 - ergodic: all states are recurrent and aperiodic
 - stationary distribution: for large t , the probability of being in state i at time t depends *only* on the transition probabilities
 - the stationary distribution π is the vector satisfying

$$B^T \pi = \pi$$

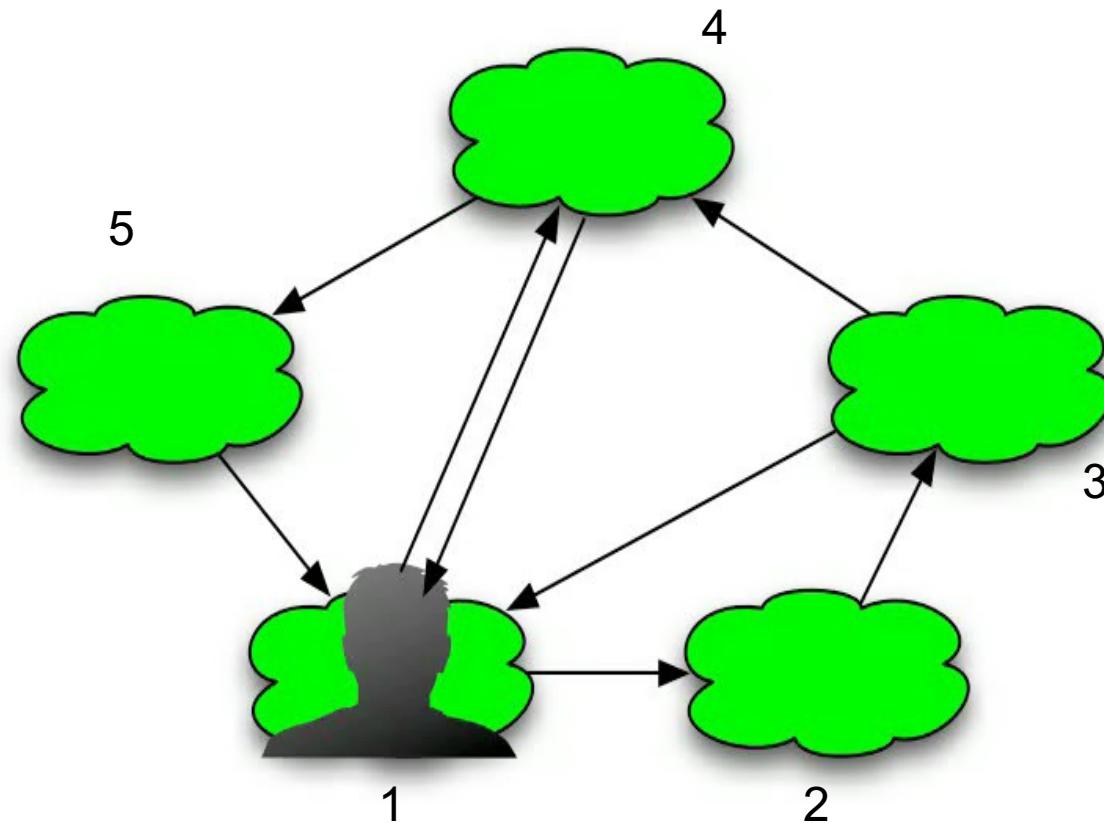
Q: At any moment in time, what's the probability that the frog is on pad 1?

A: $P(\text{Pad}=1) = ?$

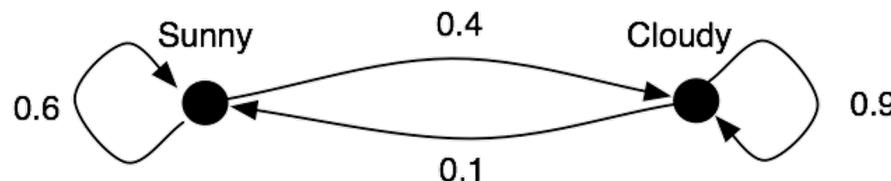


Q: At any moment in time, what's the probability that the frog is on pad 1?

A: $P(\text{Pad}=1) = ?$



Stationary distributions

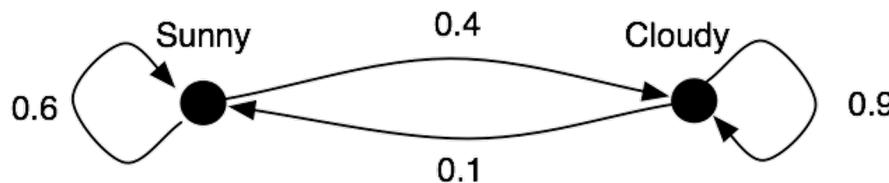


- For an *ergodic* chain, a *stationary distribution* exists
 - ergodic: all states are recurrent and aperiodic
 - stationary distribution: for large t , the probability of being in state i at time t depends *only* on the transition probabilities
 - the stationary distribution π is the vector satisfying

$$B^T \pi = \pi$$

How do we compute π ?

Stationary distribution of Markov chain



- What is the stationary distribution of this chain?

```
>> % e.g. in Matlab:
```

```
>> [v d]=eigs([0.6 0.4; 0.1 0.9]',1)
```

```
v =  
-0.24253562503633  
-0.97014250014533
```

```
d =  
1
```

```
>> v/sum(v)
```

```
ans =
```

```
0.200000000000000  
0.800000000000000
```

$$\pi = \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix}$$



Back to Markov Chain Monte Carlo (MCMC)

- Recall we want to estimate some distribution $P(X)$
 - But inference is too hard to compute it directly
- Basic idea: Construct a Markov Chain whose *stationary distribution* is exactly $P(X)$
 - Then take random walks on the Markov Chain
 - If we walk long enough, sampling from the Markov Chain is exactly equivalent to sampling from $P(X)$

MCMC in practice

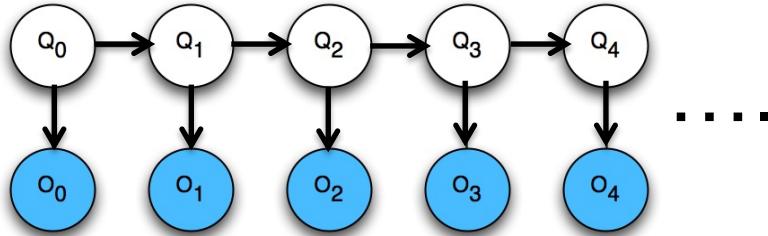
- Might take a long time for samples to be close to the stationary distribution – to “mix”
 - The “burn-in” or “warm-up” time
- The burn-in time depends on the structure of the chain and the transition probabilities
 - When might the burn-in time be particularly long?
- It’s possible to compute bounds on the burn-in time, by spectral analysis of the transition matrix
 - I.e. computing eigenvalues and eigenvectors of the Markov Chains’ transition matrix
 - But this is completely unhelpful in practice – why?

Practical solutions

- Construct a small number of identical Markov chains
 - Take random walks on each of the chains for a large number of time steps, starting from different initial states
 - Run the chains until the samples seem to be coming from the same distribution across all (or most) of the chains
 - Now use each of the chains to generate (estimated) samples from the posterior distribution

Statistical learning

Hidden Markov Models (HMMs)



- More formally, an HMM consists of:
 - Transition probabilities

$$p_{ij} = P(Q_{t+1} = j | Q_t = i)$$

- Initial state distribution

$$w_i = P(Q_0 = i)$$

- Emission probabilities

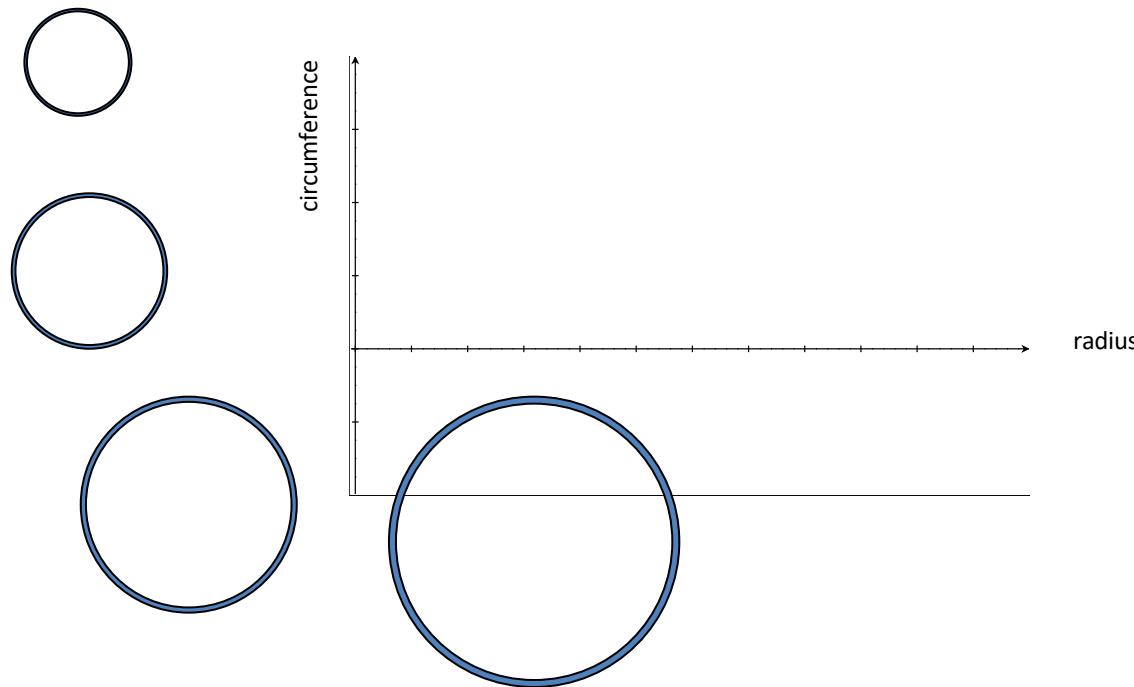
$$e_i(a) = P(O_t = a | Q_t = i)$$

Learning in General

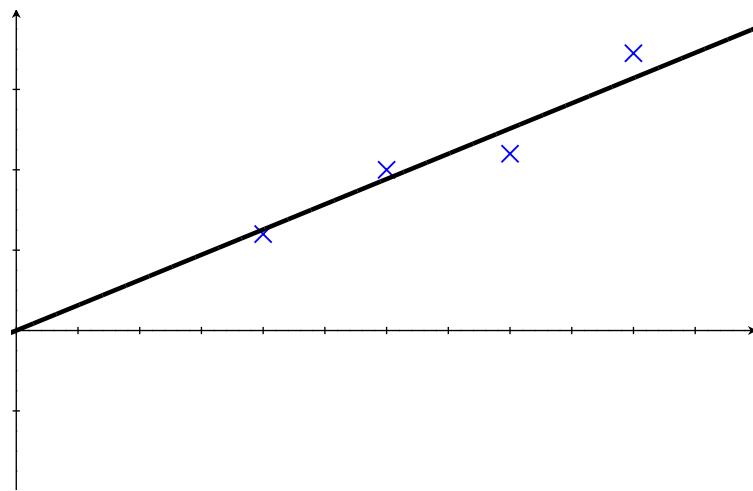
- Agent has made observations (**data**)
- Now must make sense of it (model **hypotheses**)
- *Why?*
 - Hypotheses alone may be important (e.g., in basic science)
 - For inference (e.g., forecasting)
 - To take sensible actions (decision making)
- A basic component of economics, social and physical sciences, engineering, ...

An example

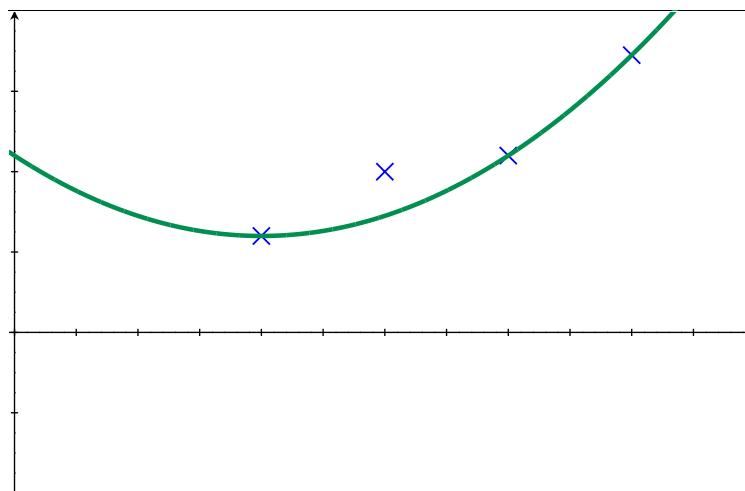
- Suppose we want to learn how to calculate the circumference of a circle from its radius.



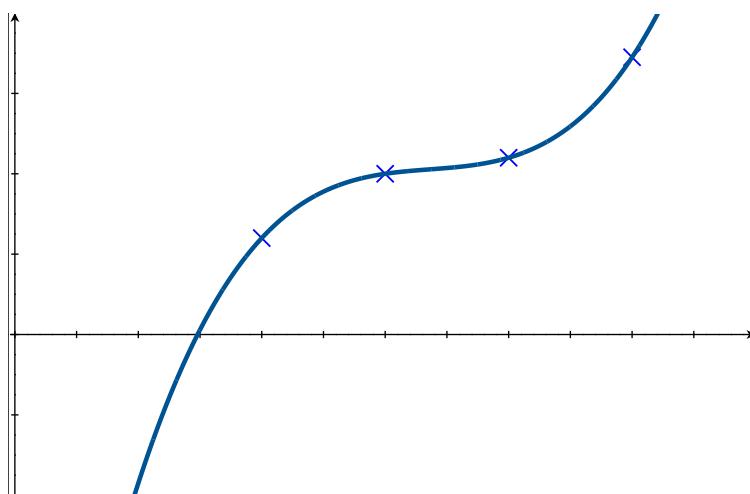
An example



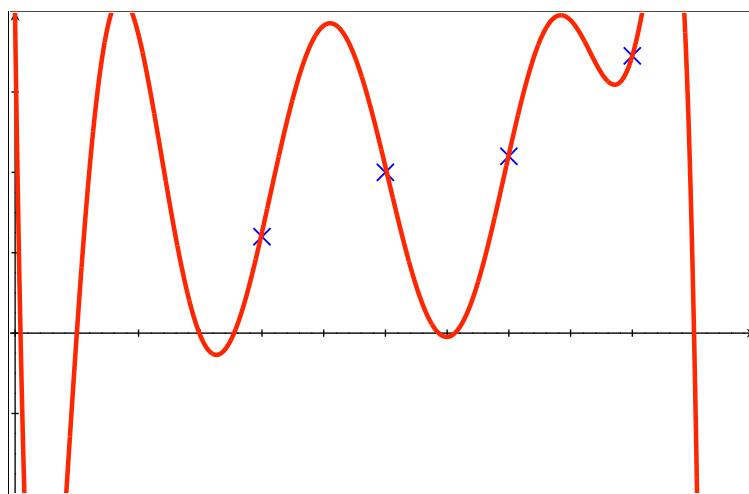
An example



An example

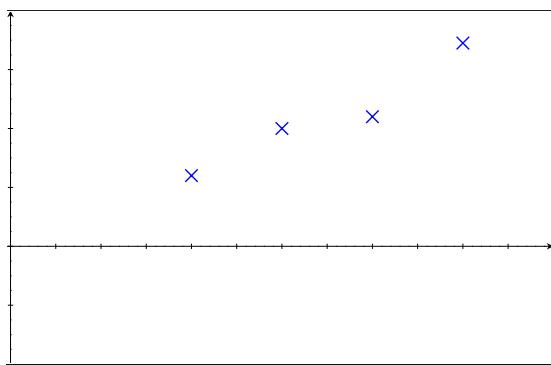


An example

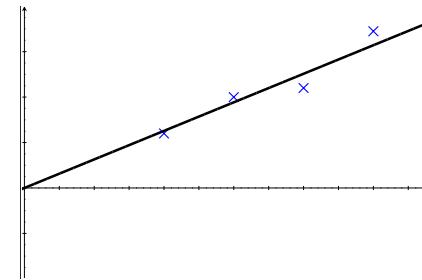


An example

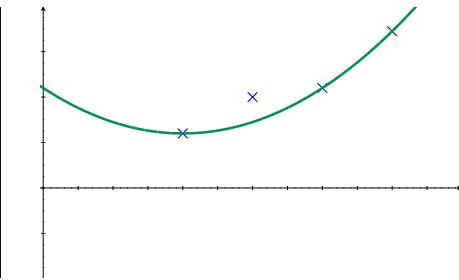
- Which is the best model?!



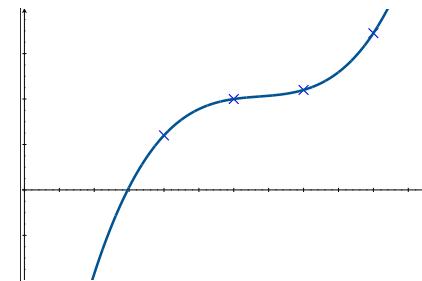
(a)



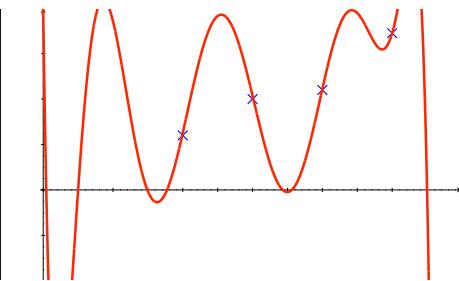
(b)



(c)



(d)



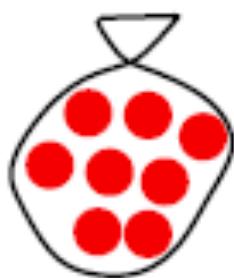
(e)

Machine Learning vs. Statistics

- Machine Learning \approx automated statistics
- Today
 - Statistical learning (aka Bayesian learning)
 - Maximum likelihood (ML) learning
 - Maximum a posteriori (MAP) learning
 - Learning Bayes Nets (R&N 20.1-3)
- Future lectures try to do more with even less data
 - Neural nets
 - Support vector machines
 - ...

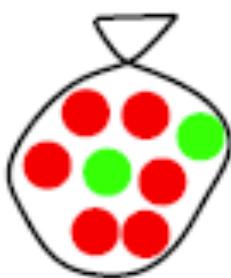
Candy Example

- Candy comes in 2 flavors, cherry and lime
- Manufacturer makes 5 types of bags:



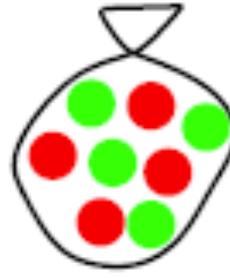
h1

C: 100%
L: 0%



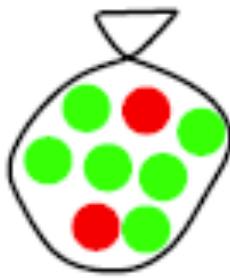
h2

C: 75%
L: 25%



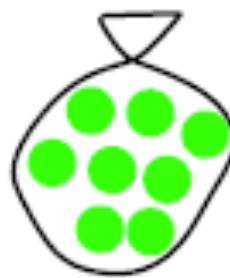
h3

C: 50%
L: 50%



h4

C: 25%
L: 75%



h5

C: 0%
L: 100%

- h1 and h5 are equally common. h2 is twice as common as h1, h4 is twice as common as h5, and h3 is twice as common as h2.
- Suppose we draw A row of five solid green circles.

Candy Example

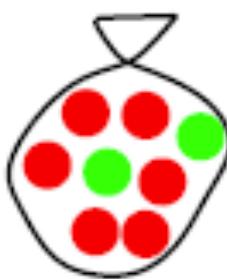
- Candy comes in 2 flavors, cherry and lime
- Manufacturer makes 5 types of bags:



h1

C: 100%

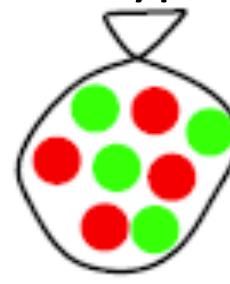
L: 0%



h2

C: 75%

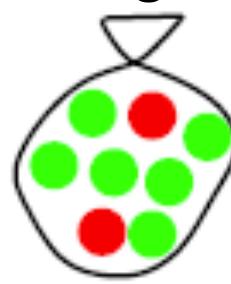
L: 25%



h3

C: 50%

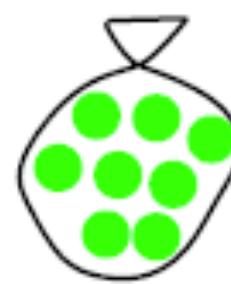
L: 50%



h4

C: 25%

L: 75%



h5

C: 0%

L: 100%

- h1 and h5 are equally common. h2 is twice as common as h1, h4 is twice as common as h5, and h3 is twice as common as h2.
- Suppose we draw $P(h1) = p(h5)$

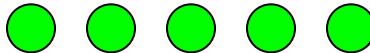
$$P(h2) = 2p(h1)$$

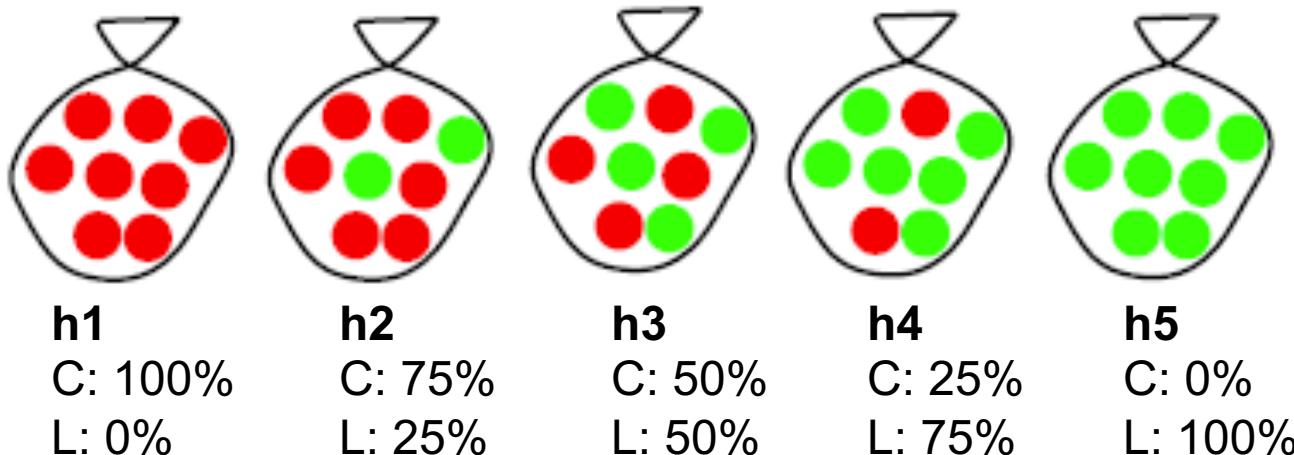
$$P(h4) = 2p(h5)$$

$$P(h3) = 2p(h2)$$

$$P(h1) + 2p(h1) + 4p(h1) + 2p(h1) + p(h1) = 1 \Rightarrow p(h1) = 0.1, p(h2) = 0.2\dots$$

Bayesian Learning

- Main idea: Compute the probability of **each** hypothesis, given the data
- Data d : 
- Hypotheses: h_1, \dots, h_5

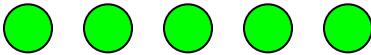


Bayesian Learning

- Main idea: Compute the probability of **each** hypothesis, given the data

$$P(h_i | d)$$

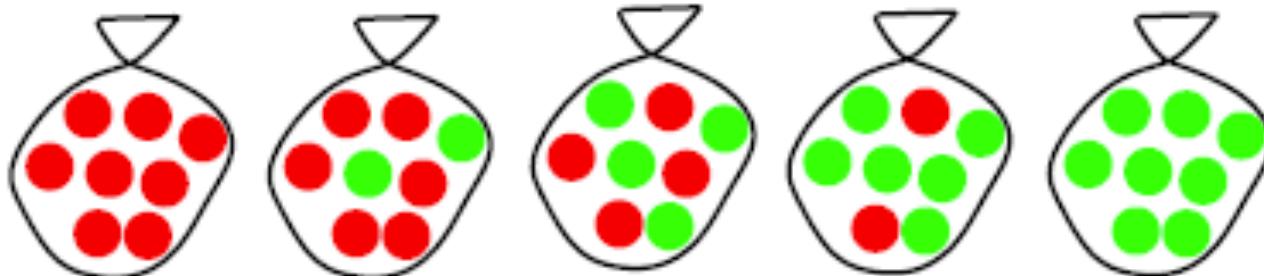
We want this...

- Data d : 

$$P(d | h_i)$$

But all we have
is this!

- Hypotheses: h_1, \dots, h_5



h_1

C: 100%
L: 0%

h_2

C: 75%
L: 25%

h_3

C: 50%
L: 50%

h_4

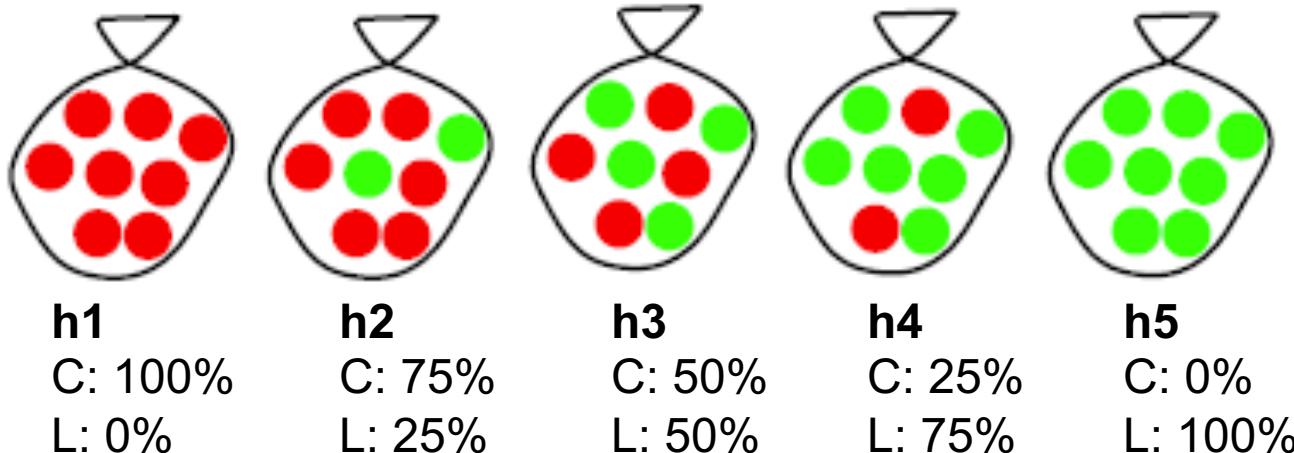
C: 25%
L: 75%

h_5

C: 0%
L: 100%

Using Bayes' Rule

- $P(h_i | d) = \alpha P(d | h_i) P(h_i)$ is the **posterior**
 - (Recall, $1/\alpha = P(d) = \sum_i P(d | h_i) P(h_i)$)
- $P(d | h_i)$ is the **likelihood**
- $P(h_i)$ is the **hypothesis prior**



Computing the Posterior

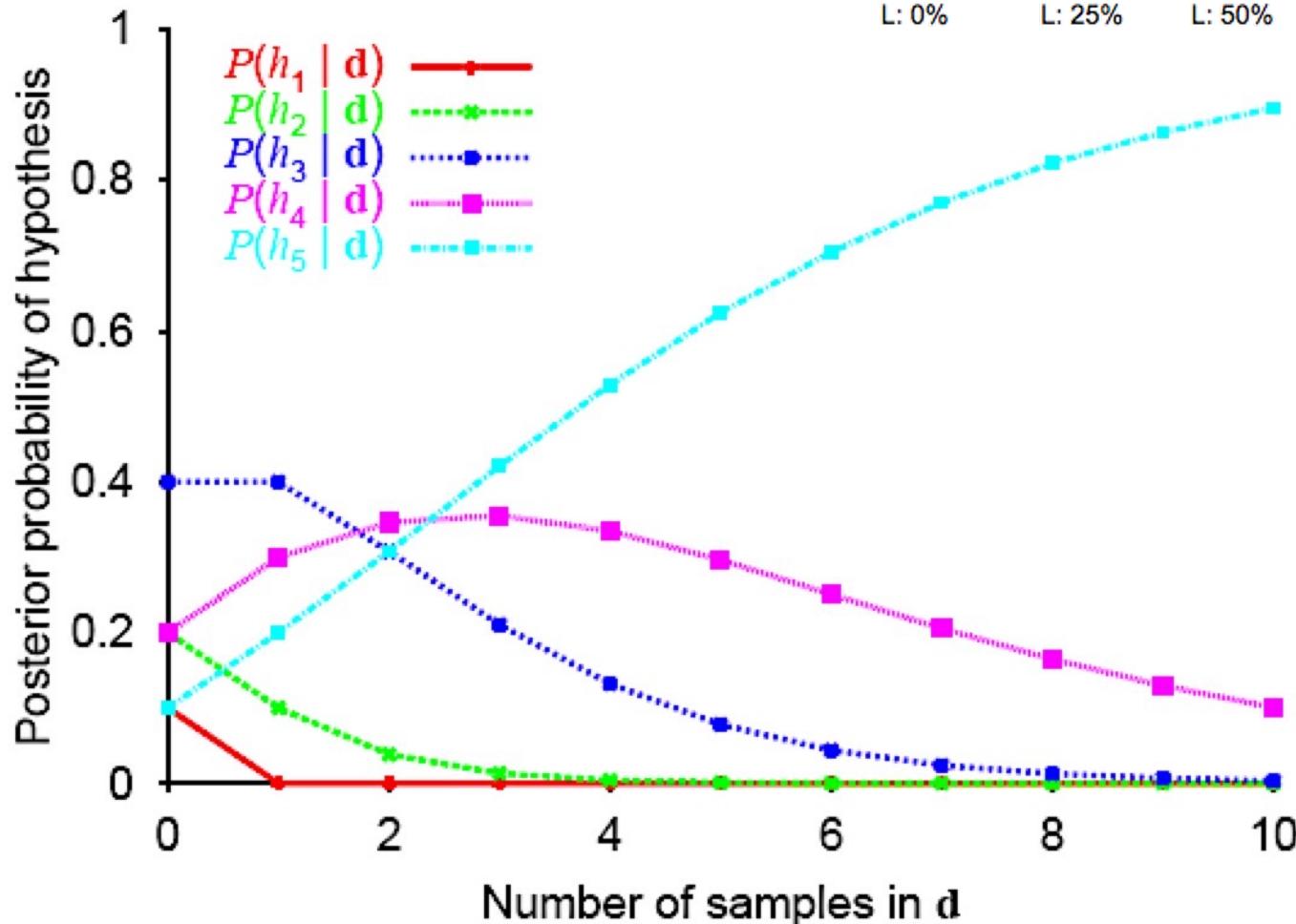
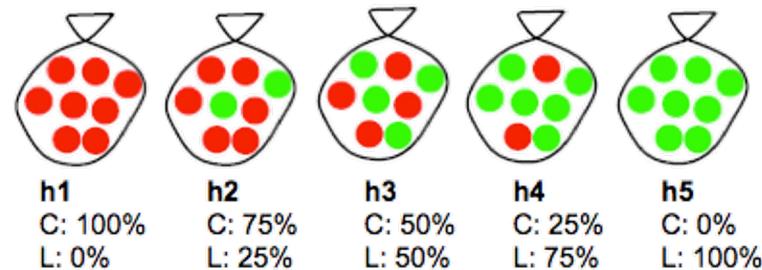
- Assume draws are independent
- Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$
- $\mathbf{d} = \{\bullet \bullet \bullet \bullet \bullet\}$

$P(d h_1) = 0$	$P(\mathbf{d} h_1)P(h_1) \approx 0$	$P(h_1 \mathbf{d}) = 0$
$P(d h_2) = 0.25^5$	$P(\mathbf{d} h_2)P(h_2) \approx 1.9e-4$	$P(h_2 \mathbf{d}) \approx 1.2e-3$
$P(d h_3) = 0.5^5$	$P(\mathbf{d} h_3)P(h_3) \approx 1.2e-2$	$P(h_3 \mathbf{d}) \approx 0.078$
$P(d h_4) = 0.75^5$	$P(\mathbf{d} h_4)P(h_4) \approx 4.7e-2$	$P(h_4 \mathbf{d}) \approx 0.29$
$P(d h_5) = 1^5$	$P(\mathbf{d} h_5)P(h_5) \approx 0.1$	$P(h_5 \mathbf{d}) \approx 0.62$
	$P(\mathbf{d}) \approx 0.16$	

Posterior Hypotheses

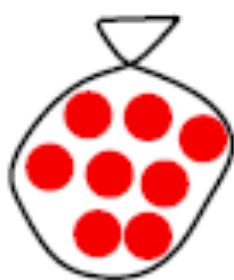
Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$

Data: All our samples are limes.



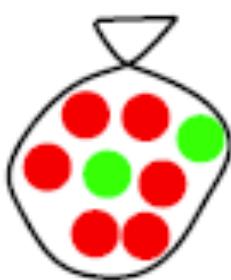
Candy Example

- Candy comes in 2 flavors, cherry and lime
- Manufacturer makes 5 types of bags:



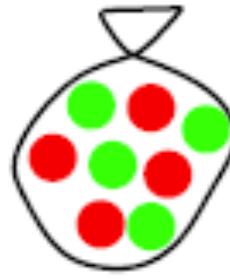
h1

C: 100%
L: 0%



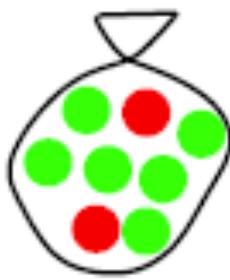
h2

C: 75%
L: 25%



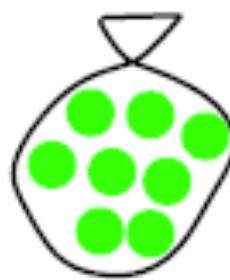
h3

C: 50%
L: 50%



h4

C: 25%
L: 75%



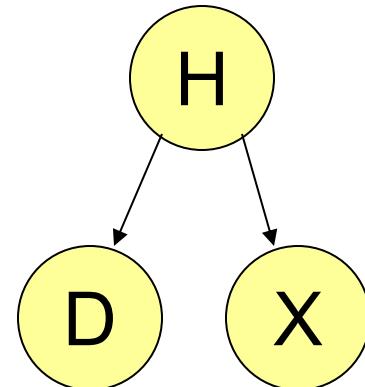
h5

C: 0%
L: 100%

- h1 and h5 are equally common. h2 is twice as common as h1, h4 is twice as common as h5, and h3 is twice as common as h2.
- Suppose we draw Which bag are we holding? **Which flavor will we draw next?**

Predicting the Next Draw

- $$\begin{aligned} P(X|\mathbf{d}) &= \sum_i P(X|h_i, \mathbf{d})P(h_i|\mathbf{d}) \\ &= \sum_i P(X|h_i)P(h_i|\mathbf{d}) \end{aligned}$$



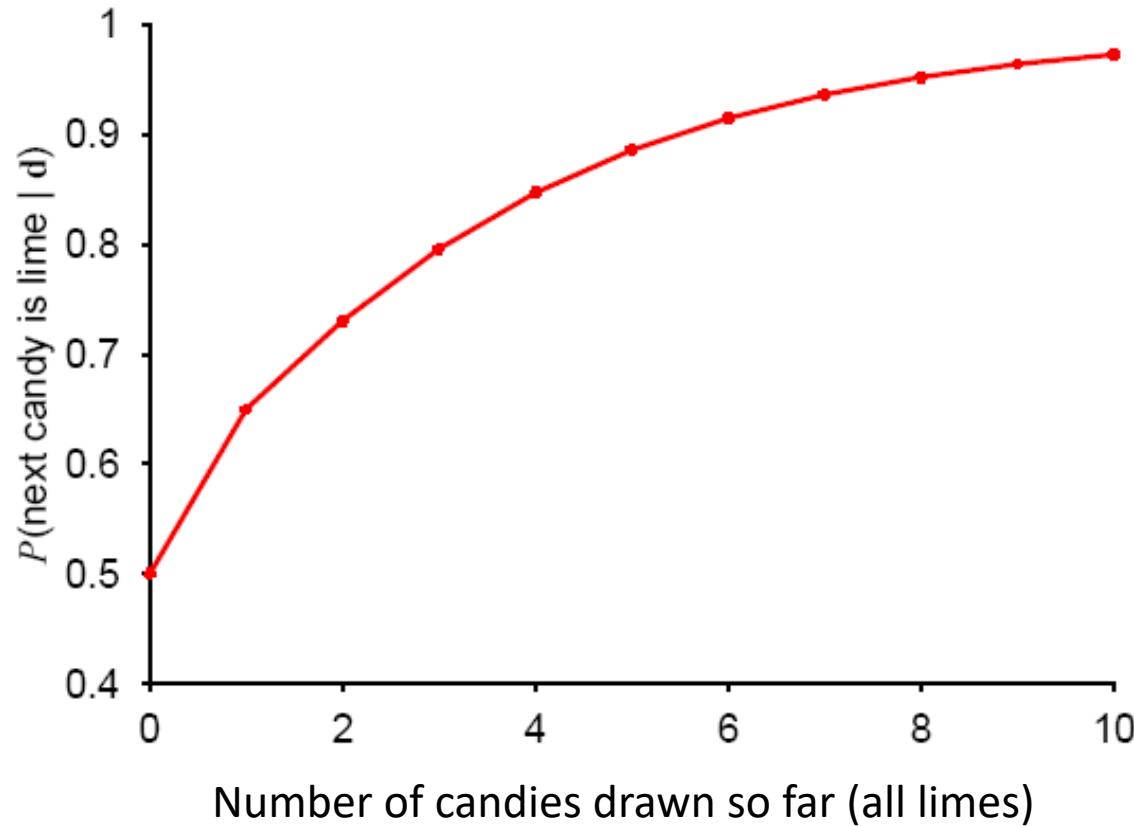
Probability that next candy drawn is a lime

$$\left. \begin{array}{ll} P(h_1|\mathbf{d}) = 0 & P(X|h_1) = 0 \\ P(h_2|\mathbf{d}) \approx 1.2e-3 & P(X|h_2) = 0.25 \\ P(h_3|\mathbf{d}) \approx 0.078 & P(X|h_3) = 0.5 \\ P(h_4|\mathbf{d}) \approx 0.29 & P(X|h_4) = 0.75 \\ P(h_5|\mathbf{d}) \approx 0.62 & P(X|h_5) = 1 \end{array} \right\} P(X|\mathbf{d}) \approx 0.890$$

$P(\text{Next Candy is Lime} \mid d)$

Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$

Data: All our samples are limes.



Properties of Bayesian Learning

- If exactly one hypothesis is correct, then the posterior probability of the correct hypothesis will tend toward 1 as more data is observed
- The effect of the prior distribution decreases as more data is observed

Hypothesis Spaces often Intractable

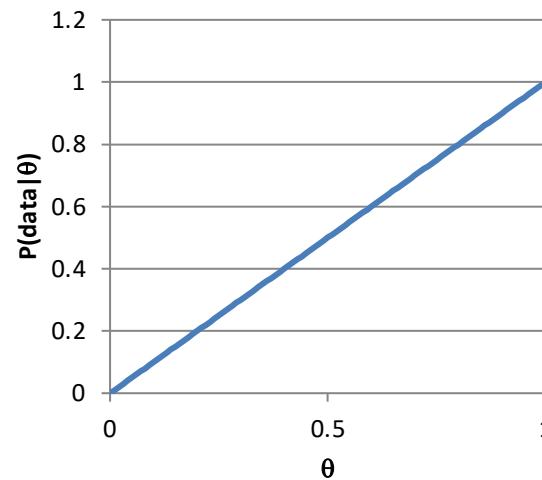
- But: a hypothesis is a joint probability table over state variables
- What if we don't have a reasonable prior?
- In practice, we'll need to use additional information about the structure of the model
 - E.g. conditional independent assumptions
 - Some parametric form for the hypotheses

Learning the bias of a coin

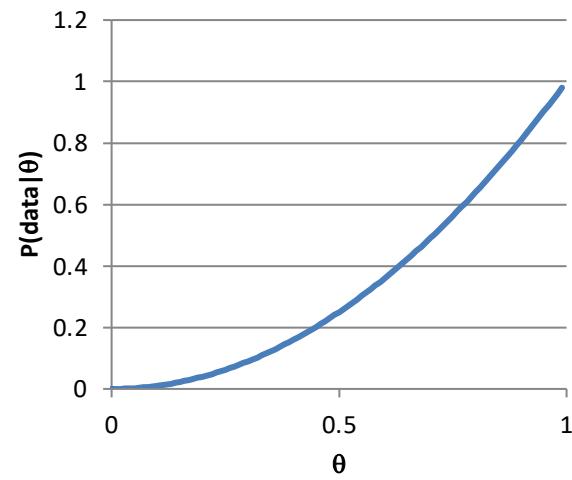
- We have a coin with unknown $P(H)$
- Let $P(H)$ be θ (**hypothesis**)
- Flips are i.i.d. – independent, identically distributed
- We flip N times to get a sequence of outcomes D . Of these, c flips are heads (**data**).
- Consider $P(D | \theta)$, the *likelihood of the data given the model*.

We flip coin N times, c of these are heads. Coin has unknown $\Theta = P(H)$.

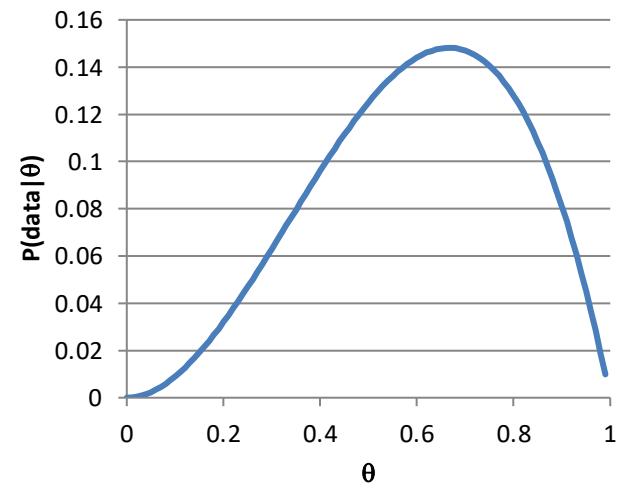
$N=1, c=1$



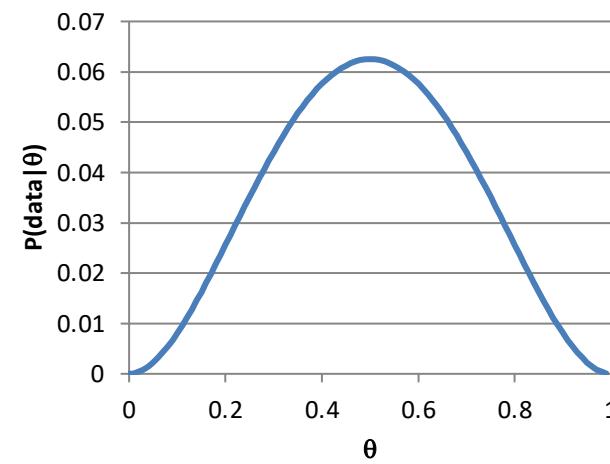
$N=2, c=2$



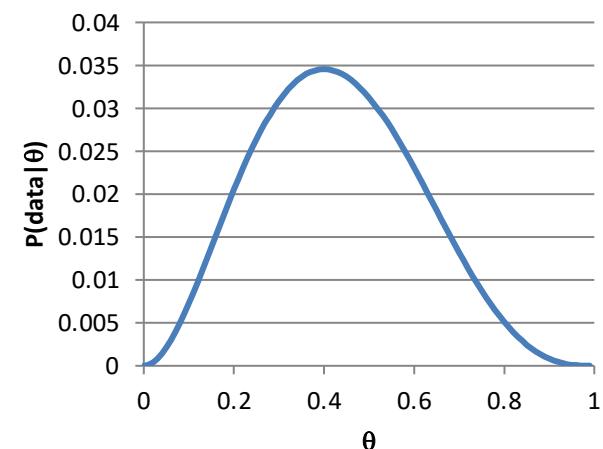
$N=3, c=2$



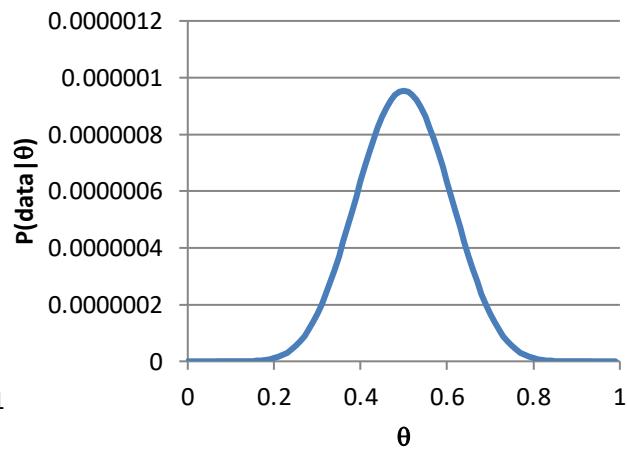
$N=4, c=2$



$N=5, c=2$

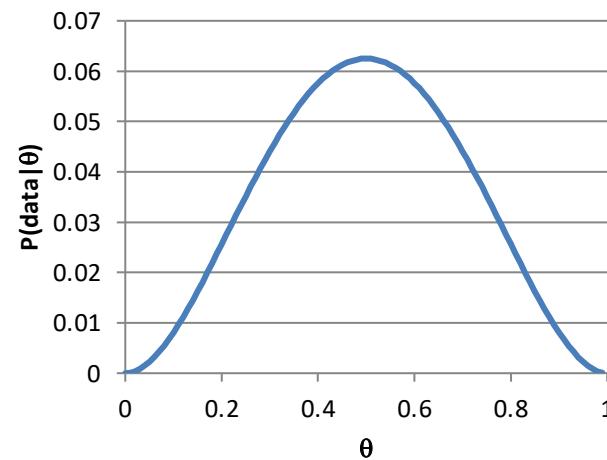


$N=20, c=10$

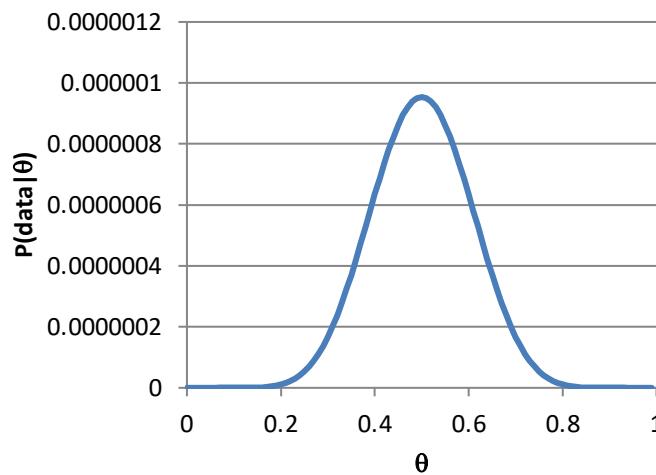


We flip coin N times, c of these are heads. Coin has unknown $\Theta=P(H)$.

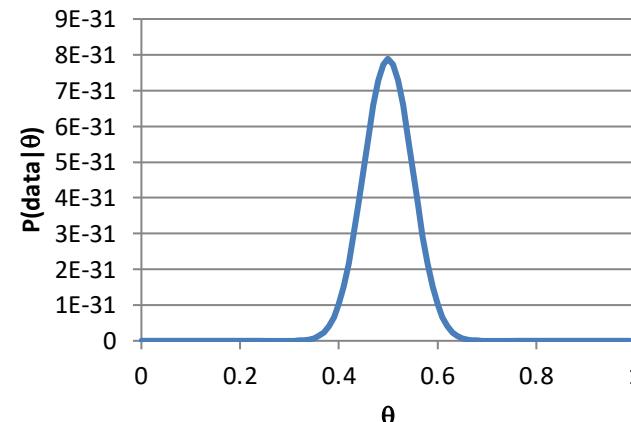
N=4, c=2



N=20, c=10



N=100, c=50



Maximum Likelihood Estimation (MLE)

- Say we have a set of training samples $D=(x_1, \dots, x_N)$
- We've decided on a parametric model for the distribution, having parameters θ
- We define a likelihood function measuring the probability of the data for a given model θ ,

$$P(D|\theta) = \prod_i^N P(D_i|\theta)$$

- In MLE, we want to find a model that *maximizes the probability of the observed data given the model*,

$$\theta^* = \arg \max_{\theta} P(D|\theta)$$

Other Closed-Form MLE results

- **Multi-valued variables:** take fraction of counts for each value
- **Continuous Gaussian distributions:** take average value as mean, standard deviation of data as standard deviation

Maximum Likelihood Properties

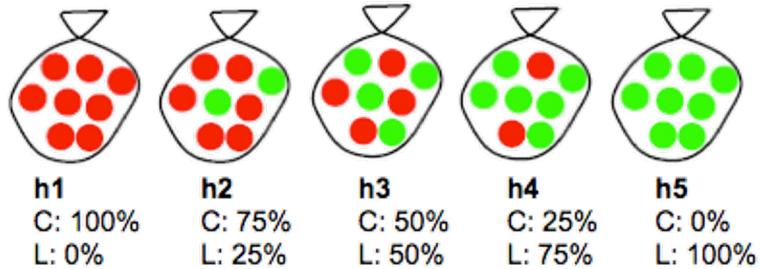
- As the number of data points approaches infinity, the MLE approaches the true estimate
- With little data, MLEs can vary wildly

MLE in candy example

Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$

Data: All our samples are limes.

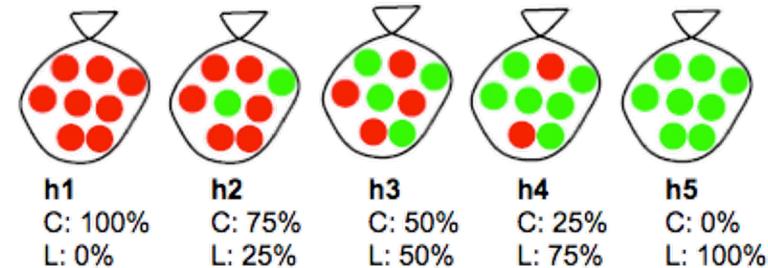
- What is the MLE?



MLE in candy example

Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$

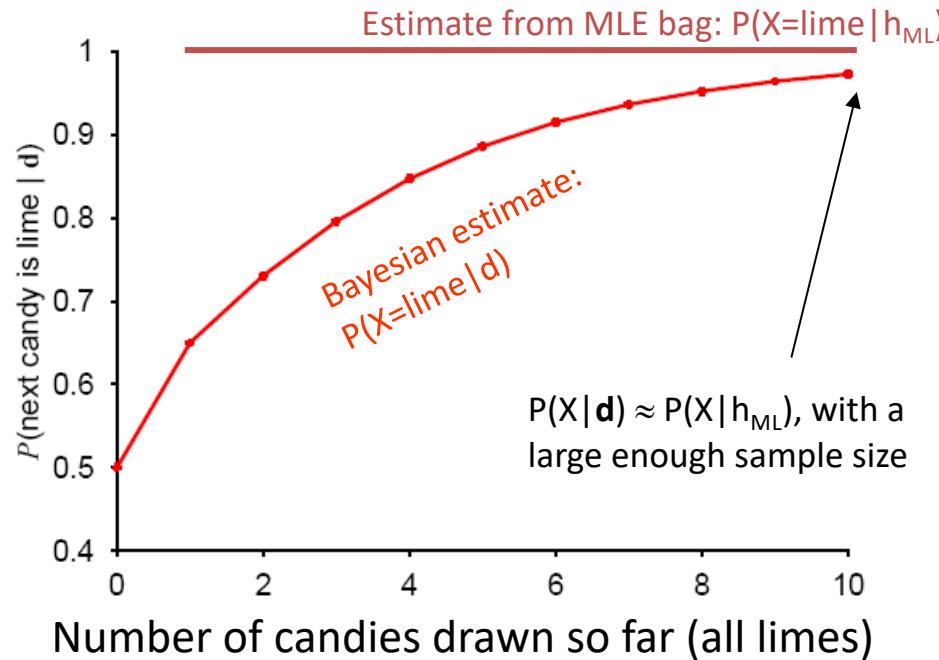
Data: All our samples are limes.



- What is the MLE?

$$h_{ML} = \operatorname{argmax}_i P(d | h_i)$$

What is probability next candy is lime?



Disadvantages of ML Estimation

- Tells us nothing about the certainty of our estimates
 - Note that we get exactly the same answer whether we flip 1 head out of 3, or 1,000 out of 3,000
- If we have no observations, can't estimate anything
 - Or: if we have a large number of variables, some values will never be seen and we can conclude nothing about them
- No way to incorporate prior evidence
 - E.g. that most coins are unbiased (or not very biased)

Maximum A Posteriori Estimation

- Maximum a posteriori (MAP) estimation
- Idea: use the hypothesis prior to get a better initial estimate than ML, without full Bayesian estimation
 - “Most coins I’ve seen have been fair coins, so I won’t let the first few tails sway my estimate much”
 - “Now that I’ve seen 100 tails in a row, I’m pretty sure it’s not a fair coin anymore”

Maximum A Posteriori

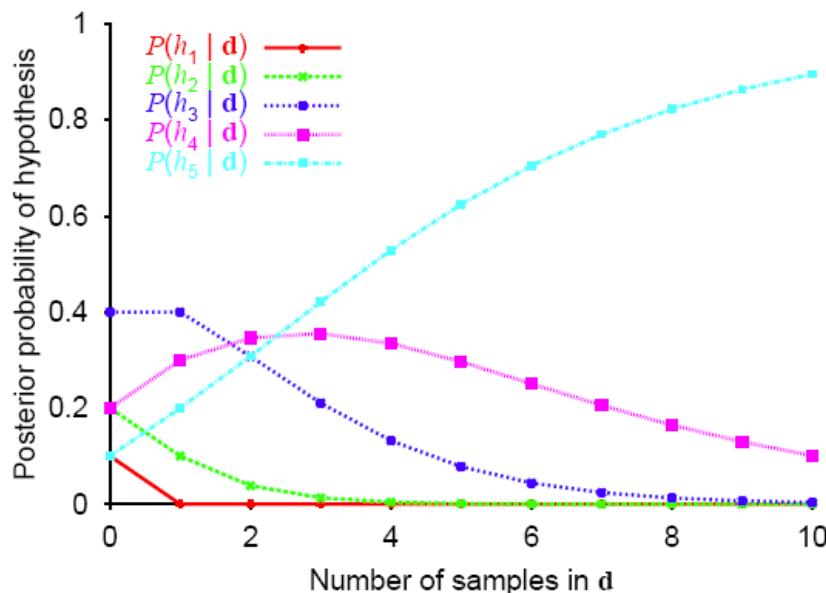
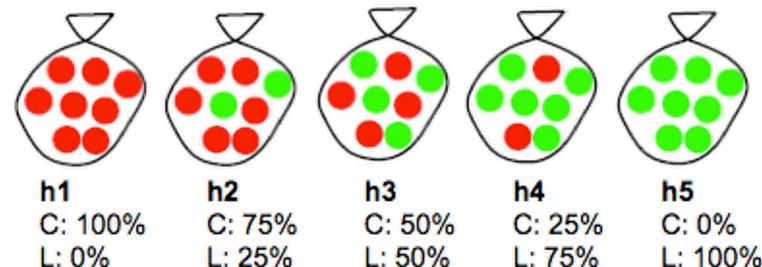
- $P(\theta | \mathbf{d})$ is the posterior probability of the hypothesis, given the data
- $\operatorname{argmax}_{\theta} P(\theta | \mathbf{d})$ is known as the **maximum a posteriori** (MAP) estimate
- Posterior of hypothesis θ given data $\mathbf{d}=\{d_1, \dots, d_N\}$
 - $P(\theta | \mathbf{d}) = 1/\alpha P(d | \theta) P(\theta)$
 - Max over θ doesn't affect α
 - So MAP estimate is $\operatorname{argmax}_{\theta} P(d | \theta) P(\theta)$

Maximum a Posteriori

Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$

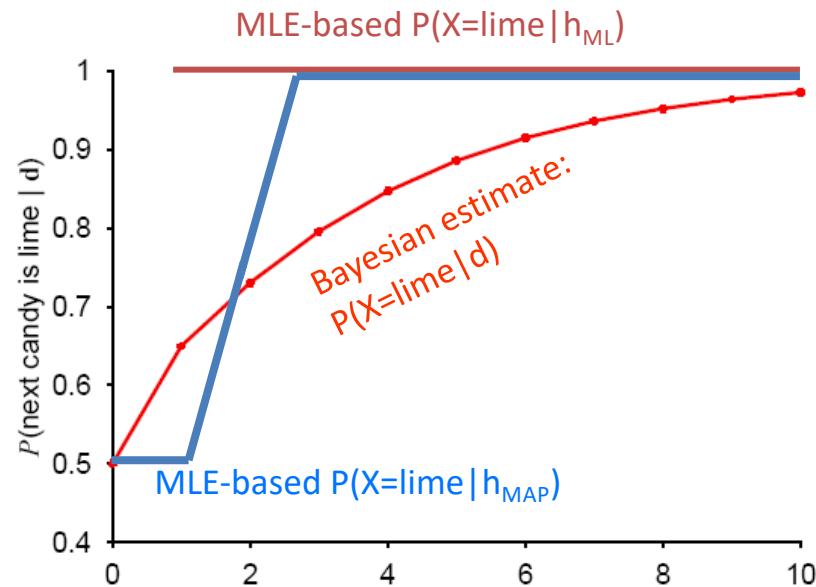
Data: All our samples are limes.

$$h_{\text{MAP}} = \operatorname{argmax}_i P(h_i | d) \quad h_{\text{ML}} = \operatorname{argmax}_i P(d | h_i)$$



# samples:	0	1	2	3	4	5	6	7	8	9	10
h_{MAP}	h3	h3	h4	h5							
h_{ML}		h5									

What is probability next candy is lime?



Advantages of MAP and MLE over Bayesian estimation

- Involves an *optimization* rather than a large summation
 - Local search techniques
- For some types of distributions, there are *closed-form* solutions that are easily computed

Next class

- More about ML and MAP

MLE in Bayes Networks

- The likelihood function can be factored into a product over variables and over exemplars,

$$P(D|\theta) = \prod_i^N P(D_i|\theta) \quad \text{where} \quad P(D|\theta_j) = \prod_i^N P(d_j^i | \text{Pa}(X_j))$$

where $\theta=(\theta_1, \dots, \theta_M)$, θ_j is the CPD for variable X_j , $D=(d_1, d_2, \dots, d_N)$, and each $d_i=(d_i^1, d_i^2, \dots, d_i^M)$

- If all variables can be observed, each of these factors can be maximized individually
 - So we can estimate parameters of each variable's probability distribution individually
- Key result: ML estimate of Bayes Net parameters is given by the set of ML estimates of CPDs of each variable.**

Maximum Likelihood for BN

- For any BN, the ML parameters of any CPT can be derived by the fraction of observed values in the data, conditioned on matched parent values

N=1000

E: 500

$P(E) = 0.5$



B: 200

$P(B) = 0.2$

E	B	$P(A E,B)$
T	T	0.95
F	T	0.95
T	F	0.34
F	F	0.003

$$\begin{aligned}A|E,B: & 19/20 \\A|\neg E,B: & 188/200 \\A|E, \neg B: & 170/500 \\A|\neg E, \neg B : & 1/380\end{aligned}$$

Bayes Net ML Algorithm

- Input: BN with nodes X_1, \dots, X_n , dataset $D = (d_1, \dots, d_N)$
 - Each $d_i = (d_i^1, d_i^2, \dots, d_i^M)$ is a sample with values for all variables
- For each node X with parents Y_1, \dots, Y_k :
 - For all $y_1 \in \text{Val}(Y_1), \dots, y_k \in \text{Val}(Y_k)$
 - For all $x \in \text{Val}(X)$
 - Count the number of times $(X=x, Y_1=y_1, \dots, Y_k=y_k)$ is observed in D . Let this be m_x
 - Count the number of times $(Y_1=y_1, \dots, Y_k=y_k)$ is observed in D . Let this be m . (note $m = \sum_x m_x$)
 - Set $P(x | y_1, \dots, y_k) = m_x / m$ for all $x \in \text{Val}(X)$

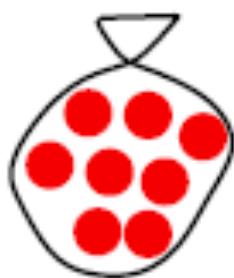
Maximum Likelihood (ML) and Maximum A Posteriori (MAP) estimation

Announcements

- A2 posted, sign up and create your teams
- Midterm exam 10/26 6:30pm-7:45pm
 - Practice materials posted on Canvas (+tophat)
- Don't forget the quiz (deadline on Friday)
- A0 grades, submit your regrade requests following instructions in the syllabus
- Next class (Monday) will be via zoom (we will do another coding exercise)

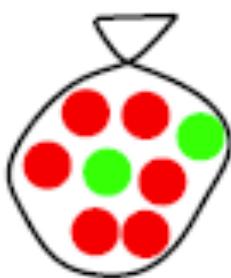
Candy Example

- Candy comes in 2 flavors, cherry and lime
- Manufacturer makes 5 types of bags:



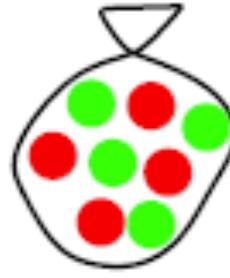
h1

C: 100%
L: 0%



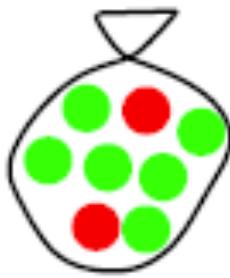
h2

C: 75%
L: 25%



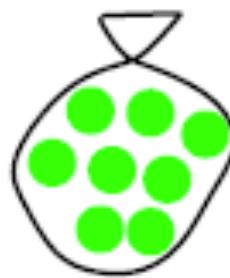
h3

C: 50%
L: 50%



h4

C: 25%
L: 75%

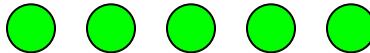


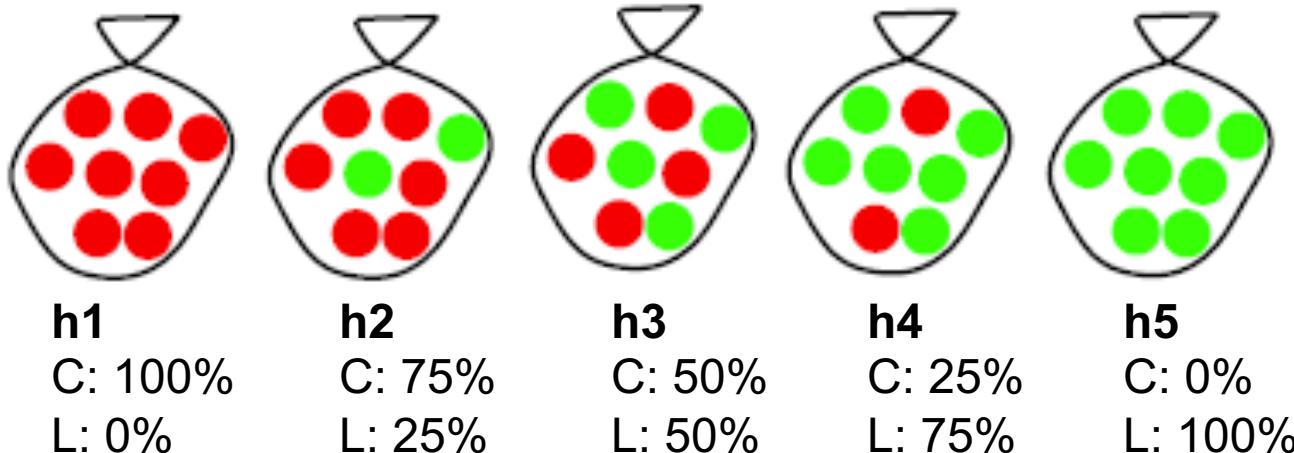
h5

C: 0%
L: 100%

- h1 and h5 are equally common. h2 is twice as common as h1, h4 is twice as common as h5, and h3 is twice as common as h2.
- Suppose we draw Which bag are drawing from?

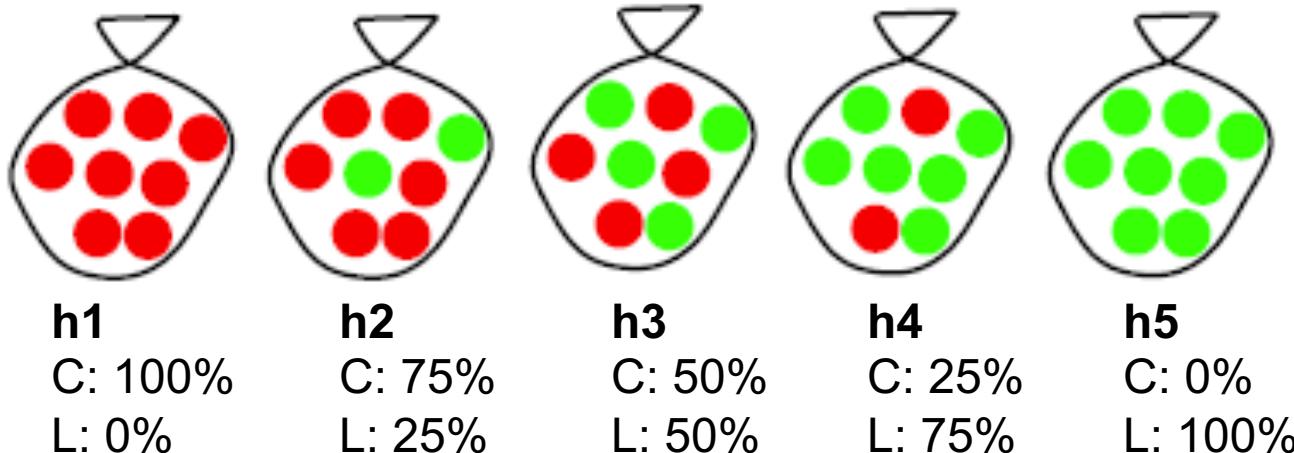
Bayesian Learning

- Main idea: Compute the probability of **each** hypothesis, given the data
- Data d : 
- Hypotheses: h_1, \dots, h_5



Using Bayes' Rule

- $P(h_i | d) = \alpha P(d | h_i) P(h_i)$ is the **posterior**
 - (Recall, $1/\alpha = P(d) = \sum_i P(d | h_i) P(h_i)$)
- $P(d | h_i)$ is the **likelihood**
- $P(h_i)$ is the **hypothesis prior**



Computing the Posterior

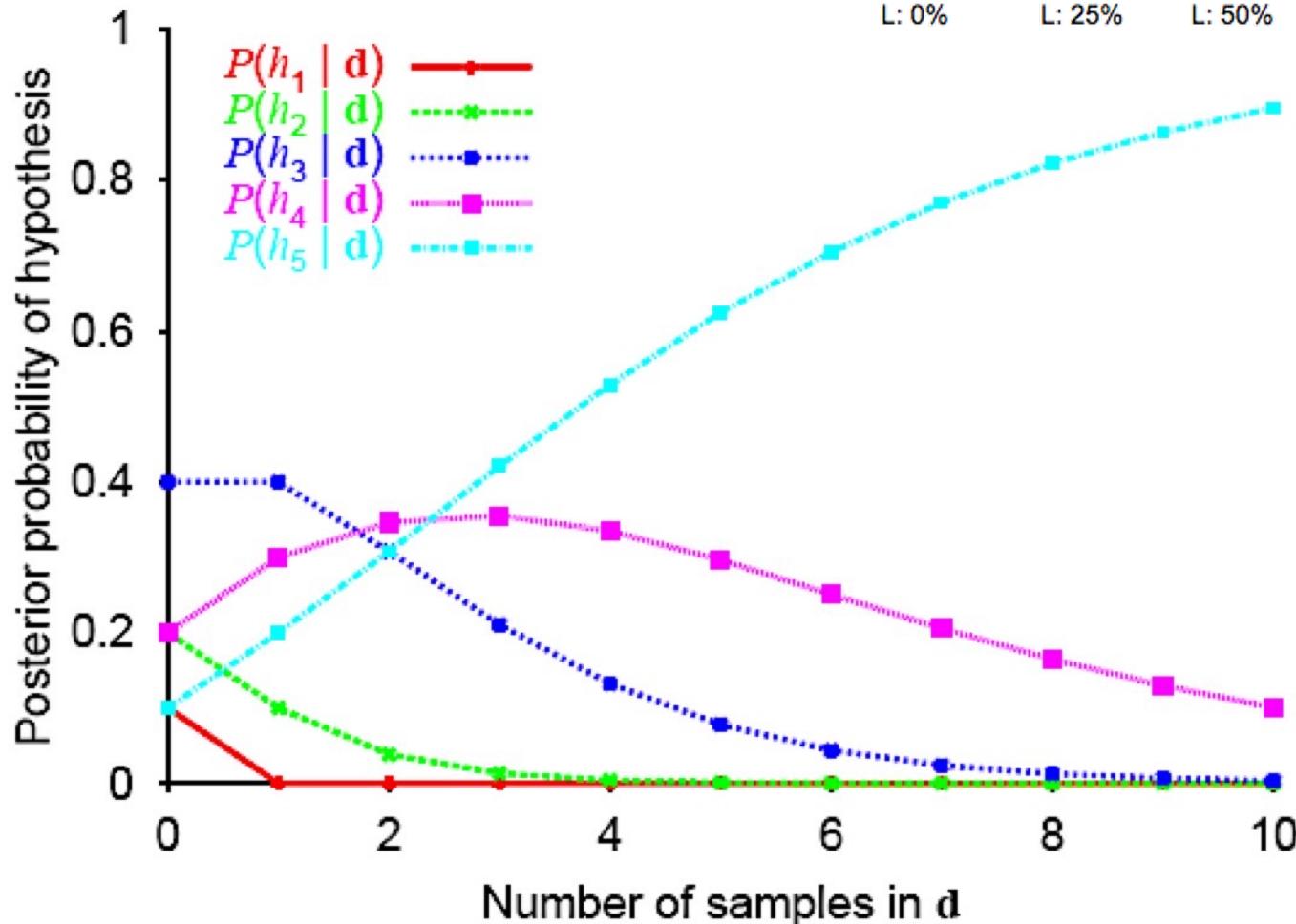
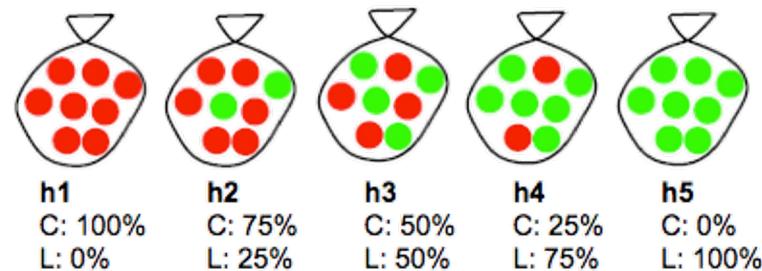
- Assume draws are independent
- Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$
- $\mathbf{d} = \{\bullet \bullet \bullet \bullet \bullet\}$

$P(d h_1) = 0$	$P(\mathbf{d} h_1)P(h_1) \approx 0$	$P(h_1 \mathbf{d}) = 0$
$P(d h_2) = 0.25^5$	$P(\mathbf{d} h_2)P(h_2) \approx 1.9e-4$	$P(h_2 \mathbf{d}) \approx 1.2e-3$
$P(d h_3) = 0.5^5$	$P(\mathbf{d} h_3)P(h_3) \approx 1.2e-2$	$P(h_3 \mathbf{d}) \approx 0.078$
$P(d h_4) = 0.75^5$	$P(\mathbf{d} h_4)P(h_4) \approx 4.7e-2$	$P(h_4 \mathbf{d}) \approx 0.29$
$P(d h_5) = 1^5$	$P(\mathbf{d} h_5)P(h_5) \approx 0.1$	$P(h_5 \mathbf{d}) \approx 0.62$
	$P(\mathbf{d}) \approx 0.16$	

Posterior Hypotheses

Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$

Data: All our samples are limes.



Parameter estimation

- Assume that data is believed to follow some distribution or model (e.g. Gaussian, Poisson, etc.), represented as M
 - Maybe by looking at the histogram of the data we suspect that data is Gaussian or Uniform
 - Maybe we know something about the process that generated the data
- Unfortunately
 - The model has unknown parameter(s) Θ (e.g. model parameters $\rightarrow \mu$ or σ)
 - Observe a random sample from distribution (independent, identically distributed): X_1, \dots, X_n (i.i.d) $\sim P(X | \Theta)$
 - Want to estimate parameter (Θ) from the data, $D = \{X_i; \widehat{\Theta}\}, i = 1 \dots N$
- How do we compute the “best” or “optimal” parameter estimates from the data?

Parameter estimation

- Generally, there are two types of parameter estimation approaches:

- Maximum Likelihood Estimation (MLE)

$$M_{ML} = \arg \max_{M \in \mathcal{M}} \{p(\mathcal{D}|M)\}.$$

- Maximum a posteriori (MAP) Estimation

$$M_{MAP} = \arg \max_{M \in \mathcal{M}} \{p(M|\mathcal{D})\}$$

Probability components

- From Bayes Rule

$$p(M|\mathcal{D}) = \frac{p(\mathcal{D}|M) \cdot p(M)}{p(\mathcal{D})},$$

- $P(M|\mathcal{D})$ is the **posterior** distribution of the model given the data (or observation)
- $P(\mathcal{D}|M)$ is the **likelihood** of the data given the model
- $P(M)$ is the **prior** distribution of the model
- $P(\mathcal{D})$ is the marginal distribution of the data

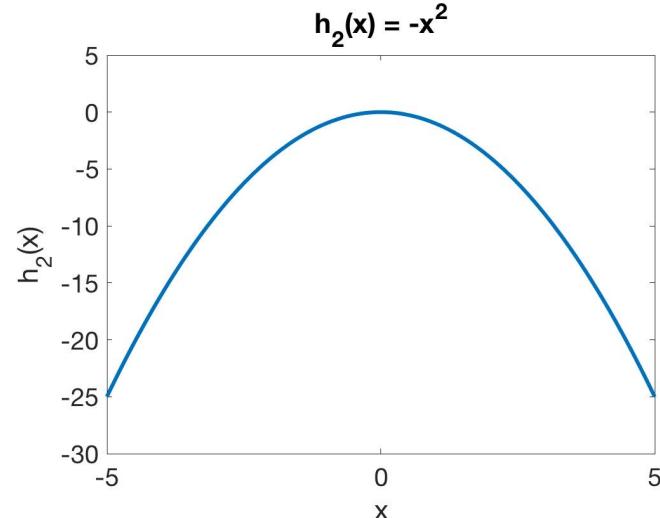
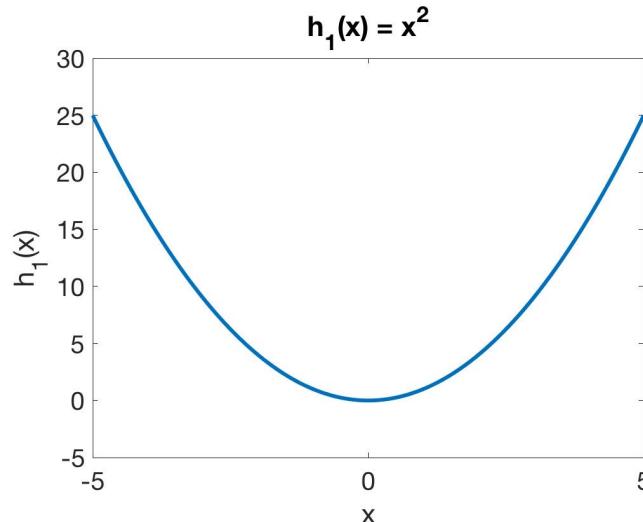
$$p(\mathcal{D}) = \begin{cases} \sum_{f \in \mathcal{F}} p(\mathcal{D}|f)p(f) & f : \text{discrete} \\ \int_{\mathcal{F}} p(\mathcal{D}|f)p(f)df & f : \text{continuous} \end{cases}$$

Calculus Review: Function Max or Min

- To find max or min of a function $h(x)$:
 1. Take the first and second derivatives of $h(x)$ with respect to x
 2. Set the first derivative to zero and solve for x (*i.e.* “value”)
 3. Evaluate the second derivative of $h(x)$ using the solution(s) from step 2
 1. If $h''(\text{“value”}) > 0$, then minimum at $x = \text{“value”}$
 2. If $h''(\text{“value”}) < 0$, then maximum at $x = \text{“value”}$

Calculus Review: Function Max or Min

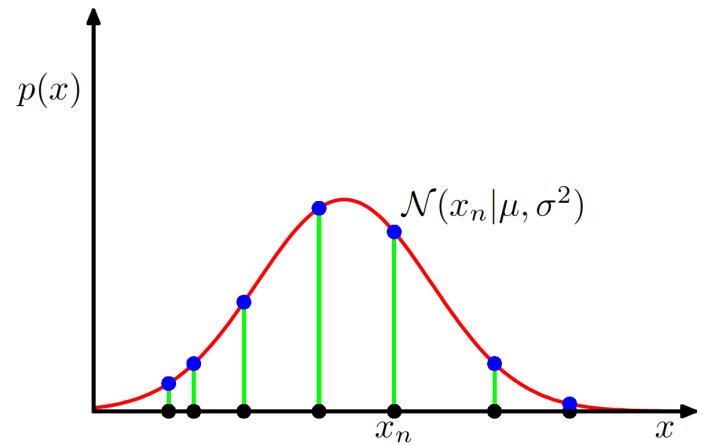
- Examples: $h_1(x) = x^2$ and $h_2(x) = -x^2$
 - $h_1'(x) = 2x, h_1''(x) = 2; h_2'(x) = -2x, h_2''(x) = -2$
 - $h_1'(x) = 0 \Rightarrow x = 0 \dots h_2'(x) = 0 \Rightarrow x = 0$
 - For h_1 : minimum at $x = 0$; For h_2 : maximum at $x = 0$
- Note: maximizing $f(x)$ is equivalent to minimizing $-f(x)$



ML Estimation

$$M_{ML} = \arg \max_{M \in \mathcal{M}} \{p(\mathcal{D}|M)\}.$$

- MLE chooses Θ to best explain the data (assuming i.i.d)
 - Assumes no knowledge of prior distribution of the model or data



Example: ML Estimation for Poisson distribution

Example 8: Suppose data set $\mathcal{D} = \{2, 5, 9, 5, 4, 8\}$ is an i.i.d. sample from a Poisson distribution with an unknown parameter λ_t . Find the maximum likelihood estimate of λ_t .

The probability density function of a Poisson distribution is expressed as $p(x|\lambda) = \lambda^x e^{-\lambda} / x!$, with some parameter $\lambda \in \mathbb{R}^+$. We will estimate this parameter as

$$\lambda_{ML} = \arg \max_{\lambda \in (0, \infty)} \{p(\mathcal{D}|\lambda)\}. \quad (2.2)$$

Example: ML Estimation for Poisson distribution

- Poisson Distribution $p(x|\lambda) = \lambda^x e^{-\lambda} / x!$
- We can write the likelihood function as

$$\begin{aligned} p(\mathcal{D}|\lambda) &= p(\{x_i\}_{i=1}^n | \lambda) \\ &= \prod_{i=1}^n p(x_i | \lambda) \\ &= \frac{\lambda^{\sum_{i=1}^n x_i} \cdot e^{-n\lambda}}{\prod_{i=1}^n x_i!}. \end{aligned}$$

- To make it easier to find λ that maximizes the likelihood we take a log (since log is a monotonic function it won't affect the result)
- We express the log-likelihood as $ll(\mathcal{D}, \lambda) = \ln p(\mathcal{D}|\lambda)$

$$ll(\mathcal{D}, \lambda) = \ln \lambda \sum_{i=1}^n x_i - n\lambda - \sum_{i=1}^n \ln(x_i!)$$

Example: ML Estimation for Poisson distribution

- Next we take the first derivative with respect to λ

$$\frac{\partial \ln(\mathcal{D}, \lambda)}{\partial \lambda} = \frac{1}{\lambda} \sum_{i=1}^n x_i - n$$
$$= 0.$$
$$\ln(\mathcal{D}, \lambda) = \ln \lambda \sum_{i=1}^n x_i - n\lambda - \sum_{i=1}^n \ln(x_i!)$$

- And we find that λ_{ML} is equal to the sample mean

$$\lambda_{ML} = \frac{1}{n} \sum_{i=1}^n x_i$$

- When a distribution is parametrized by its mean (as with Poisson, remember $E(p(x)) = \lambda$) it often happens that

$$\hat{\Theta}_{MLE} = \bar{x}$$

MAP Estimation

$$M_{MAP} = \arg \max_{M \in \mathcal{M}} \{p(M|\mathcal{D})\}$$

$$p(M|\mathcal{D}) = \frac{p(\mathcal{D}|M) \cdot p(M)}{p(\mathcal{D})}$$

- As before have X_1, \dots, X_n (i.i.d) $\sim P(X|\Theta)$, and want to estimate Θ
- Suppose now we have a prior belief on Θ , expressed as a prob. distribution: $\Theta \sim P(\Theta)$
- Using Bayes Rule, can compute the posterior distribution

$$P(\Theta|x_1, \dots, x_n) = \frac{P(x_1, \dots, x_n|\Theta)P(\Theta)}{P(x_1, \dots, x_n)}$$

This reflects everything that we know about Θ after observation

- Then, MAP estimate is: $\hat{\Theta}_{MAP} = \arg \max_{\Theta} P(\Theta|x_1, \dots, x_n)$

Most likely Θ given knowledge

MAP Estimation

$$P(\Theta|x_1, \dots, x_n) = \frac{P(x_1, \dots, x_n|\Theta)P(\Theta)}{P(x_1, \dots, x_n)}$$

$$\hat{\Theta}_{MAP} = \arg \max_{\Theta} P(\Theta|x_1, \dots, x_n)$$

- Since $P(X_1, \dots, X_n)$ is constant once observed,

$$\hat{\Theta}_{MAP} = \arg \max_{\Theta} P(\Theta|x_1, \dots, x_n) = \arg \max_{\Theta} P(\Theta)P(x_1, \dots, x_n|\Theta)$$

MAP Estimation

- The same thing written in a different way

$$p(M|\mathcal{D}) = \frac{p(\mathcal{D}|M) \cdot p(M)}{p(\mathcal{D})}$$
$$\propto p(\mathcal{D}|M) \cdot p(M)$$

$$M_{MAP} = \arg \max_{M \in \mathcal{M}} \{p(M|\mathcal{D})\}$$

$$p(\mathcal{D}) = \begin{cases} \sum_{M \in \mathcal{M}} p(\mathcal{D}|M)p(M) & M : \text{discrete} \\ \int_{\mathcal{M}} p(\mathcal{D}|M)p(M)dM & M : \text{continuous} \end{cases}$$

$$M_{MAP} = \arg \max_{M \in \mathcal{M}} \{p(\mathcal{D}|M)p(M)\}$$

$$M_{ML} = \arg \max_{M \in \mathcal{M}} \{p(\mathcal{D}|M)\}$$

- Where \propto is the proportionality symbol

Example: MAP Estimation for Poisson distribution (with Gamma prior on parameters)

Example 9: Let $\mathcal{D} = \{2, 5, 9, 5, 4, 8\}$ again be an i.i.d. sample from $\text{Poisson}(\lambda_t)$, but now we are also given additional information. Suppose the prior knowledge about λ_t can be expressed using a gamma distribution $\Gamma(x|k, \theta)$ with parameters $k = 3$ and $\theta = 1$. Find the maximum a posteriori estimate of λ_t .

First, we write the probability density function of the gamma family as

$$\Gamma(x|k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)},$$

$x > 0$, $k > 0$, and $\theta > 0$

$$\Gamma(k) = (k - 1)!$$

Example: MAP Estimation for Poisson distribution (with Gamma prior on parameters)

As before, we can write the likelihood function as

$$p(\mathcal{D}|\lambda) = \frac{\lambda^{\sum_{i=1}^n x_i} \cdot e^{-n\lambda}}{\prod_{i=1}^n x_i!}$$

and the prior distribution as

$$\lambda_{MAP} = \arg \max_{\lambda \in (0, \infty)} \{p(\mathcal{D}|\lambda)p(\lambda)\}$$
$$p(\lambda) = \frac{\lambda^{k-1} e^{-\frac{\lambda}{\theta}}}{\theta^k \Gamma(k)}.$$

Now, we can maximize the logarithm of the posterior distribution $p(\lambda|\mathcal{D})$ using

$$\begin{aligned} \ln p(\lambda|\mathcal{D}) &\propto \ln p(\mathcal{D}|\lambda) + \ln p(\lambda) \\ &= \ln \lambda(k - 1 + \sum_{i=1}^n x_i) - \lambda(n + \frac{1}{\theta}) - \sum_{i=1}^n \ln x_i! - k \ln \theta - \ln \Gamma(k) \end{aligned}$$

Example: MAP Estimation for Poisson distribution (with Gamma prior on parameters)

$$\ln p(\lambda|\mathcal{D}) \propto \ln p(\mathcal{D}|\lambda) + \ln p(\lambda)$$

$$= \ln \lambda(k - 1 + \sum_{i=1}^n x_i) - \lambda(n + \frac{1}{\theta}) - \sum_{i=1}^n \ln x_i! - k \ln \theta - \ln \Gamma(k)$$

After taking partial derivative with respect to λ and equaling to 0, we can solve for λ

$$\begin{aligned}\lambda_{MAP} &= \frac{k - 1 + \sum_{i=1}^n x_i}{n + \frac{1}{\theta}} & \mathcal{D} &= \{2, 5, 9, 5, 4, 8\} \\ &= 5 & k &= 3 \\ & & & \theta = 1\end{aligned}$$

ML converges to MAP when we have many samples

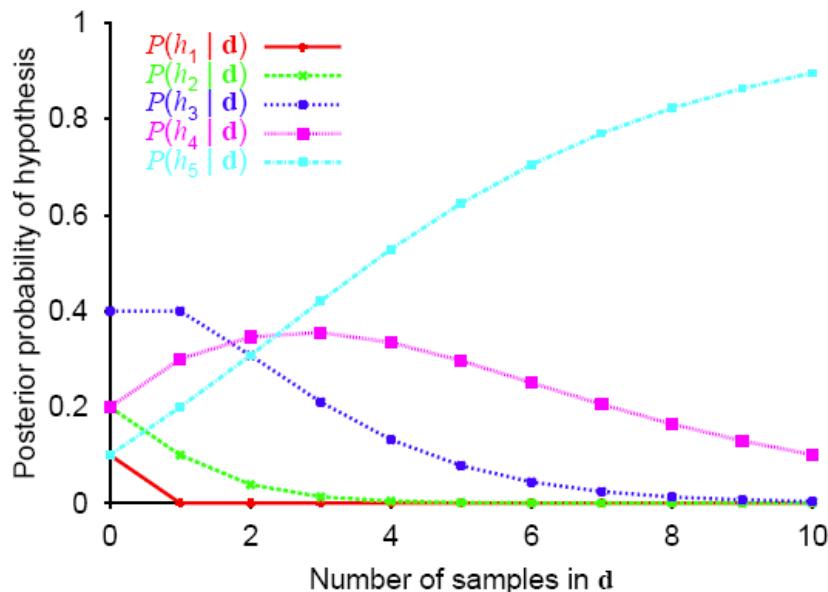
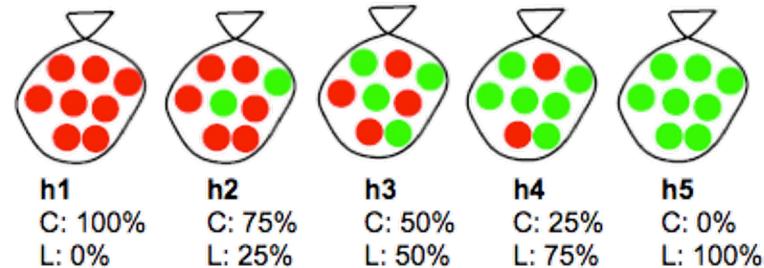
- Notice that with ML estimation we had $\lambda_{\text{ML}}=5.5$ and with MAP estimation we got $\lambda_{\text{MAP}}=5$.
- In the limit of infinite samples, both the MAP and ML converge to the same model, M (as long as the prior does not have zero probability on M).
- In other words, large data diminishes the importance of prior knowledge.
- To get some intuition for this result, we will show that the MAP and ML estimates converge to the same solution for the above example with a Poisson distribution.

Back to candy

Let $P(h_1), \dots, P(h_5) = (0.1, 0.2, 0.4, 0.2, 0.1)$

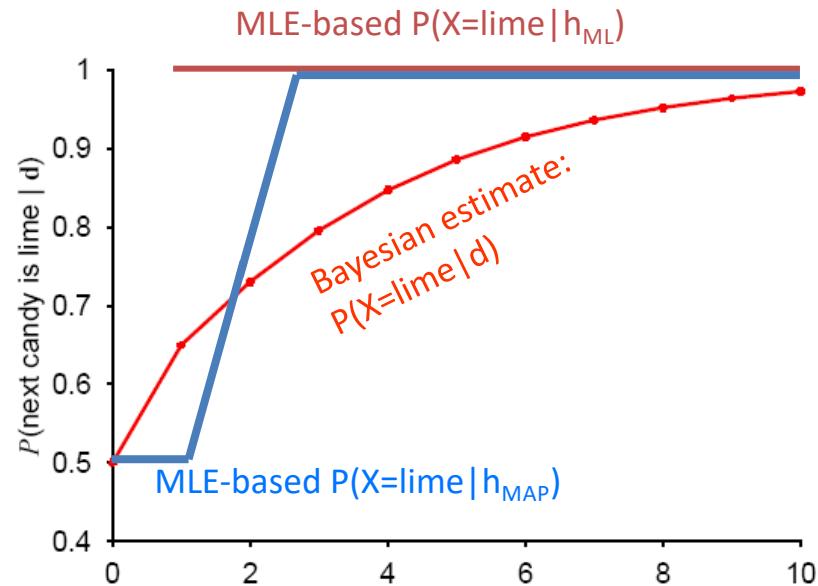
Data: All our samples are limes.

$$h_{\text{MAP}} = \operatorname{argmax}_i P(h_i | d) \quad h_{\text{ML}} = \operatorname{argmax}_i P(d | h_i)$$



# samples:	0	1	2	3	4	5	6	7	8	9	10
h_{MAP}	h3	h3	h4	h5							
h_{ML}		h5									

What is probability next candy is lime?



Next class

- Zoom coding exercise on Viterbi algorithm

Learning & Decision Trees

Announcements

- A2 due November 7
- A3 and optional A4 to follow (A4 will have a deadline during the last week of classes)
- Midterm grade distribution:

μ Average Score

39%

\nearrow High Score

87%

\searrow Low Score

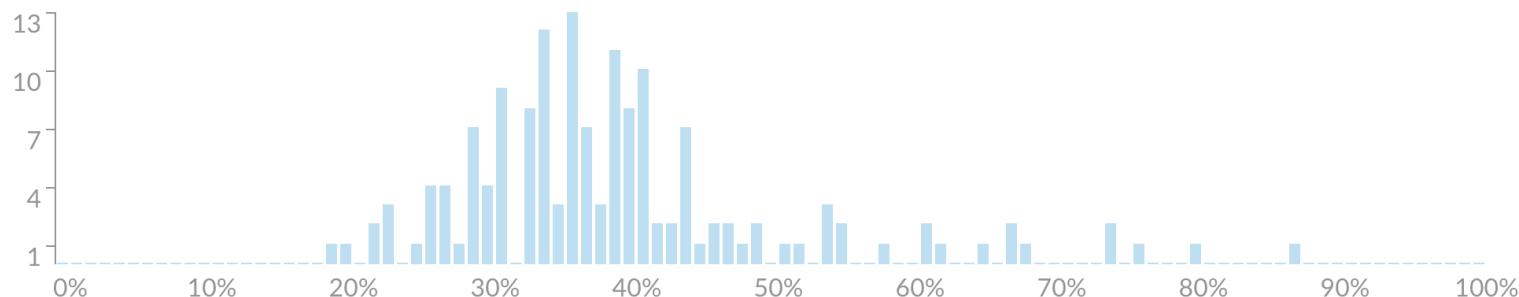
19%

σ Standard Deviation

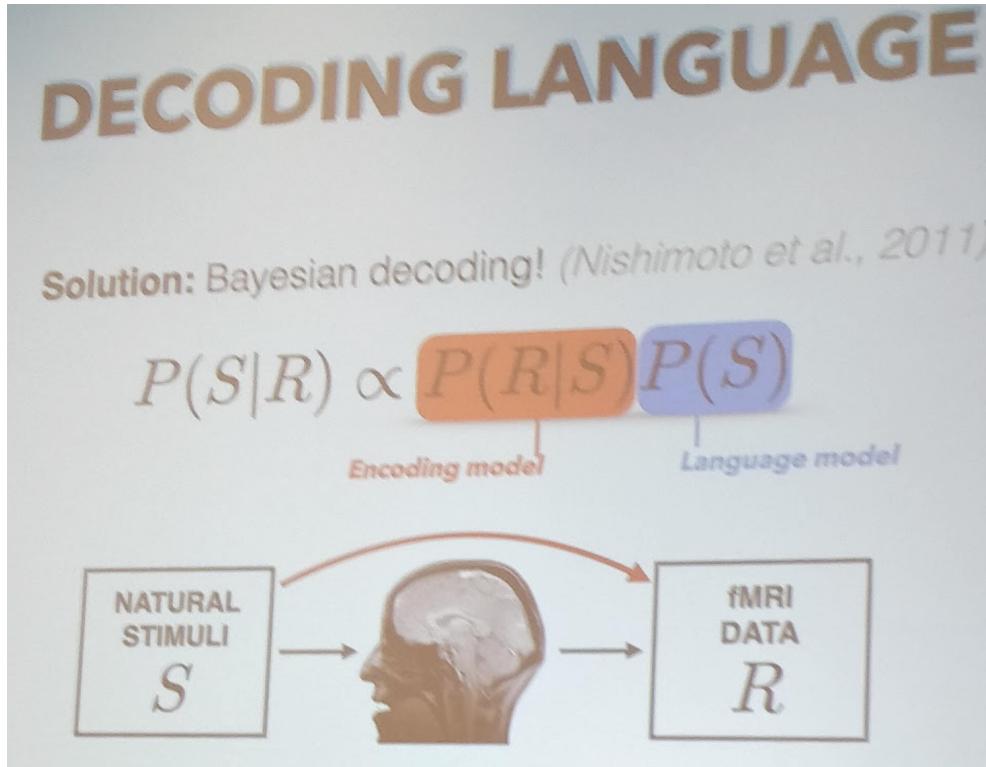
8.28

⌚ Average Time

01:14:88



Fun example of Bayes' theorem application in decoding of brain recordings



Actual stimulus

i got up from the air mattress and pressed my face against the glass of the bedroom window expecting to see eyes staring back at me but instead finding only darkness

Decoded stimulus

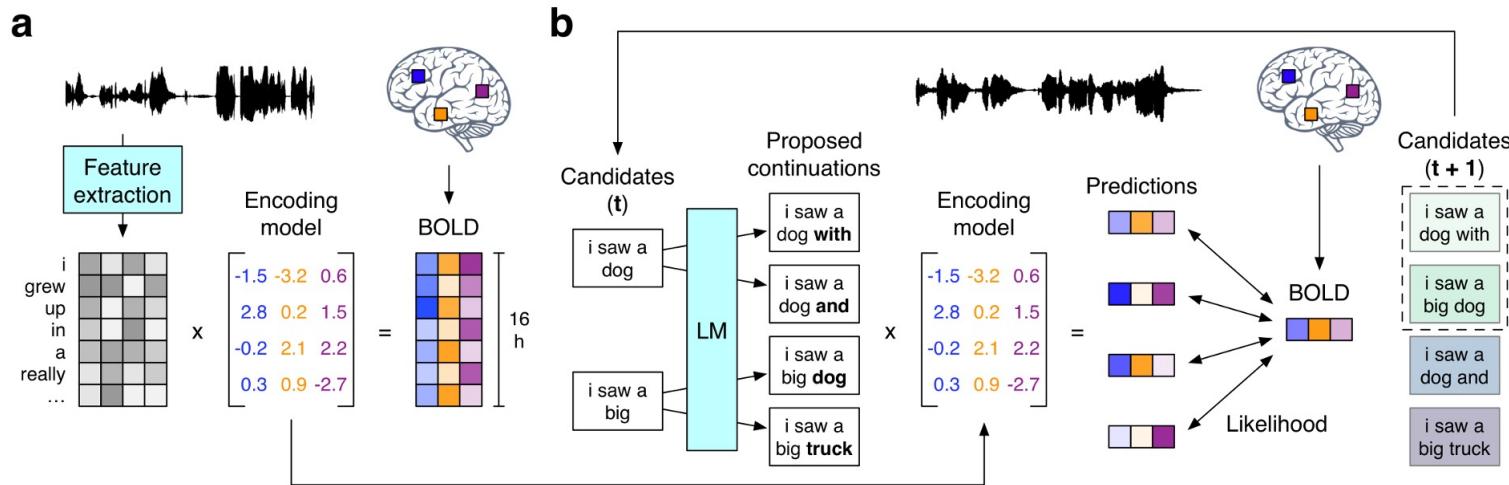
i just continued to walk up to the window and open the glass i stood on my toes and peered out i didn't see anything and looked up again i saw nothing

Self-supervised models of audio effectively explain human cortical responses to speech

Aditya R Vaidya, Shailee Jain, Alexander G Huth

<https://arxiv.org/pdf/2205.14252.pdf>

Fun example of Bayes' theorem application in decoding of brain recordings



c		Actual stimulus	Decoded stimulus	
<i>i got up from the air mattress and pressed my face against the glass of the bedroom window expecting to see eyes staring back at me but instead finding only darkness</i>		<i>i just continued to walk up to the window and open the glass i stood on my toes and peered out i didn't see anything and looked up again i saw nothing</i>		Exact
<i>i didn't know whether to scream cry or run away instead i said leave me alone i don't need your help adam disappeared and i cleaned up alone crying</i>		<i>started to scream and cry and then she just said i told you to leave me alone you can't hurt me i'm sorry and then he stormed off i thought he had left i started to cry</i>		Gist
<i>that night i went upstairs to what had been our bedroom and not knowing what else to do i turned out the lights and lay down on the floor</i>		<i>we got back to my dorm room i had no idea where my bed was i just assumed i would sleep on it but instead i lay down on the floor</i>		Error
<i>i don't have my driver's license yet and i just jumped out right when i needed to and she says well why don't you come back to my house and i'll give you a ride i say ok</i>		<i>she is not ready she has not even started to learn to drive yet i had to push her out of the car i said we will take her home now and she agreed</i>		

Learning

- Generalization from experience

“Our experience of the world is specific, yet we are able to formulate general theories that account for the past and predict the future.”

--M.R. Genesereth and N.J. Nilsson, *Logical Foundations of AI*, 1987

- Agent has made observations (**data**)
- Now must make sense of it (**hypotheses**)

Tasks & settings

Classification
Ranking
Clustering
Regression
Decision-making

Supervised
Unsupervised
Semi-supervised
Active
Reinforcement

Techniques

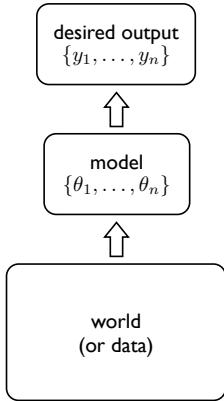
Bayesian learning
Decision trees
Neural networks
Support vector machines
Boosting
Case-based reasoning
Dimensionality reduction
...

Applications

Document retrieval
Document classification
Data mining
Computer vision
Scientific discovery
Robotics
...

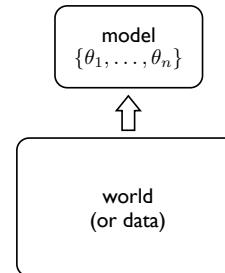
Supervised

Given pairs (x, y) with $y=f(x)$, agent builds model to predict $f(x)$ for new x



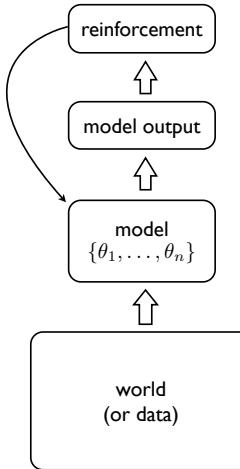
Unsupervised

Given data points x , agent learns patterns in the data (e.g. clusters)



Reinforcement

Agent performs actions $a_1 \dots a_n$, receives reward R ; decides which actions



Semi-supervised

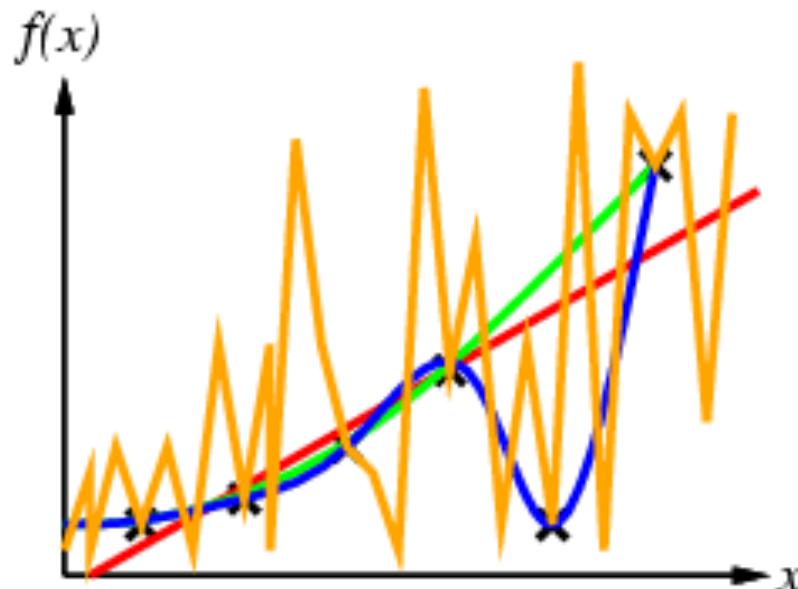
Some labels are missing in the training set, or some labels are erroneous

Active

Learner asks an oracle for correct output $y=f(x)$ for some input points x

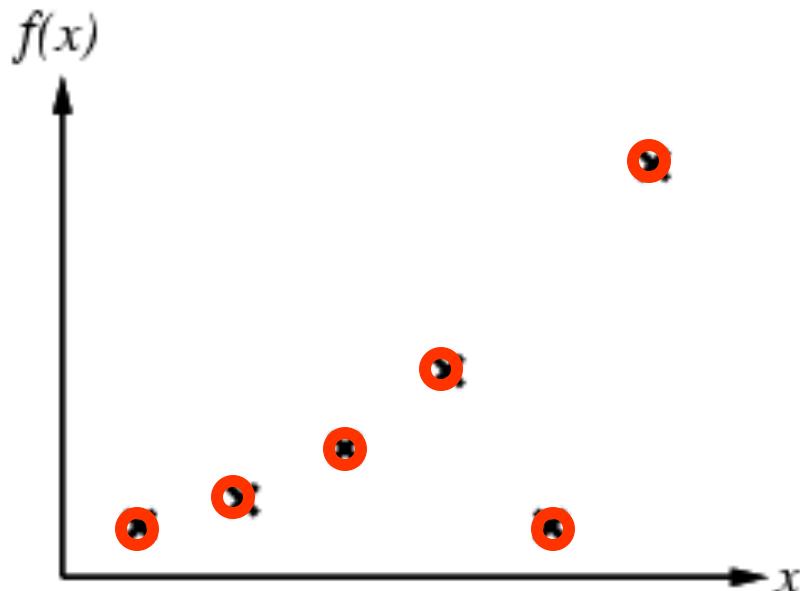
Supervised learning

- Construct/adjust h to agree with f on training set
- h is **consistent** if it agrees with f on all examples
- E.g., curve fitting:



Supervised learning

- Construct/adjust h to agree with f on training set
- h is **consistent** if it agrees with f on all examples
- E.g., curve fitting:



$h=D$ is a trivial, but perhaps uninteresting solution (memorization)

Tasks

- Depending on the task, the goal is to produce values for $f(x)$ of different forms:
 - **Regression:** Predict continuous values
 - **Classification:** Predict one of a discrete set of labels
 - **Binary Classification:** Predict positive or negative
 - **Structured:** Predict complex structure (e.g. tree, sequence, etc.)

Logic-Based Inductive Learning

- Here, examples $(x, f(x))$ take on discrete values

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

Learning a Logical Predicate (Concept Classifier)

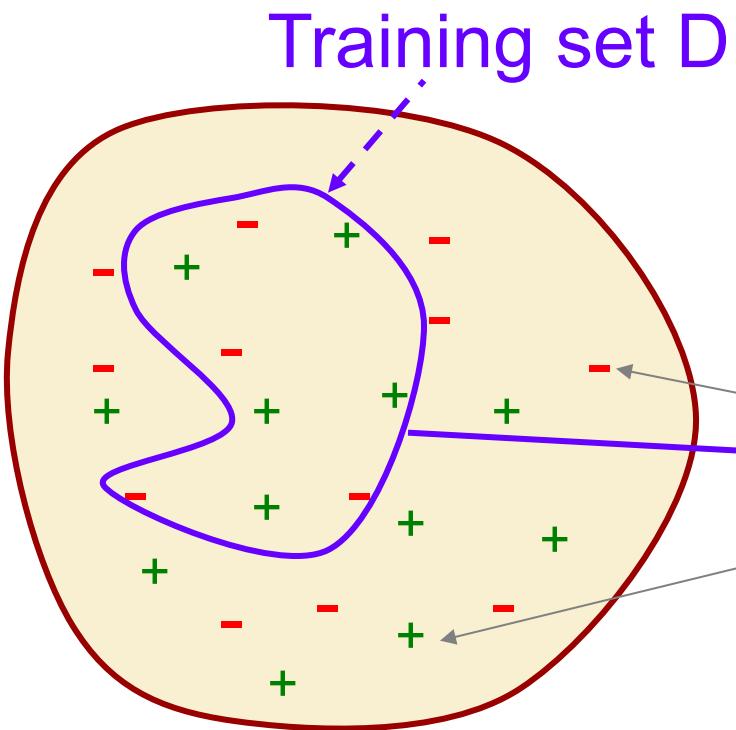
- Given set E of objects, and predicates $A(x)$, $B(x)$, ... for $x \in E$,
the goal is to predict $\text{CONCEPT}(x) \in \{\text{T}, \text{F}\}$, where CONCEPT is a sentence of predicates, e.g.:

$$\text{CONCEPT}(x) \Leftrightarrow A(x) \wedge (\neg B(x) \vee C(x))$$

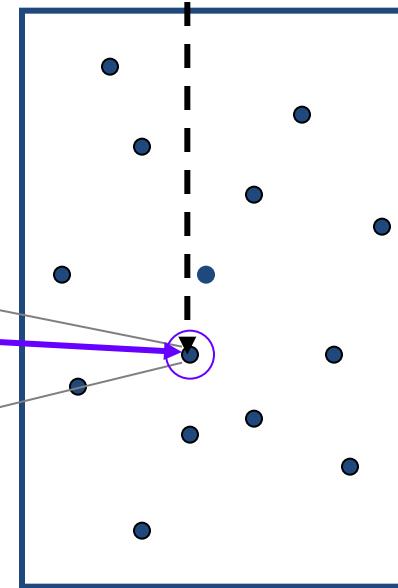
$$\text{CONCEPT}(x) \Leftrightarrow A(x) \text{ and } (\text{not } B(x) \text{ or } C(x))$$

- Training set:** values of CONCEPT for subset of predicate values
 - A hypothesis** is a possible model, $\text{CONCEPT}(x) \Leftrightarrow S(A, B, \dots)$
 - The set of all hypotheses is called the **hypothesis space**
 - A hypothesis **agrees** with an example if it gives the correct value of CONCEPT
-
- https://www.rapidtables.com/math/symbols/Logic_Symbols.html

Inductive Learning Scheme



Inductive hypothesis h



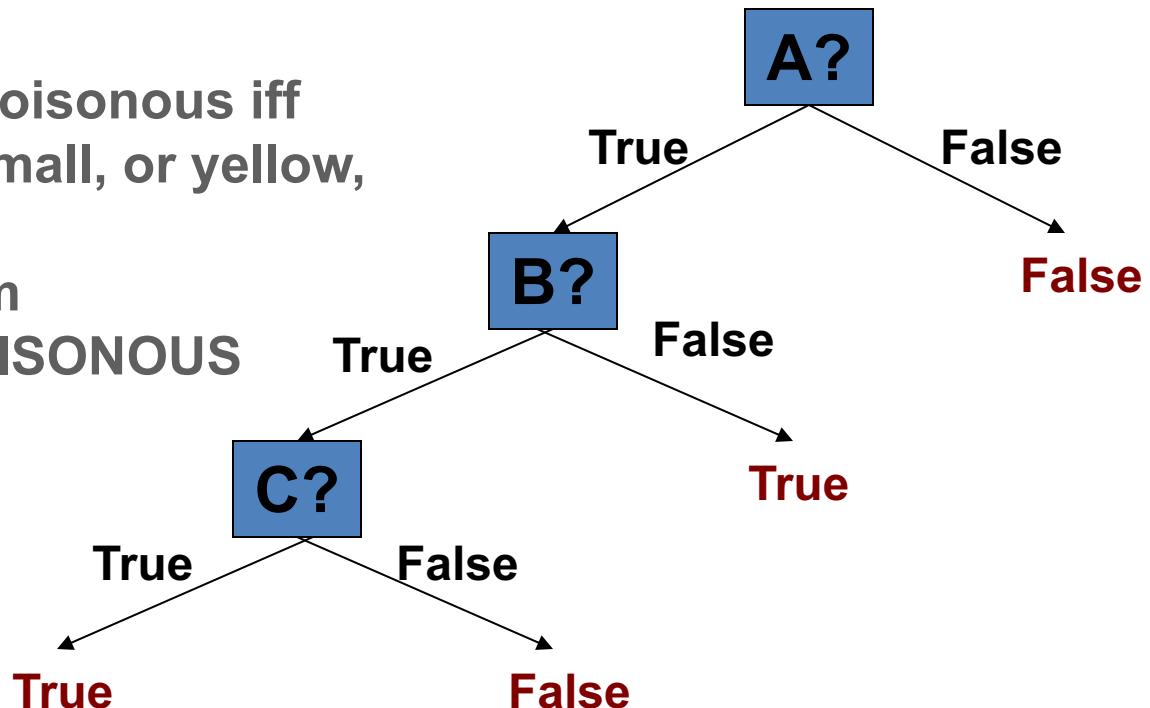
Predicate as a Decision Tree

The predicate $\text{CONCEPT}(x) \Leftrightarrow A(x) \wedge (\neg B(x) \vee C(x))$ can be represented by the following decision tree:

Example:

A mushroom is poisonous iff it is yellow and small, or yellow, big and spotted

- x is a mushroom
- $\text{CONCEPT} = \text{POISONOUS}$
- $A = \text{YELLOW}$
- $B = \text{BIG}$
- $C = \text{SPOTTED}$



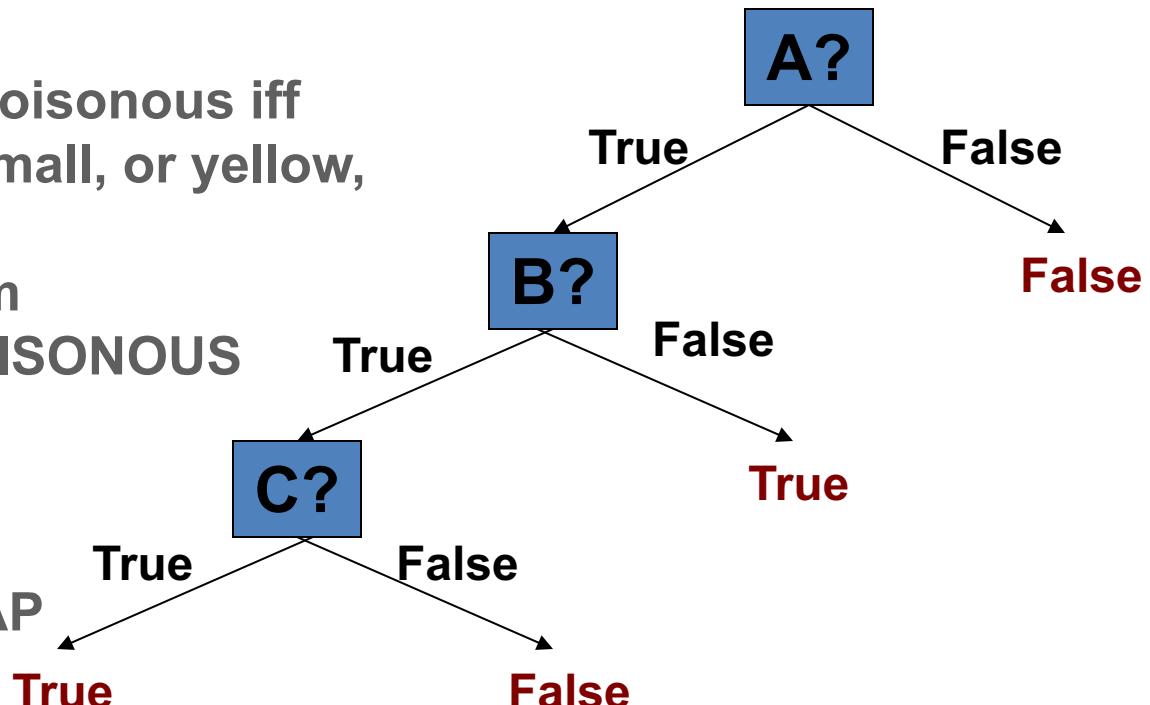
Predicate as a Decision Tree

The predicate $\text{CONCEPT}(x) \Leftrightarrow A(x) \wedge (\neg B(x) \vee C(x))$ can be represented by the following decision tree:

Example:

A mushroom is poisonous iff it is yellow and small, or yellow, big and spotted

- x is a mushroom
- $\text{CONCEPT} = \text{POISONOUS}$
- $A = \text{YELLOW}$
- $B = \text{BIG}$
- $C = \text{SPOTTED}$
- $D = \text{FUNNEL-CAP}$
- $E = \text{BULKY}$



Training Set

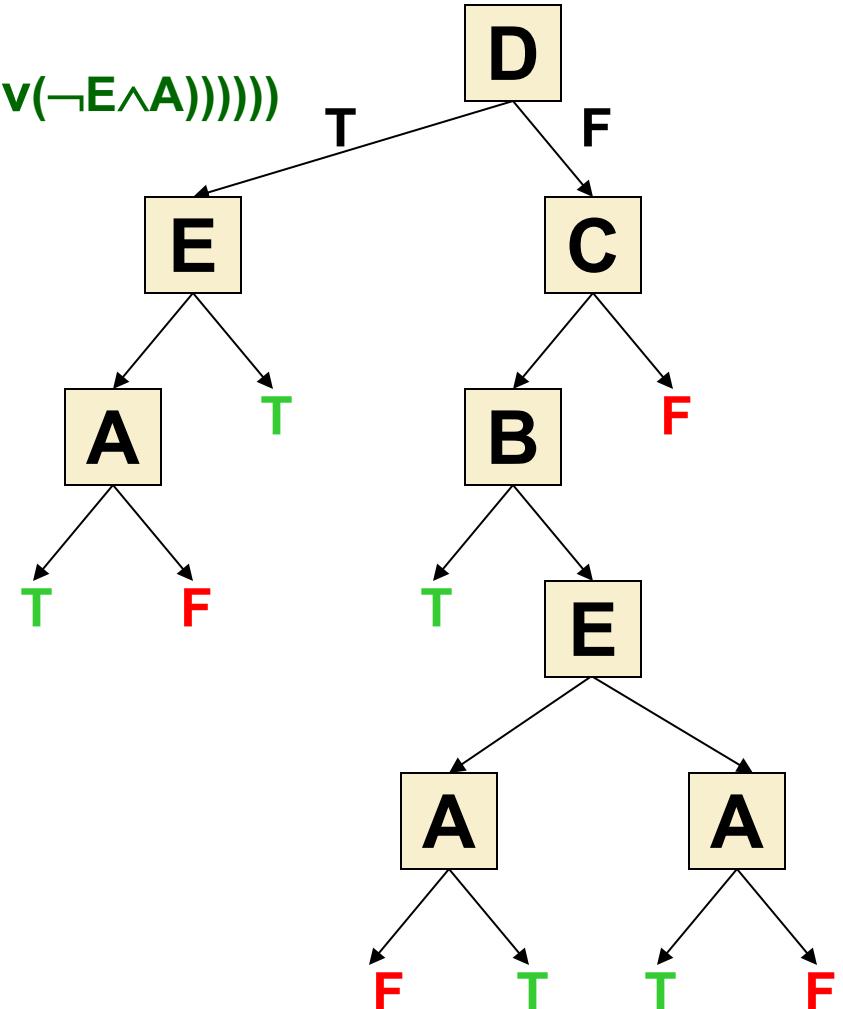
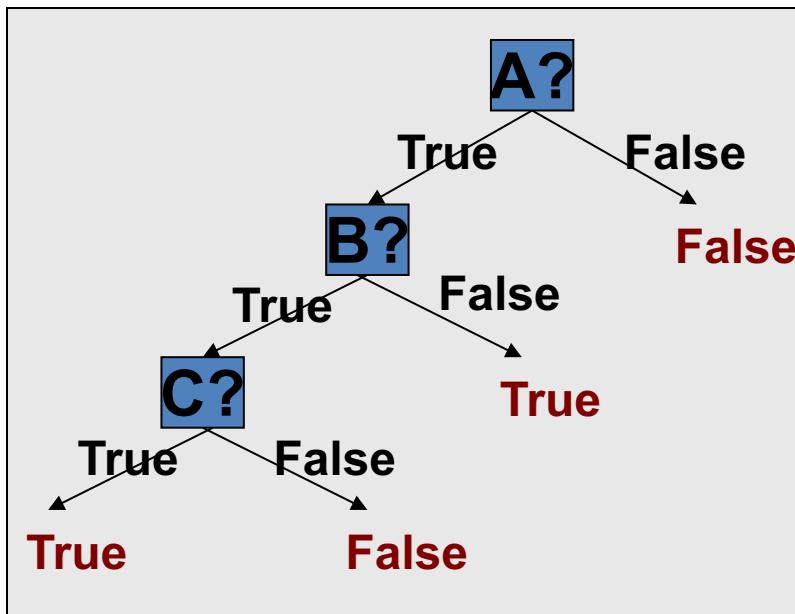
Ex. #	A	B	C	D	E	CONCEPT
1	False	False	True	False	True	False
2	False	True	False	False	False	False
3	False	True	True	True	True	False
4	False	False	True	False	False	False
5	False	False	False	True	True	False
6	True	False	True	False	False	True
7	True	False	False	True	False	True
8	True	False	True	False	True	True
9	True	True	True	False	True	True
10	True	True	True	True	True	True
11	True	True	False	False	False	False
12	True	True	False	False	True	False
13	True	False	True	True	True	True

Possible Decision Tree

CONCEPT \Leftrightarrow

$$(D \wedge (\neg E \vee A)) \vee (\neg D \wedge (C \wedge (B \vee (\neg B \wedge ((E \wedge \neg A) \vee (\neg E \wedge A))))))$$

CONCEPT $\Leftrightarrow A \wedge (\neg B \vee C)$

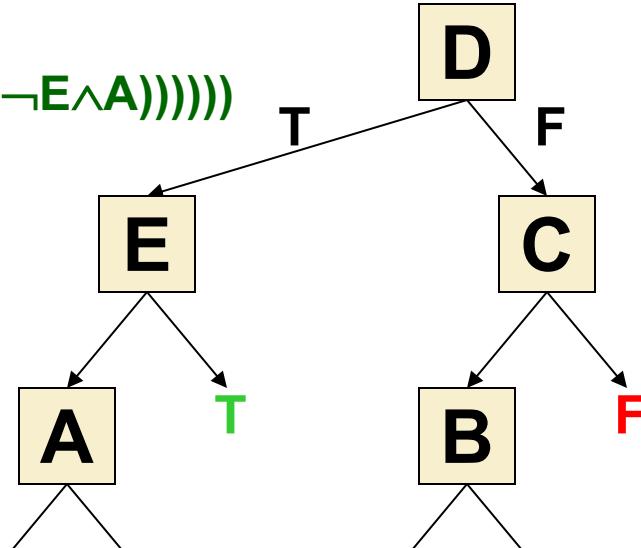


Possible Decision Tree

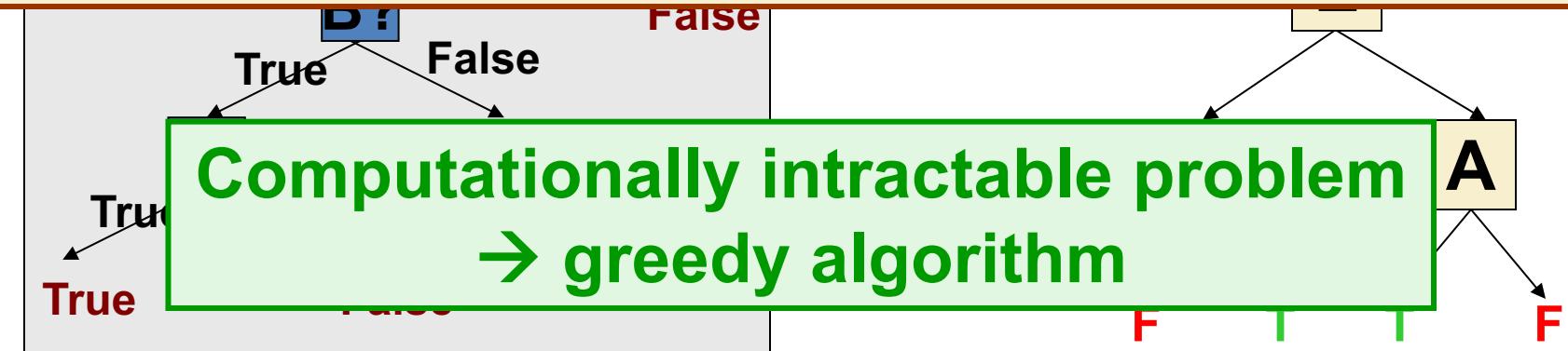
CONCEPT \Leftrightarrow

$$(D \wedge (\neg E \vee A)) \vee (\neg D \wedge (C \wedge (B \vee (\neg B \wedge ((E \wedge \neg A) \vee (\neg E \wedge A))))))$$

CONCEPT $\Leftrightarrow A \wedge (\neg B \vee C)$



KIS (keep it simple) bias \rightarrow Build smallest decision tree

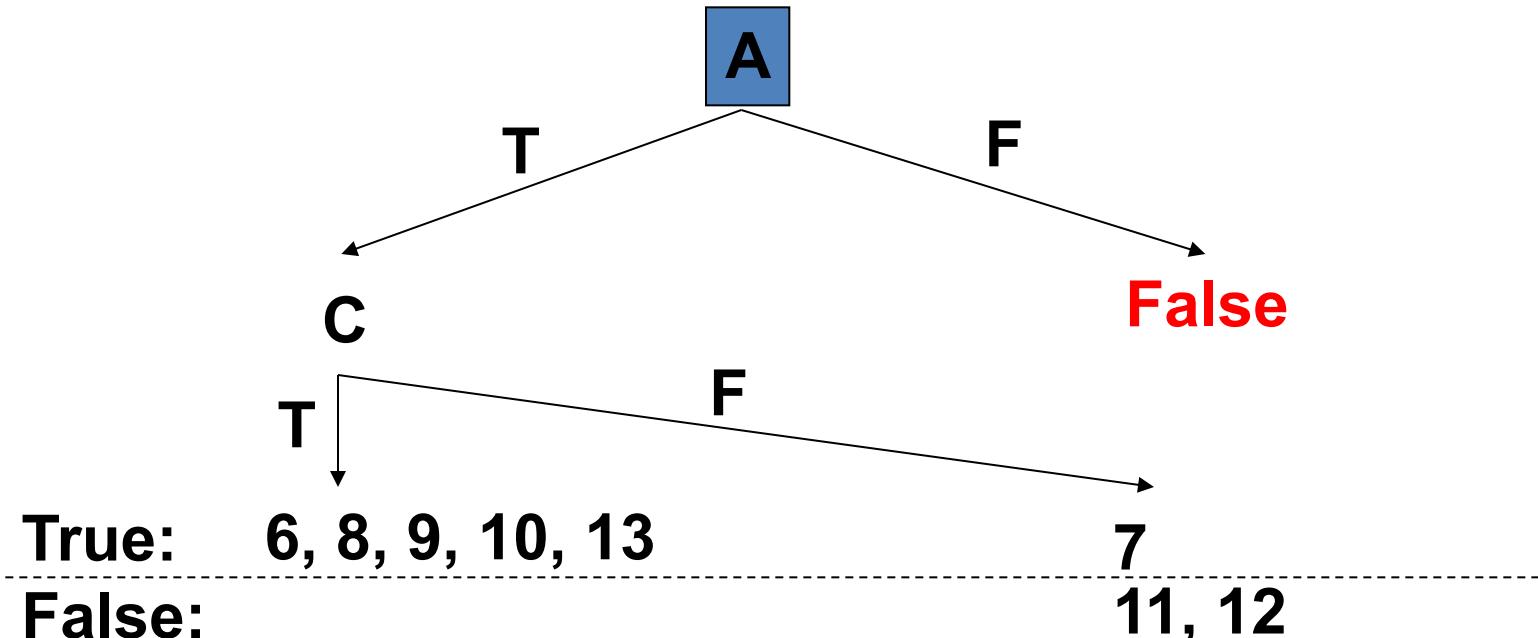


Top-down induction of decision tree

Ex. #	A	B	C	D	E	CONCEPT
1	<u>False</u>	<u>False</u>	True	<u>False</u>	True	<u>False</u>
2	<u>False</u>	True	<u>False</u>	<u>False</u>	<u>False</u>	<u>False</u>
3	<u>False</u>	True	True	True	True	<u>False</u>
4	<u>False</u>	<u>False</u>	True	<u>False</u>	<u>False</u>	<u>False</u>
5	<u>False</u>	<u>False</u>	<u>False</u>	True	True	<u>False</u>
6	True	<u>False</u>	True	<u>False</u>	<u>False</u>	True
7	True	<u>False</u>	<u>False</u>	True	<u>False</u>	True
8	True	<u>False</u>	True	<u>False</u>	True	True
9	True	True	True	<u>False</u>	True	True
10	True	True	True	True	True	True
11	True	True	<u>False</u>	<u>False</u>	<u>False</u>	<u>False</u>
12	True	True	<u>False</u>	<u>False</u>	True	<u>False</u>
13	True	<u>False</u>	True	True	True	True

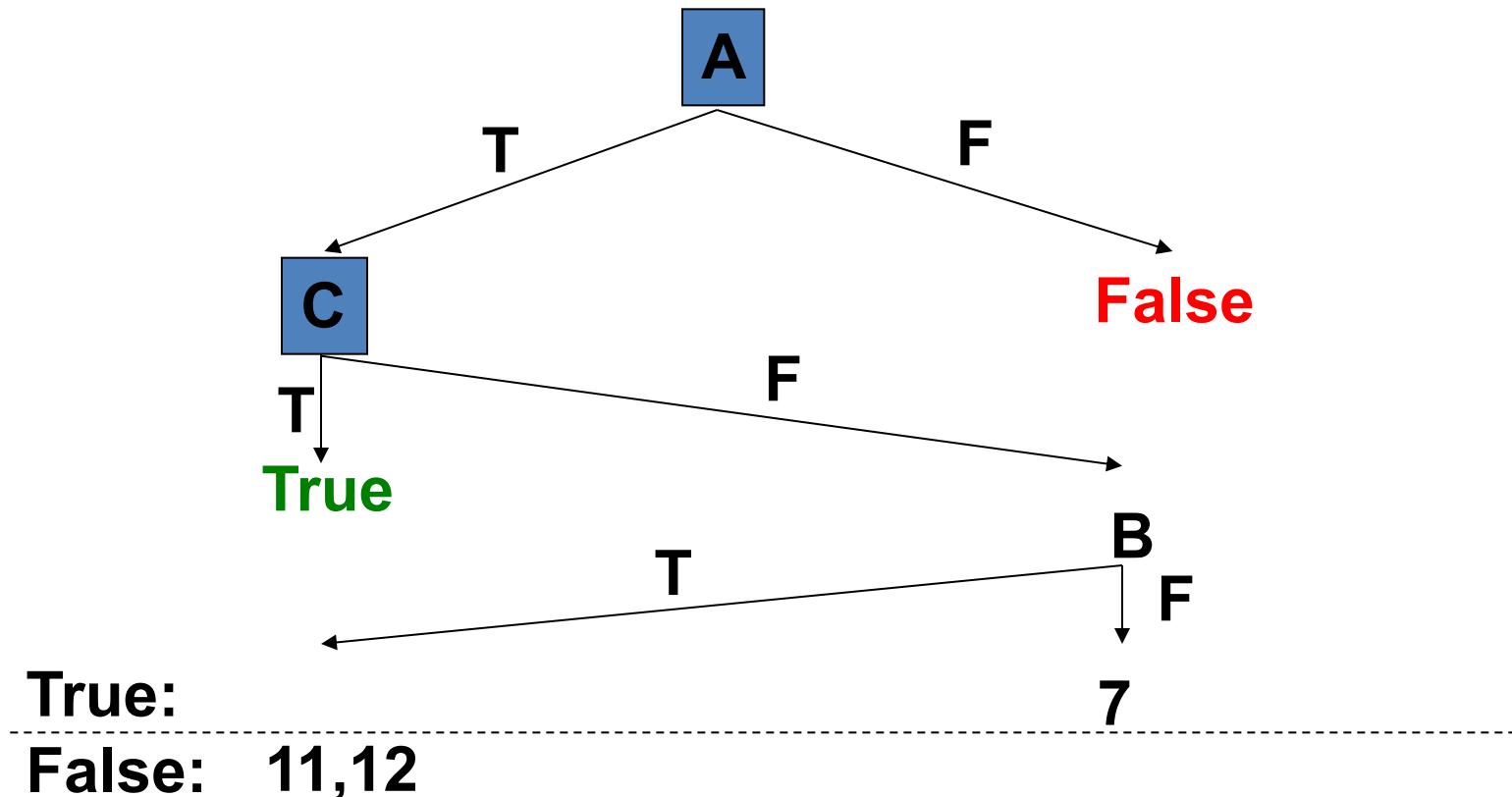
- What should the classifier do if it observes none of the predicates?
- What should it do if it can choose only one predicate?

Choice of Second Predicate

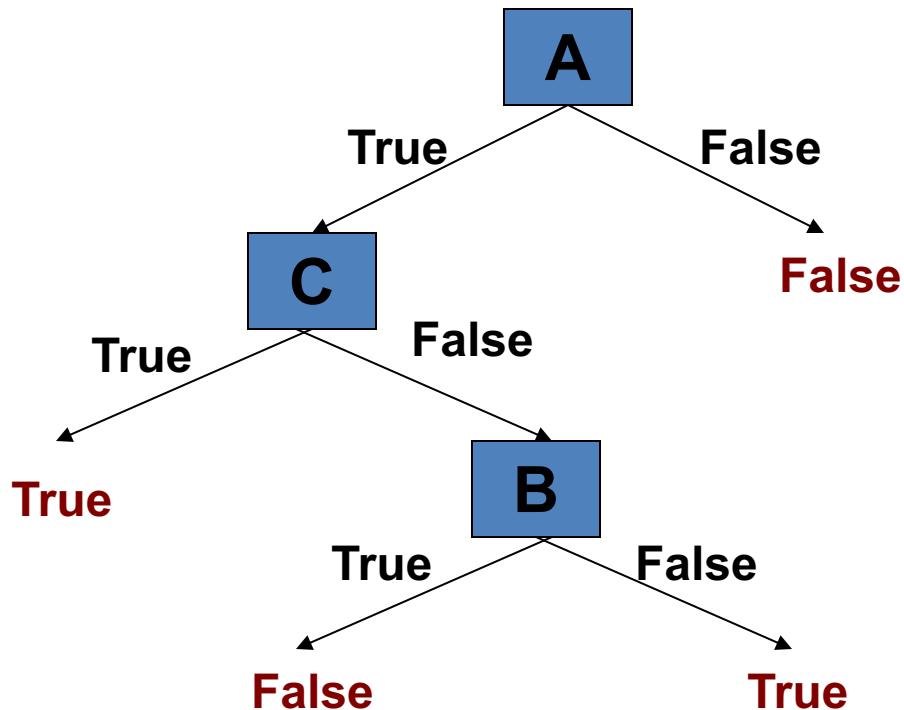


→ The number of misclassified examples from the training set is 1

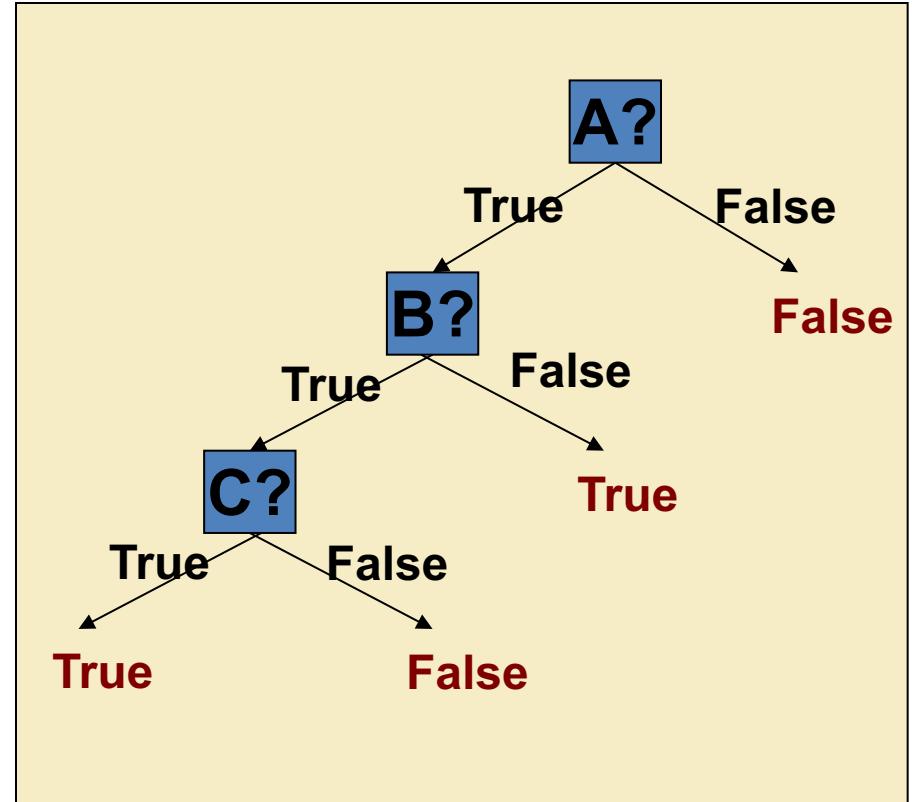
Choice of Third Predicate



Final Tree



CONCEPT $\Leftrightarrow A \wedge (C \vee \neg B)$

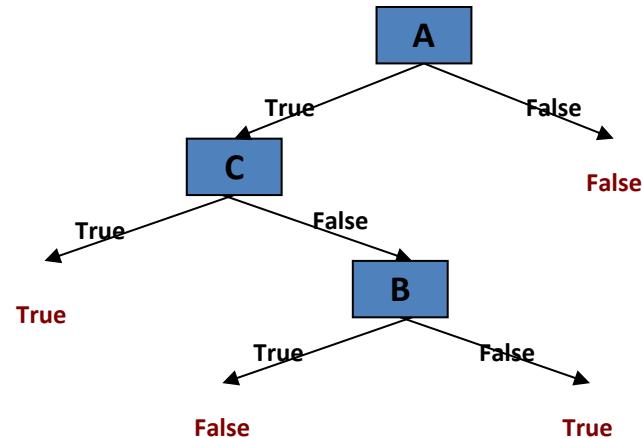


CONCEPT $\Leftrightarrow A \wedge (\neg B \vee C)$

Top-Down Induction of a DT

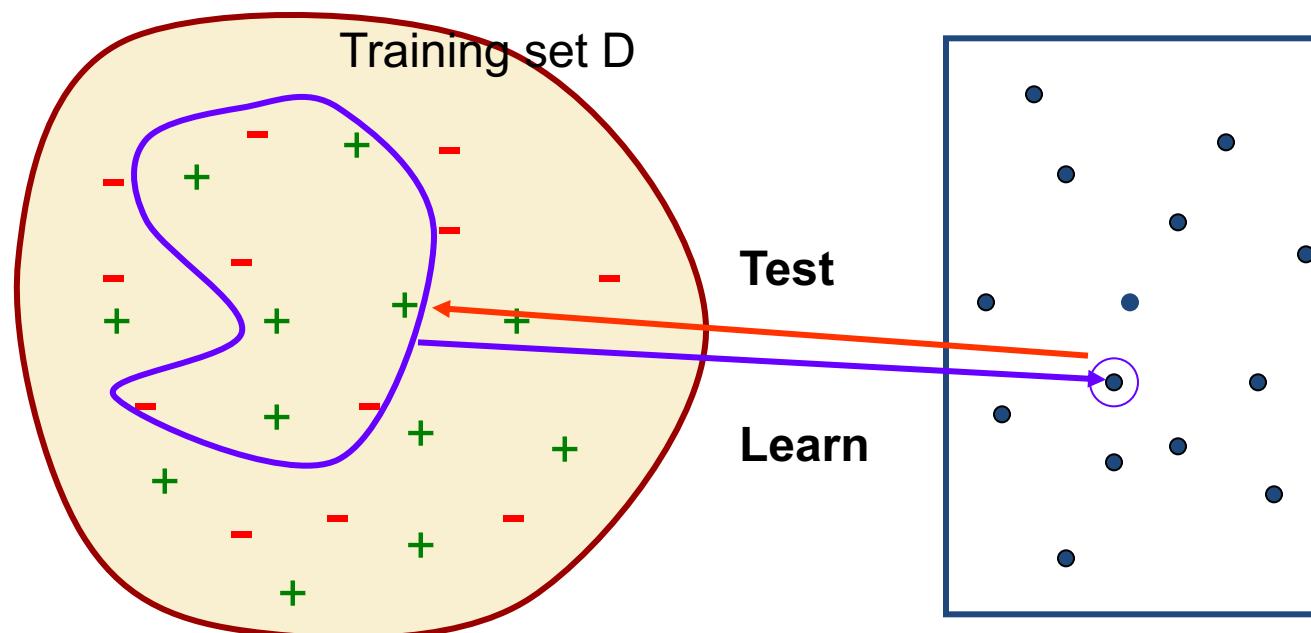
DTL(D, Predicates)

1. If all examples in D are positive then return True
2. If all examples in D are negative then return False
3. If Predicates is empty then return *failure*
4. $A \leftarrow \text{error-minimizing predicate in Predicates}$
5. Return the tree whose:
 - root is A,
 - left branch is $\text{DTL}(D^A, \text{Predicates}-A)$,
 - right branch is $\text{DTL}(D^{-A}, \text{Predicates}-A)$



Capacity is Not the Only Criterion

- Accuracy on training set isn't the best measure of performance

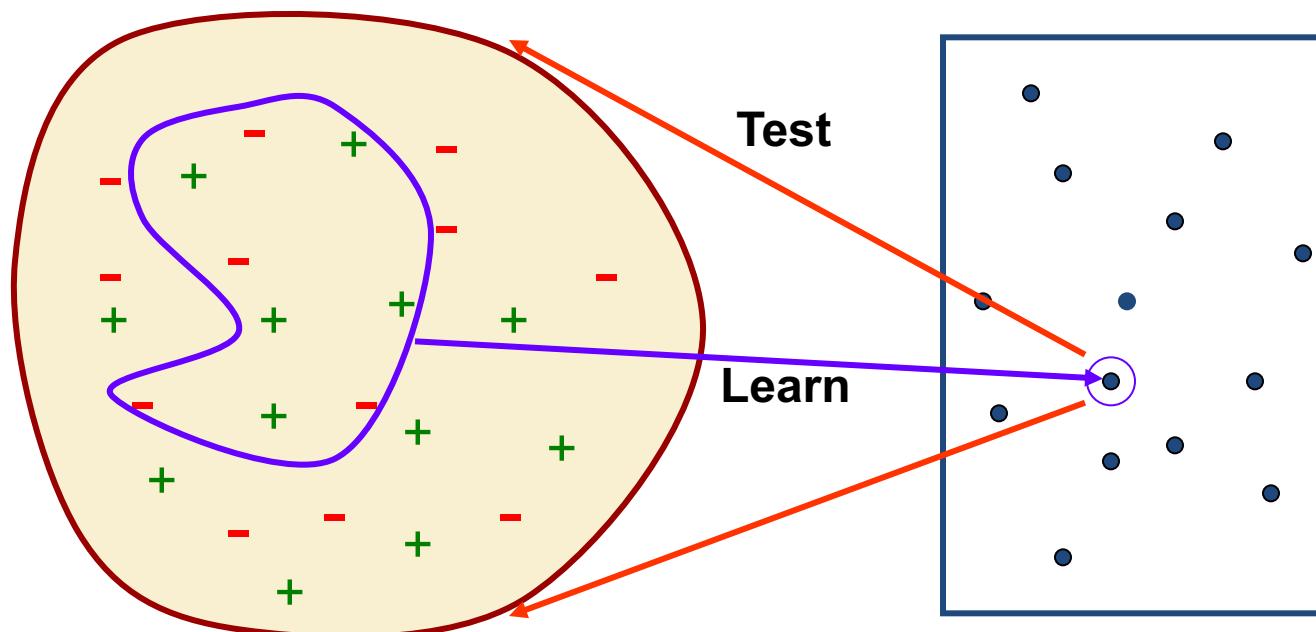


Example set X

Hypothesis space H

Generalization Error

- A hypothesis h is said to *generalize* well if it achieves low error on all examples in X



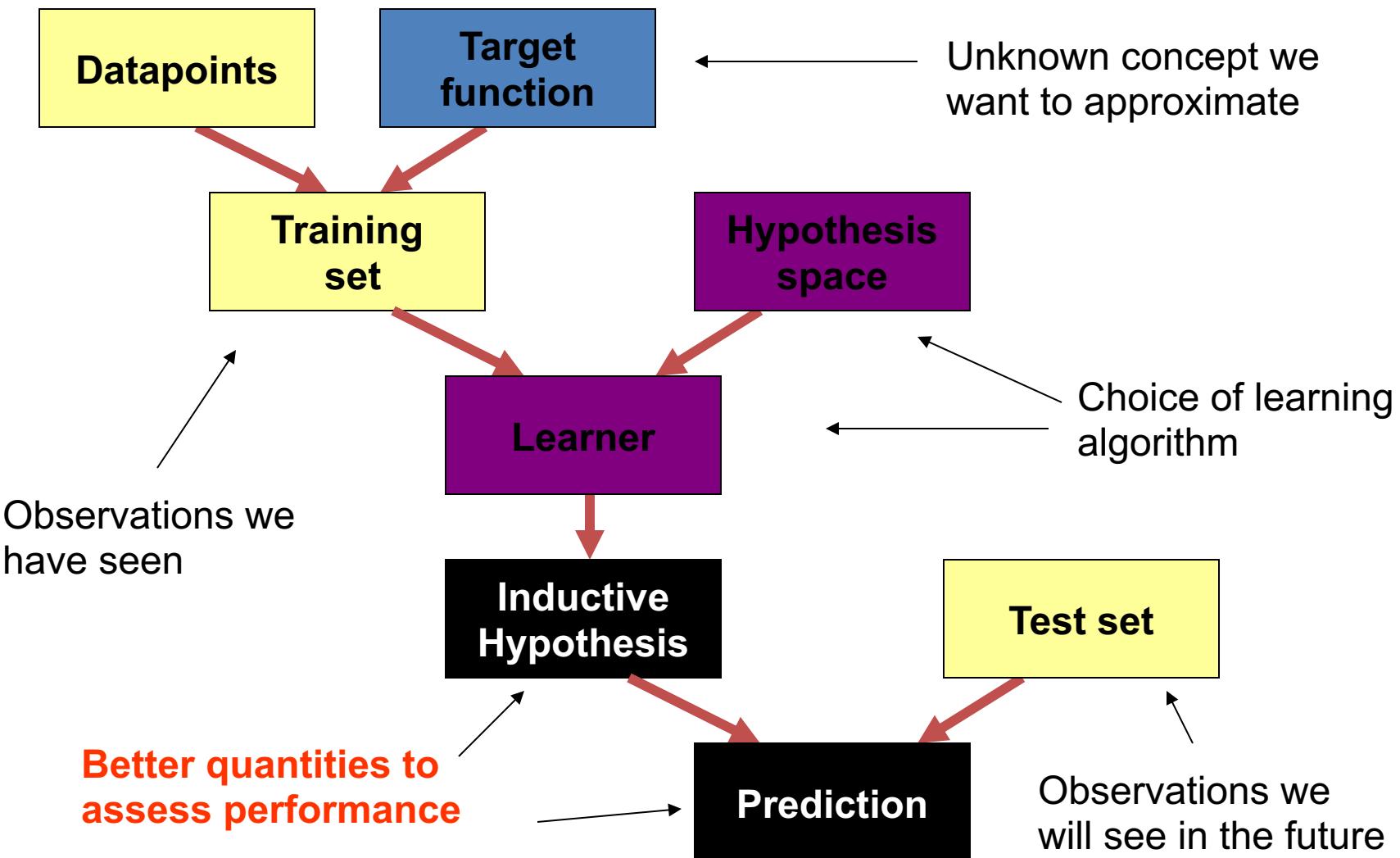
Example set X

Hypothesis space H

Assessing Performance of a Learning Algorithm

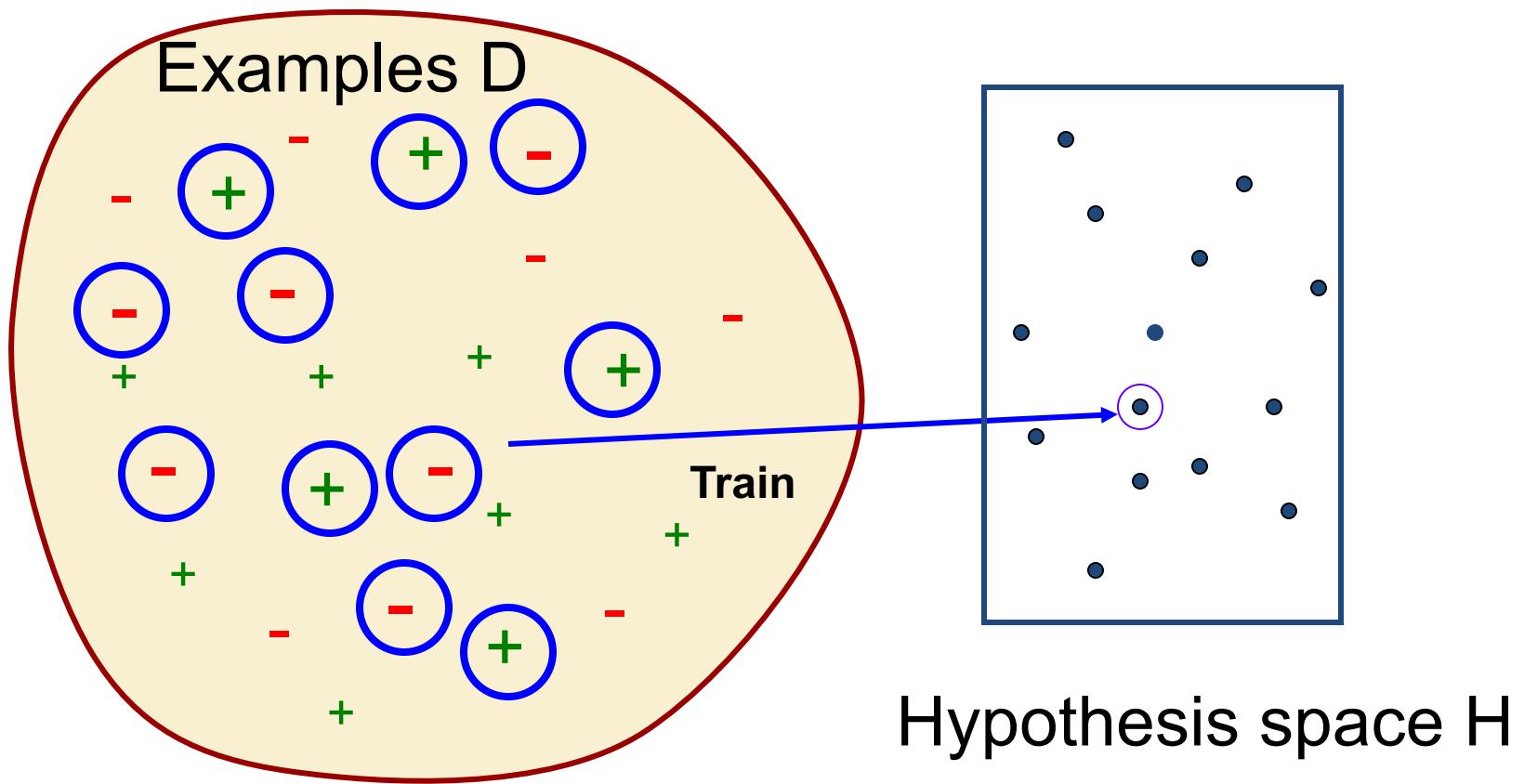
- Samples from X are typically unavailable
- Take out some of the training set
 - Train on the remaining training set
 - Test on the excluded instances
 - *Cross-validation*

Supervised Learning Flow Chart



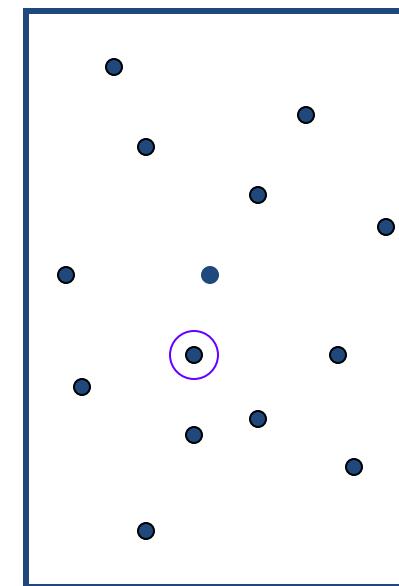
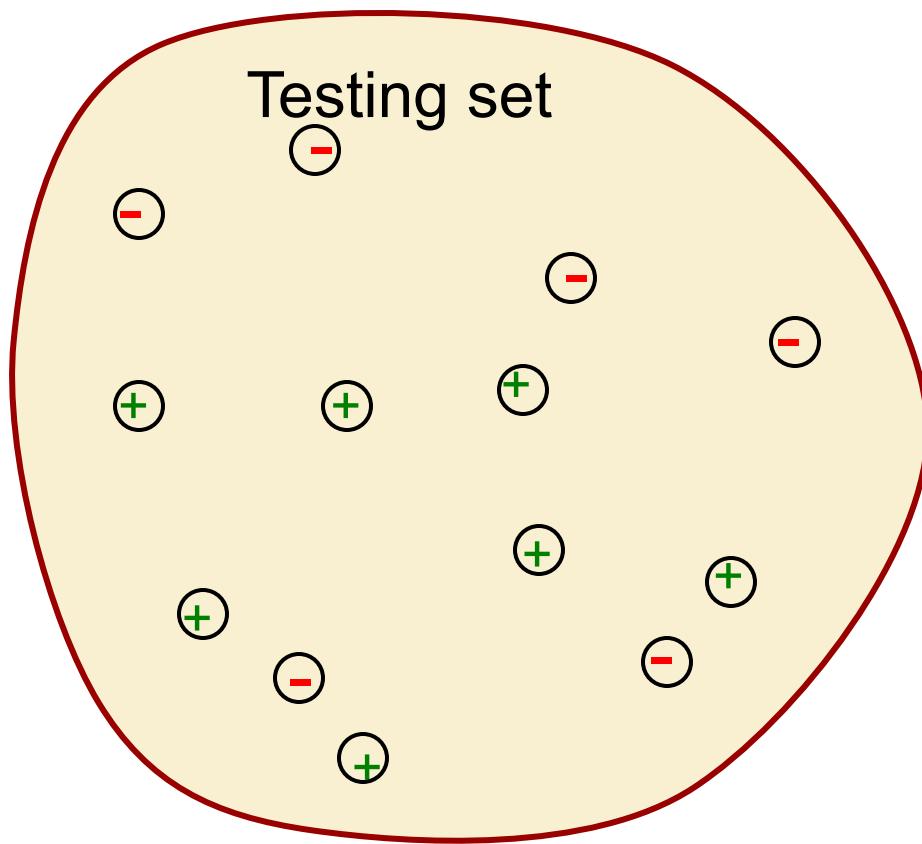
Cross-Validation

- Split original set of examples, train



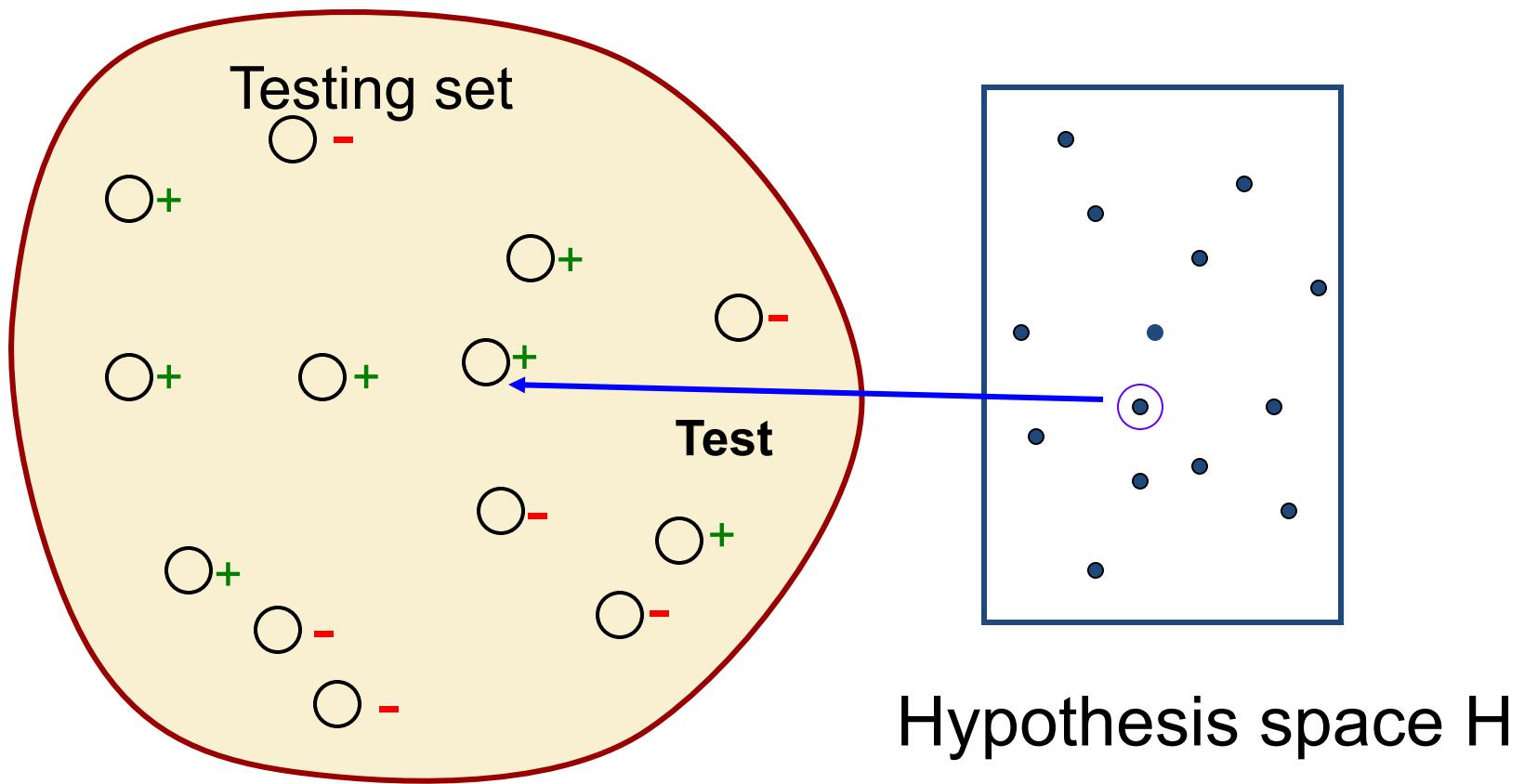
Cross-Validation

- Evaluate hypothesis on testing set



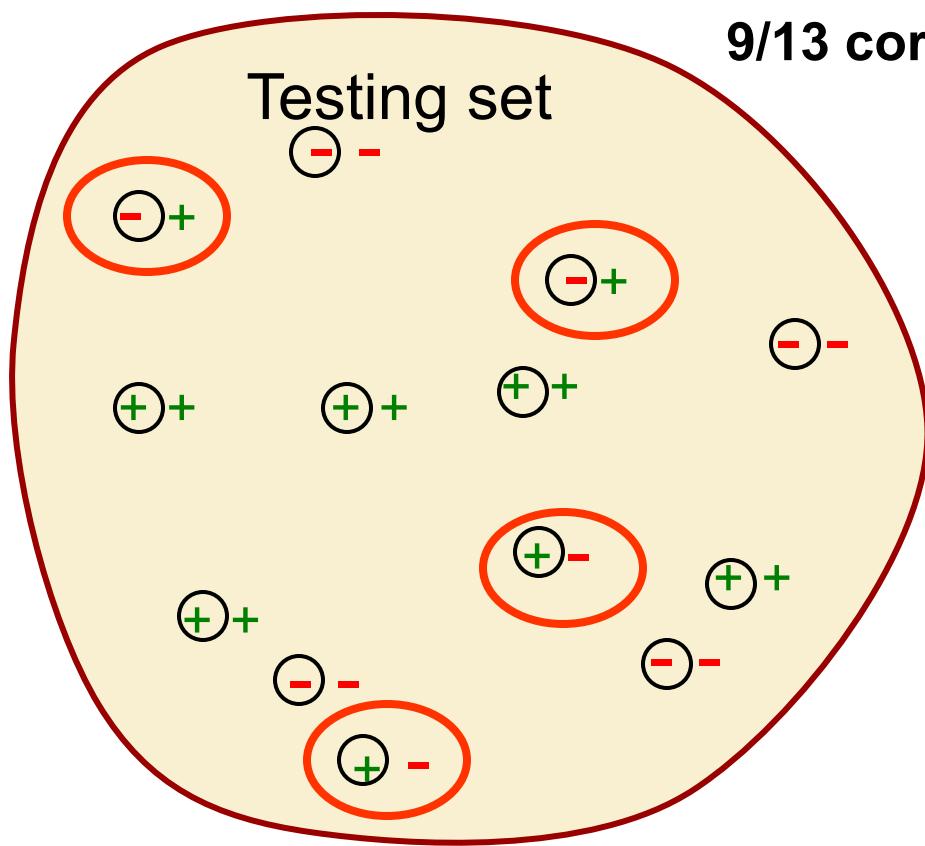
Cross-Validation

- Evaluate hypothesis on testing set



Cross-Validation

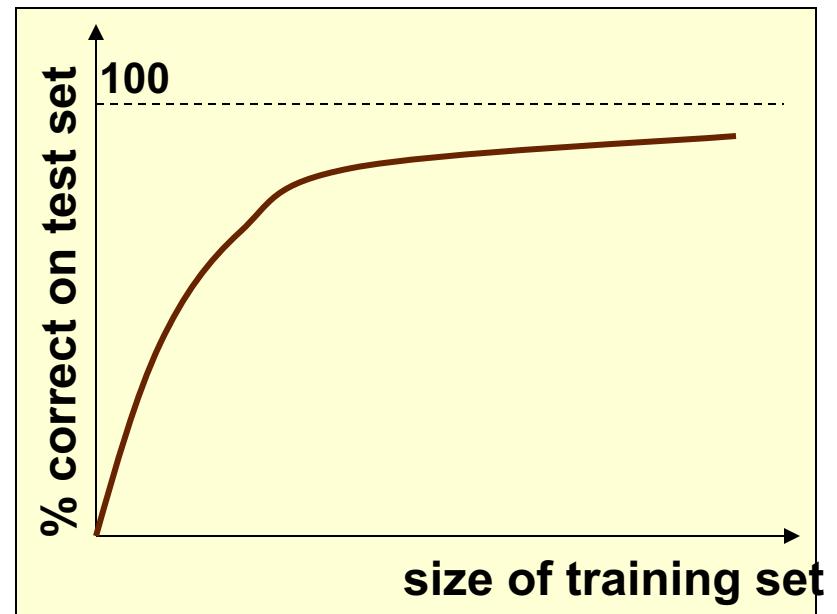
- Compare true concept against prediction



Hypothesis space H

Performance Issues

- Assessing performance:
 - Training set and test set
 - Learning curve

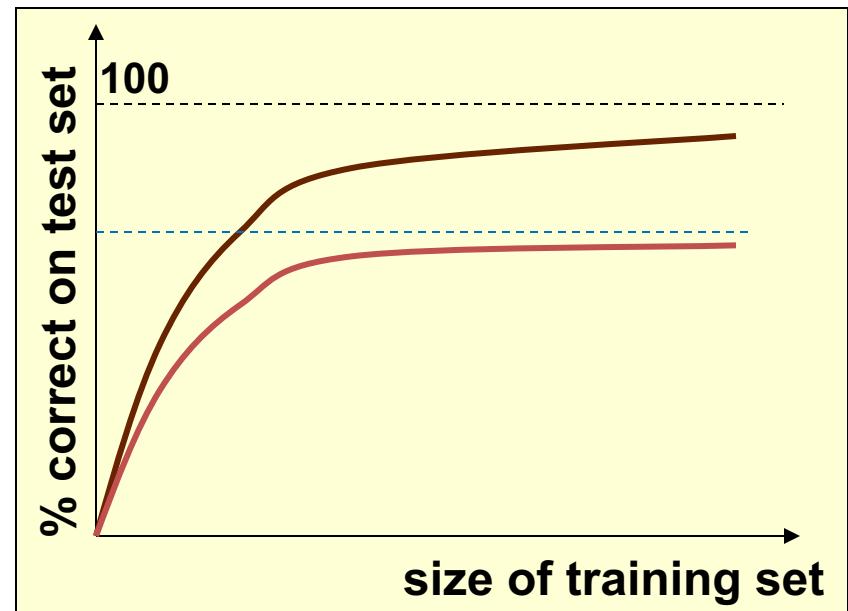


Typical learning curve

Performance Issues

- Assessing performance:
 - Training set and test set
 - Learning curve

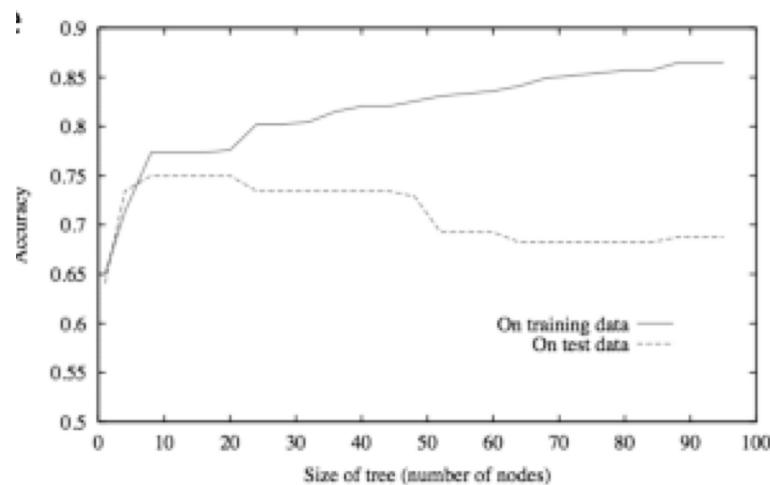
Some concepts are unrealizable within a machine's capacity



Typical learning curve

Performance Issues

- Assessing performance:
 - Training set and test set
 - Learning curve
- Overfitting
 - Classifier works well on training set, poorly on test set



- Next class: more decision trees

Decision Trees – part 2



*"I know you're a new
analyst but there's no
need to hide from
our clients."*

*"I'm not hiding!
I'm trying a new
technique I learned
on the web:
The Decision Tree".*

Announcements

- A2 due November 7 (make sure you are using the latest pdf – there was a change in the first paragraph of part 1, updated on October 25)
- A3 and optional A4 to follow (A4 will have a deadline during the last week of classes)
- Regrade requests are being processed – if it's been more than a week since you submitted your request, you can follow up on Q&A

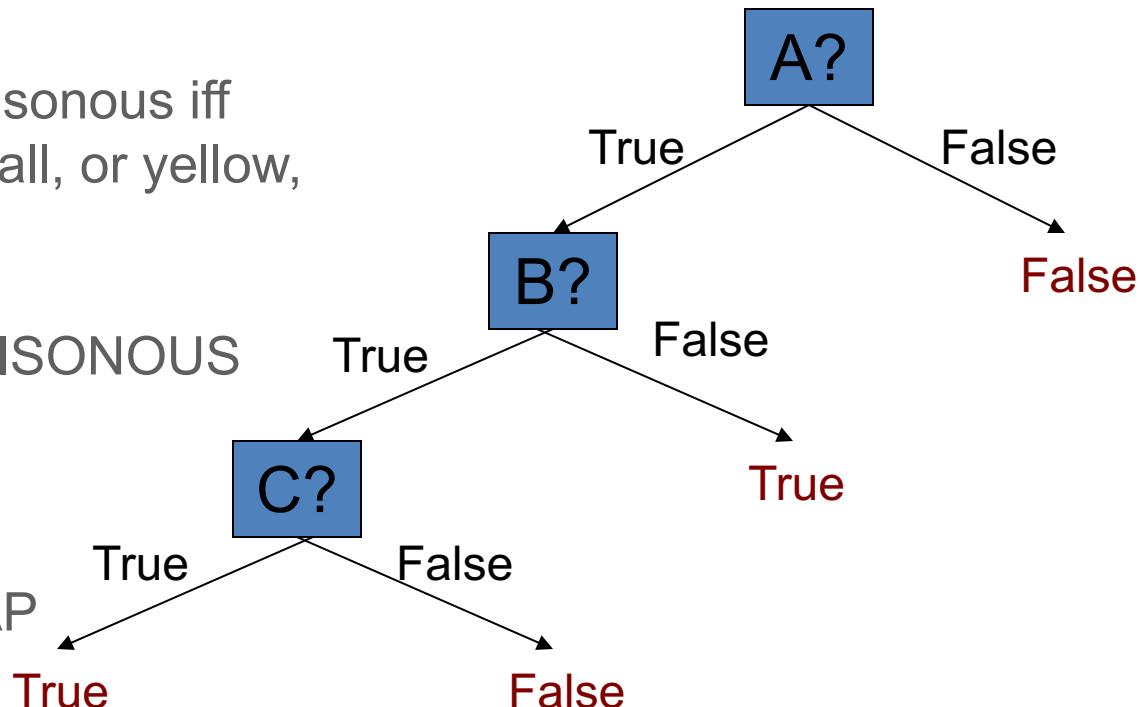
Predicate as a Decision Tree

The predicate $\text{CONCEPT}(x) \Leftrightarrow A(x) \wedge (\neg B(x) \vee C(x))$ can be represented by the following decision tree:

Example:

A mushroom is poisonous iff it is yellow and small, or yellow, big and spotted

- x is a mushroom
- $\text{CONCEPT} = \text{POISONOUS}$
- $A = \text{YELLOW}$
- $B = \text{BIG}$
- $C = \text{SPOTTED}$
- $D = \text{FUNNEL-CAP}$
- $E = \text{BULKY}$



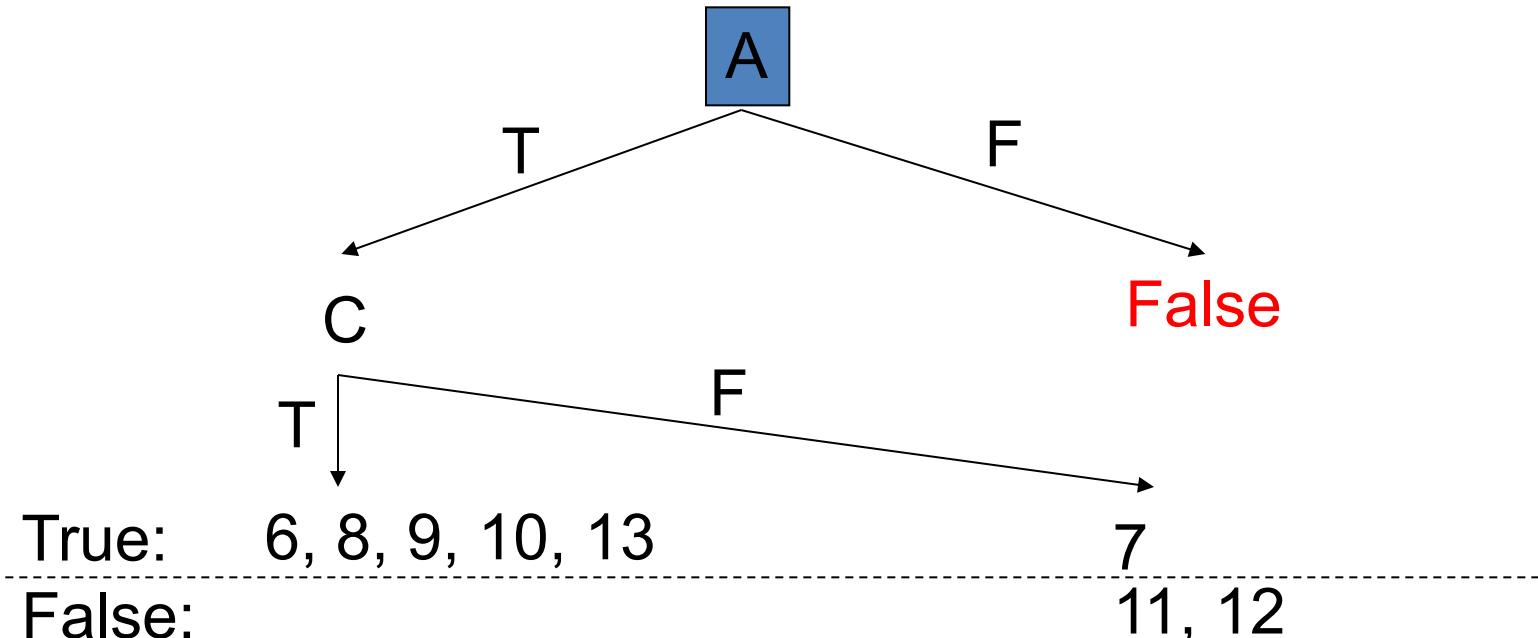
Top-down induction of decision tree

Ex. #	A	B	C	D	E	CONCEPT
1	<u>False</u>	<u>False</u>	True	<u>False</u>	True	<u>False</u>
2	<u>False</u>	True	<u>False</u>	<u>False</u>	<u>False</u>	<u>False</u>
3	<u>False</u>	True	True	True	True	<u>False</u>
4	<u>False</u>	<u>False</u>	True	<u>False</u>	<u>False</u>	<u>False</u>
5	<u>False</u>	<u>False</u>	<u>False</u>	True	True	<u>False</u>
6	True	<u>False</u>	True	<u>False</u>	<u>False</u>	True
7	True	<u>False</u>	<u>False</u>	True	<u>False</u>	True
8	True	<u>False</u>	True	<u>False</u>	True	True
9	True	True	True	<u>False</u>	True	True
10	True	True	True	True	True	True
11	True	True	<u>False</u>	<u>False</u>	<u>False</u>	<u>False</u>
12	True	True	<u>False</u>	<u>False</u>	True	<u>False</u>
13	True	<u>False</u>	True	True	True	True

DTL(D, Predicates)

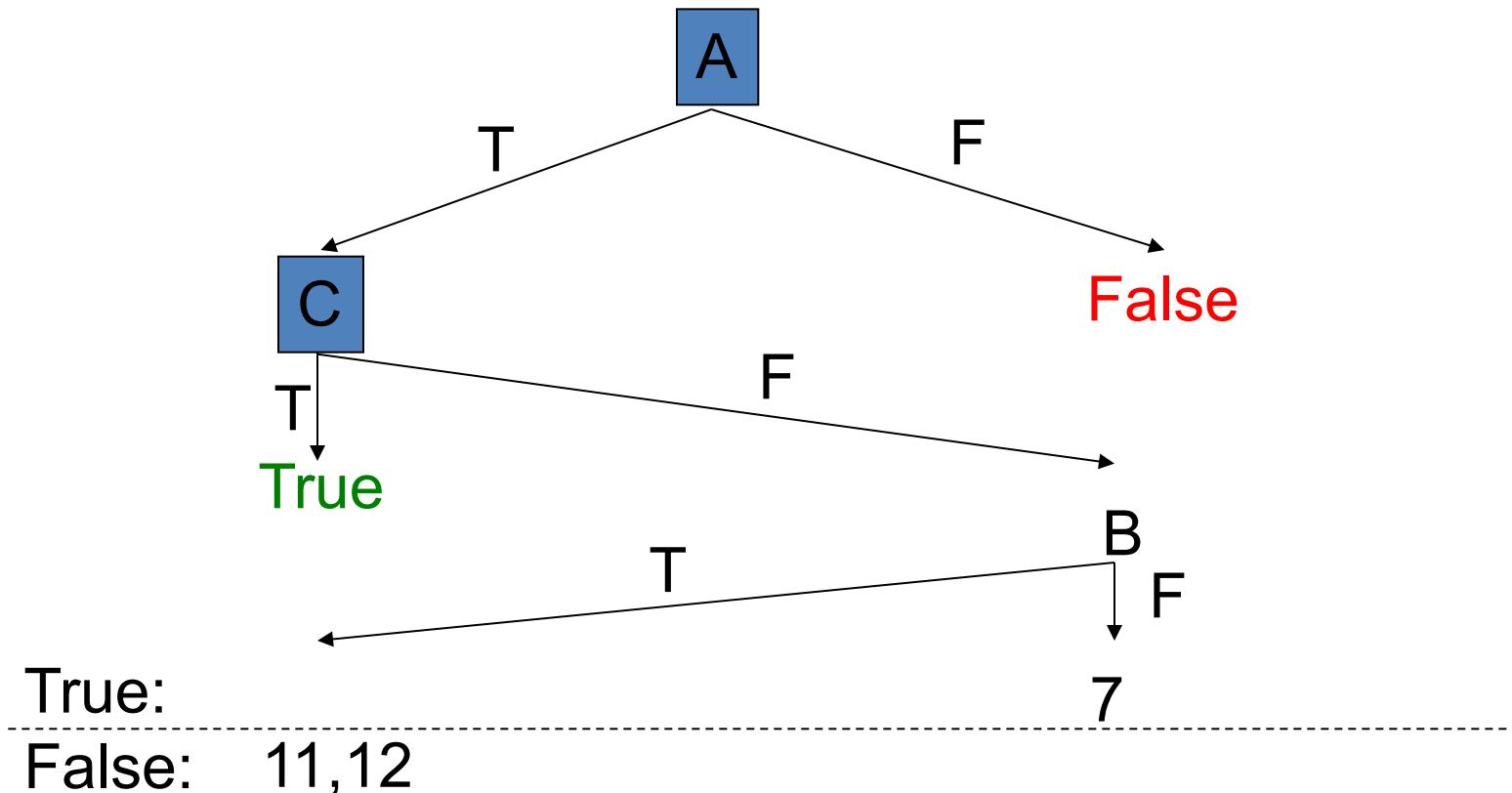
1. If all examples in D are positive then return True
2. If all examples in D are negative then return False
3. If Predicates is empty then return *failure*
4. A \leftarrow error-minimizing predicate in Predicates
5. Return the tree whose:
 - root is A,
 - left branch is DTL(D^{+A} , Predicates-A),
 - right branch is DTL(D^{-A} , Predicates-A)

Choice of Second Predicate

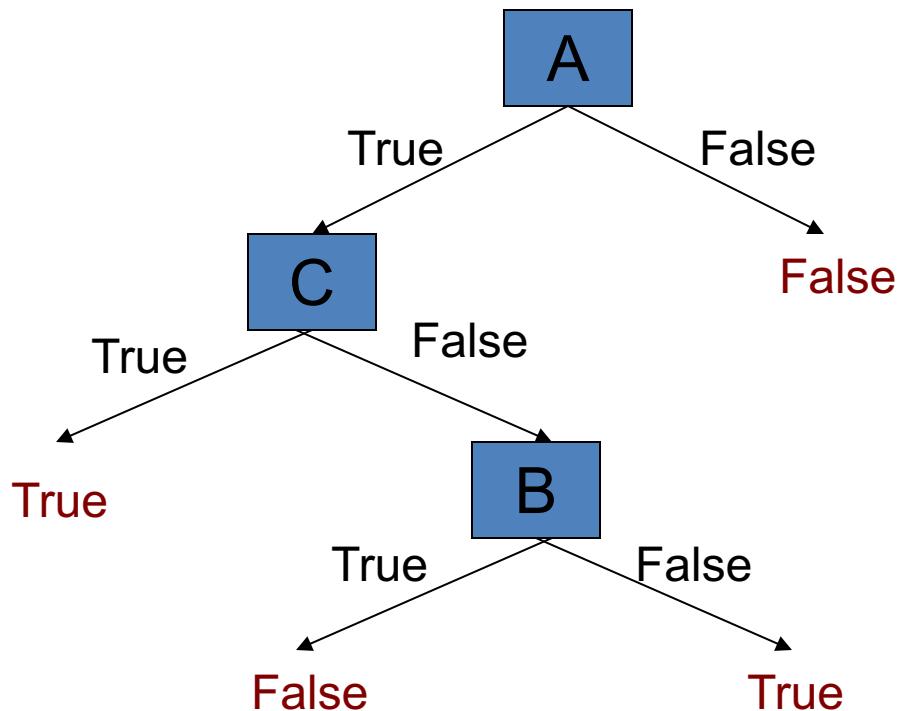


→ The number of misclassified examples from the training set is 1

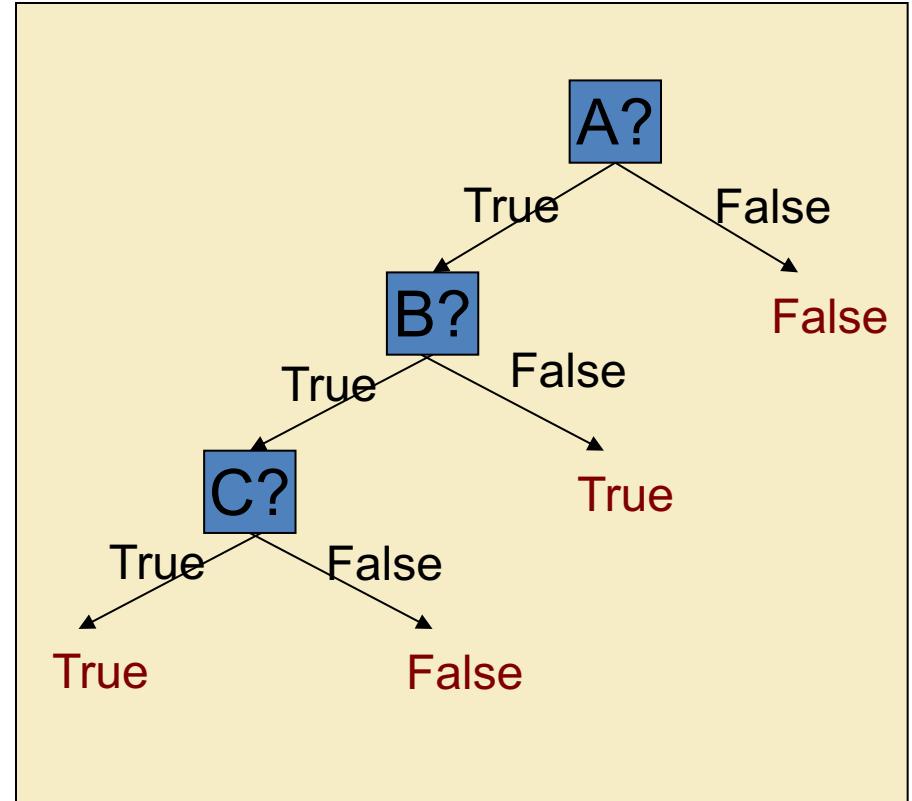
Choice of Third Predicate



Final Tree



CONCEPT $\Leftrightarrow A \wedge (C \vee \neg B)$



CONCEPT $\Leftrightarrow A \wedge (\neg B \vee C)$

Generalizing decision trees

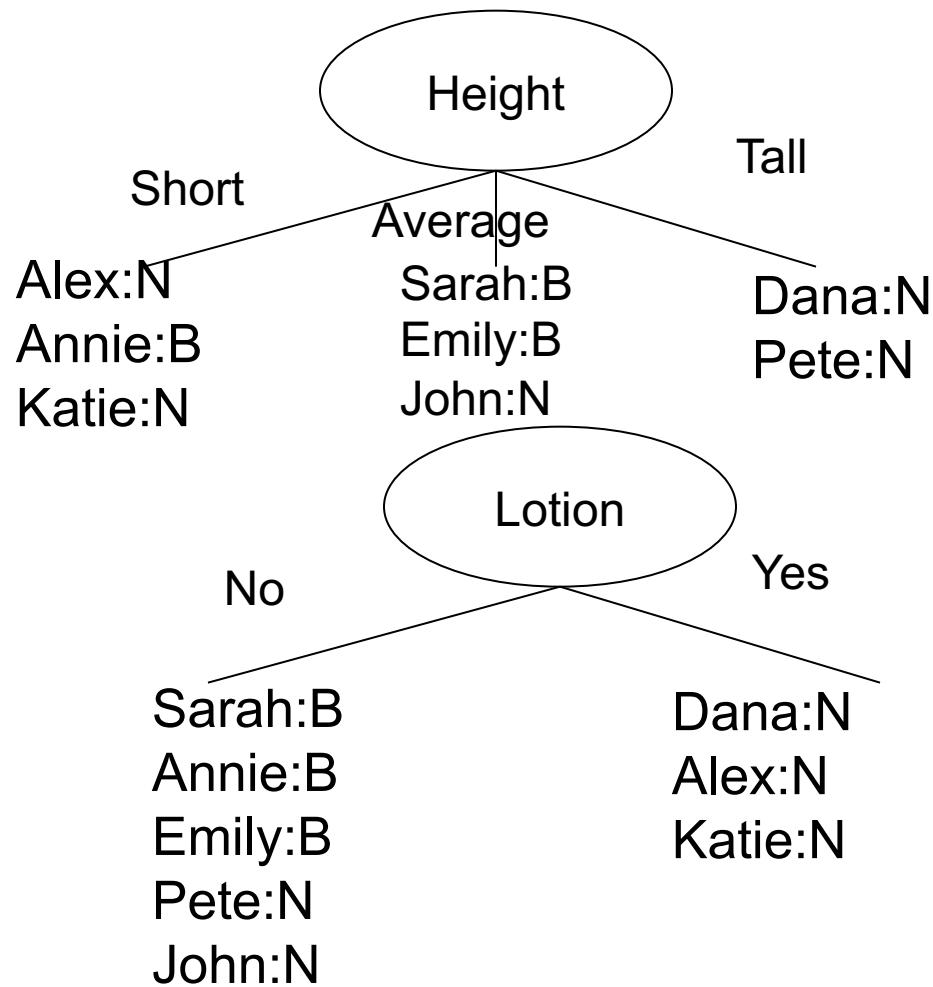
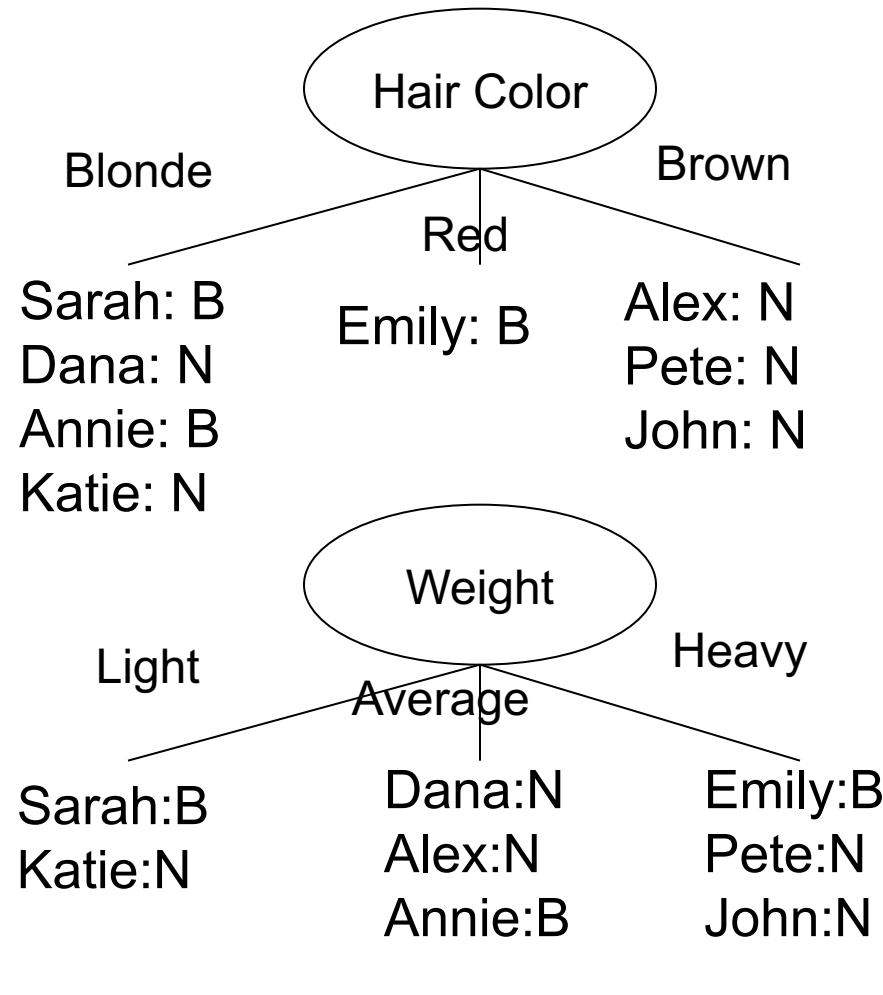
Name	Hair	Height	Weight	Lotion	Result
Sarah	Blonde	Average	Light	No	Burn
Dana	Blonde	Tall	Average	Yes	None
Alex	Brown	Short	Average	Yes	None
Annie	Blonde	Short	Average	No	Burn
Emily	Red	Average	Heavy	No	Burn
Pete	Brown	Tall	Heavy	No	None
John	Brown	Average	Heavy	No	None
Katie	Blonde	Short	Light	Yes	None

Building Decision Trees

- Goal: Build a small tree such that all samples at leaves have same class
- Greedy solution:
 - At each node, pick test such that branches are closest to having same class
 - Split into subsets with least “disorder”
 - (Disorder \sim Entropy)
 - Find test that minimizes disorder

Name	Hair	Height	Weight	Lotion	Result
Sarah	Blonde	Average	Light	No	Burn
Dana	Blonde	Tall	Average	Yes	None
Alex	Brown	Short	Average	Yes	None
Annie	Blonde	Short	Average	No	Burn
Emily	Red	Average	Heavy	No	Burn
Pete	Brown	Tall	Heavy	No	None
John	Brown	Average	Heavy	No	None
Katie	Blonde	Short	Light	Yes	None

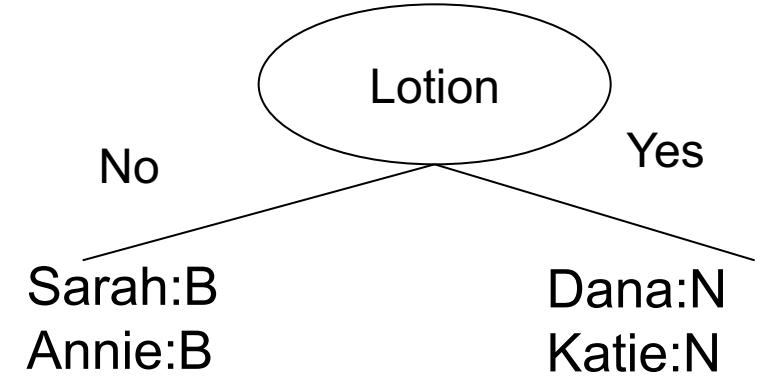
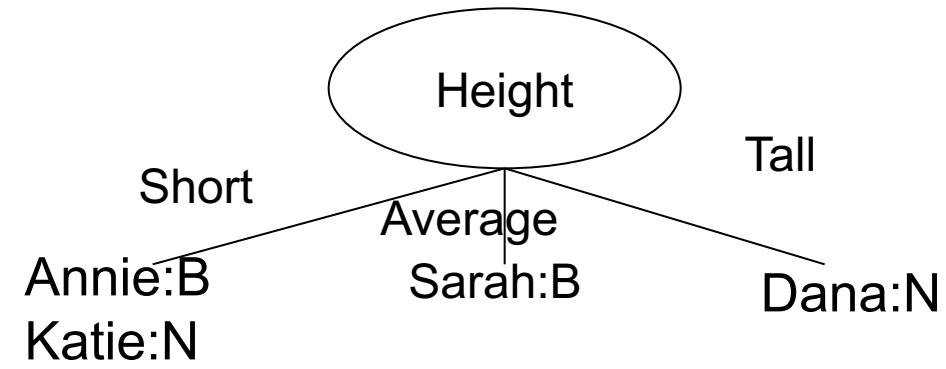
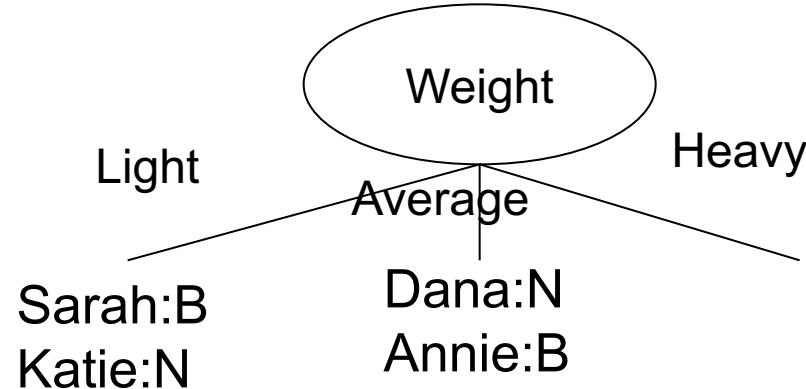
Minimizing Disorder



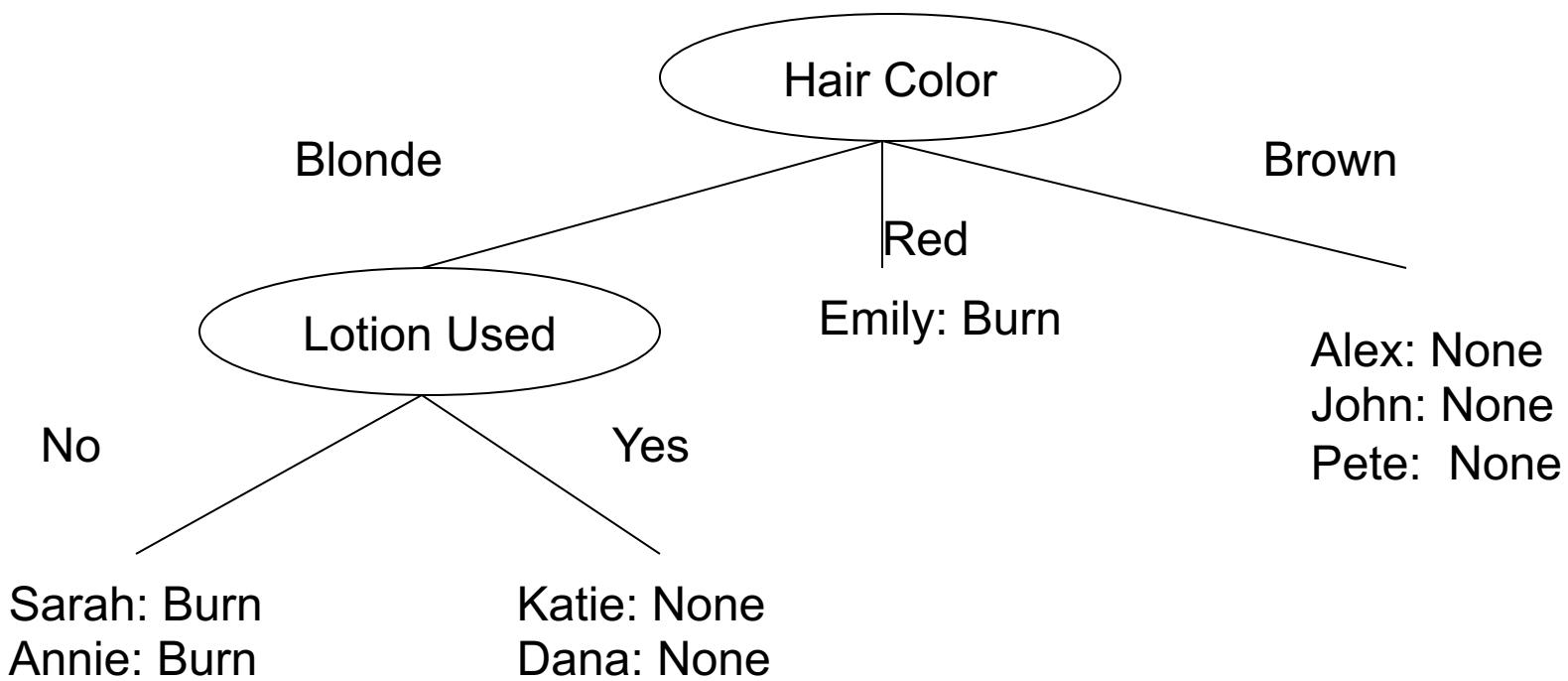
Name	Hair	Height	Weight	Lotion	Result
Sarah	Blonde	Average	Light	No	Burn
Dana	Blonde	Tall	Average	Yes	None
Alex	Brown	Short	Average	Yes	None
Annie	Blonde	Short	Average	No	Burn
Emily	Red	Average	Heavy	No	Burn
Pete	Brown	Tall	Heavy	No	None
John	Brown	Average	Heavy	No	None
Katie	Blonde	Short	Light	Yes	None

Minimizing Disorder

Splitting Blonde!



Sunburn Identification Tree



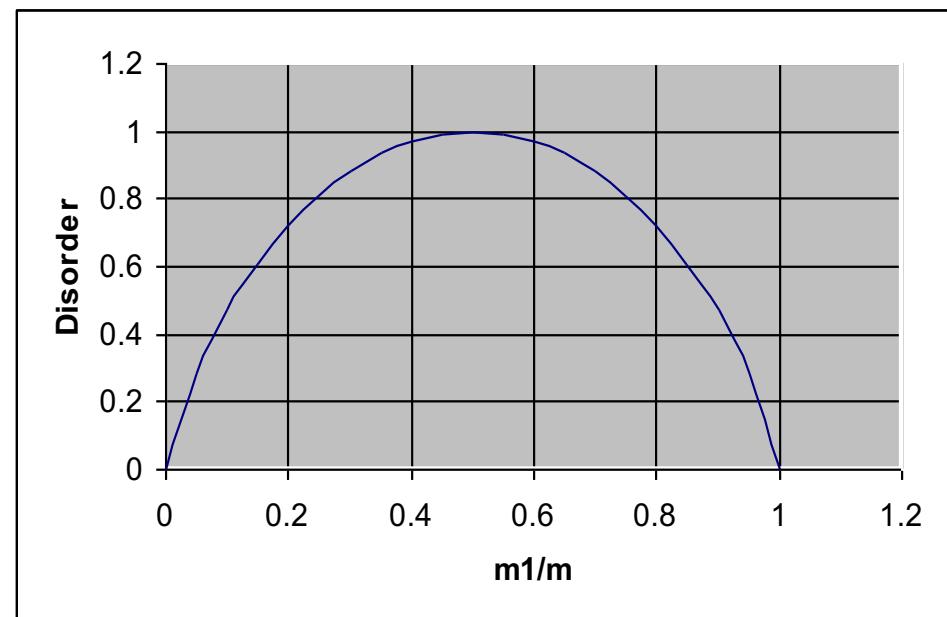
Measuring Disorder

- Problem:
 - In general, tests on large DB's don't yield homogeneous subsets
- Solution:
 - General information theoretic measure of disorder
 - Desired features:
 - Homogeneous set: least disorder = 0
 - Even split: most disorder = 1

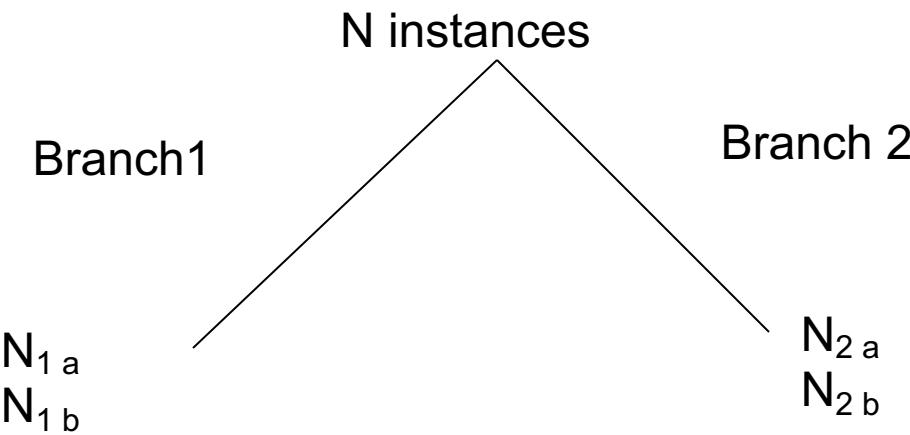
Measuring Entropy

- If split we m objects into 2 bins of size m_1 and m_2 , what is the entropy?

$$\sum_i -\frac{m_i}{m} \log_2 \frac{m_i}{m} =$$
$$-\frac{m_1}{m} \log_2 \frac{m_1}{m} - \frac{m_2}{m} \log_2 \frac{m_2}{m}$$



Computing Disorder



$$AvgDisorder = \sum_{i=1}^k \left(\frac{n_i}{n_t} \sum_{c \in class} -\frac{n_{i,c}}{n_i} \log_2 \frac{n_{i,c}}{n_i} \right)$$

Fraction of samples down branch i

Disorder of class distribution on branch i

Entropy in Sunburn Example

$$AvgDisorder = \sum_{i=1}^k \left(\frac{n_i}{n_t} \sum_{c \in class} -\frac{n_{i,c}}{n_i} \log_2 \frac{n_{i,c}}{n_i} \right)$$

Name	Hair	Height	Weight	Lotion	Result
Sarah	Blonde	Average	Light	No	Burn
Dana	Blonde	Tall	Average	Yes	None
Alex	Brown	Short	Average	Yes	None
Annie	Blonde	Short	Average	No	Burn
Emily	Red	Average	Heavy	No	Burn
Pete	Brown	Tall	Heavy	No	None
John	Brown	Average	Heavy	No	None
Katie	Blonde	Short	Light	Yes	None

Entropy in Sunburn Example

$$AvgDisorder = \sum_{i=1}^k \left(\frac{n_i}{n_t} \sum_{c \in class} -\frac{n_{i,c}}{n_i} \log_2 \frac{n_{i,c}}{n_i} \right)$$

Name	Hair	Height	Weight	Lotion	Result
Sarah	Blonde	Average	Light	No	Burn
Dana	Blonde	Tall	Average	Yes	None
Alex	Brown	Short	Average	Yes	None
Annie	Blonde	Short	Average	No	Burn
Emily	Red	Average	Heavy	No	Burn
Pete	Brown	Tall	Heavy	No	None
John	Brown	Average	Heavy	No	None
Katie	Blonde	Short	Light	Yes	None

$$\begin{aligned} \text{Hair color} &= 4/8(-2/4 \log 2/4 - 2/4 \log 2/4) + 1/8 * 0 + 3/8 * 0 \\ &= 0.5 \end{aligned}$$

$$\text{Height} = 0.69$$

$$\text{Weight} = 0.94$$

$$\text{Lotion} = 0.61$$

Entropy in Sunburn Example

$$AvgDisorder = \sum_{i=1}^k \left(\frac{n_i}{n_t} \sum_{c \in class} -\frac{n_{i,c}}{n_i} \log_2 \frac{n_{i,c}}{n_i} \right)$$

Name	Hair	Height	Weight	Lotion	Result
Sarah	Blonde	Average	Light	No	Burn
Dana	Blonde	Tall	Average	Yes	None
Alex	Brown	Short	Average	Yes	None
Annie	Blonde	Short	Average	No	Burn
Emily	Red	Average	Heavy	No	Burn
Pete	Brown	Tall	Heavy	No	None
John	Brown	Average	Heavy	No	None
Katie	Blonde	Short	Light	Yes	None

Hair color already happened!

$$\text{Height} = 2/4(-1/2\log 1/2 - 1/2\log 1/2) + 1/4*0 + 1/4*0 = 0.5$$

$$\text{Weight} = 2/4(-1/2\log 1/2 - 1/2\log 1/2) + 2/4(-1/2\log 1/2 - 1/2\log 1/2) = 1$$

$$\text{Lotion} = 0$$

So after hair color, we should expand lotion

ID3 Algorithm

- Iterative Dichotomiser 3 -- Ross Quinlan (1986)

ID3 (Examples, Target_Attribute, Attributes)

- Create a root node for the tree
- If all examples are positive, Return the single-node tree Root, with label = +.
- If all examples are negative, Return the single-node tree Root, with label = -.
- If # of attributes is empty, return Root, with label = most common value of target.
- Otherwise:
 - $A \leftarrow$ Attribute that best classifies examples, use this attribute as Root
 - For each possible value, v_i , of A,
 - Add a new tree branch below Root, corresponding to the test $A = v_i$.
 - Let $\text{Examples}(v_i)$ be the subset of examples that have the value v_i for A
 - If $\text{Examples}(v_i)$ empty, add leaf with label = most common value of target, otherwise add subtree ID3 ($\text{Examples}(v_i)$, Target_Attribute, Attributes – {A})
 - Return Root

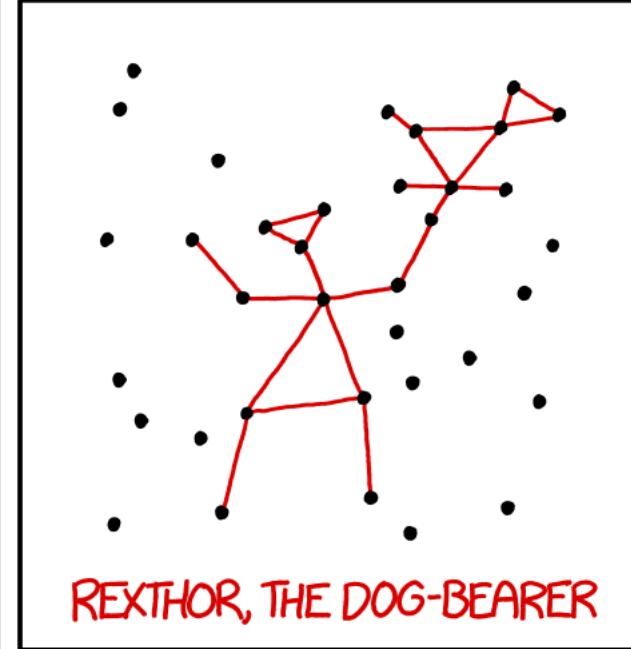
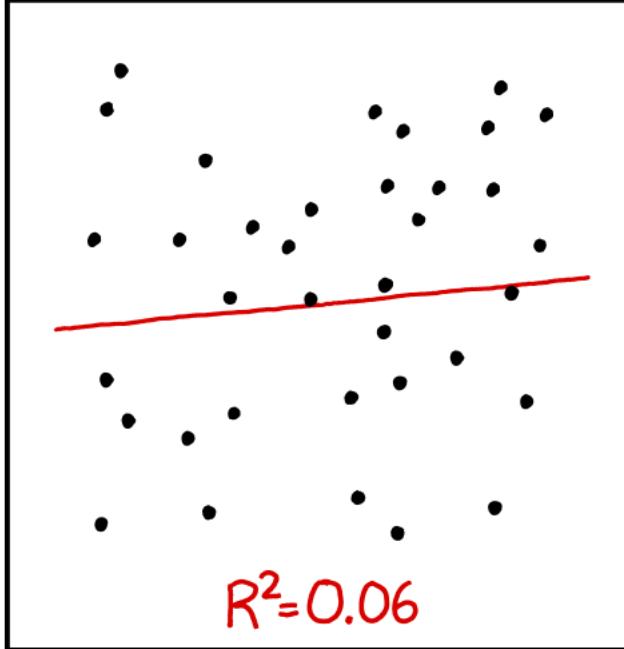
Decision tree properties

- Widely used
- Greedy
- Robust to noise (incorrect examples)
- Easily understood by humans; this is important in medical, financial, military applications

Linear models

Announcements

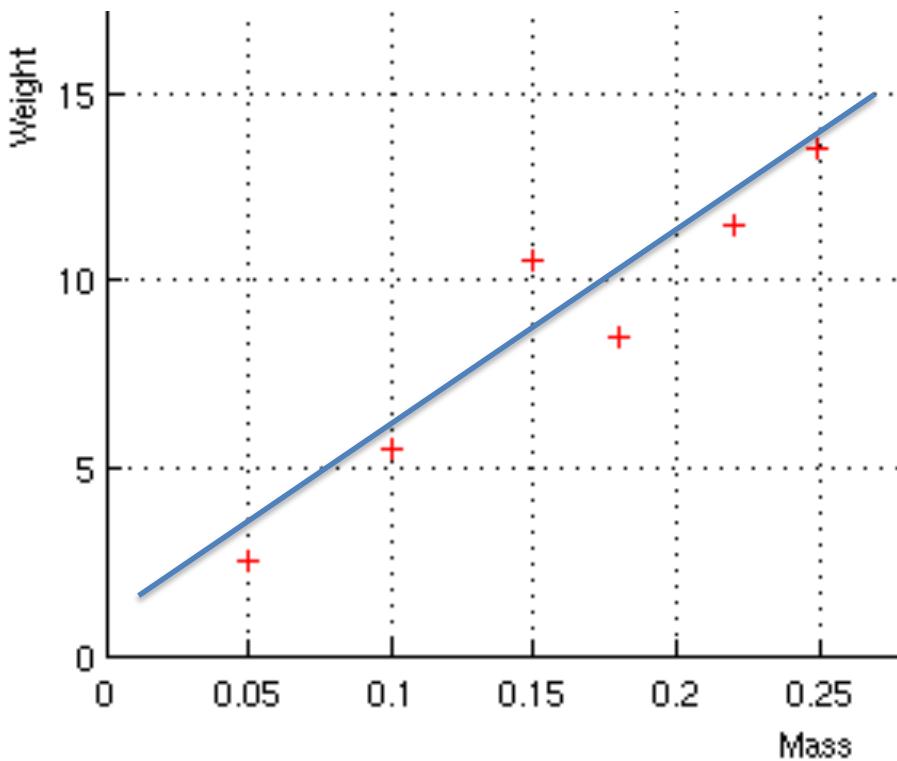
- A2 due date extended to November 11 (you can test your agents by playing against others – see Q&A post)
- A3 and optional A4 to follow (A4 will have a deadline during the last week of classes)
- Grades for A1 will be posted later tonight/tomorrow.
- Survey to follow



I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER
TO GUESS THE DIRECTION OF THE CORRELATION FROM THE
SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

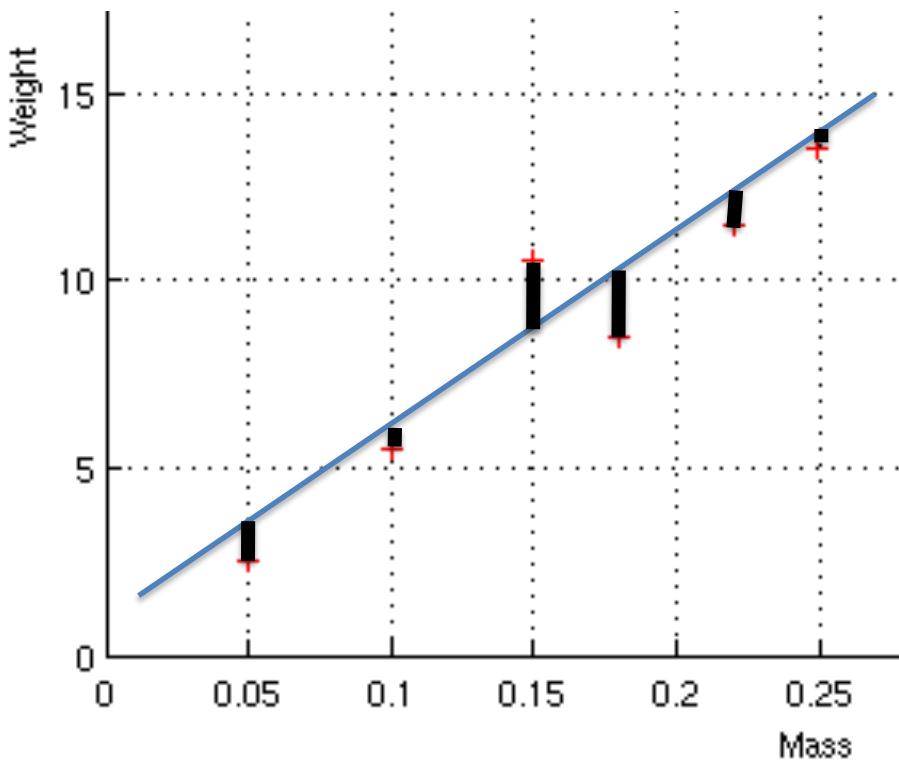
A step back: Line fitting

- Suppose you want to fit a line to some data points
 - I.e. find a line that minimizes the sum squared distances between each data point to the line



A step back: Line fitting

- Suppose you want to fit a line to some data points
 - I.e. find a line that minimizes the sum squared distances between each data point to the line

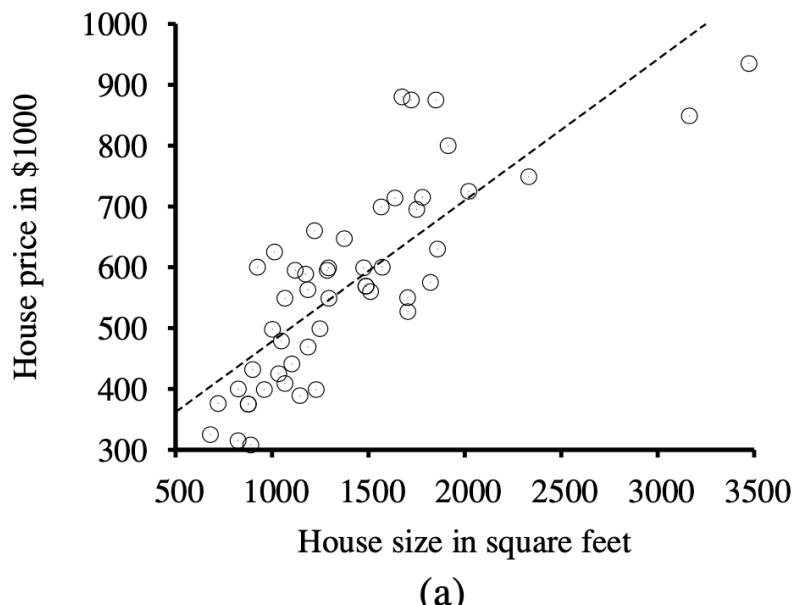


- **Linear regression** gives unique solution in closed form:

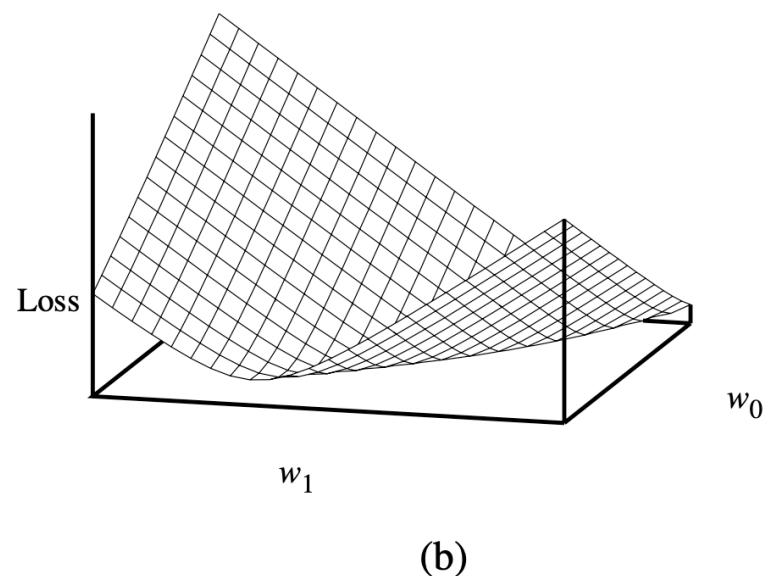
$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}$$

$$w_0 = (\sum y_j - w_1(\sum x_j))/N$$

Example



(a)

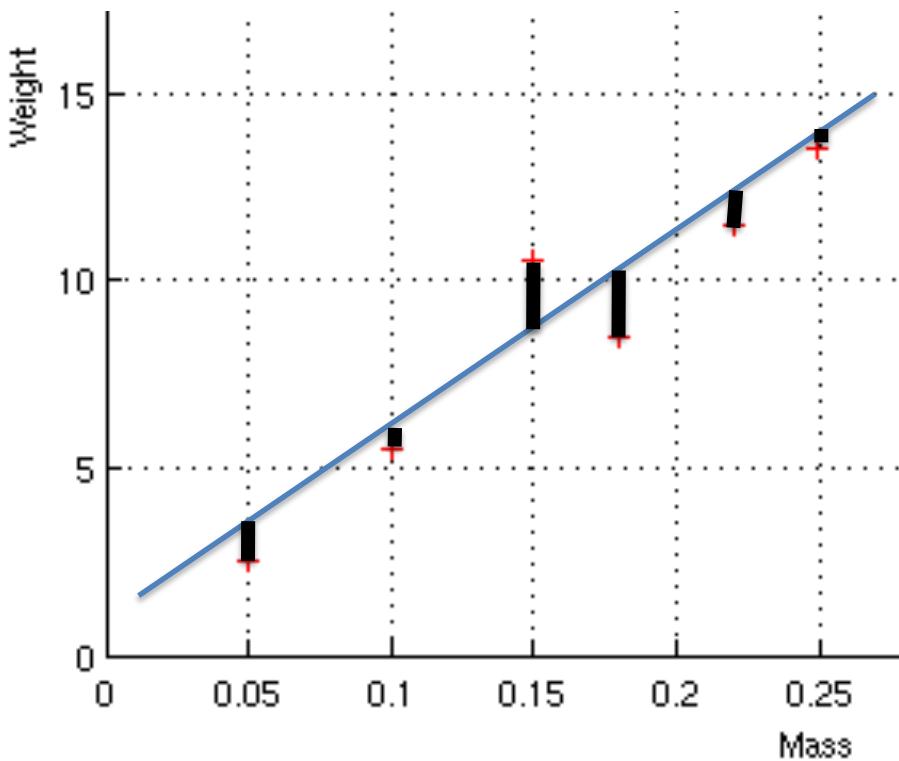


(b)

Figure 18.13 (a) Data points of price versus floor space of houses for sale in Berkeley, CA, in July 2009, along with the linear function hypothesis that minimizes squared error loss: $y = 0.232x + 246$. (b) Plot of the loss function $\sum_j (w_1 x_j + w_0 - y_j)^2$ for various values of w_0, w_1 . Note that the loss function is convex, with a single global minimum.

A step back: Line fitting

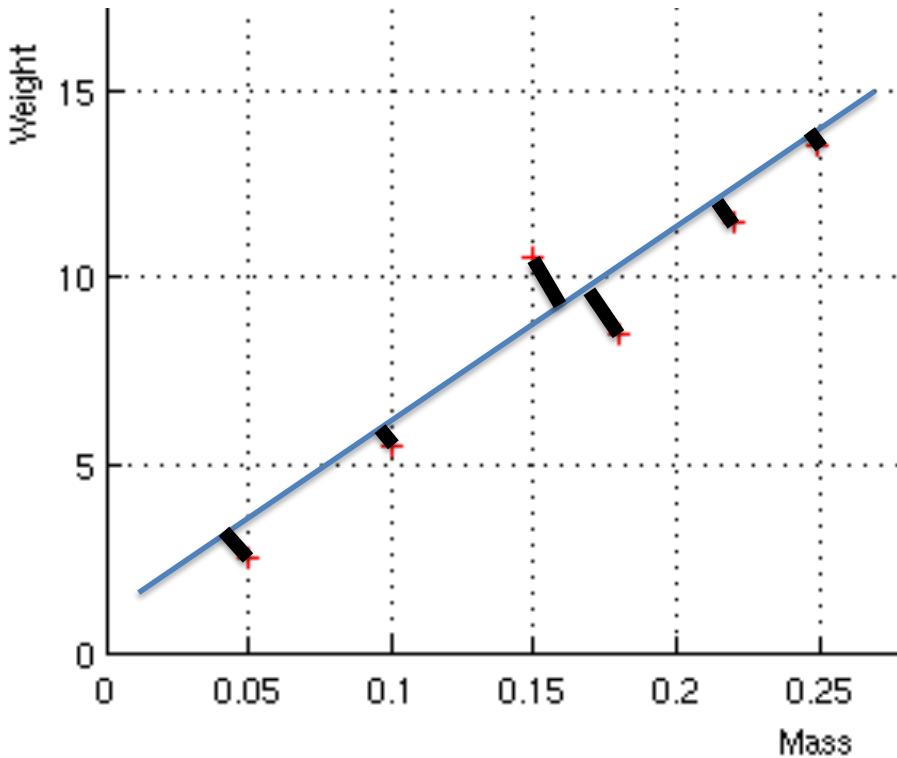
- Suppose you want to fit a line to some data points
 - I.e. find a line that minimizes the sum squared distances between each data point to the line



- Linear regression
 - aka “ordinary least squares”
 - Measures errors along y-axis only
 - Assumes observations on x-axis are error free

A step back: Line fitting

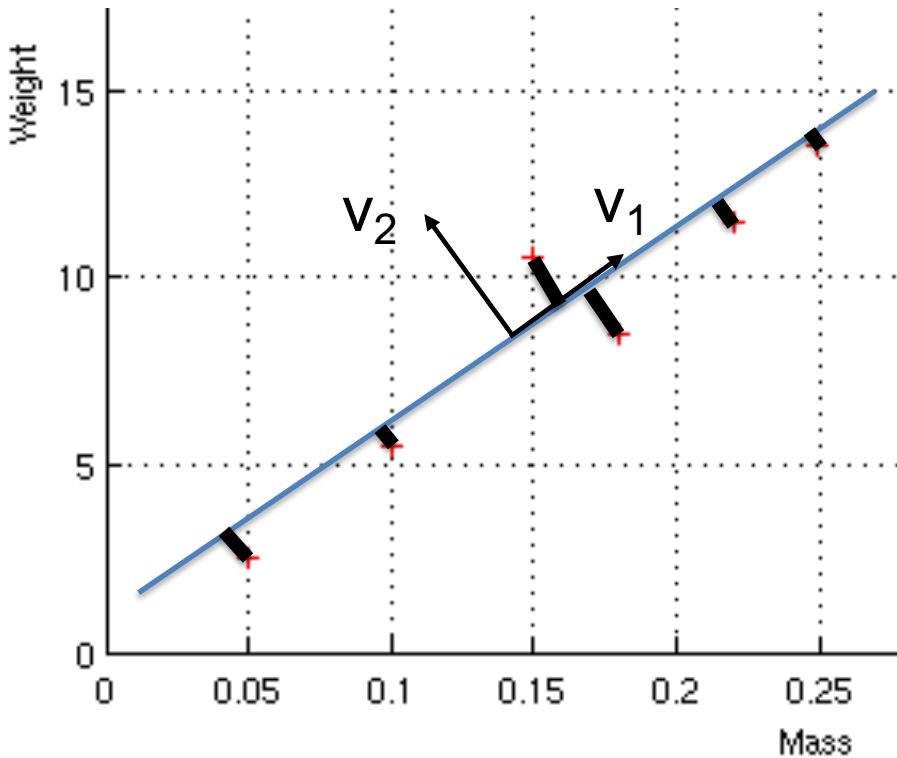
- Suppose you want to fit a line to some data points
 - I.e. find a line that minimizes the sum of distances between each data point to the line



- Total least squares
 - Measures Euclidean distance from point to line

A step back: Line fitting

- Suppose you want to fit a line to some data points
 - I.e. find a line that minimizes the sum of distances between each data point to the line

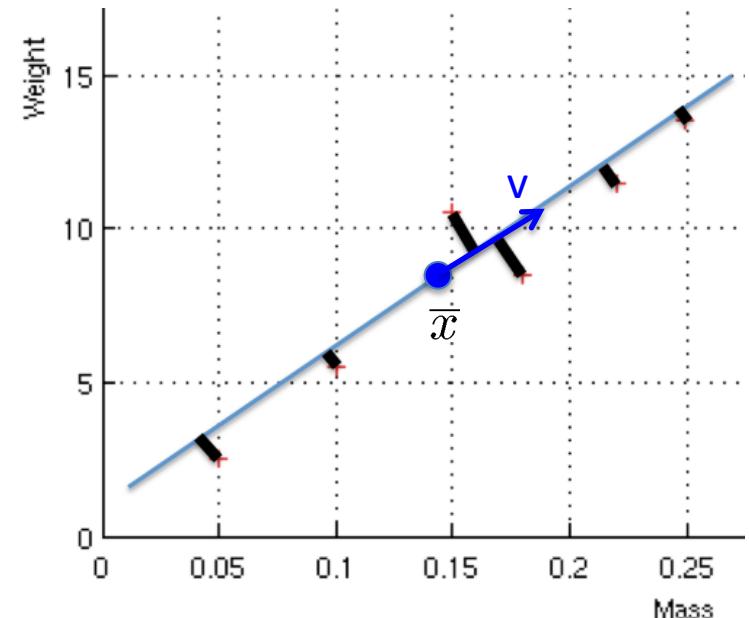


- Total least squares
 - Measures Euclidean distance from point to line

Total least squares

- Can show that the best line passes through the centroid (mean) of the points, \bar{x}
- What about the direction?
 - Need to solve for vector v by minimizing error,

$$\begin{aligned}\text{Total error} &= \sum_x ((x - \bar{x})^T \cdot v)^2 \\ &= \sum_x v^T (x - \bar{x})(x - \bar{x})^T v \\ &= v^T \left[\sum_x (x - \bar{x})(x - \bar{x})^T \right] v \\ &= v^T A v \quad \text{where } A = \sum_x (x - \bar{x})(x - \bar{x})^T\end{aligned}$$



v that minimizes
 $v^T A v$ is the largest
eigenvector of A

Works in
multivariate case
too!

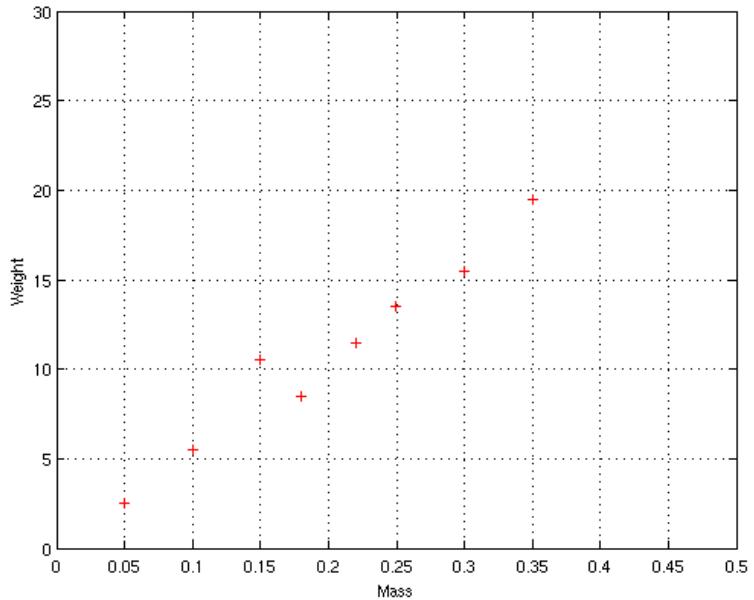
Total least squares

- A line minimizing the total least squares can be found using eigendecomposition
 1. Put the data in a matrix, X , where each row is a vector representing a data point
 2. Compute the mean of the columns of X , giving a vector \mathbf{x}
 3. Compute the covariance matrix, $A = (X-\mathbf{x})^T(X-\mathbf{x})$
 4. Find the eigenvectors and eigenvalues of A

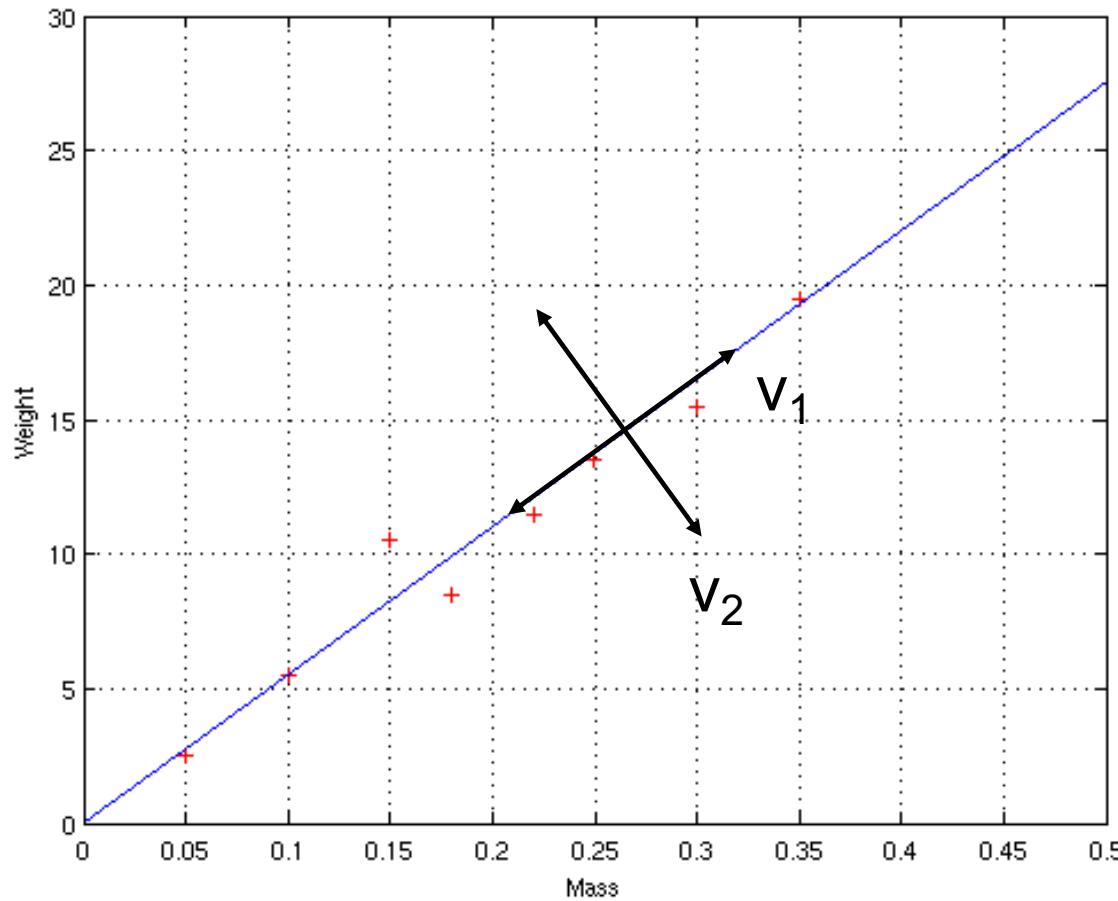
Example

$$X = \begin{bmatrix} 0.15 & 10.5 \\ 0.05 & 2.5 \\ 0.18 & 8.5 \\ 0.10 & 5.5 \\ 0.25 & 13.5 \\ 0.35 & 19.5 \\ 0.30 & 15.5 \\ 0.22 & 11.5 \end{bmatrix} \quad Y = \begin{bmatrix} -0.05 & -0.375 \\ -0.15 & -8.375 \\ -0.02 & -2.375 \\ -0.10 & -5.375 \\ 0.05 & 2.625 \\ 0.15 & 8.625 \\ 0.10 & 4.625 \\ 0.02 & 0.625 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.0708 & 3.7600 \\ 3.7600 & 207.8750 \end{bmatrix}$$



- Eigenvectors and eigenvalues of A:
 - $v_1 = (0.0181, 0.999)$, $\lambda_1 = 208$
 - $v_2 = (-0.999, 0.0181)$, $\lambda_2 = 0.0181$



- Eigenvectors and eigenvalues of A :
 - $v_1 = (0.0181, 0.999)$, $\lambda_1 = 208$
 - $v_2 = (-0.999, 0.0181)$, $\lambda_2 = 0.0181$

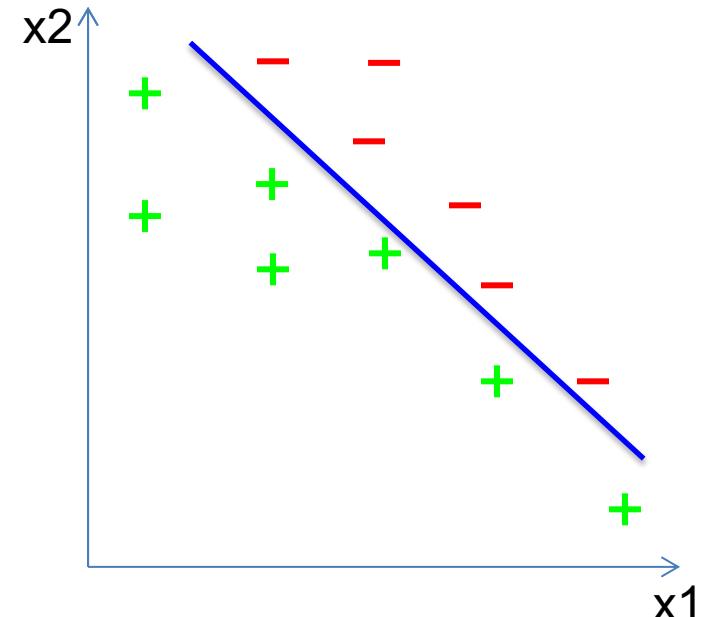
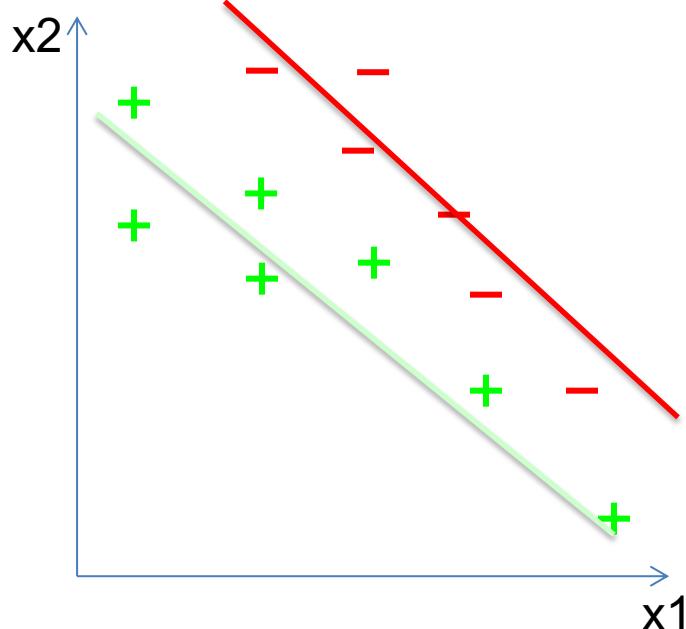
Classification

- We can model the class itself, but often want to model the *difference* between two classes

Generative

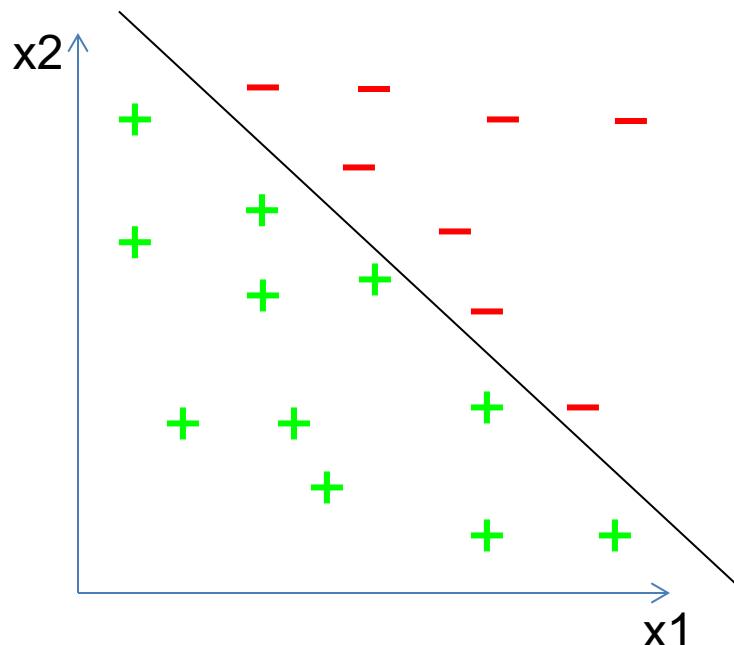
vs

Discriminative



Linear classifiers : Motivation

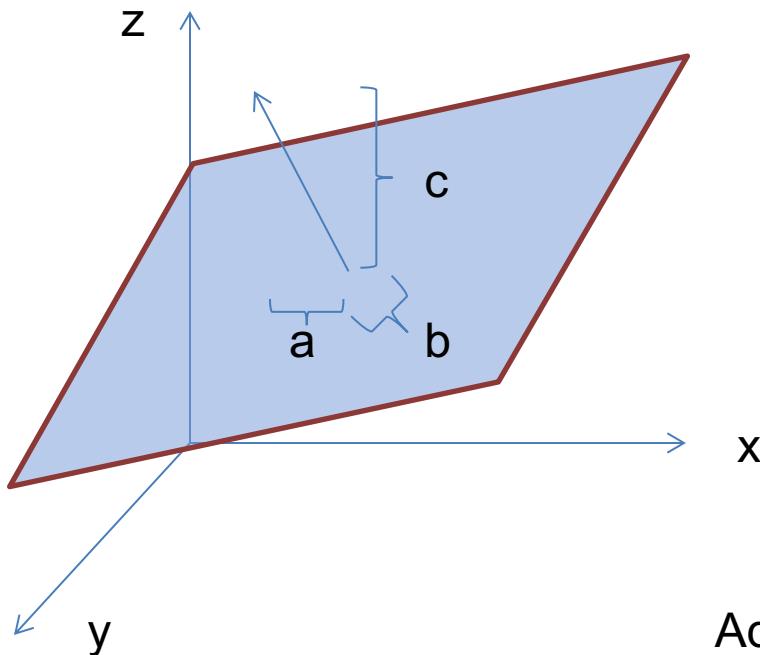
- Decision tree produces axis-aligned decision boundaries and can't accurately classify data like this:



- Linear classifiers model this boundary directly

Plane Geometry

- In 3D, a plane can be expressed as the set of solutions (x,y,z) to the equation $ax+by+cz+d=0$
 - $ax+by+cz+d > 0$ is one side of the plane
 - $ax+by+cz+d < 0$ is the other side
 - $ax+by+cz+d = 0$ is the plane itself

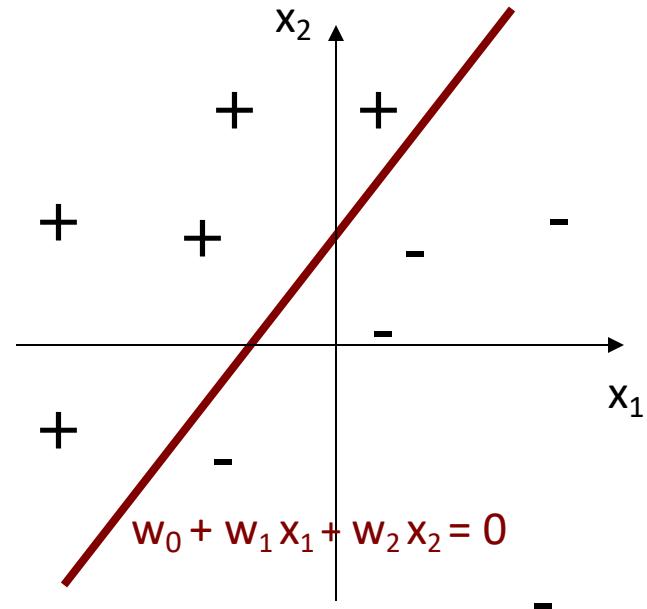
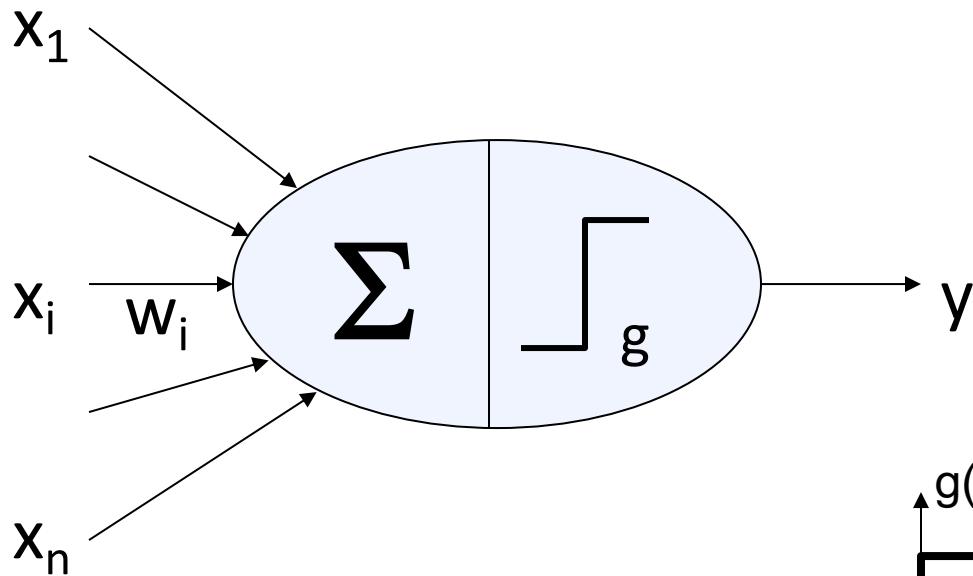


Adapted from K. Hauser's slide

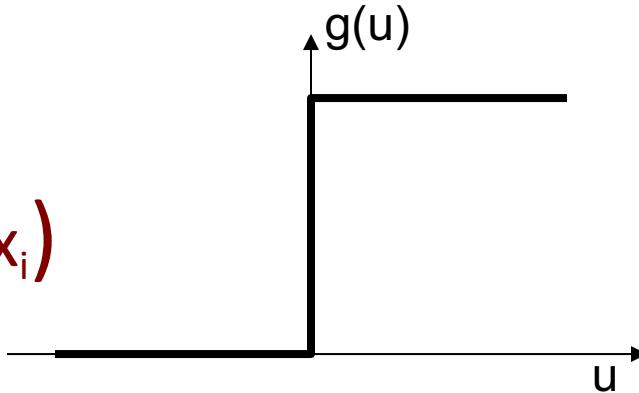
Linear Classifier

- In d dimensions,
 - $w_0 + w_1 * x_1 + \dots + w_d * x_d = 0$ is a **hyperplane**.
- Idea:
 - Use $w_0 + w_1 * x_1 + \dots + w_d * x_d \geq 0$ to denote positive classifications
 - Use $w_0 + w_1 * x_1 + \dots + w_d * x_d < 0$ to denote negative classifications

Perceptron

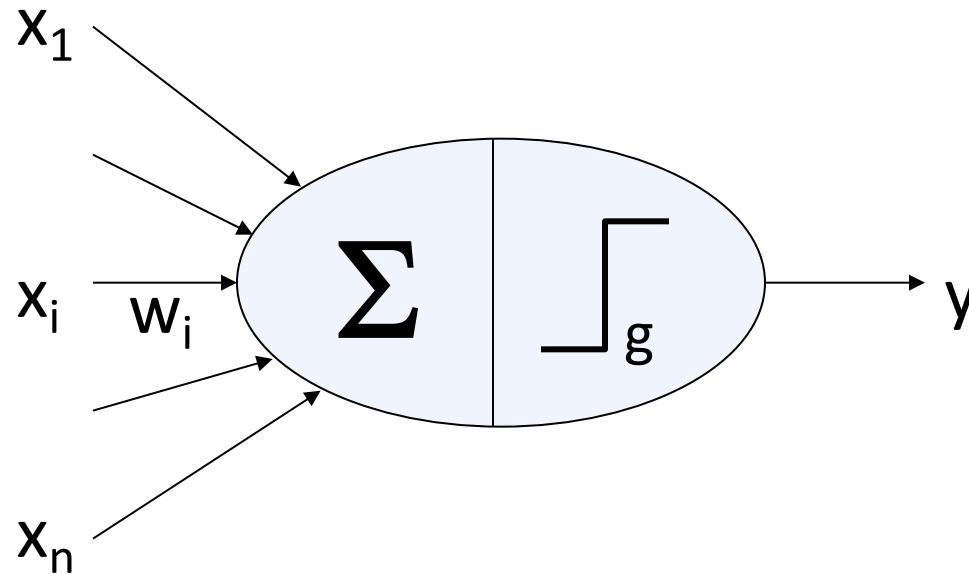


$$\begin{aligned}y &= f(\mathbf{x}, \mathbf{w}) = g\left(\sum_{i=1, \dots, n} w_i x_i\right) \\&= g(\mathbf{w} \bullet \mathbf{x})\end{aligned}$$



Adapted from K. Hauser's slide

Perceptrons can model different functions



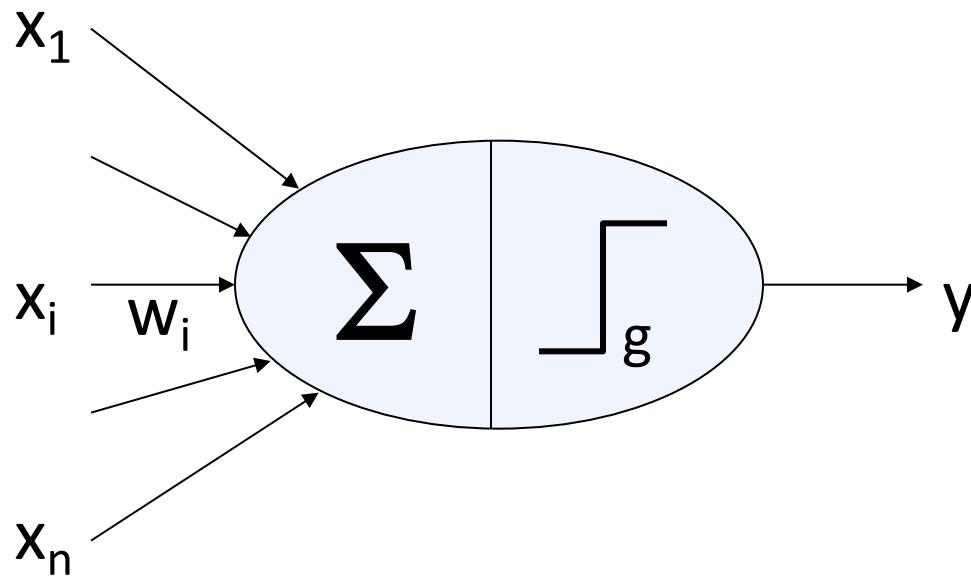
The function $x_1 \wedge x_2 \wedge \neg x_3$?

Majority function (if most of inputs are 1 return 1, otherwise return 0)?

Learning perceptrons

- How do we learn w ?
 - Take derivative, set equal to 0, solve for w ?
- Simple update rule:
 - Start with initial guess of w .
 - For each exemplar (x, y) , and each weight w_i , update:
$$w_i \leftarrow w_i + \alpha x_i (y - g(w^T x))$$
(where y is either 0 or 1 and $g()$ is either 0 or 1)
- Converges if data is linearly separable, but oscillates otherwise

Perceptrons can model different functions

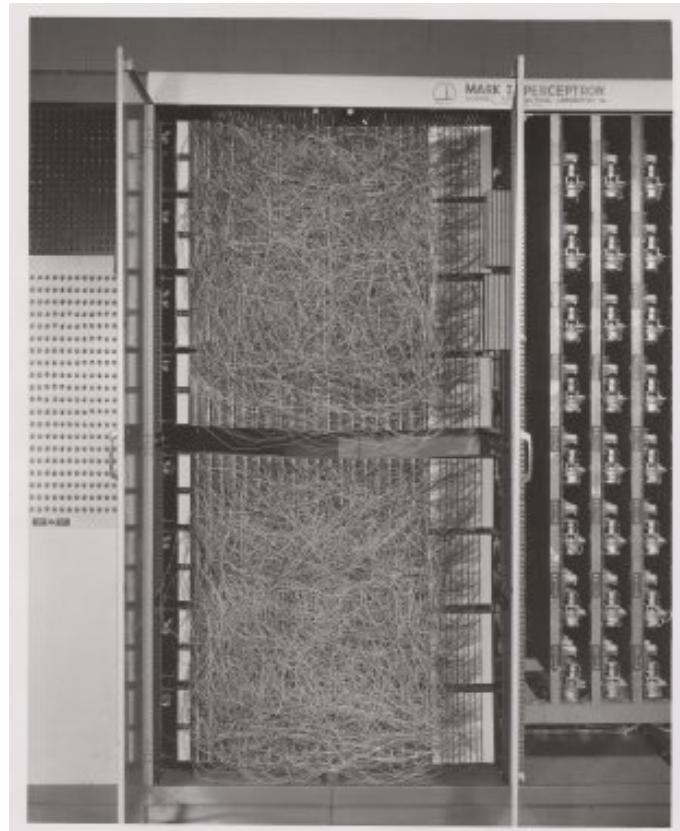
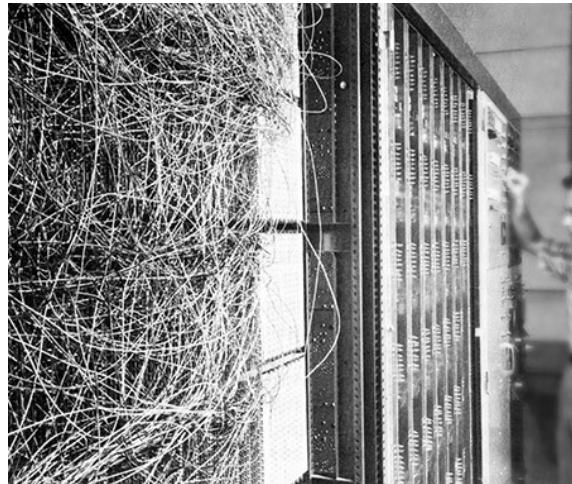


The function $x_1 \wedge x_2 \wedge \neg x_3$?

Majority function ?

XOR ?

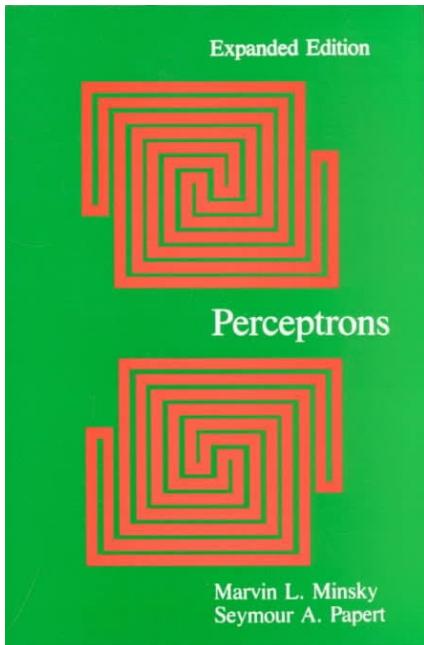
Perceptrons



“the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”

Frank Rosenblatt, 1958

Perceptrons



...until Minky & Papert showed they
couldn't even learn XOR. (1969)

Next class

- Learning in neural networks

Neural networks



STORY BY DATA

HISTORY OF NEURAL NETWORKS

1943-2019



Intuition: visual hierarchy in the brain

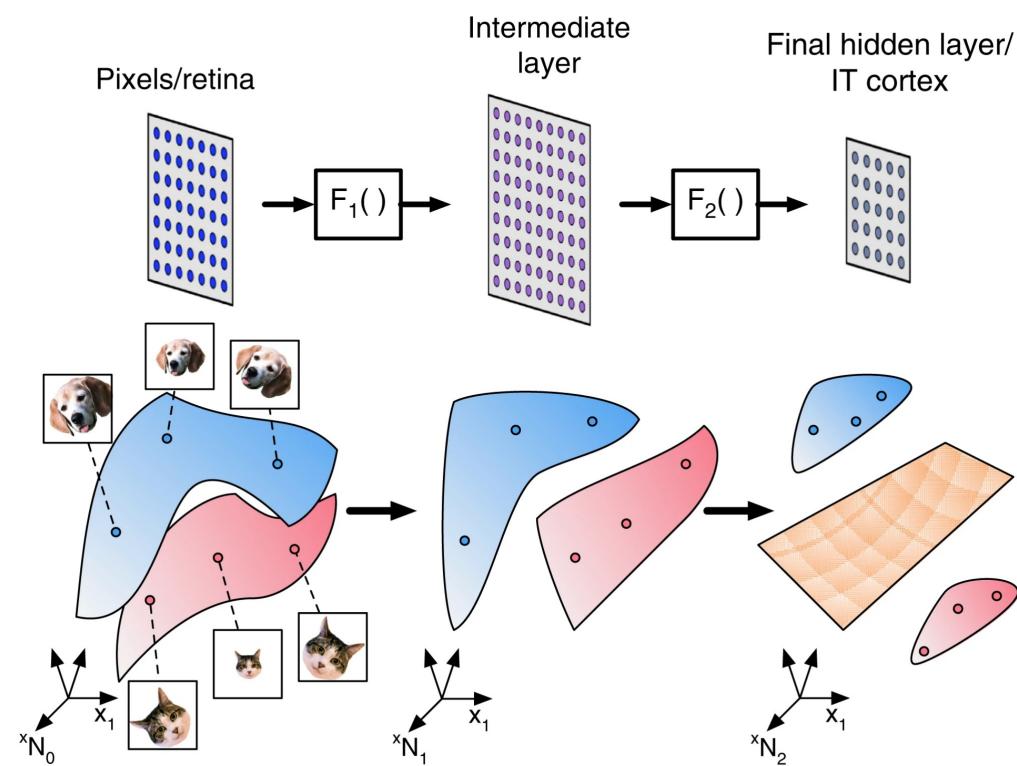
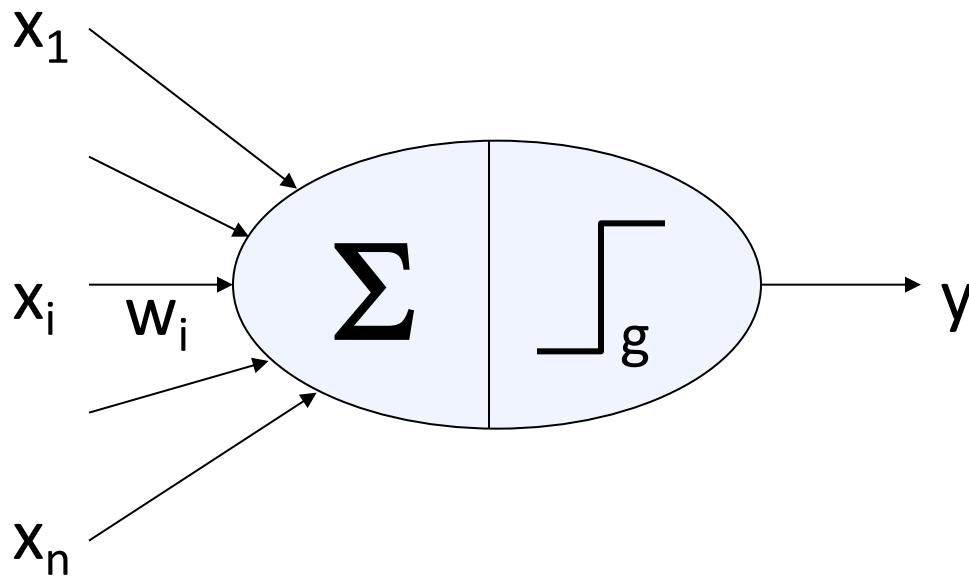


Illustration of three layers in a visual hierarchy where the population response of the first layer is mapped into intermediate layer by F_1 and into the last layer by F_2 (top). The transformation of per-stimuli responses is associated with changes in the geometry of the object manifold, the collection of responses to stimuli of the same object (colored blue for a 'dog' manifold and pink for a 'cat' manifold). Changes in geometry may result in transforming object manifolds which are not linearly separable (in the first and intermediate layers) into separable ones in the last layer (separating hyperplane, colored orange).

Perceptrons can model different functions



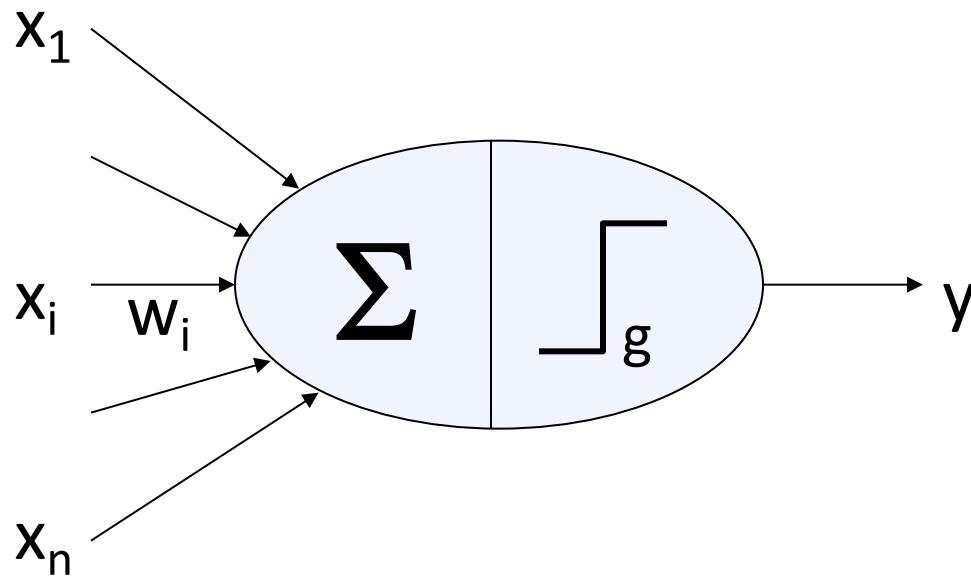
The function $x_1 \wedge x_2 \wedge \neg x_3$?

Majority function ?

Learning perceptrons

- How do we learn w ?
 - Take derivative, set equal to 0, solve for w ?
- Simple update rule:
 - Start with initial guess of w .
 - For each exemplar (x, y) , and each weight w_i , update:
$$w_i \leftarrow w_i + \alpha x_i (y - g(w^T x))$$
(where y is either 0 or 1 and $g()$ is either 0 or 1)
- Converges if data is linearly separable, but oscillates otherwise

Perceptrons can model different functions

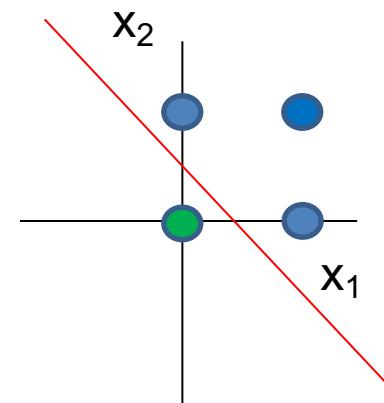
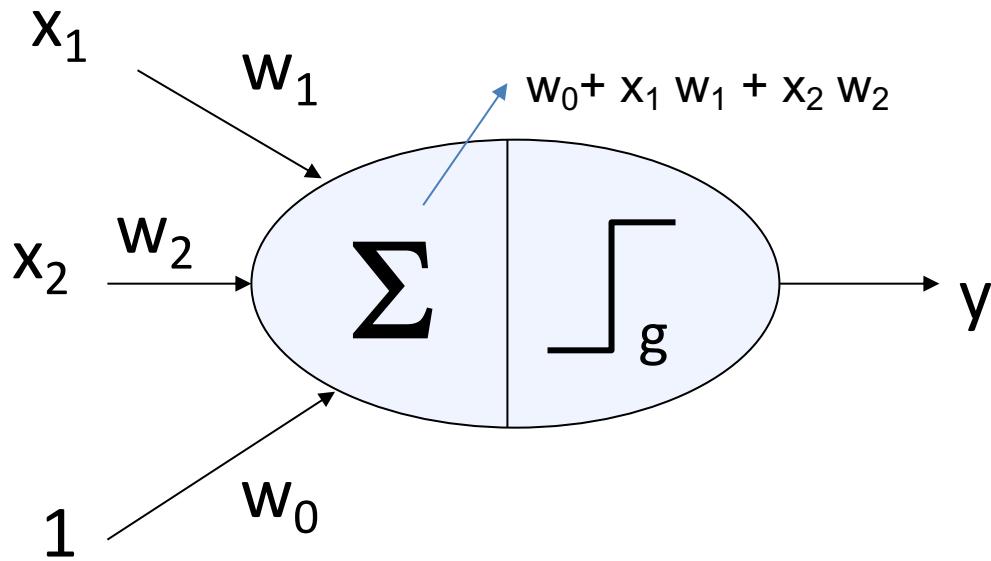


The function $x_1 \wedge x_2 \wedge \neg x_3$?

Majority function ?

XOR ?

Perceptrons can model different functions



OR function

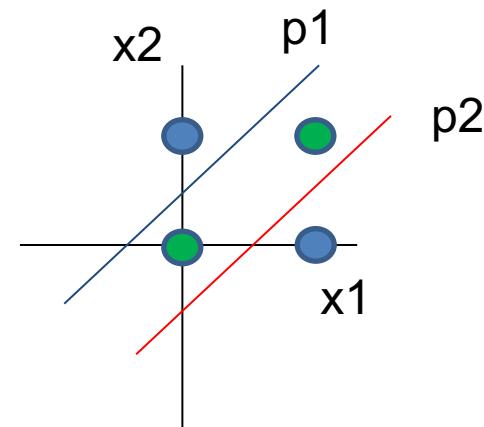
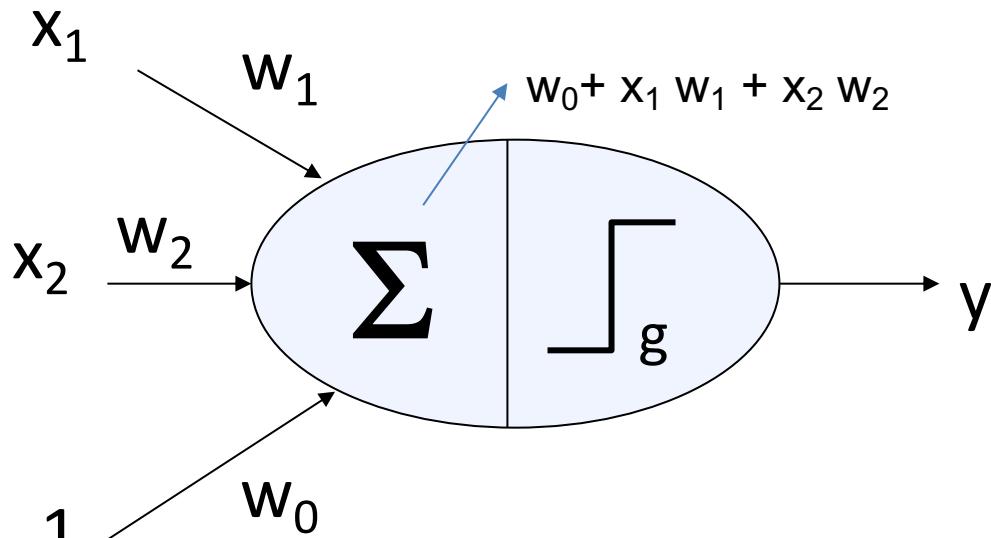
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$$w_0 = -0.5$$

$$w_1 = 1$$

$$w_2 = 1$$

Perceptrons can model different functions

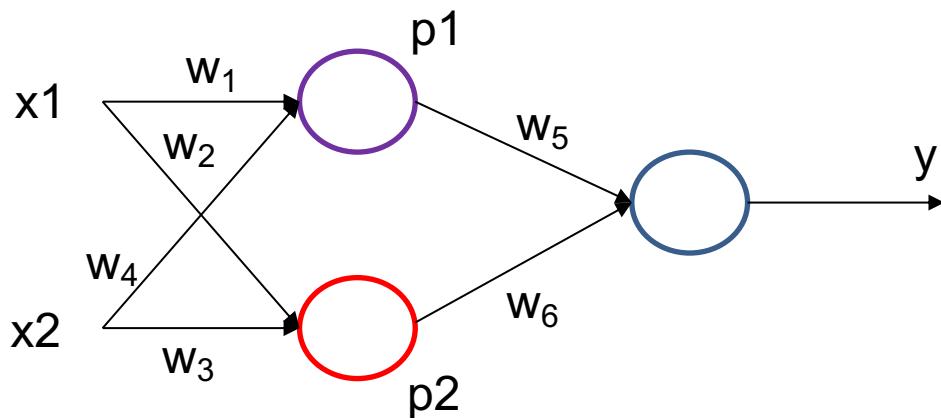


XOR ?

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Adapted from K. Hauser's slide

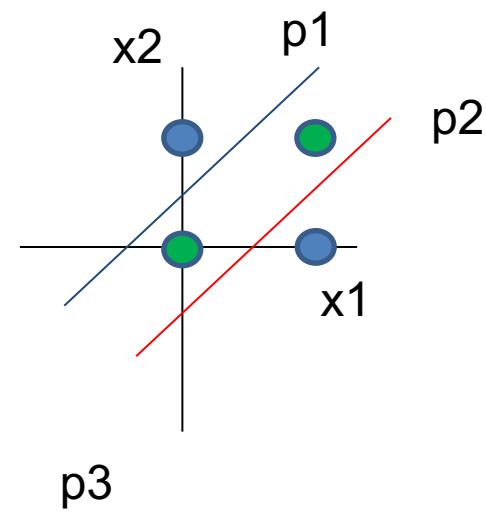
XOR with perceptrons



p_1	p_2				
x_1	x_2	p_1	x_1	x_2	p_1
0	0	0	0	0	0
0	1	0	0	1	1
1	0	1	1	0	0
1	1	0	1	1	0

$$\begin{aligned} w_1 &= 1 \\ w_2 &= -1 \end{aligned}$$

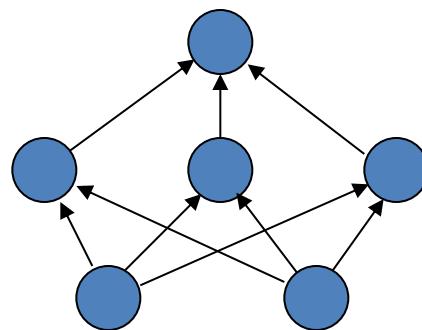
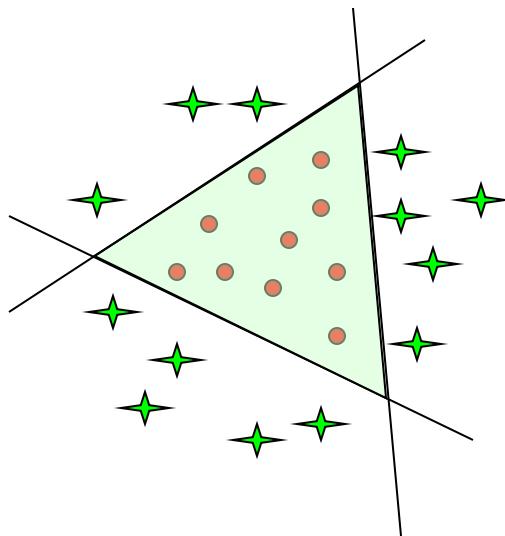
$$\begin{aligned} w_3 &= -1 \\ w_4 &= 1 \end{aligned}$$



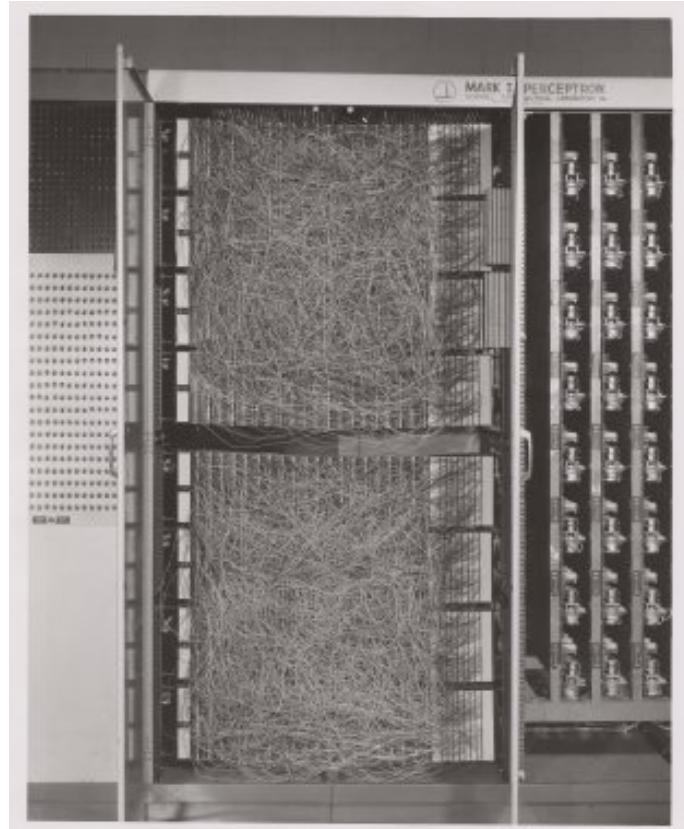
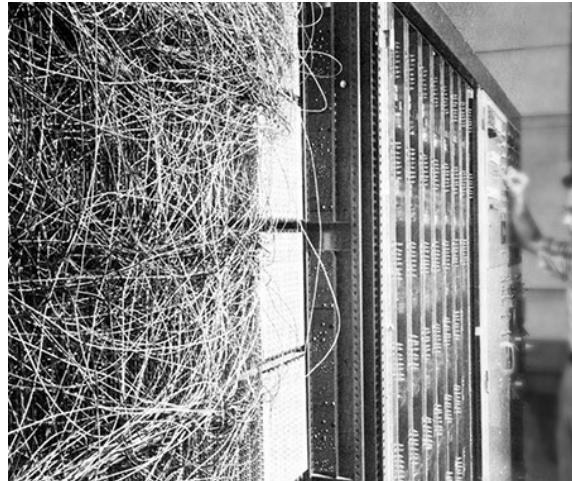
x_1	x_2	p_1	p_2	y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

$$\begin{aligned} w_5 &= 1 \\ w_6 &= 1 \end{aligned}$$

Multi-Layer Generalization



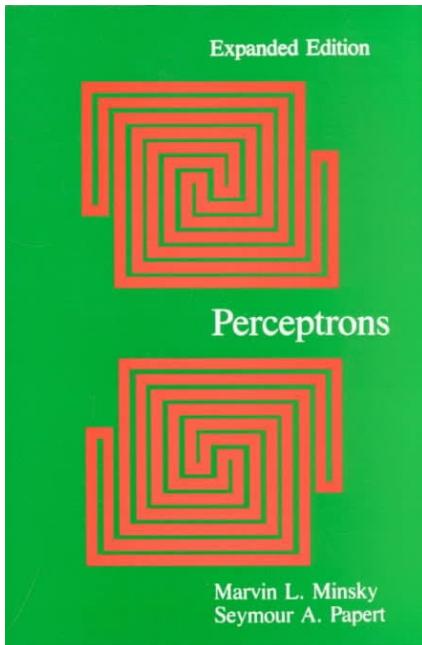
Perceptrons



“the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”

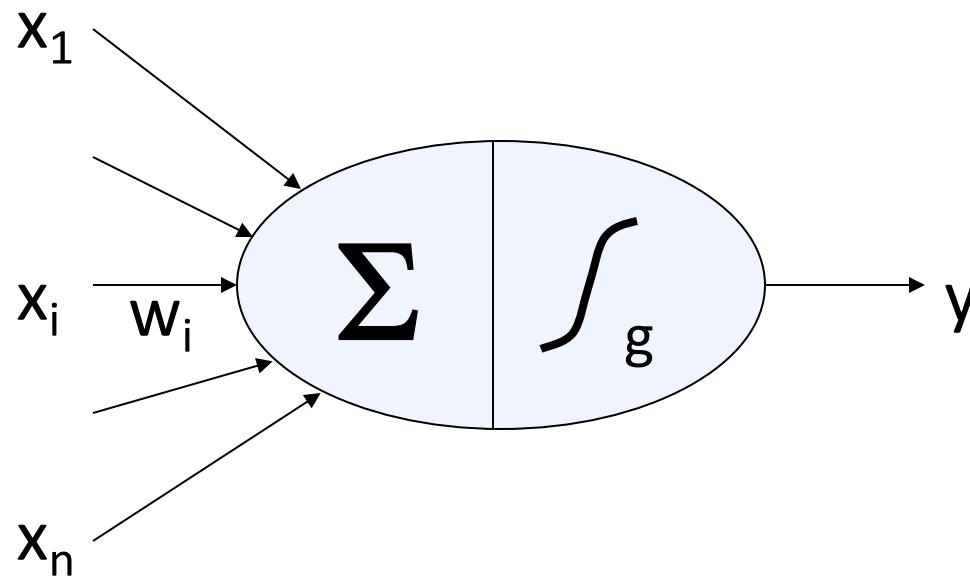
Frank Rosenblatt, 1958

Perceptrons



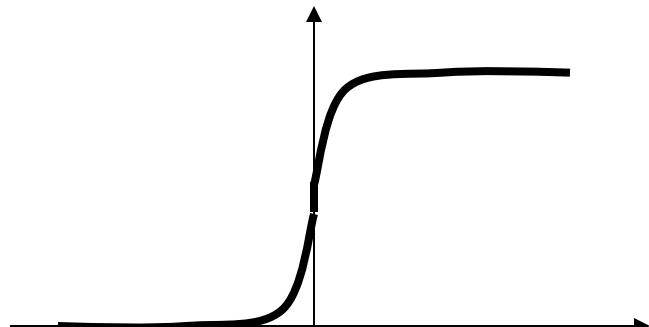
...until Minky & Papert showed they
couldn't even learn XOR. (1969)

Unit (Neuron)



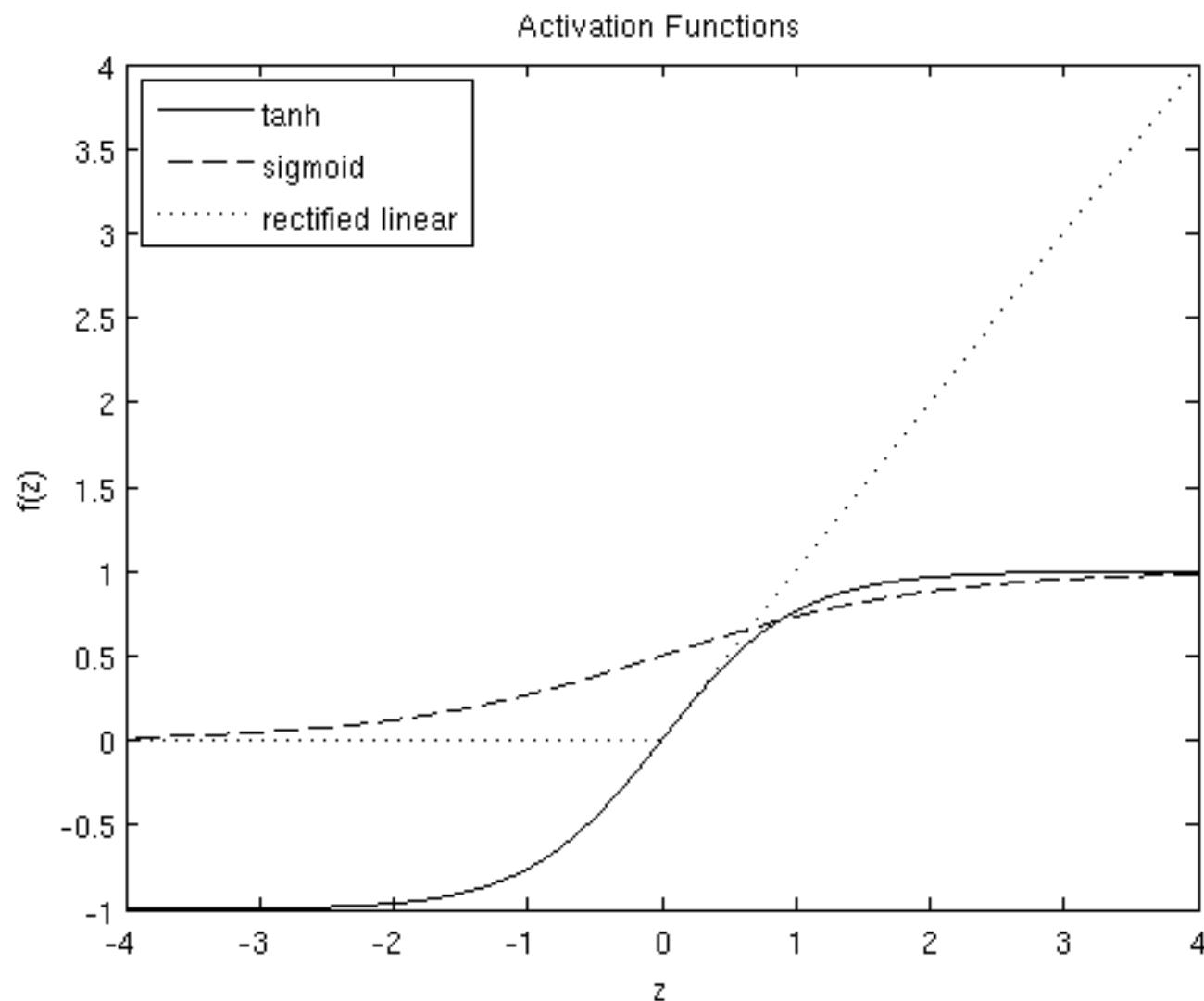
$$y = g(\sum_{i=1, \dots, n} w_i x_i)$$

$$g(u) = 1/[1 + \exp(-au)]$$



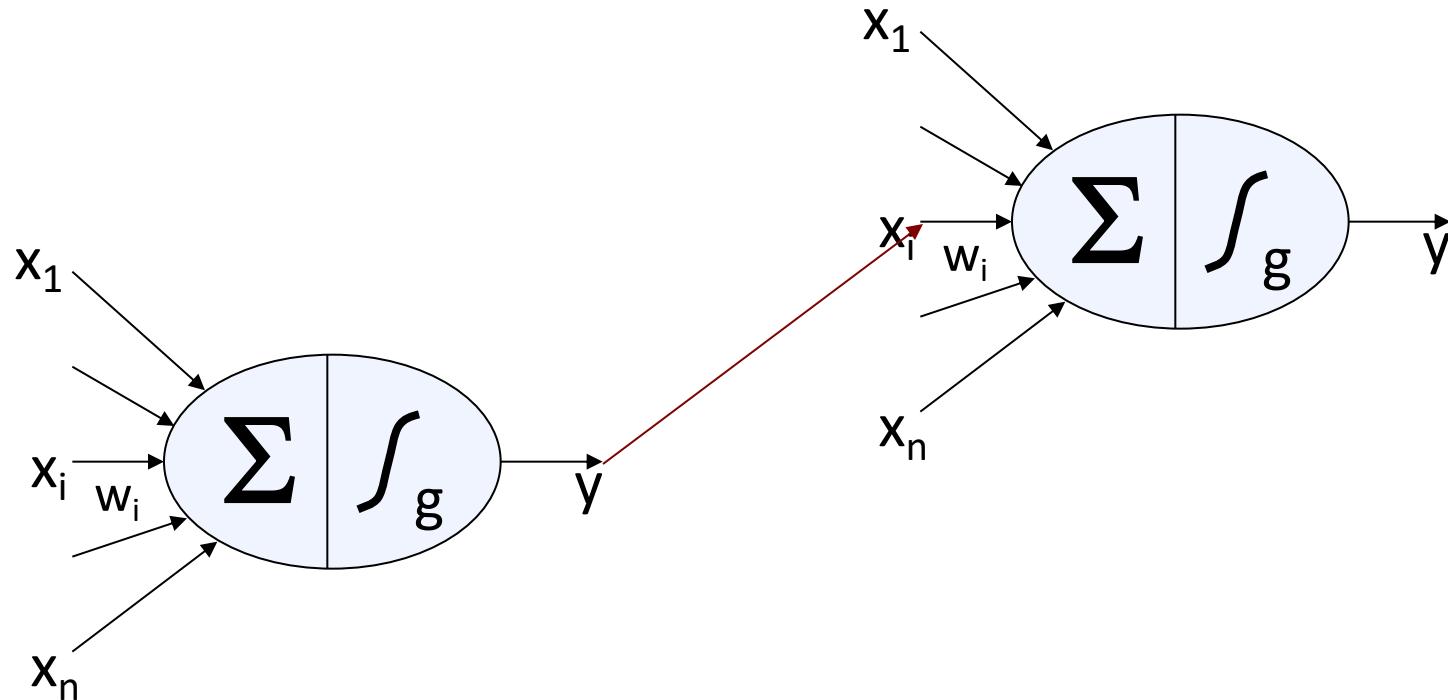
Adapted from K. Hauser's slide

Common activation functions



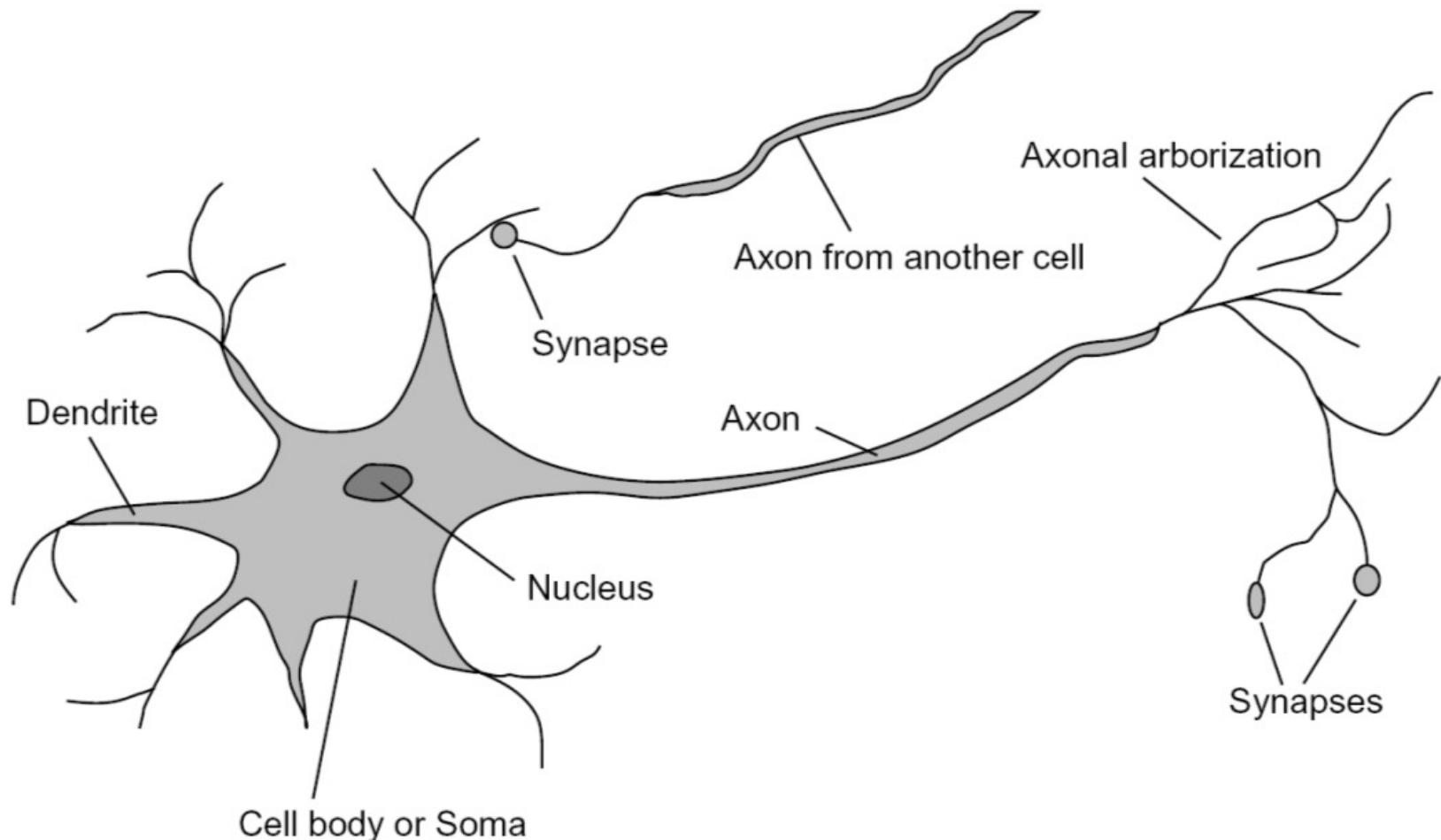
Neural Network

- Network of interconnected neurons

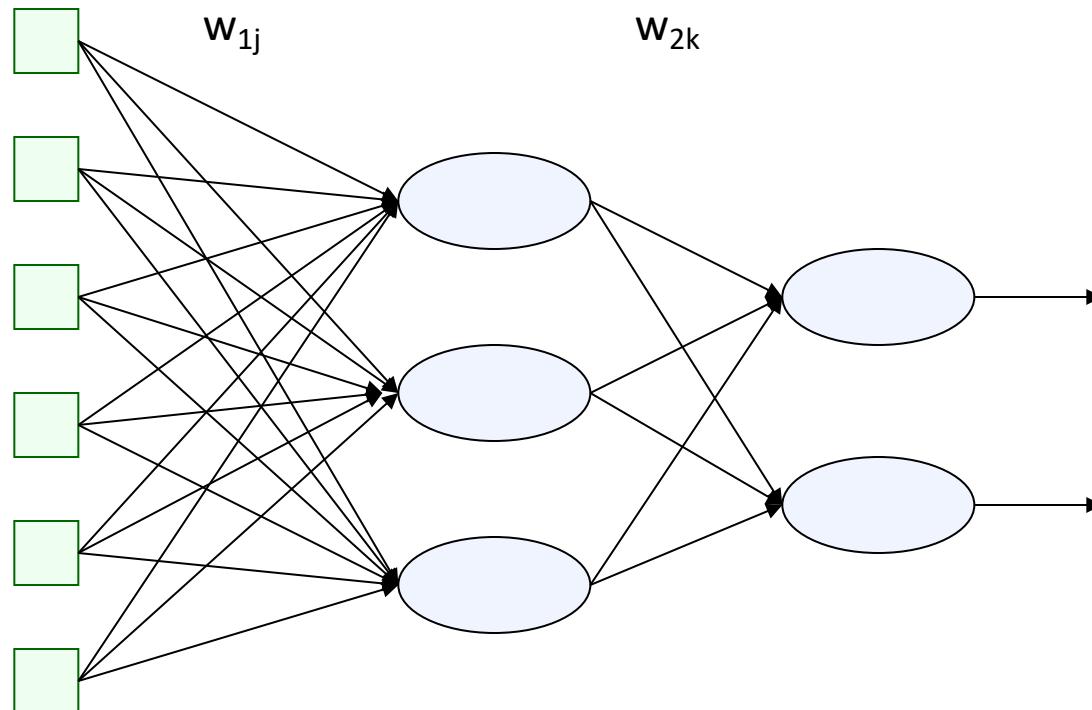


Acyclic (feed-forward) vs. recurrent networks

Inspiration: Neuron cells



Two-Layer Feed-Forward Neural Network



Inputs

Hidden
layer

Output
layer

Networks with hidden layers

- Can represent XORs, other nonlinear functions
- Many, many variants:
 - Different network structures
 - Different activation functions
 - Etc...
- As the number of hidden units increases, the network's capacity to learn more complicated functions also increases
- *How to train hidden layers?*

Next Class

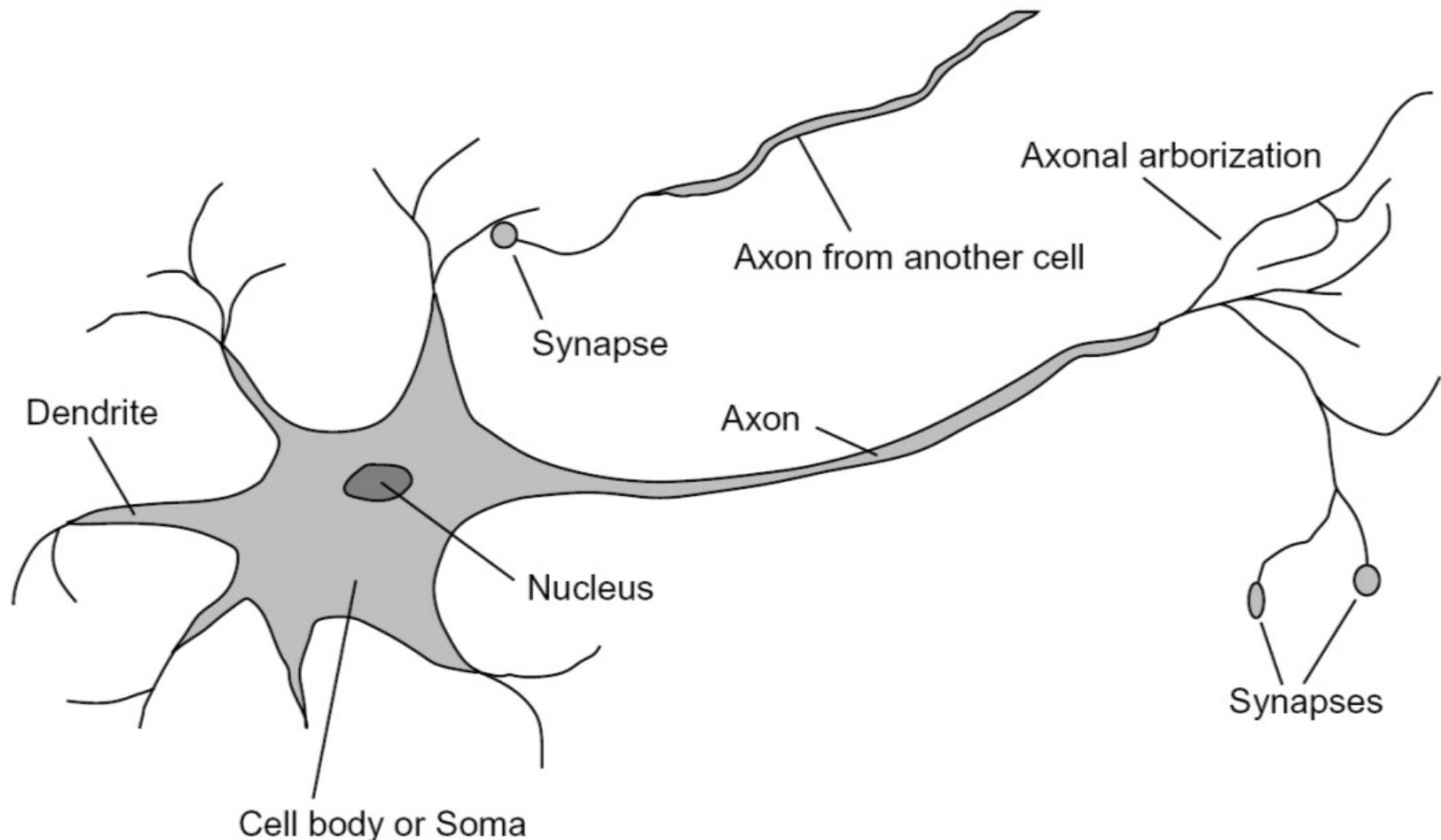
- Training Neural Networks

Training neural networks

Announcements

- A3 posted, sign up and create your teams

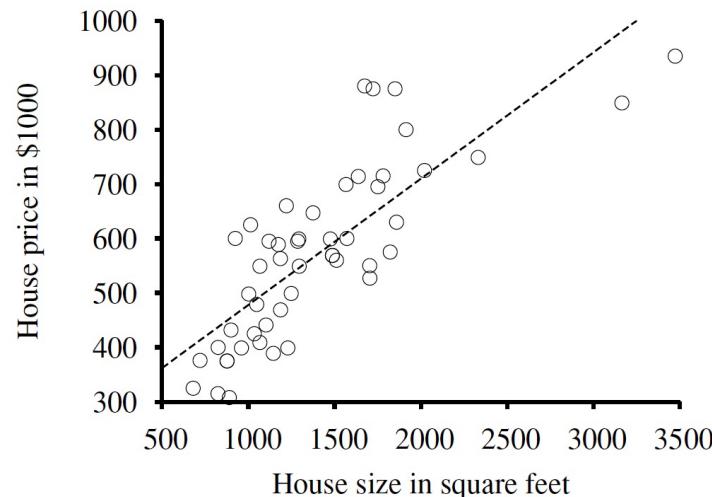
Inspiration: Neuron cells



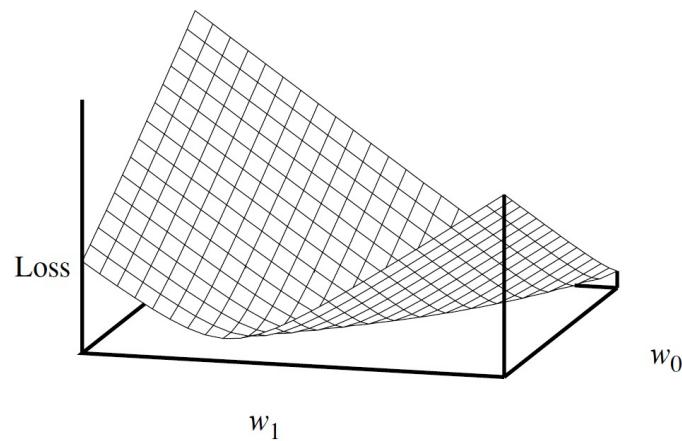
Networks with hidden layers

- Can represent XORs, other nonlinear functions
- Many, many variants:
 - Different network structures
 - Different activation functions
 - Etc...
- As the number of hidden units increases, the network's capacity to learn more complicated functions also increases
- *How to train hidden layers?*

Sometimes we can estimate parameters analytically: linear regression



(a)



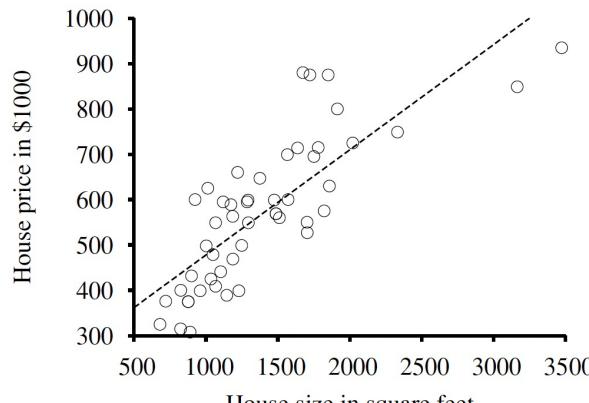
(b)

Figure 18.13 (a) Data points of price versus floor space of houses for sale in Berkeley, CA, in July 2009, along with the linear function hypothesis that minimizes squared error loss: $y = 0.232x + 246$. (b) Plot of the loss function $\sum_j (w_1 x_j + w_0 - y_j)^2$ for various values of w_0, w_1 . Note that the loss function is convex, with a single global minimum.

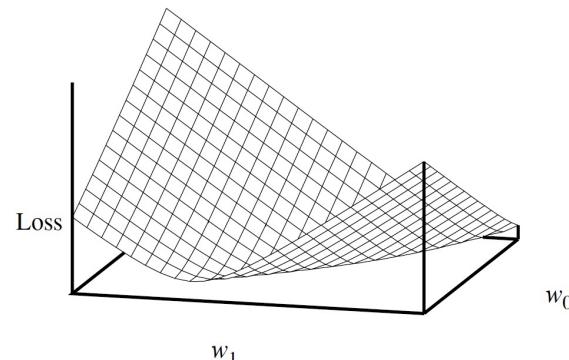
$$h_{\mathbf{w}}(x) = w_1 x + w_0$$

$$\text{Loss}(h_{\mathbf{w}}) = \sum_{j=1}^N L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^N (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2$$

Sometimes we can estimate parameters analytically: linear regression



(a)



(b)

Figure 18.13 (a) Data points of price versus floor space of houses for sale in Berkeley, CA, in July 2009, along with the linear function hypothesis that minimizes squared error loss: $y = 0.232x + 246$. (b) Plot of the loss function $\sum_j (w_1 x_j + w_0 - y_j)^2$ for various values of w_0, w_1 . Note that the loss function is convex, with a single global minimum.

We would like to find $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \text{Loss}(h_{\mathbf{w}})$. The sum $\sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2$ is minimized when its partial derivatives with respect to w_0 and w_1 are zero:

$$\frac{\partial}{\partial w_0} \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 = 0 \text{ and } \frac{\partial}{\partial w_1} \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 = 0 . \quad (18.2)$$

These equations have a unique solution:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = (\sum y_j - w_1(\sum x_j))/N . \quad (18.3)$$

- For models more complicated than linear regression, typically there is not closed-form solution (we can not estimate parameters analytically).
- Such problems can be addressed by a hill-climbing algorithm that follows the gradient of the function to be optimized.
- To minimize the loss, we will use gradient descent:

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

for each w_i **in** \mathbf{w} **do**

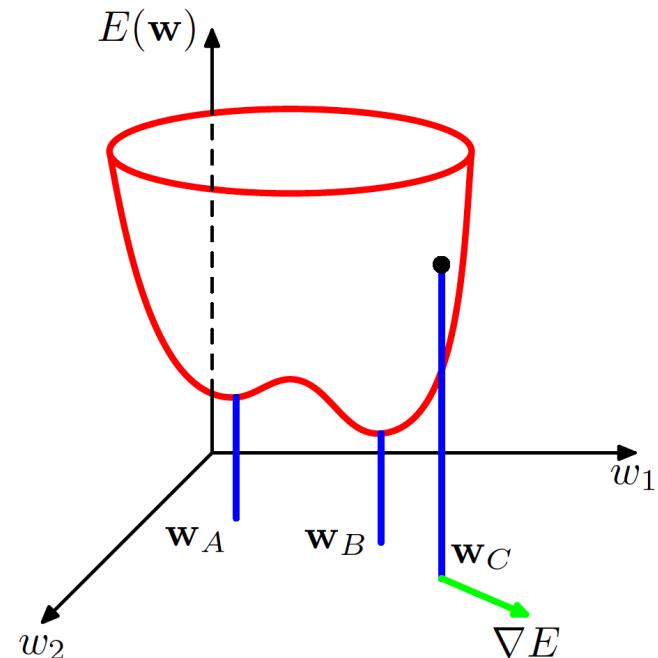
$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

α is the learning rate

Network training

If we make a small step in weight space from \mathbf{w} to $\mathbf{w} + \delta\mathbf{w}$ then the change in the error function (Loss) is $\delta E = \delta\mathbf{w}^\top \nabla E(\mathbf{w})$.

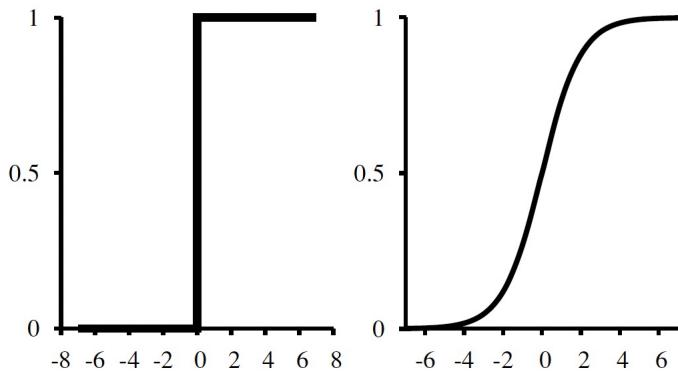
- where the vector $\nabla E(\mathbf{w})$ points in the direction of greatest rate of increase of the error function.
- Our goal is to find a vector \mathbf{w} such that $E(\mathbf{w})$ takes its smallest value.
- However, the error function typically has a highly nonlinear dependence on the weights and bias parameters, and so there will be many points in weight space at which the gradient vanishes (or is numerically very small).
- A minimum that corresponds to the smallest value of the error function for any weight vector is said to be a **global minimum** (\mathbf{w}_B in the figure).
- Any other minima corresponding to higher values of the error function are said to be **local minima** (\mathbf{w}_A in the figure).



Linear classification with logistic regression

Unlike step function, logistic function (also known as sigmoid) is differentiable.

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



Derivative of the logistic function:

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

Linear classification with logistic regression

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Remember chain rule: $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$

Linear classification with logistic regression

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Logistic}(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Remember chain rule: $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$

Let's derive weight update for minimizing the loss in logistic regression

$$\begin{aligned}\frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x})) \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i} \mathbf{w} \cdot \mathbf{x} \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i .\end{aligned}$$

Linear classification with logistic regression

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Logistic}(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Remember chain rule: $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$

Let's derive weight update for minimizing the loss in logistic regression

$$\begin{aligned}\frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x})) \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i} \mathbf{w} \cdot \mathbf{x} \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i .\end{aligned}$$

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

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

Linear classification with logistic regression

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Logistic}(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Remember chain rule: $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$

Let's derive weight update for minimizing the loss in logistic regression

$$\begin{aligned}\frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x})) \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i} \mathbf{w} \cdot \mathbf{x} \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i .\end{aligned}$$

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

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

$$w_i \leftarrow w_i + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

Learning in multilayer networks: error backpropagation

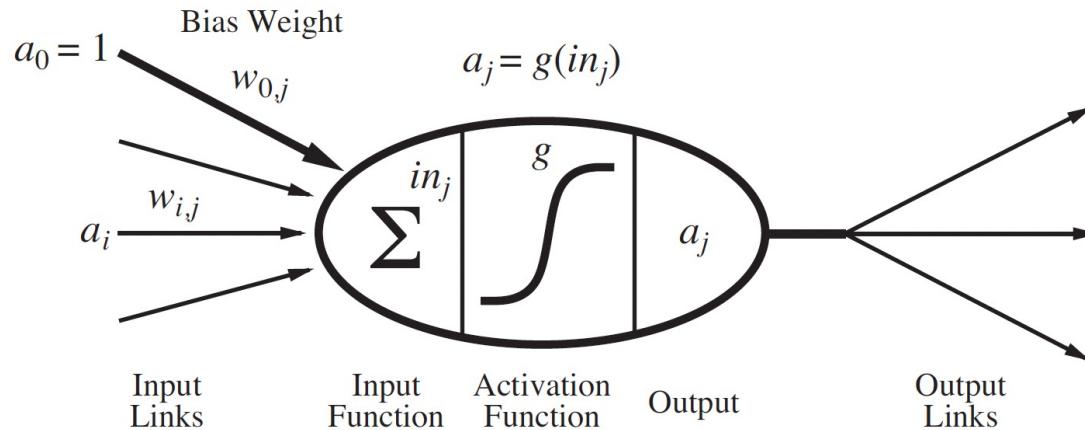


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^n w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

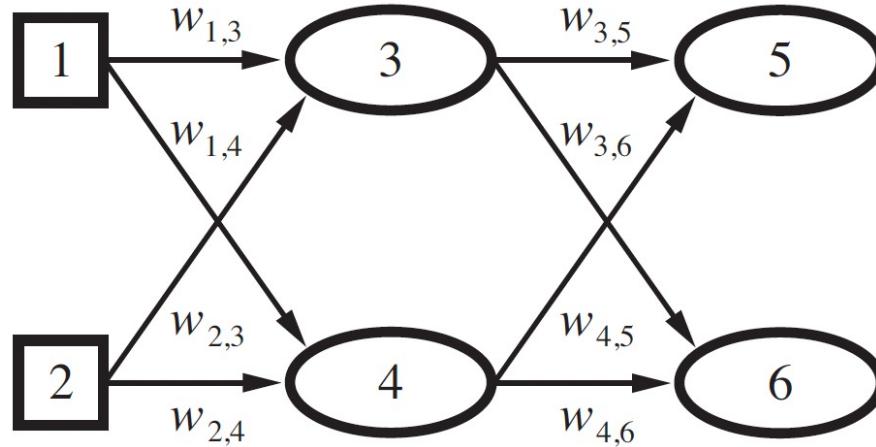
$$in_j = \sum_{i=0}^n w_{i,j} a_i$$

$$a_j = g(in_j) = g\left(\sum_{i=0}^n w_{i,j} a_i\right)$$

Learning in multilayer networks: error backpropagation

$$\frac{\partial}{\partial w} Loss(\mathbf{w}) = \frac{\partial}{\partial w} |\mathbf{y} - \mathbf{h}_{\mathbf{w}}(\mathbf{x})|^2 = \frac{\partial}{\partial w} \sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w} (y_k - a_k)^2$$

where the index k ranges over nodes in the output layer.



Learning in multilayer networks: error backpropagation

$$\frac{\partial}{\partial w} Loss(\mathbf{w}) = \frac{\partial}{\partial w} |\mathbf{y} - \mathbf{h}_{\mathbf{w}}(\mathbf{x})|^2 = \frac{\partial}{\partial w} \sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w} (y_k - a_k)^2$$

First, let's compute the gradient for loss at k-th output:

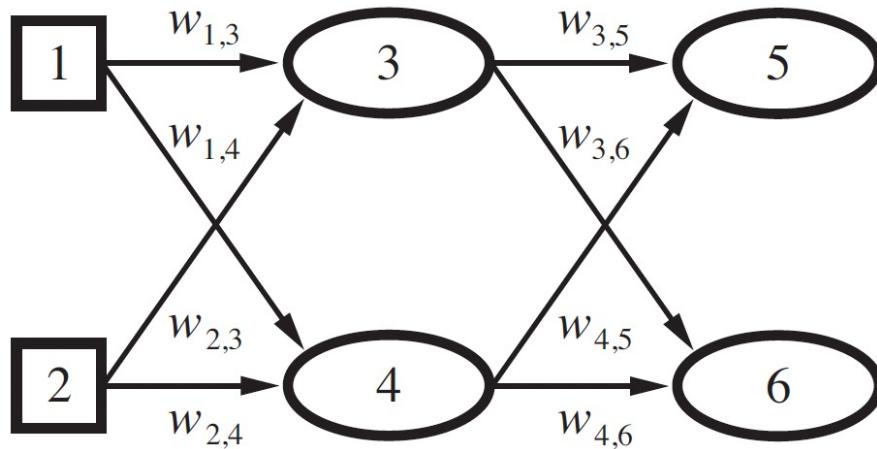
Remember chain rule: $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$

$$\begin{aligned}\frac{\partial Loss_k}{\partial w_{j,k}} &= -2(y_k - a_k) \frac{\partial a_k}{\partial w_{j,k}} = -2(y_k - a_k) \frac{\partial g(in_k)}{\partial w_{j,k}} \\ &= -2(y_k - a_k)g'(in_k) \frac{\partial in_k}{\partial w_{j,k}} = -2(y_k - a_k)g'(in_k) \frac{\partial}{\partial w_{j,k}} \left(\sum_j w_{j,k} a_j \right) \\ &= -2(y_k - a_k)g'(in_k)a_j = -a_j \Delta_k ,\end{aligned}$$

$$\Delta_k = Err_k \times g'(in_k)$$

Learning in multilayer networks: error backpropagation

Next we compute gradient with respect to the $w_{i,j}$ weights connecting the input layer to the hidden layer



Learning in multilayer networks: error backpropagation

Next, we compute gradient with respect to the $w_{i,j}$ weights connecting the input layer to the hidden layer

$$\begin{aligned}\frac{\partial Loss_k}{\partial w_{i,j}} &= -2(y_k - a_k) \frac{\partial a_k}{\partial w_{i,j}} = -2(y_k - a_k) \frac{\partial g(in_k)}{\partial w_{i,j}} \\ &= -2(y_k - a_k)g'(in_k) \frac{\partial in_k}{\partial w_{i,j}} = -2\Delta_k \frac{\partial}{\partial w_{i,j}} \left(\sum_j w_{j,k} a_j \right) \\ &= -2\Delta_k w_{j,k} \frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k} \frac{\partial g(in_j)}{\partial w_{i,j}} \\ &= -2\Delta_k w_{j,k}g'(in_j) \frac{\partial in_j}{\partial w_{i,j}} \\ &= -2\Delta_k w_{j,k}g'(in_j) \frac{\partial}{\partial w_{i,j}} \left(\sum_i w_{i,j} a_i \right) \\ &= -2\Delta_k w_{j,k}g'(in_j)a_i = -a_i \Delta_j , \\ &\qquad\qquad\qquad \Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k\end{aligned}$$

Backpropagation Algorithm

- Werbos, Rumelhart, Hinton, Williams (1974)
- Until convergence:
 - Present a training pattern to network
 - Calculate the error of the output nodes
 - Calculate the error of the hidden nodes, based on the output node error which is propagated back
 - Continue back-propagating error until the input layer
 - Update all weights in the network

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network

inputs: *examples*, a set of examples, each with input vector \mathbf{x} and output vector \mathbf{y}

network, a multilayer network with L layers, weights $w_{i,j}$, activation function g

local variables: Δ , a vector of errors, indexed by network node

repeat

for each weight $w_{i,j}$ in *network* **do**

$w_{i,j} \leftarrow$ a small random number

for each example (\mathbf{x}, \mathbf{y}) in *examples* **do**

/* Propagate the inputs forward to compute the outputs */

for each node i in the input layer **do**

$a_i \leftarrow x_i$

for $\ell = 2$ **to** L **do**

for each node j in layer ℓ **do**

$in_j \leftarrow \sum_i w_{i,j} a_i$

$a_j \leftarrow g(in_j)$

/* Propagate deltas backward from output layer to input layer */

for each node j in the output layer **do**

$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

for $\ell = L - 1$ **to** 1 **do**

for each node i in layer ℓ **do**

$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$

/* Update every weight in network using deltas */

for each weight $w_{i,j}$ in *network* **do**

$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$

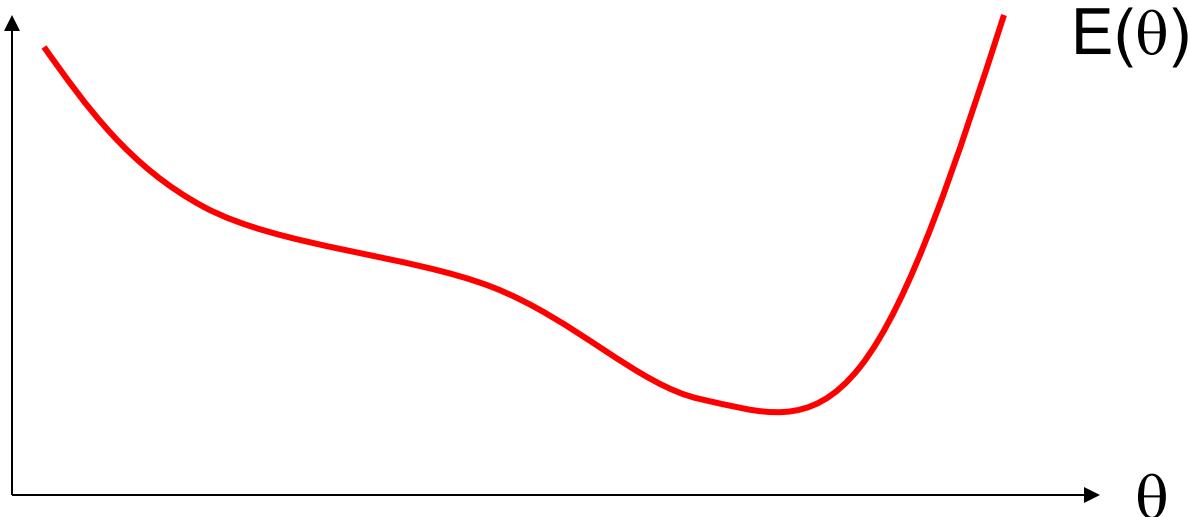
until some stopping criterion is satisfied

return *network*

Figure 18.24 The back-propagation algorithm for learning in multilayer networks.

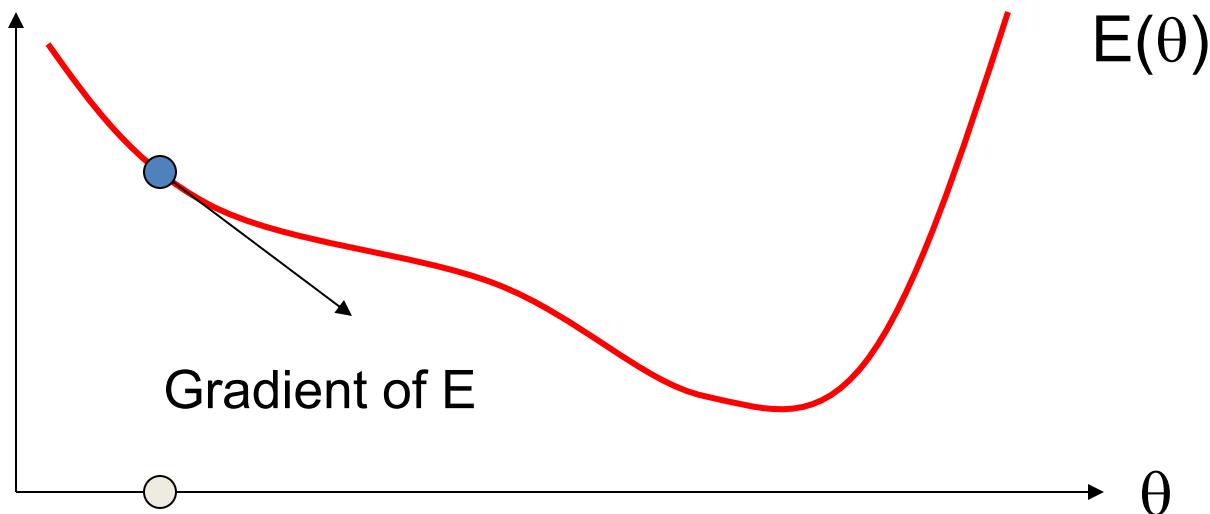
Understanding Backpropagation

- Minimize $E(\theta)$
- Gradient Descent...



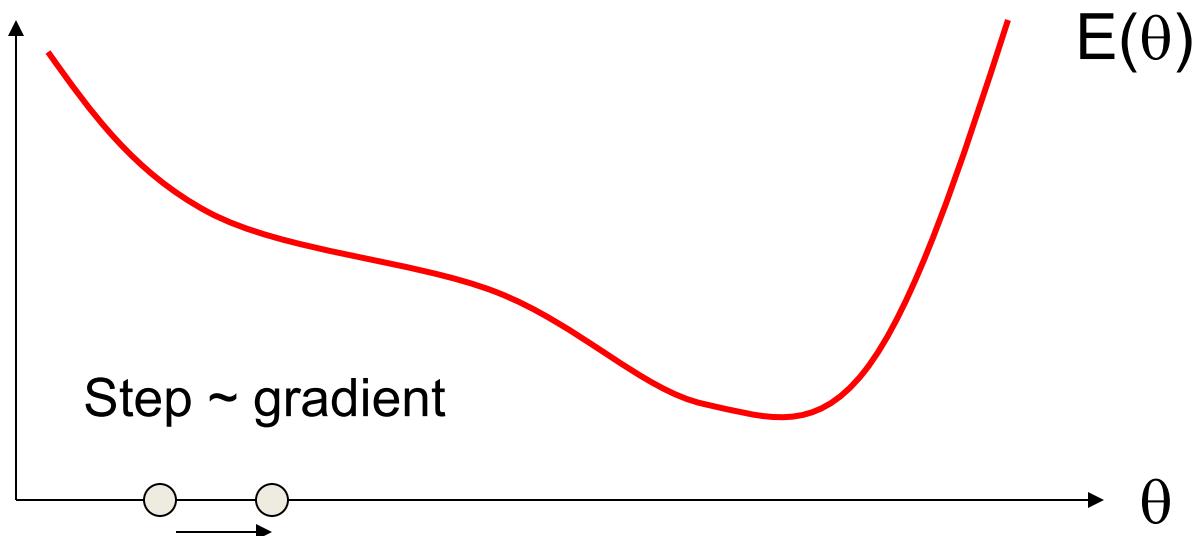
Understanding Backpropagation

- Minimize $E(\theta)$
- Gradient Descent...



Understanding Backpropagation

- Minimize $E(\theta)$
- Gradient Descent...

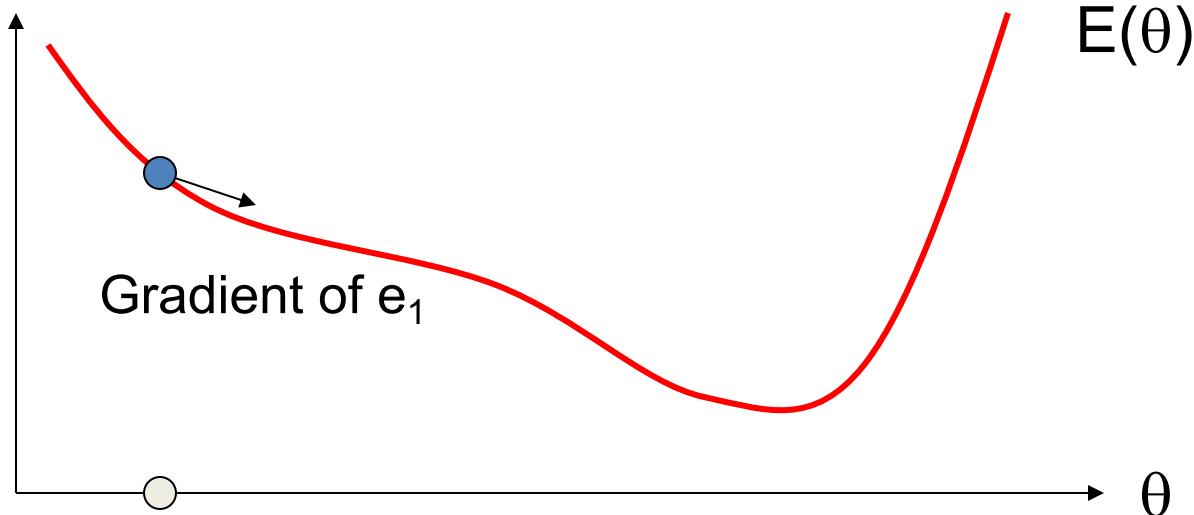


Stochastic Gradient Descent

- Classic backprop computes weight changes after scanning through the entire training set
 - Theoretically justified
 - But this is very slow
- Stochastic gradient descent randomizes the input data, then takes a step after each training exemplar

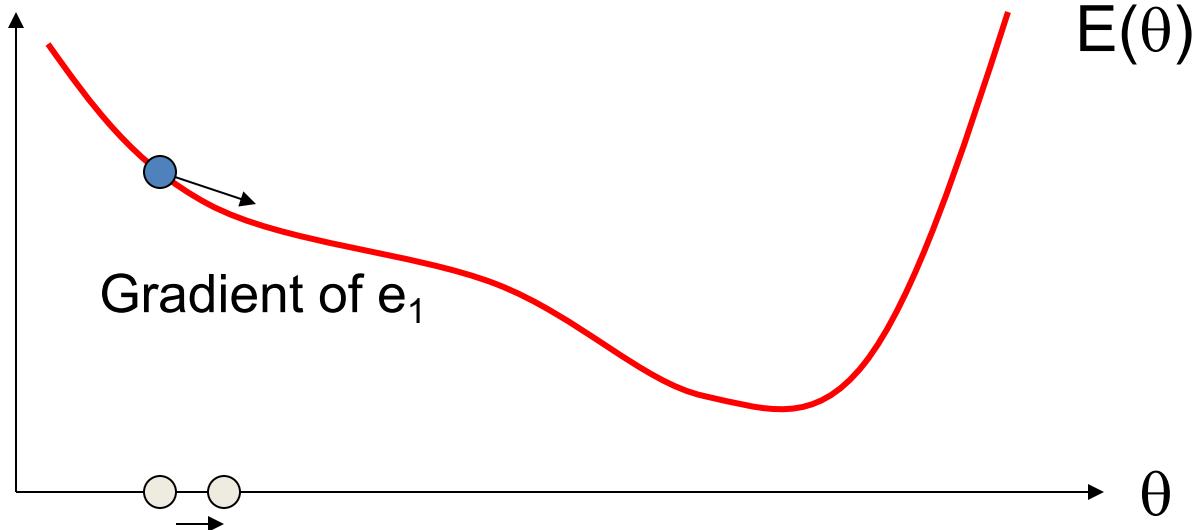
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q)+e_2(q)+\dots+e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



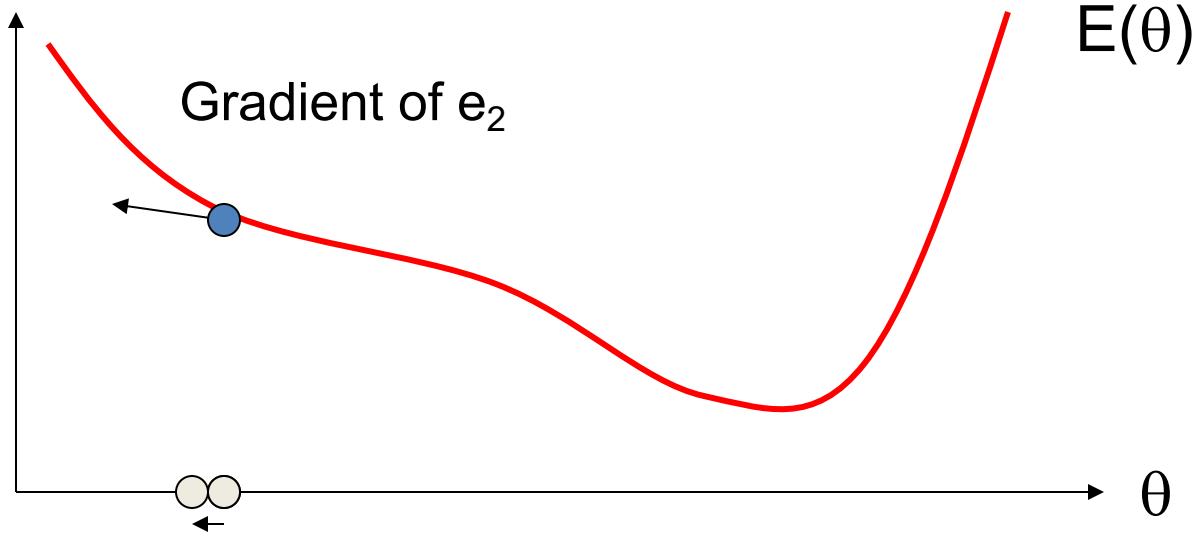
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



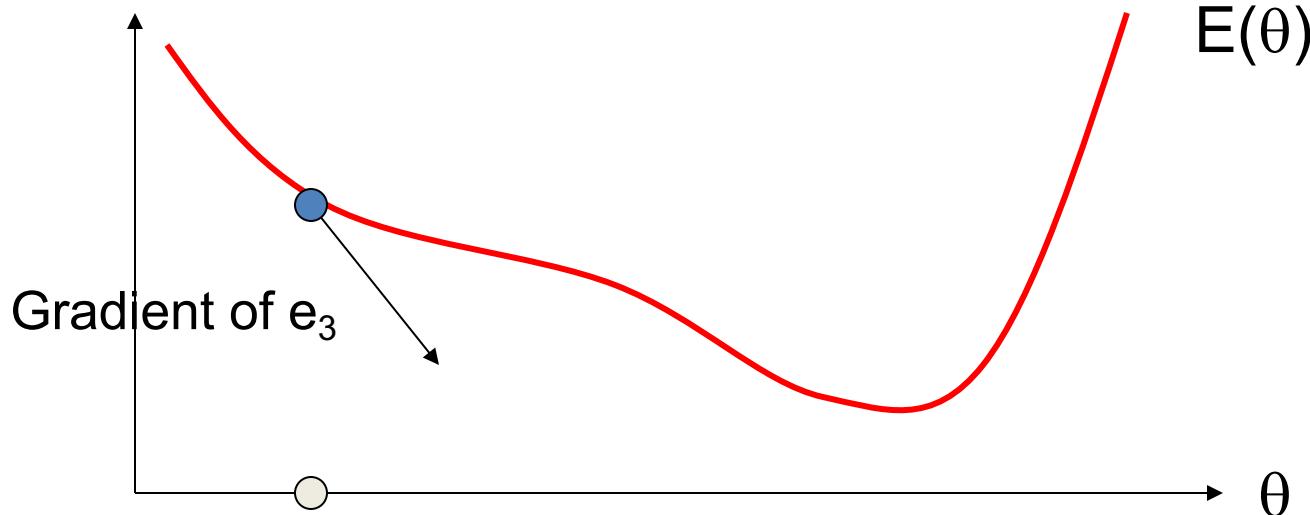
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q)+e_2(q)+\dots+e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



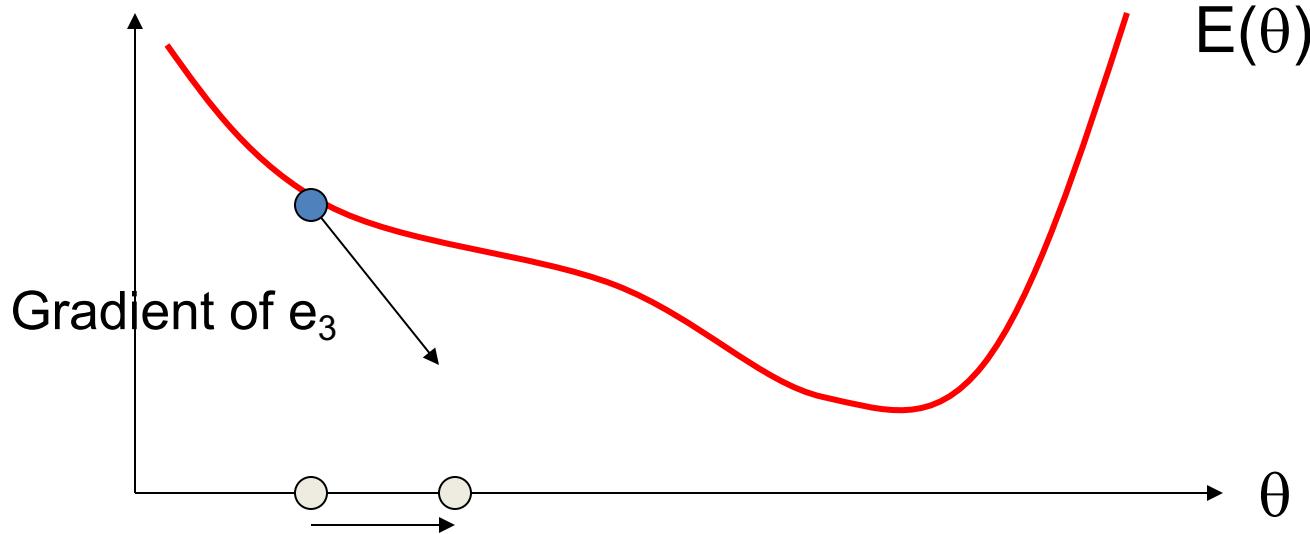
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



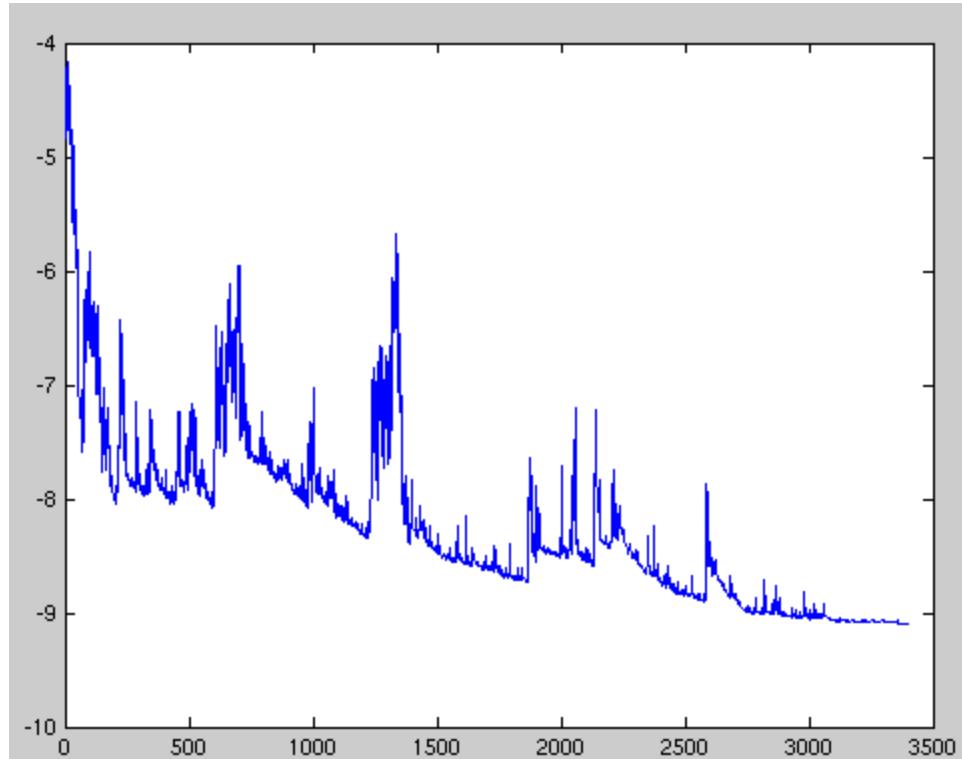
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



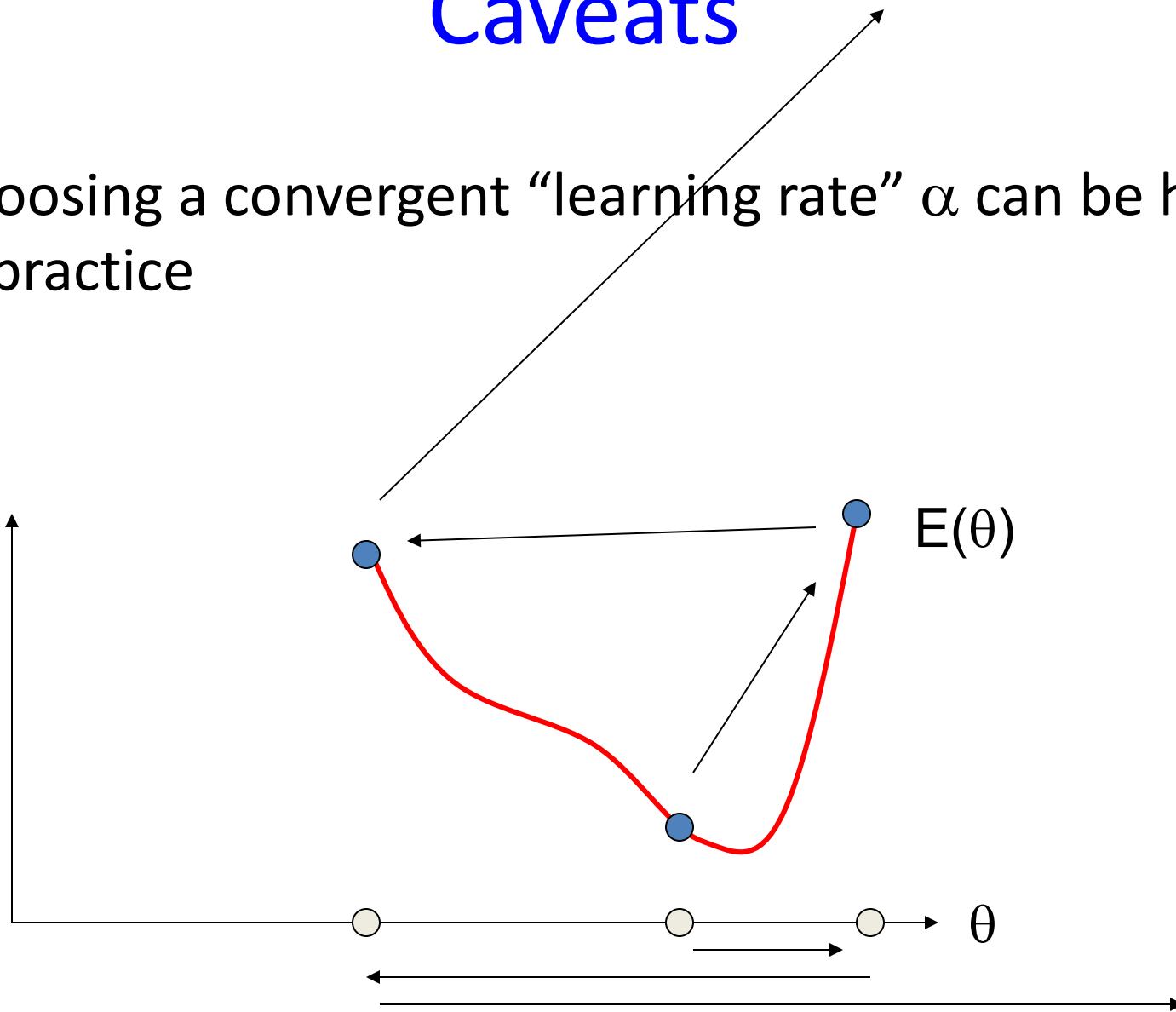
Stochastic Gradient Descent

- Objective function values (measured over all examples) over time settle into local minimum
- Step size must be reduced over time, e.g., $O(1/t)$



Caveats

- Choosing a convergent “learning rate” α can be hard in practice



Neural networks

- Neural networks are *universal function approximators*
 - Given any function, and a complicated enough network, they can accurately model that function



- How to choose the size and structure of networks?
 - If network is too large, risk of over-fitting (data caching)
 - If network is too small, representation may not be rich enough

Pros and cons of different classifiers

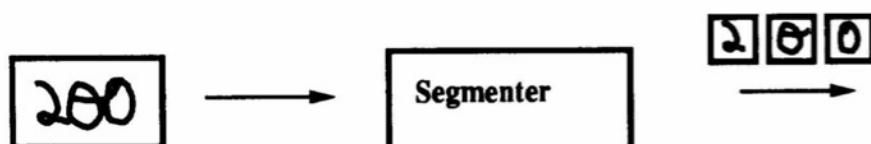
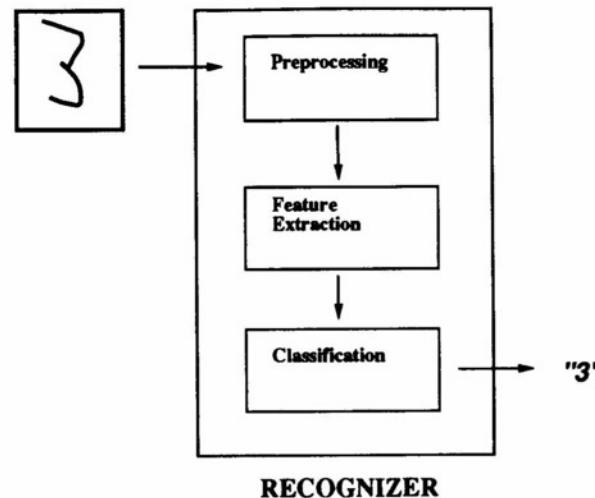
- Nearest neighbors
 - Can model any data, very prone to overfitting, requires distance function, fast learning, slow classification
- Neural networks
 - Models any function, requires structure, can suffer from local minima, slow learning, fast classification, difficult to interpret.
- Bayes nets
 - Requires setting network structure, fast learning, fast classification, intuitive interpretation of parameters.
- Decision trees
 - Limited modeling power, mostly automatic, moderate learning speed, fast classification, intuitive interpretation of parameters.
- Perceptrons
 - Very limited modeling power, fast training, fast classification, intuitive interpretation of parameters.

Neural Nets: 1960s-1990s

- Failure to deliver perceptron promises during 1960s-1970s led to “AI winter”
- In 1980s, multi-layer networks and the backpropagation algorithm led to new excitement, new era of neural network research

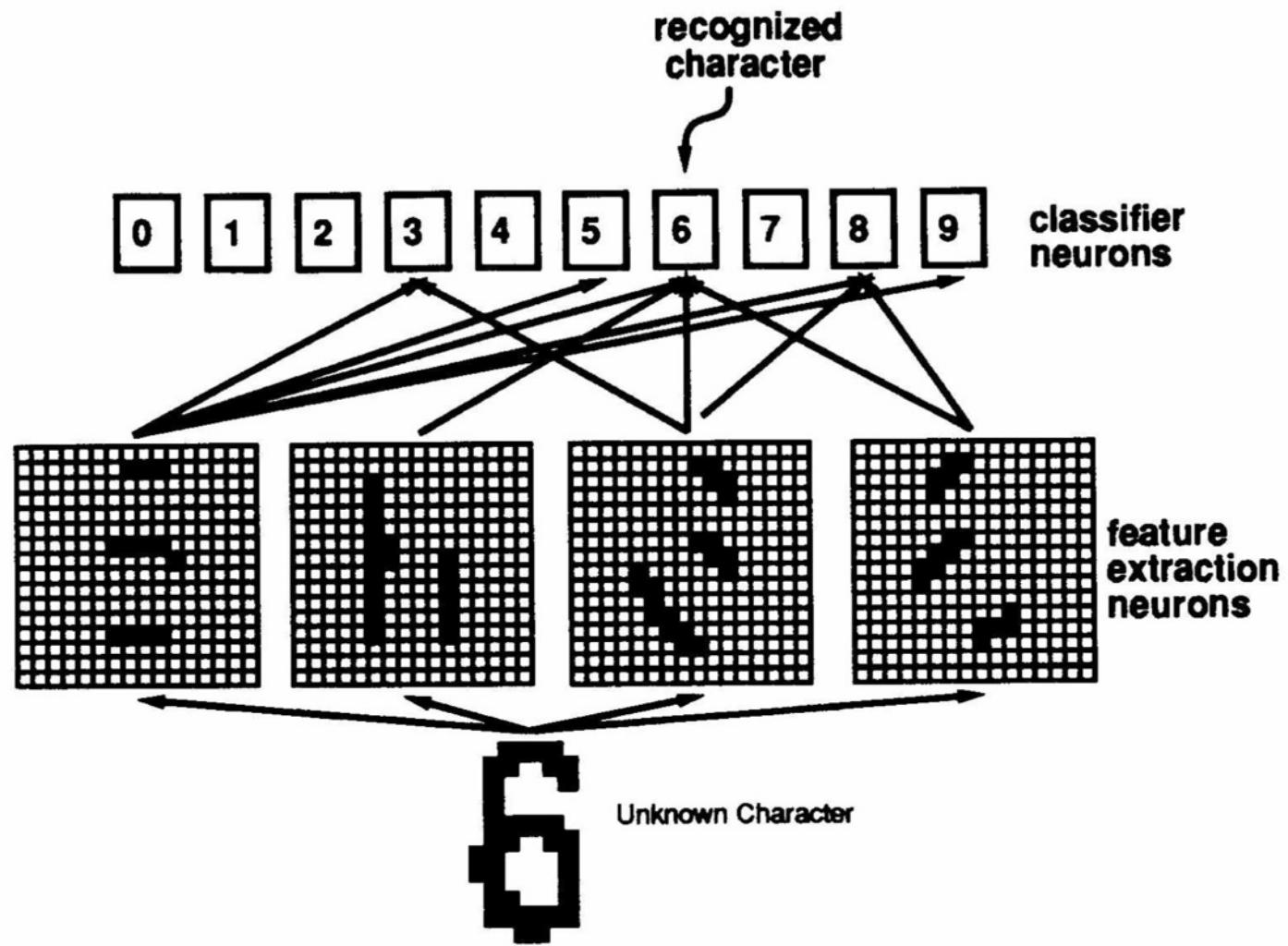
Success story: handwritten digit recognition (LeCun, 1989)

40004 75216
14199-2087 23505
96203 14310
44151 05753

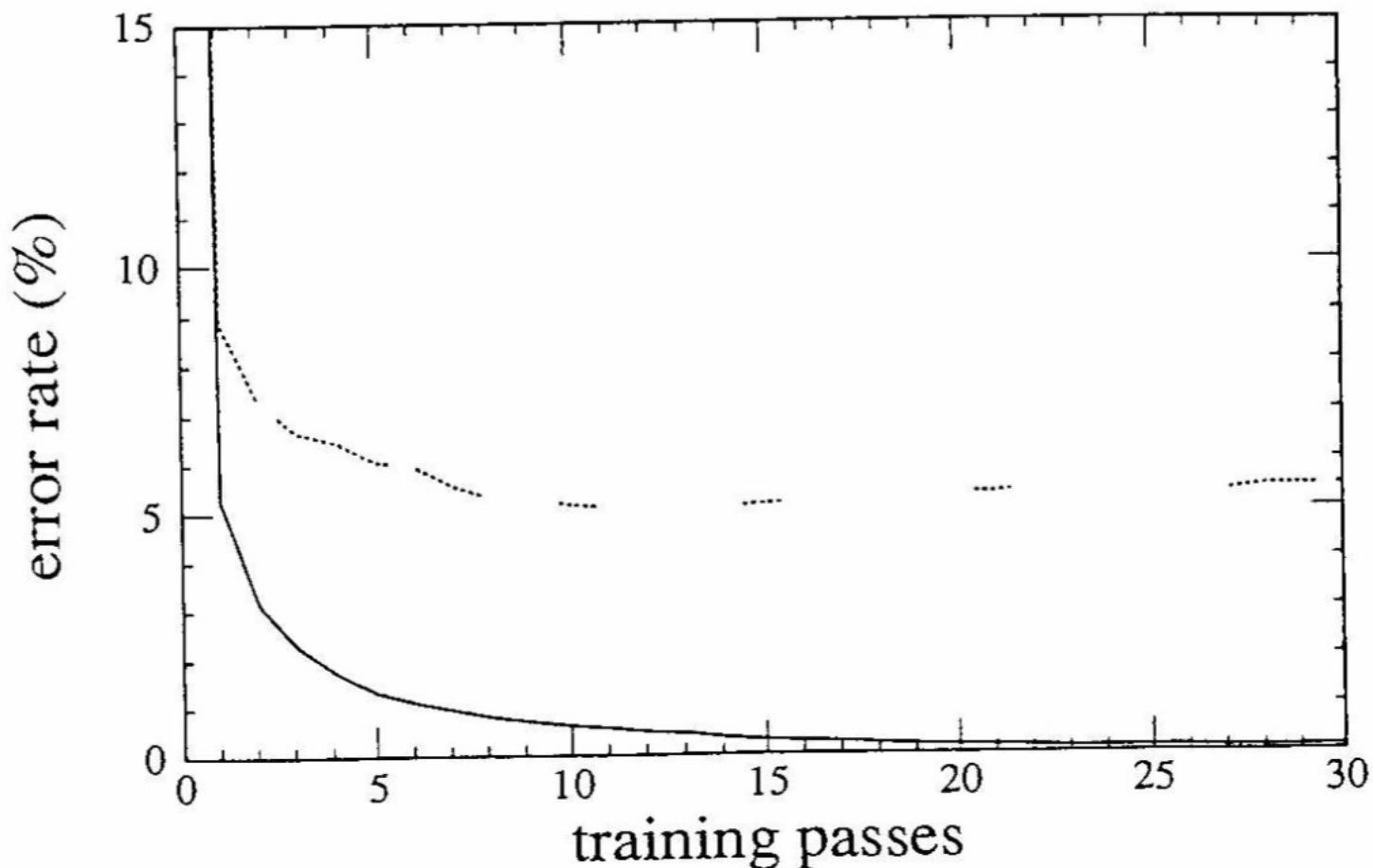


1 4 1 6 1 1 9 1 3 4 8 5 7 2 6 8 0 3 2 2
8 6 6 3 5 9 7 2 0 2 9 9 2 9 9 7 2 2 5 1
0 1 3 0 8 4 1 1 1 5 9 1 0 1 0 6 1 5 4 0
3 1 1 0 6 4 1 1 1 0 3 0 4 7 5 2 6 2 0 0
6 6 8 9 1 2 0 7 6 7 0 8 5 5 7 1 3 1 4 2
6 0 2 0 1 7 2 3 0 1 8 7 1 1 2 9 9 1 0 8
8 4 0 1 0 9 7 0 7 5 9 7 3 3 1 9 7 2 0 1
5 5 1 0 7 5 5 1 8 2 5 5 1 8 2 8 1 4 3 5
4 3 1 7 8 7 5 4 1 6 5 5 4 6 0 3 5 4 6 0
5 5 1 8 2 5 5 1 0 8 5 0 3 0 4 7 5 2 0 4

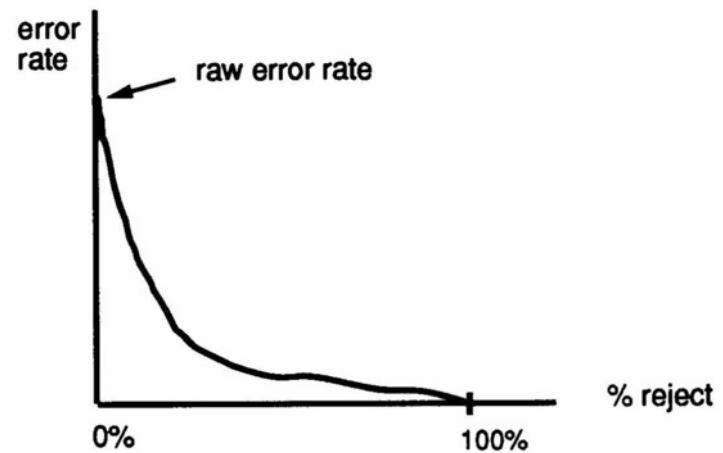
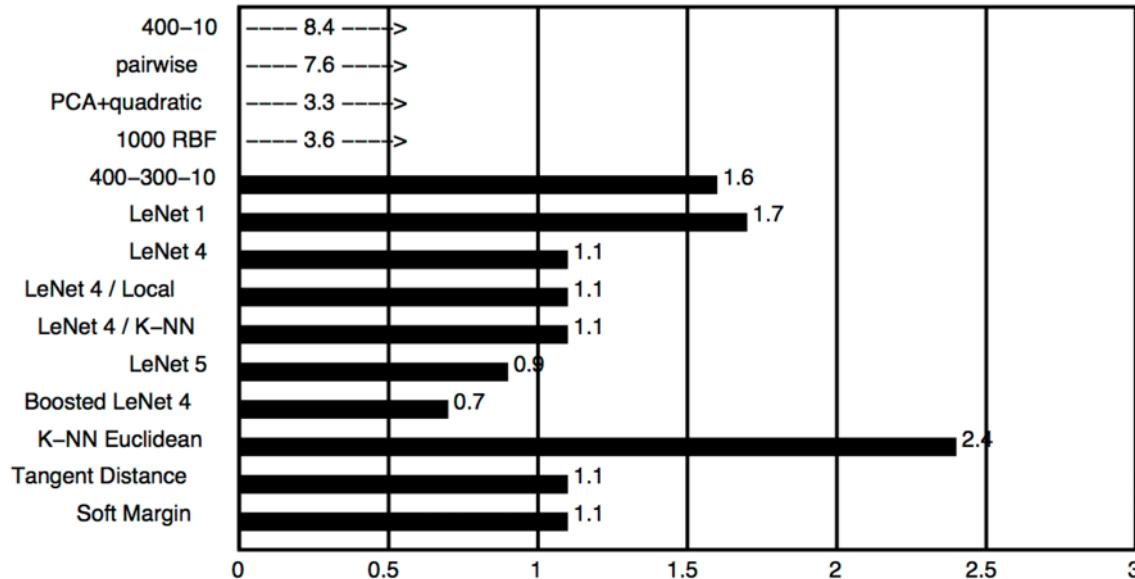
Network structure



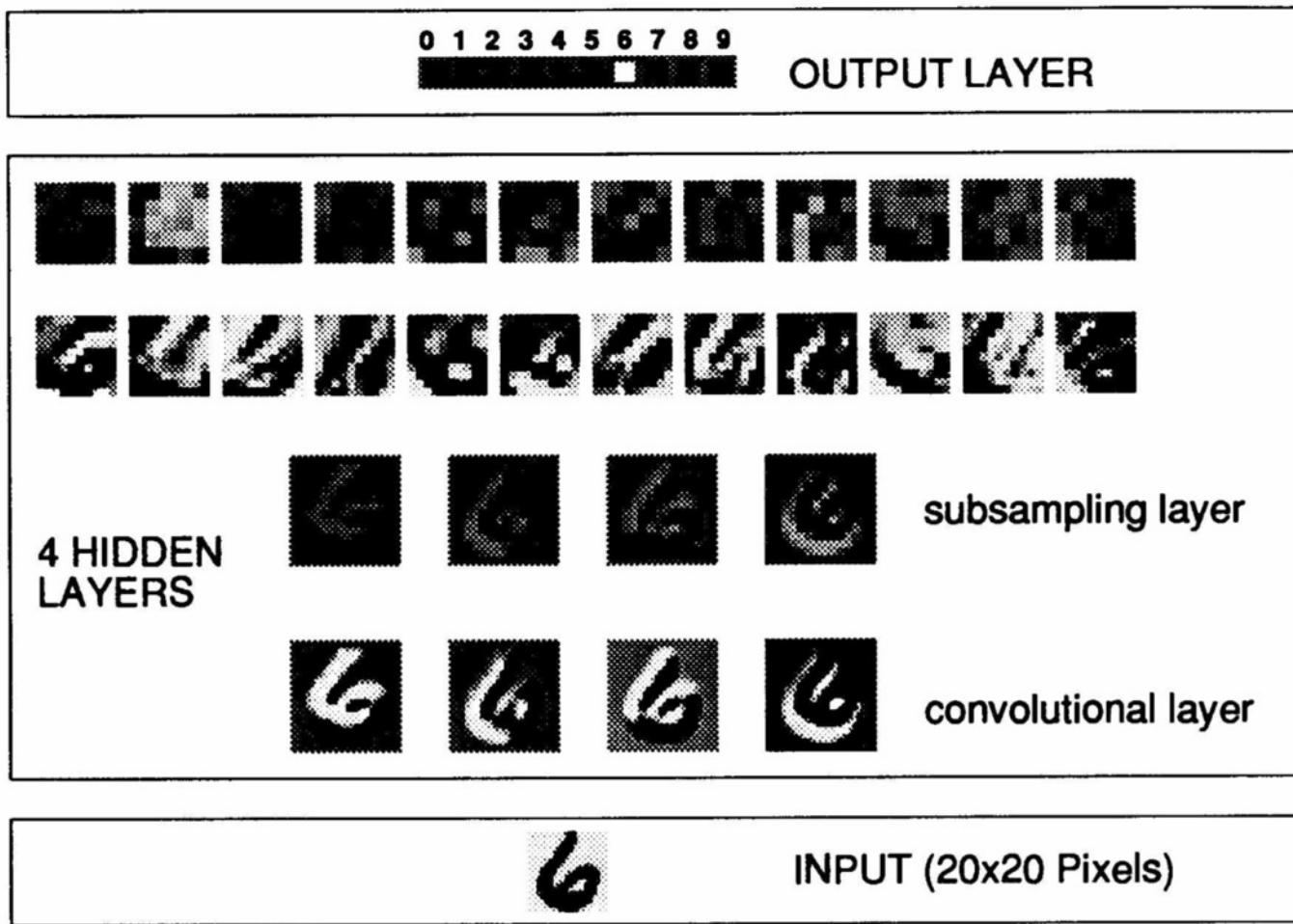
Use backprop to train



Worked better than other techniques (LeCun 1989)

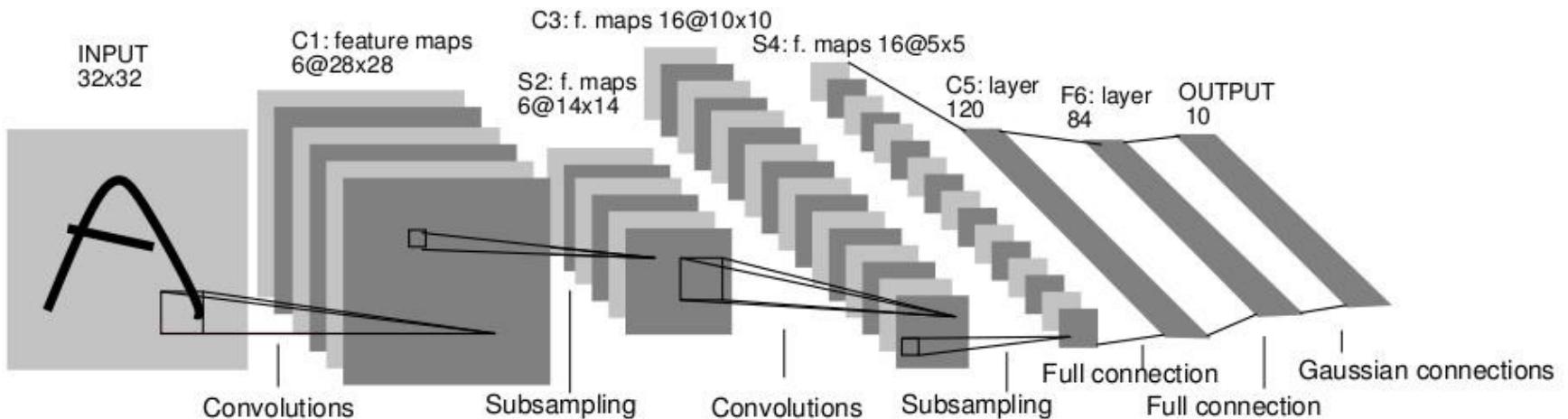
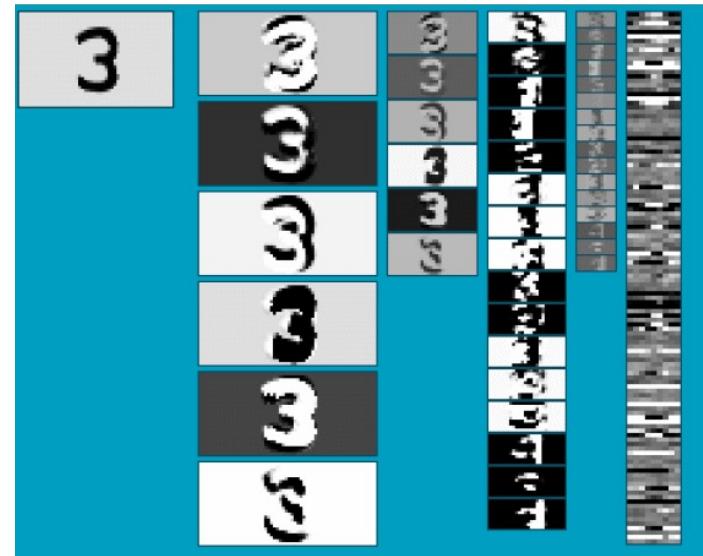


More complex architectures...

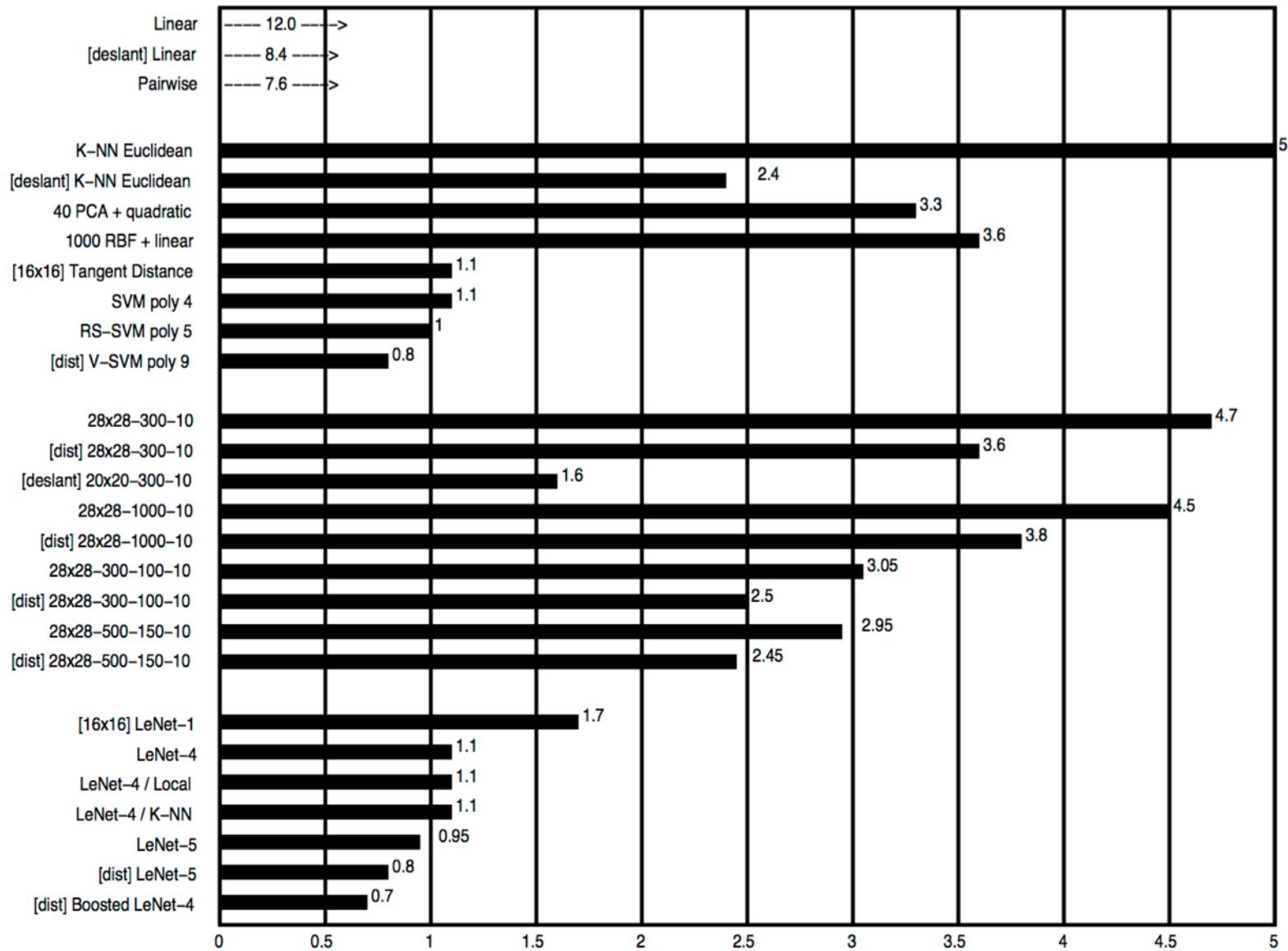


Convolutional Neural Networks

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end



But other techniques catching up (LeCun 1998)



Late 1990s-2010: Another decline

- Neural networks failed to work equally well on more complicated problems
 - E.g. recognition in real images, real audio streams, etc.
- Mix of practical and theoretical problems
 - How to decide network structure and many learning parameters (e.g. step sizes)?
 - Required too much computation
 - Required too much data
 - Very difficult to “debug” failures

2000's: Return to the simple

- Return to simpler techniques, like linear classifiers
 - But in high dimensions
 - Simpler learning algorithms, easier to justify theoretically
- Learn classifiers on manually-created features
 - E.g. not images themselves, but statistical features like color histograms, edge distributions, etc.

Next class

- Support Vector Machines (SVM)

More neural networks and support vector machines (SVMs)

Announcements

- A3 released (due on December 2nd), fill out the teams form by the end of the day (everyone should fill out the form)
- Five classes left! And some work:
 - Assignment 4
 - Final exam

Backpropagation Algorithm

- Werbos, Rumelhart, Hinton, Williams (1974)
- Until convergence:
 - Present a training pattern to network
 - Calculate the error of the output nodes
 - Calculate the error of the hidden nodes, based on the output node error which is propagated back
 - Continue back-propagating error until the input layer
 - Update all weights in the network

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network

inputs: *examples*, a set of examples, each with input vector \mathbf{x} and output vector \mathbf{y}

network, a multilayer network with L layers, weights $w_{i,j}$, activation function g

local variables: Δ , a vector of errors, indexed by network node

repeat

for each weight $w_{i,j}$ in *network* **do**

$w_{i,j} \leftarrow$ a small random number

for each example (\mathbf{x}, \mathbf{y}) in *examples* **do**

/* Propagate the inputs forward to compute the outputs */

for each node i in the input layer **do**

$a_i \leftarrow x_i$

for $\ell = 2$ **to** L **do**

for each node j in layer ℓ **do**

$in_j \leftarrow \sum_i w_{i,j} a_i$

$a_j \leftarrow g(in_j)$

/* Propagate deltas backward from output layer to input layer */

for each node j in the output layer **do**

$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

for $\ell = L - 1$ **to** 1 **do**

for each node i in layer ℓ **do**

$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$

/* Update every weight in network using deltas */

for each weight $w_{i,j}$ in *network* **do**

$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$

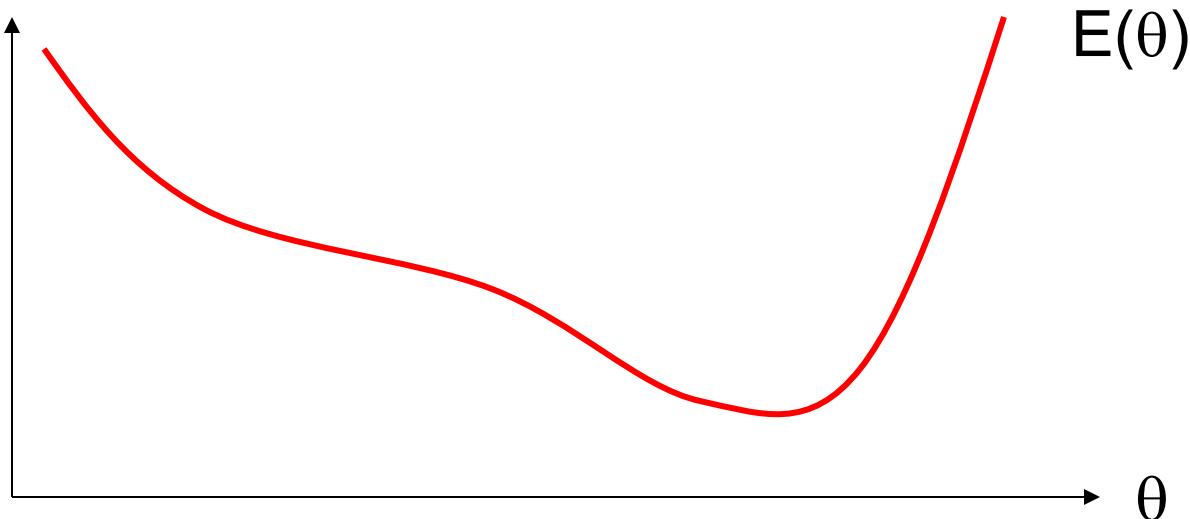
until some stopping criterion is satisfied

return *network*

Figure 18.24 The back-propagation algorithm for learning in multilayer networks.

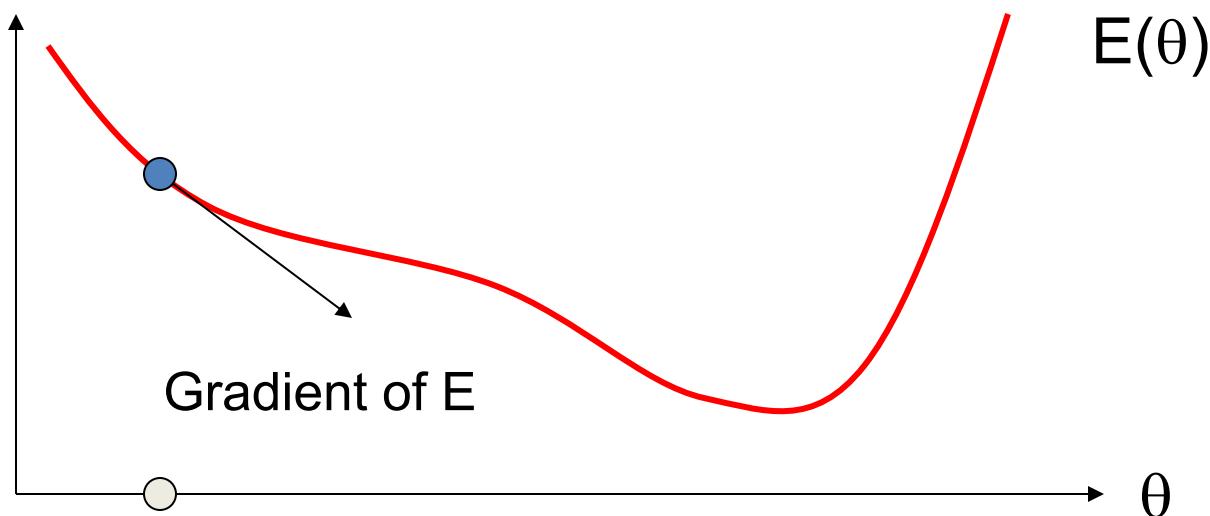
Understanding Backpropagation

- Minimize $E(\theta)$
- Gradient Descent...



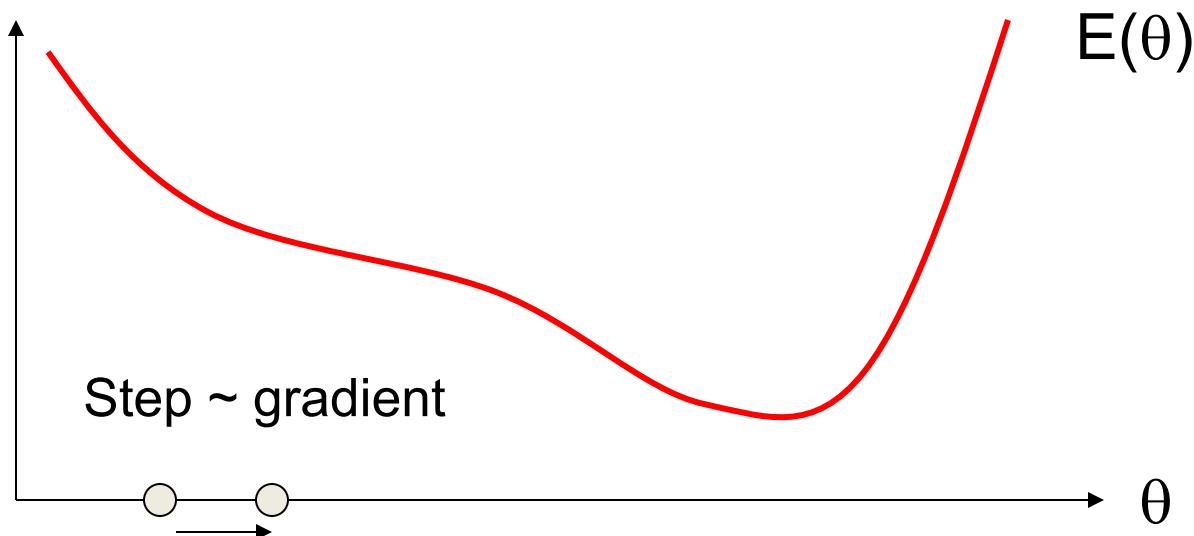
Understanding Backpropagation

- Minimize $E(\theta)$
- Gradient Descent...



Understanding Backpropagation

- Minimize $E(\theta)$
- Gradient Descent...

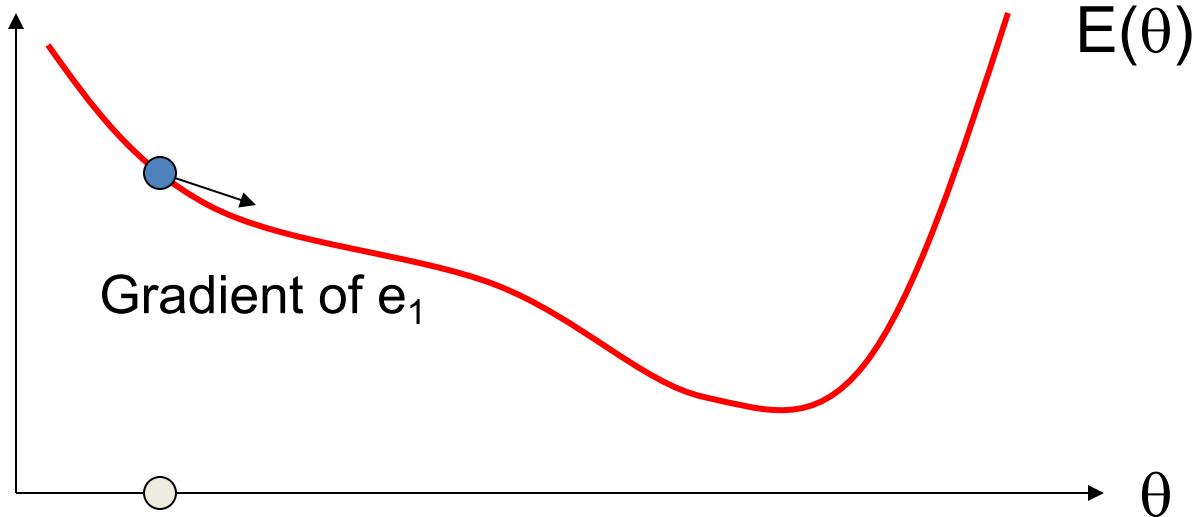


Stochastic Gradient Descent

- Classic backprop computes weight changes after scanning through the entire training set
 - Theoretically justified
 - But this is very slow
- Stochastic gradient descent randomizes the input data, then takes a step after each training exemplar

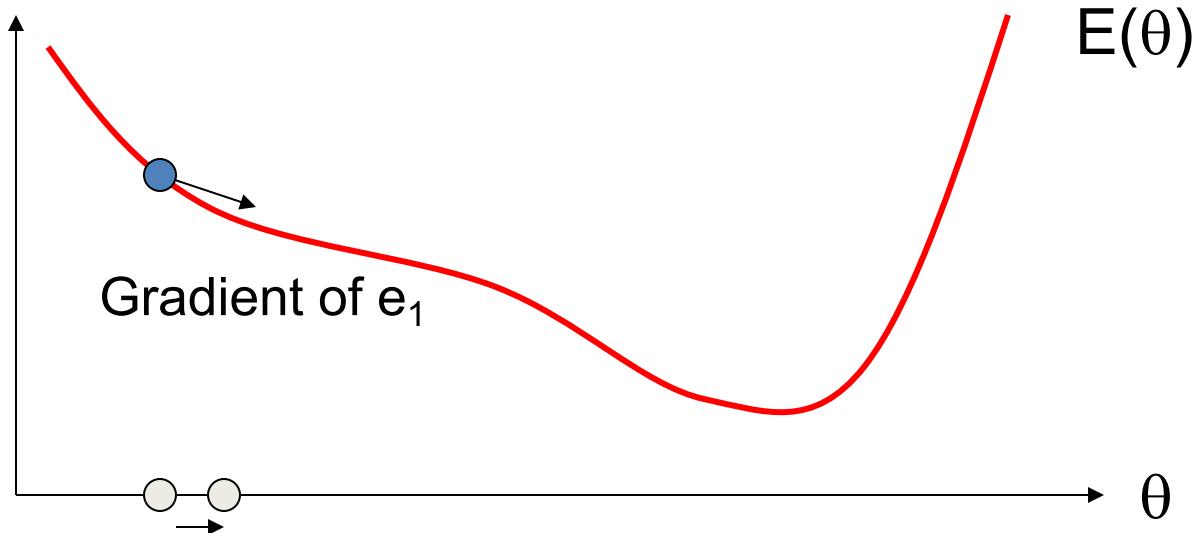
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



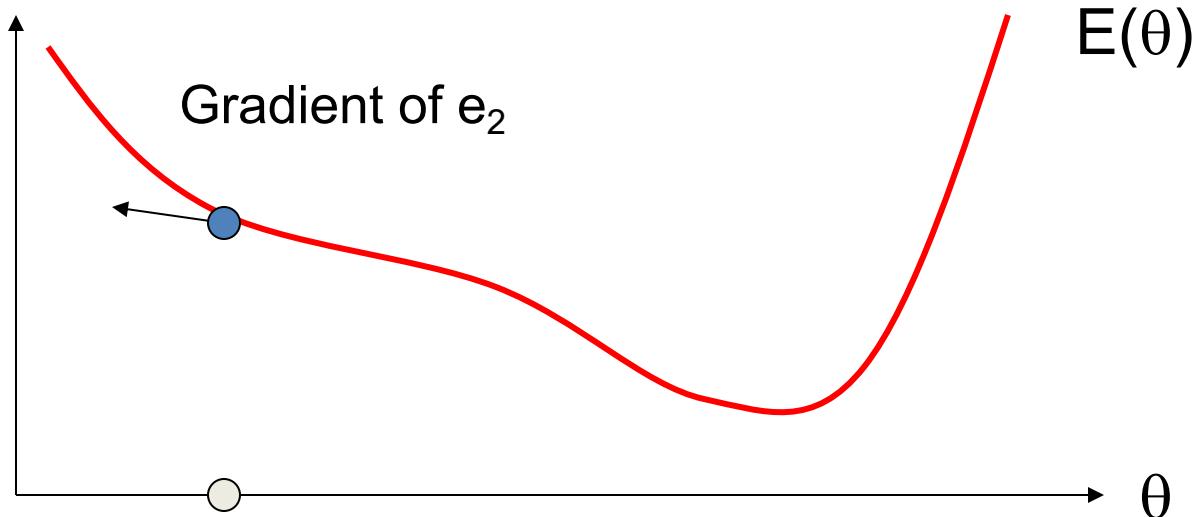
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q)+e_2(q)+\dots+e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



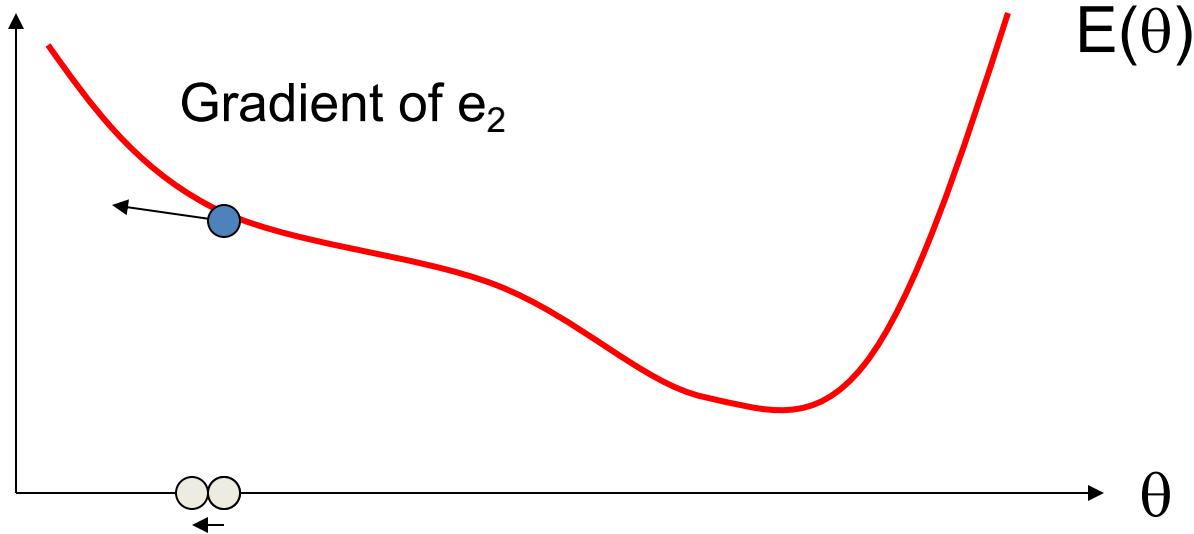
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



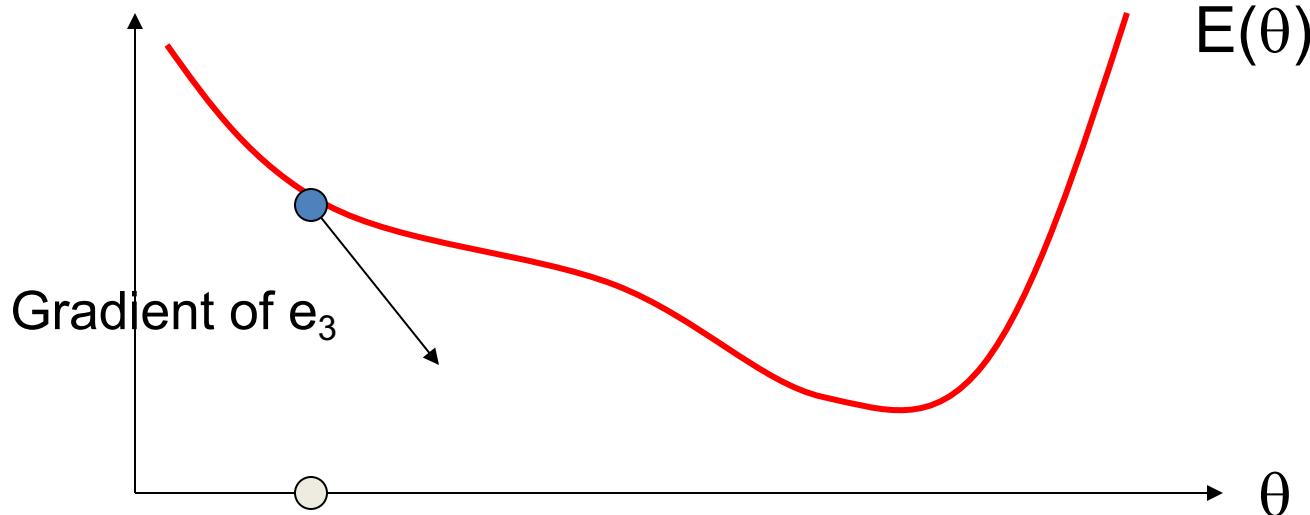
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q)+e_2(q)+\dots+e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



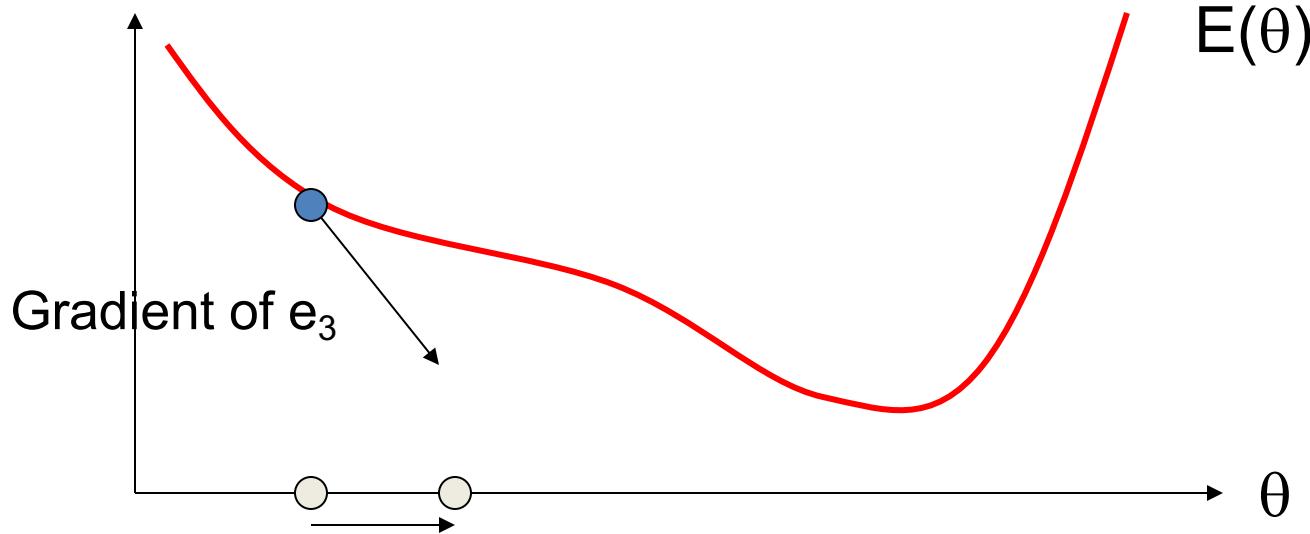
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



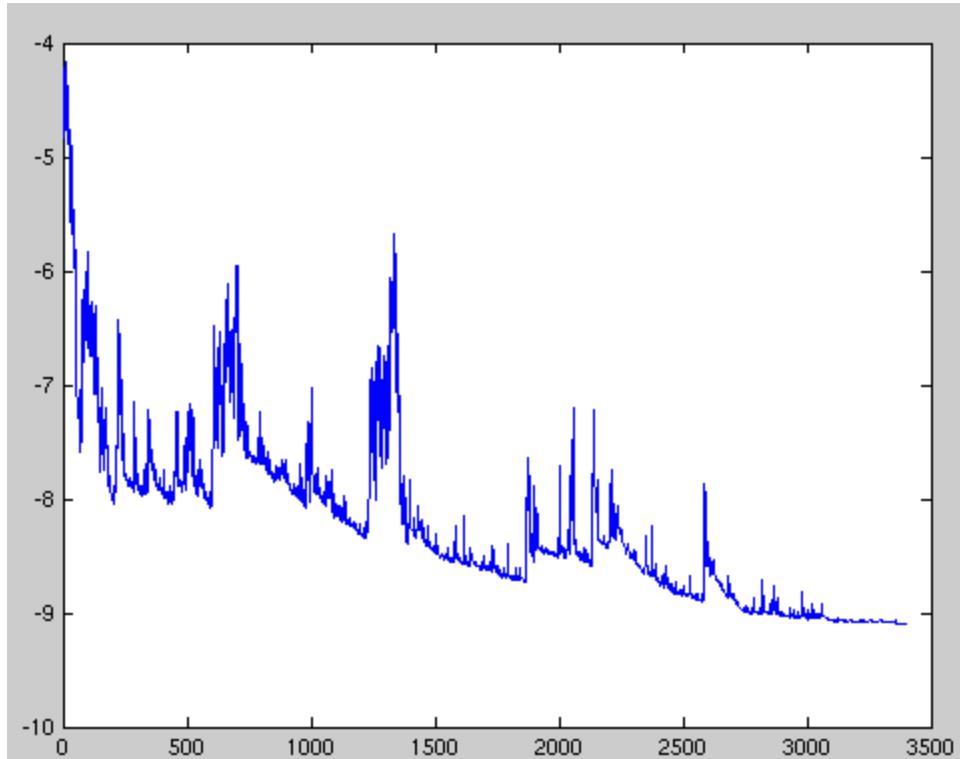
Understanding Backpropagation

- Example of Stochastic Gradient Descent
- Decompose $E(\theta) = e_1(q) + e_2(q) + \dots + e_N(q)$
 - Here $e_k = (g(x^{(k)}, \theta) - y^{(k)})^2$
- On each iteration take a step to reduce e_k



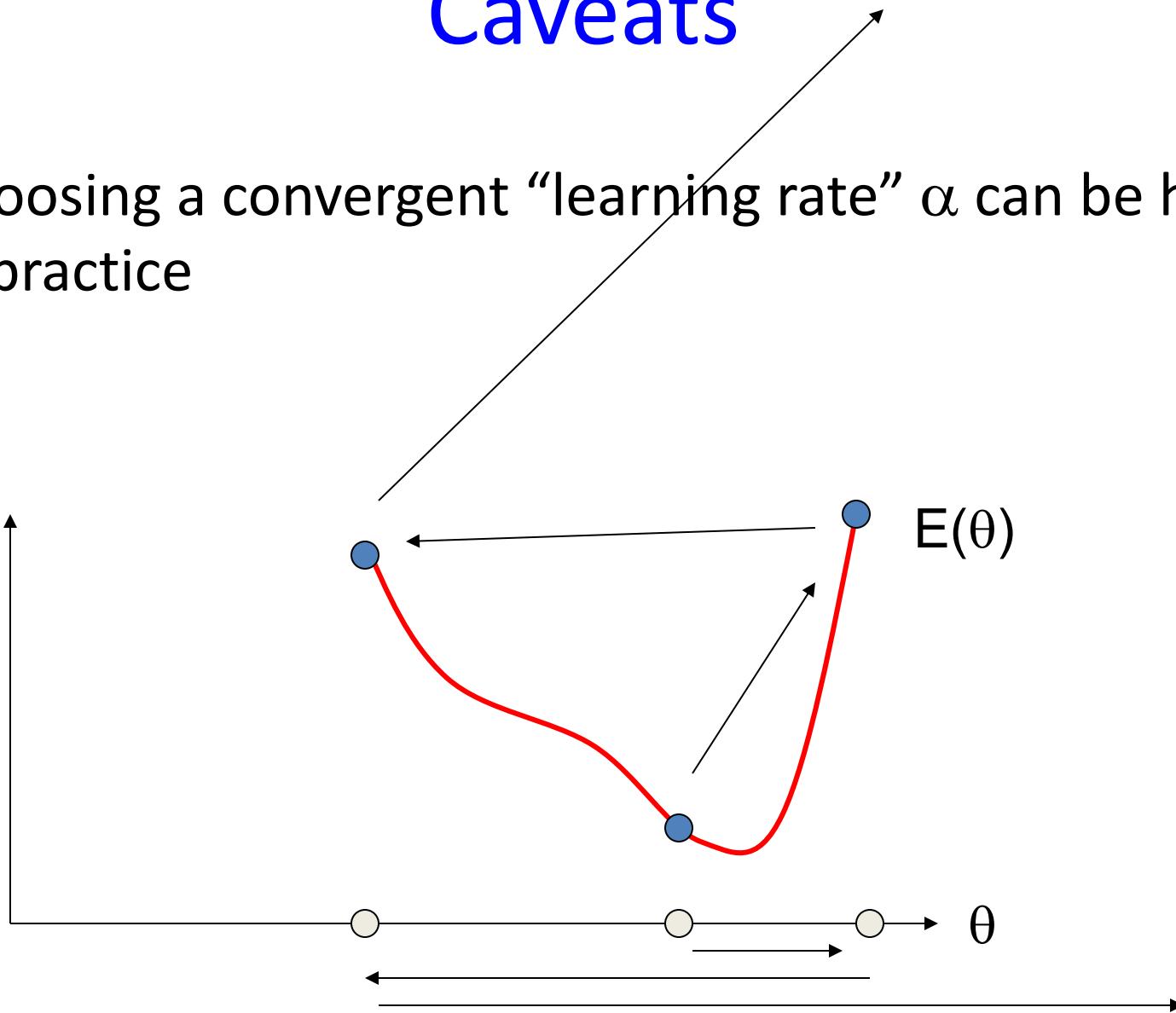
Stochastic Gradient Descent

- Objective function values (measured over all examples) over time settle into local minimum
- Step size must be reduced over time, e.g., $O(1/t)$



Caveats

- Choosing a convergent “learning rate” α can be hard in practice



Neural networks

- Neural networks are *universal function approximators*
 - Given any function, and a complicated enough network, they can accurately model that function



- How to choose the size and structure of networks?
 - If network is too large, risk of over-fitting (data caching)
 - If network is too small, representation may not be rich enough

Pros and cons of different classifiers

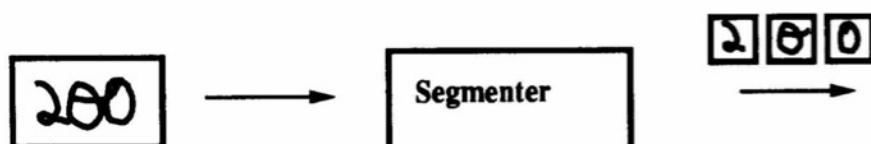
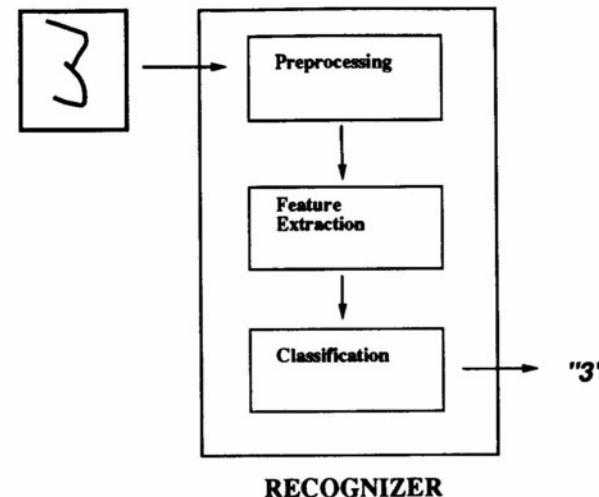
- Nearest neighbors
 - Can model any data, very prone to overfitting, requires distance function, fast learning, slow classification
- Neural networks
 - Models any function, requires structure, can suffer from local minima, slow learning, fast classification, difficult to interpret.
- Bayes nets
 - Requires setting network structure, fast learning, fast classification, intuitive interpretation of parameters.
- Decision trees
 - Limited modeling power, mostly automatic, moderate learning speed, fast classification, intuitive interpretation of parameters.
- Perceptrons
 - Very limited modeling power, fast training, fast classification, intuitive interpretation of parameters.

Neural Nets: 1960s-1990s

- Failure to deliver perceptron promises during 1960s-1970s led to “AI winter”
- In 1980s, multi-layer networks and the backpropagation algorithm led to new excitement, new era of neural network research

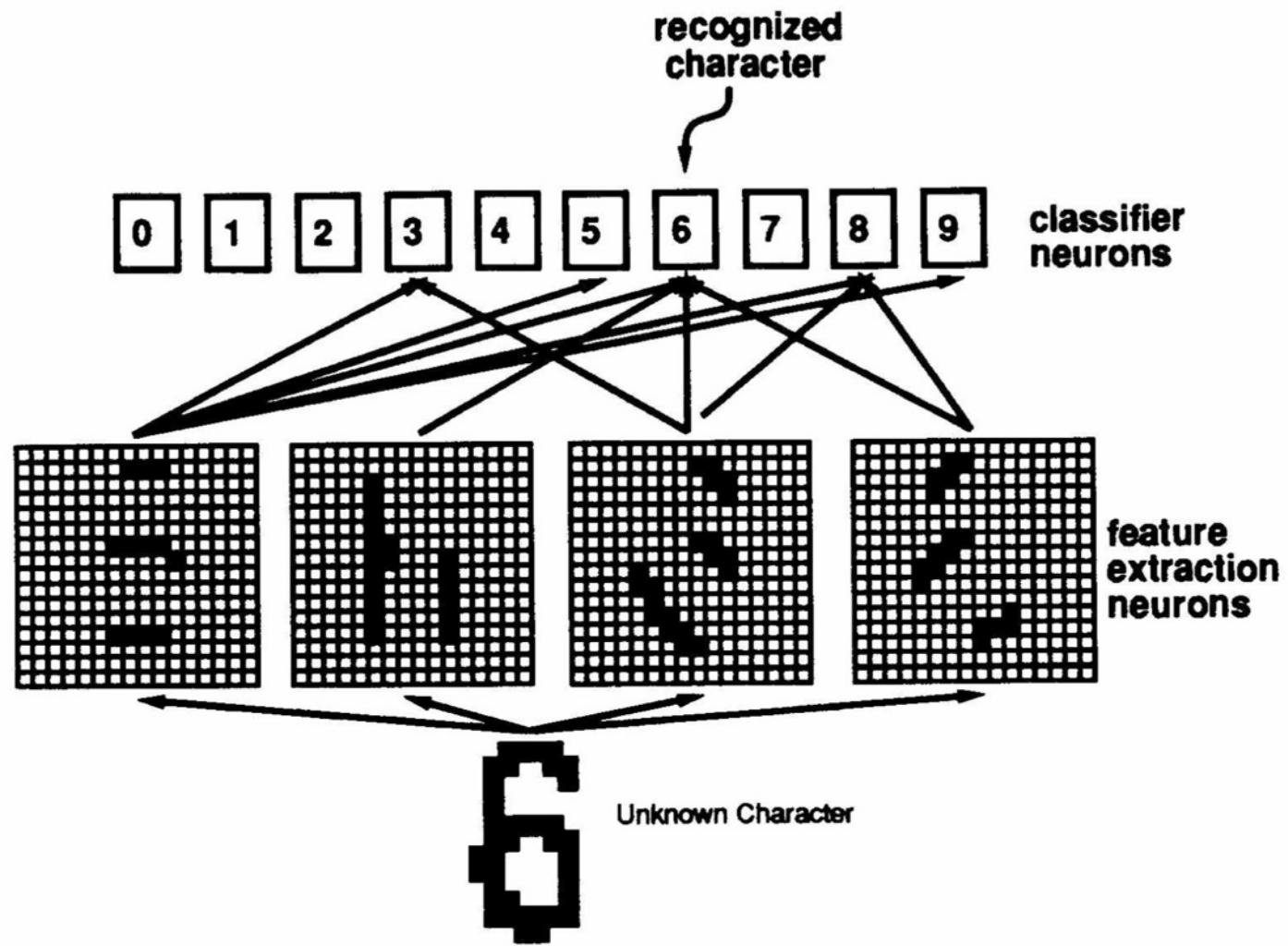
Success story: handwritten digit recognition (LeCun, 1989)

40004 75216
14199-2087 23505
96203 14310
44151 05753

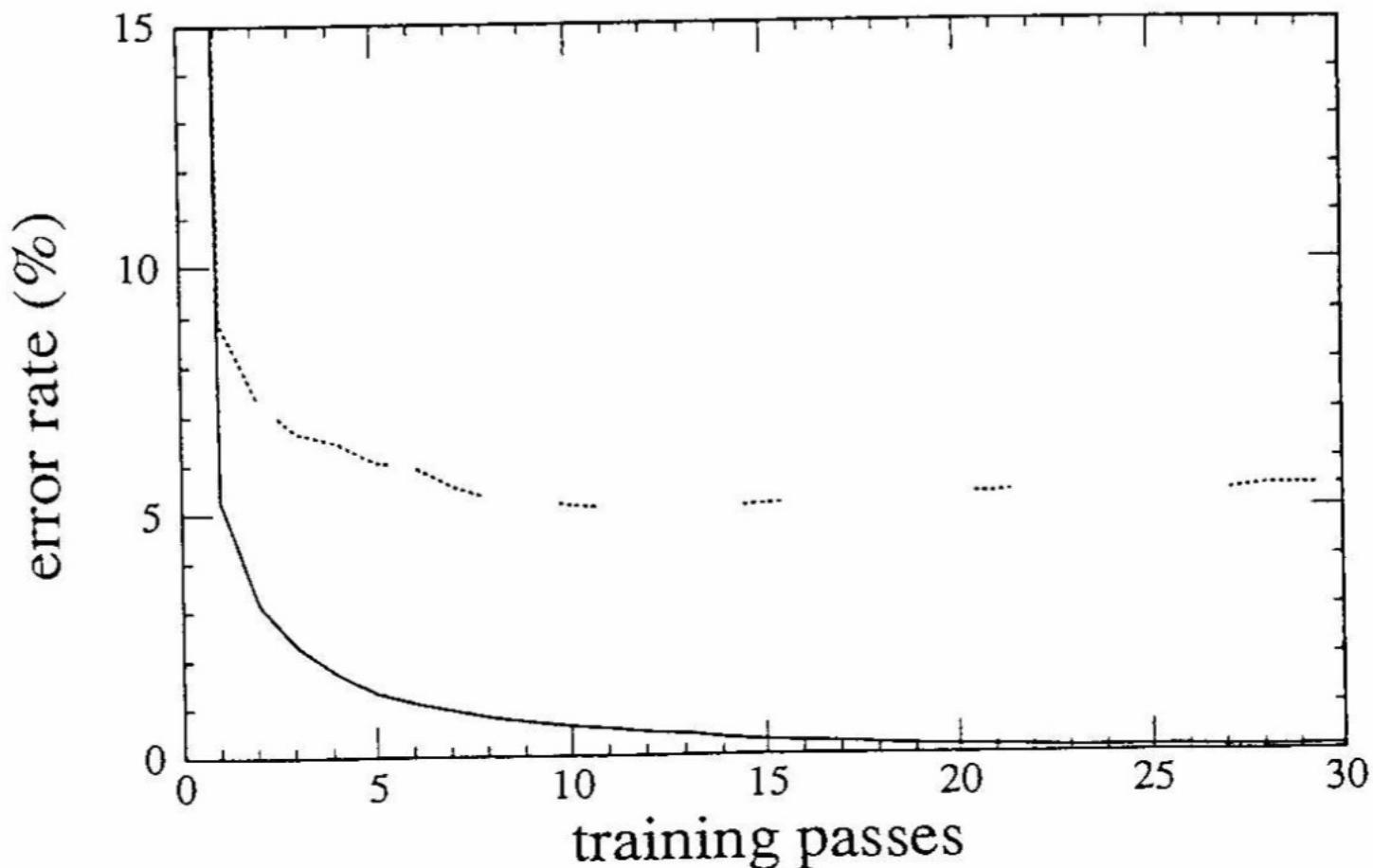


1 4 1 6 1 1 9 1 3 4 8 5 7 2 6 8 0 3 2 2
8 6 6 3 5 9 7 2 0 2 9 9 2 9 9 7 2 2 5 1
0 1 3 0 8 4 1 1 1 5 9 1 0 1 0 6 1 5 4 0
3 1 1 0 6 4 1 1 1 0 3 0 4 7 5 2 6 2 0 0
6 6 8 9 1 2 0 7 6 7 0 8 5 5 7 1 3 1 4 2
6 0 2 0 1 7 2 3 0 1 8 7 1 1 2 9 9 1 0 8
8 4 0 1 0 9 7 0 7 5 9 7 3 3 1 9 7 2 0 1
5 5 1 0 7 5 5 1 8 2 5 5 1 8 2 8 1 4 3 5
4 3 1 7 8 7 5 4 1 6 5 5 4 6 0 3 5 4 6 0
5 5 1 8 2 5 5 1 0 8 5 0 3 0 4 7 5 2 0 4

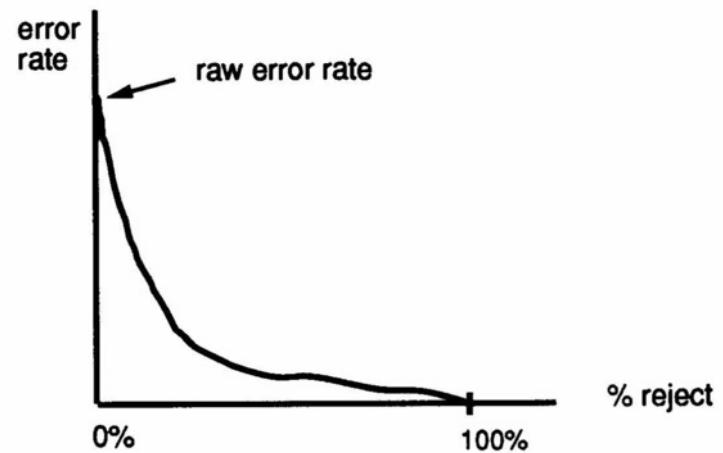
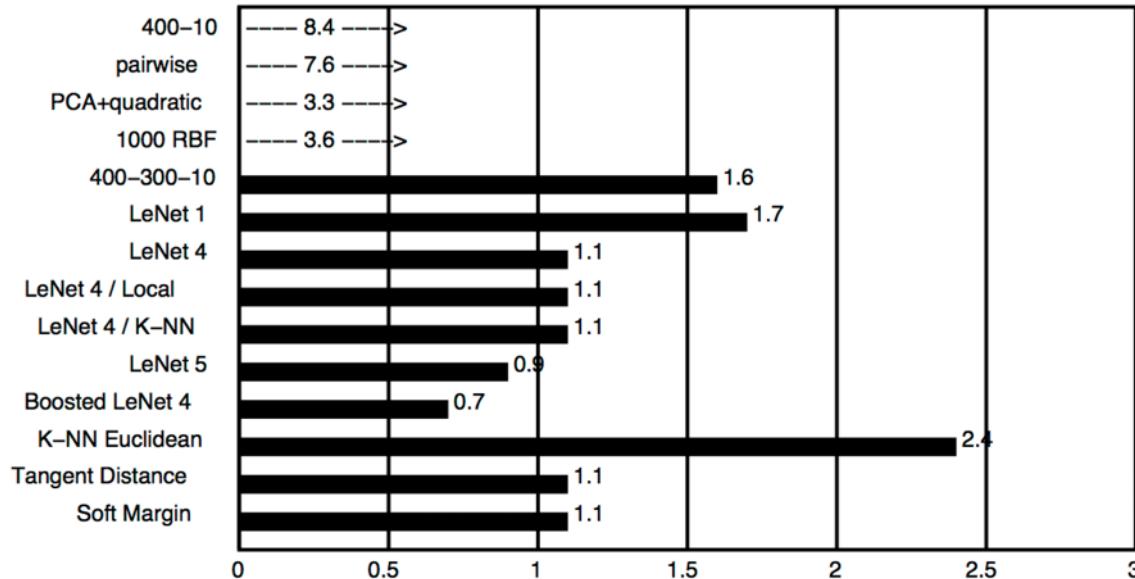
Network structure



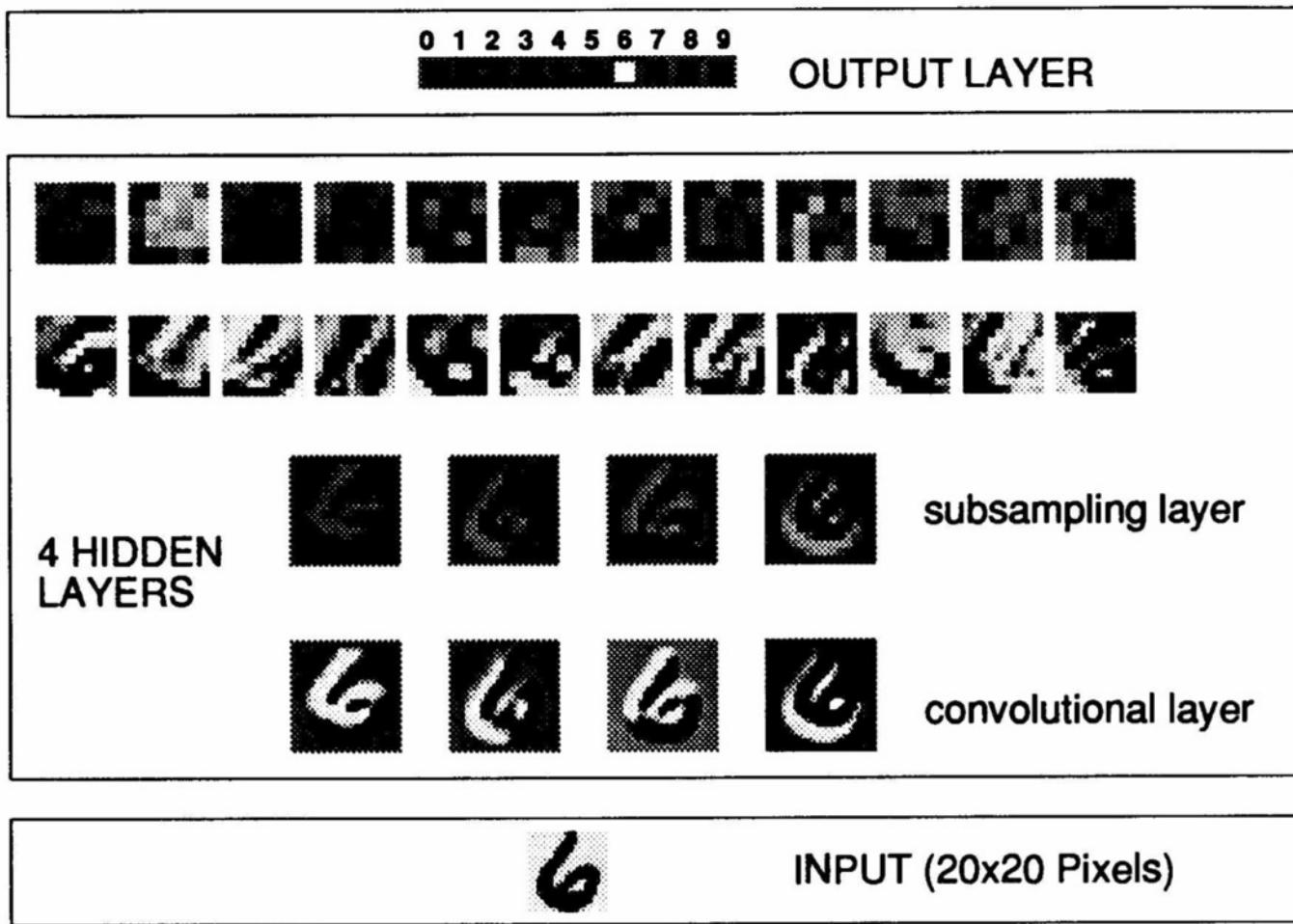
Use backprop to train



Worked better than other techniques (LeCun 1989)

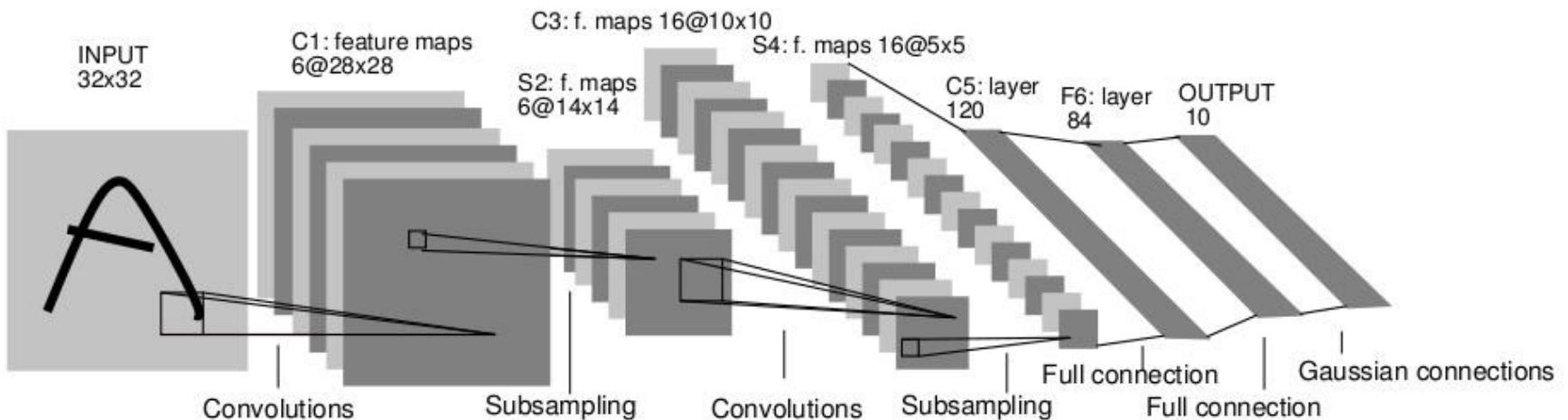
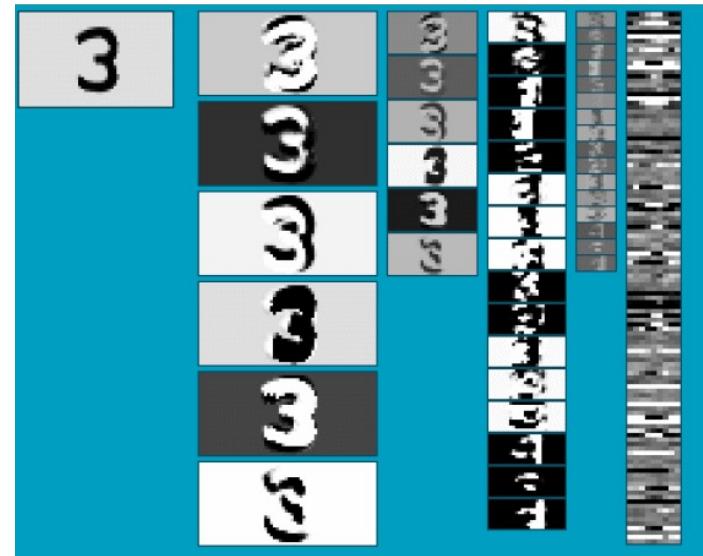


More complex architectures...

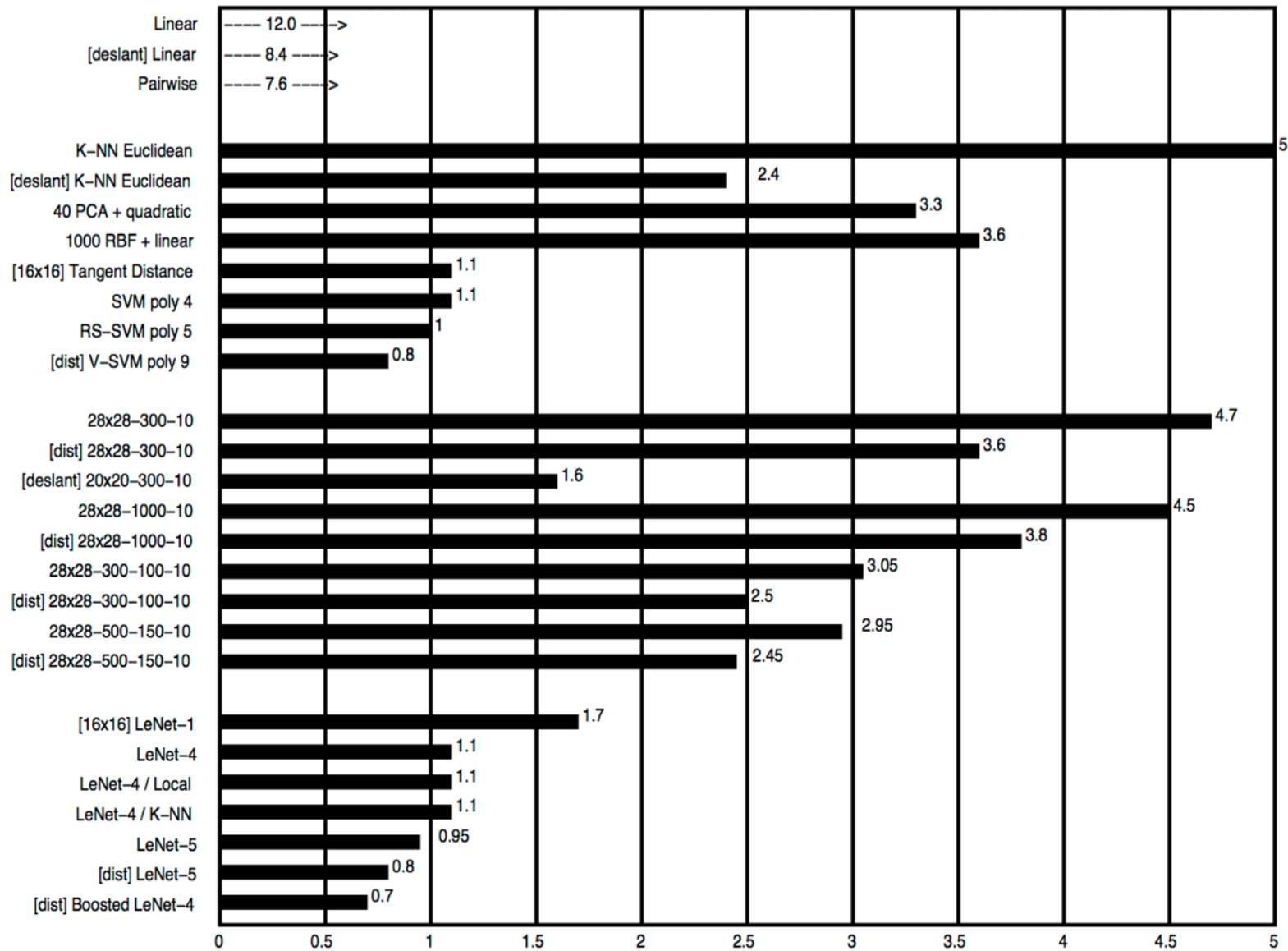


Convolutional Neural Networks

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end



But other techniques catching up (LeCun 1998)



Late 1990s-2010: Another decline

- Neural networks failed to work equally well on more complicated problems
 - E.g. recognition in real images, real audio streams, etc.
- Mix of practical and theoretical problems
 - How to decide network structure and many learning parameters (e.g. step sizes)?
 - Required too much computation
 - Required too much data
 - Very difficult to “debug” failures

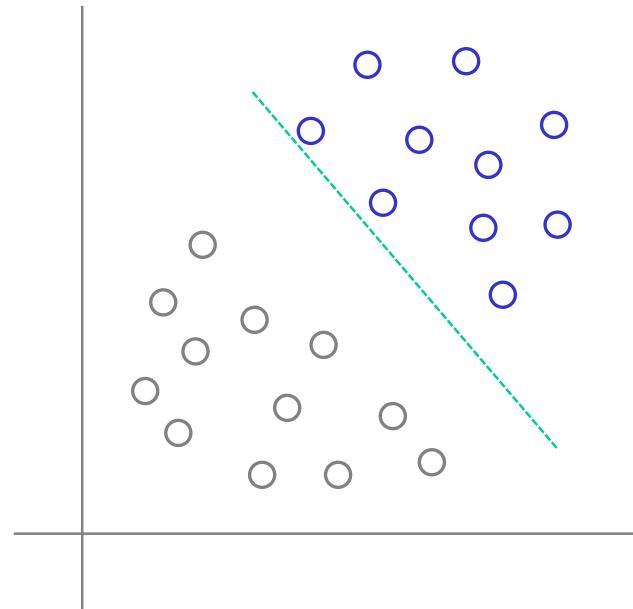
2000's: Return to the simple

- Return to simpler techniques, like linear classifiers
 - But in high dimensions
 - Simpler learning algorithms, easier to justify theoretically
- Learn classifiers on manually-created features
 - E.g. not images themselves, but statistical features like color histograms, edge distributions, etc.

Support Vector Machines (SVM)

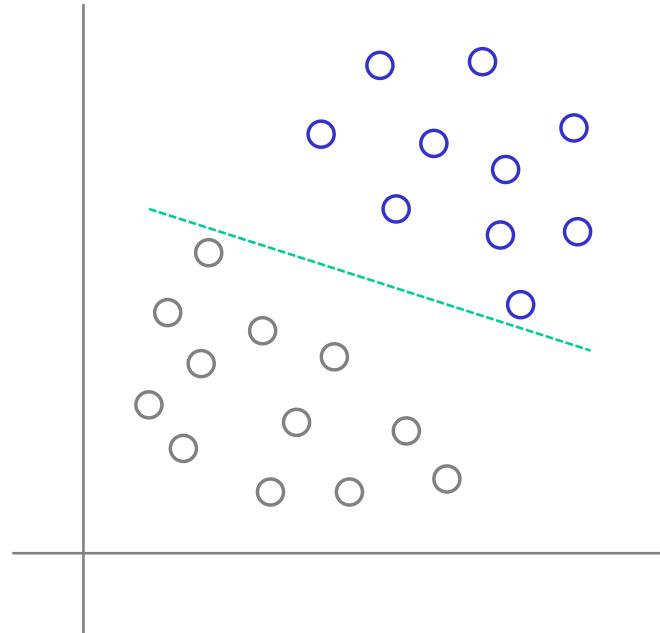
Reminder about perceptrons

- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points



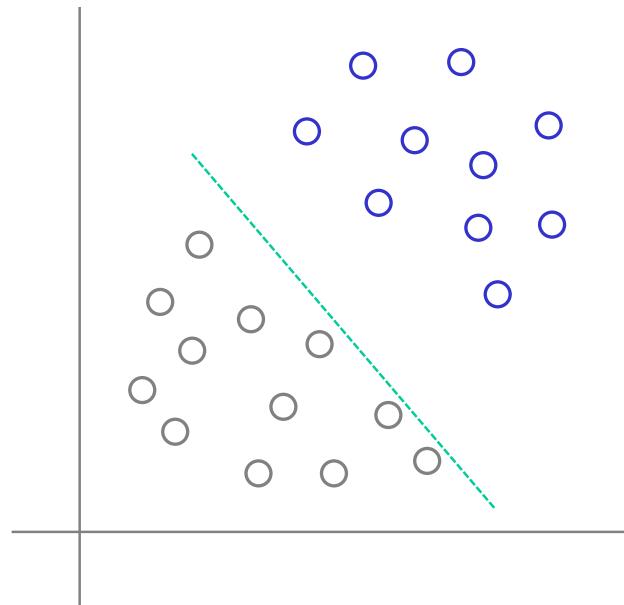
Reminder about perceptrons

- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is not unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points
- If done differently



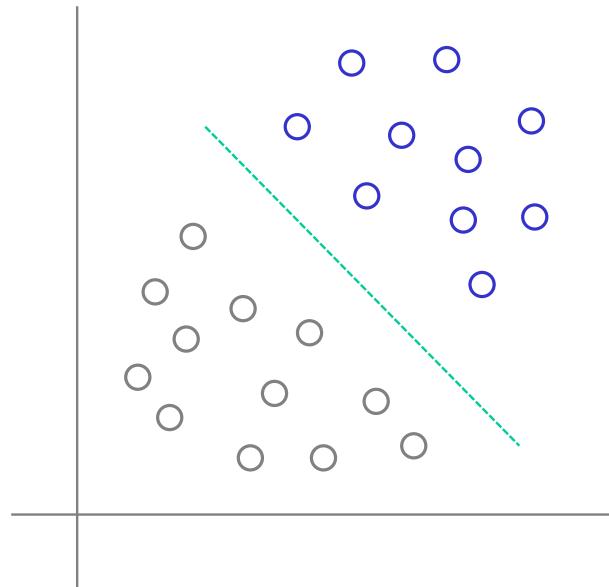
Reminder about perceptrons

- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points
- **If done differently....Again**



Reminder about perceptrons

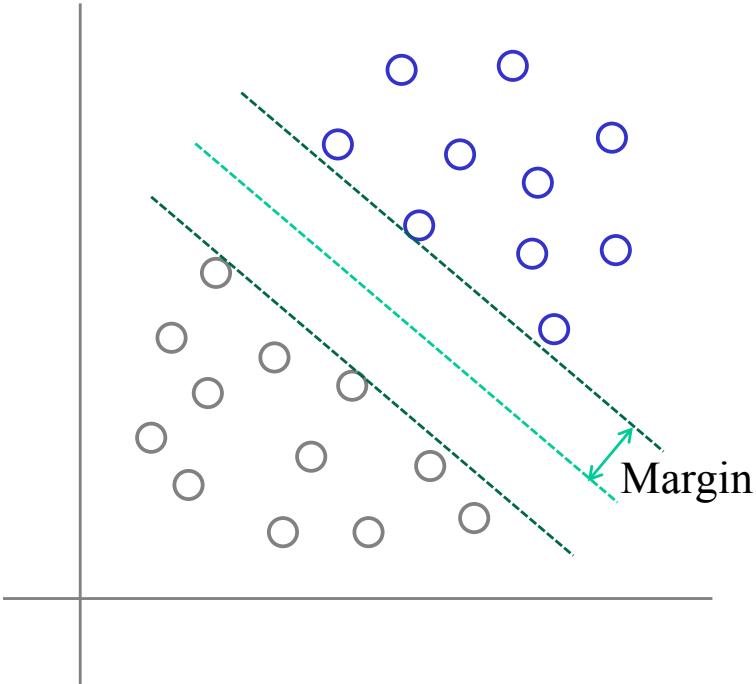
- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is not unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points
- If done differently....Again...And Again



Motivation

- For a linearly separable classification task, there are generally infinitely many separating hyperplanes
 - Perceptron learning, however, stops as soon as one of them is reached
 - Some hyperplanes may be better than others
- To improve generalization, we want to place a decision boundary as far away from training classes as possible.
 - In other words, place the boundary at equal distances from class boundaries

Optimal Hyperplane



- Given a training sample, the support vector machine constructs a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized.

Next class

- More about SVM and neural networks

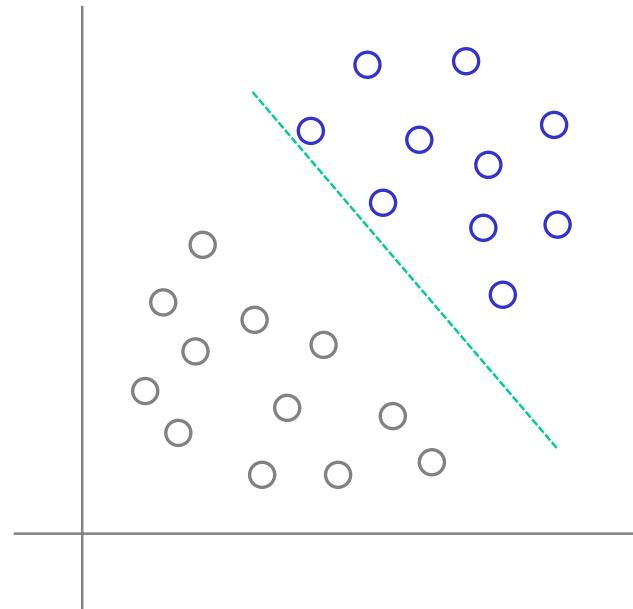
Support vector machines (SVMs)

Announcements

- A2 grades will be uploaded later today or tomorrow
- A3 due on December 2nd
- Optional A4 will be released this week (it will be due during finals week)
- Final exam is on December 16

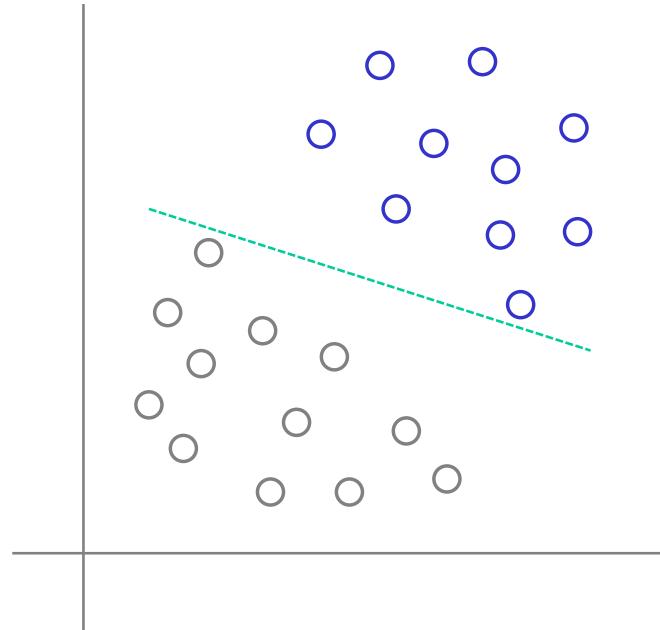
Reminder about perceptrons

- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points



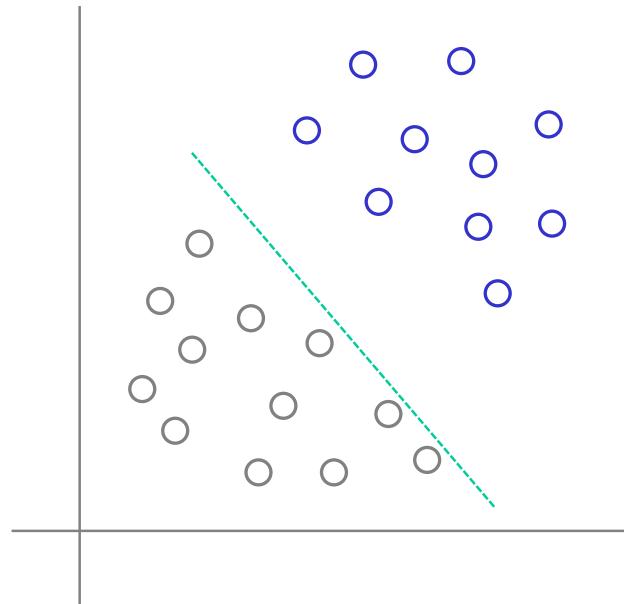
Reminder about perceptrons

- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is not unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points
- If done differently



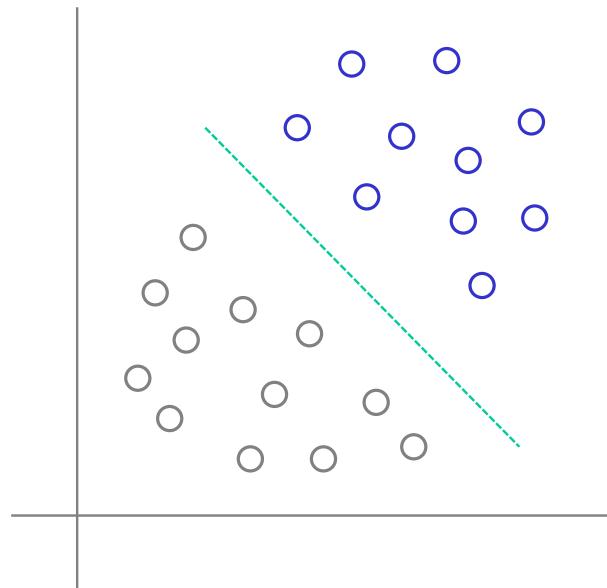
Reminder about perceptrons

- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points
- **If done differently....Again**



Reminder about perceptrons

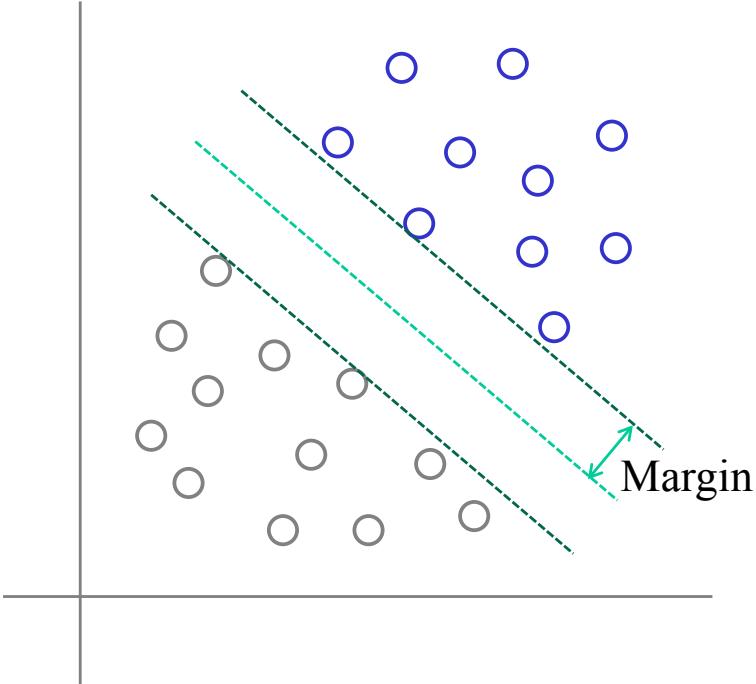
- **Perceptron Convergence Theorem:** If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is not unique. There are infinite possible solutions and decision boundaries.
 - Perceptrons find any separating hyperplane
 - The hyperplane depends on initialization and ordering of training points
- If done differently....Again...And Again



Motivation

- For a linearly separable classification task, there are generally infinitely many separating hyperplanes
 - Perceptron learning, however, stops as soon as one of them is reached
 - Some hyperplanes may be better than others
- To improve generalization, we want to place a decision boundary as far away from training classes as possible.
 - In other words, place the boundary at equal distances from class boundaries

Optimal Hyperplane



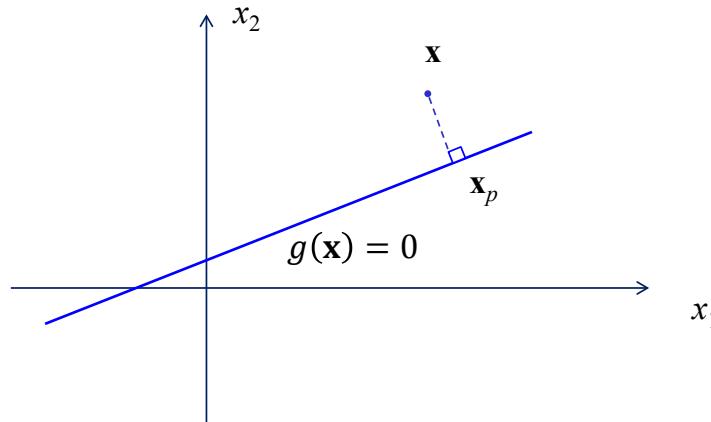
- Given a training sample, the support vector machine constructs a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized.

Decision Boundary

- Given a linear discriminant function

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$$

- To find its distance to a given pattern \mathbf{x} , project \mathbf{x} onto the decision boundary



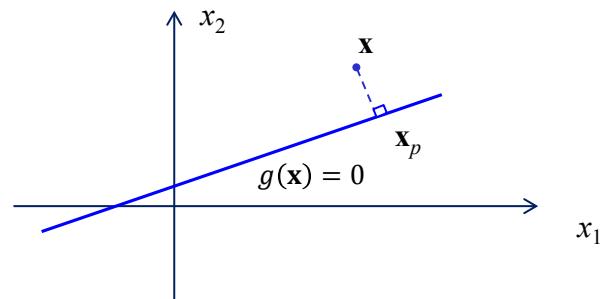
Decision Boundary (cont.)

- \mathbf{x} can be re-written as a function of the projection and the weights

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{||\mathbf{w}||}$$

- \mathbf{x}_p is \mathbf{x} 's projection
- The second term arises from the fact that the weight vector is perpendicular to the decision boundary
- The algebraic distance r is positive if \mathbf{x} is on the positive side of the boundary and negative if \mathbf{x} is on the negative side

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$$

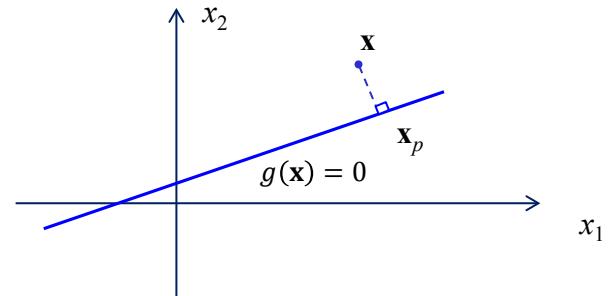


Decision Boundary (cont.)

- Since \mathbf{x} can be written in terms of its projection and r , then the decision boundary can as well:

$$\begin{aligned}g(\mathbf{x}) &= g(\mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}) \\&= \mathbf{w}^T (\mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}) + b \\&= \mathbf{w}^T \mathbf{x}_p + b + r \|\mathbf{w}\| \\&= r \|\mathbf{w}\|\end{aligned}$$

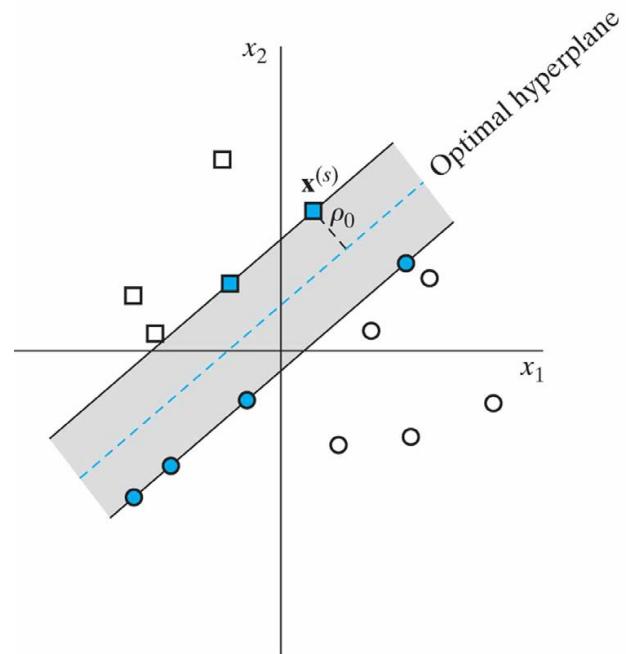
$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$$



- Thus, $r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$

Margin of Separation

- The ***margin of separation*** is the smallest distance of the hyperplane to a data set
- The training patterns (i.e. inputs) closest to the optimal hyperplane are called ***support vectors***
 - We'll show that non-support vectors have no role



Finding the Optimal Hyperplane for Linearly Separable Problems

- **Question:** Given N pairs of inputs and desired outputs $\langle \mathbf{x}_i, d_i \rangle$, How to find \mathbf{w}_o and b_o for the optimal hyperplane?
- Without loss of generality, \mathbf{w}_o and b_o must satisfy

$$\mathbf{w}_o^T \mathbf{x}_i + b_o \geq 1 \quad \text{for } d_i = 1$$

$$\mathbf{w}_o^T \mathbf{x}_i + b_o \leq -1 \quad \text{for } d_i = -1$$

or

$$d_i(\mathbf{w}_o^T \mathbf{x}_i + b_o) \geq 1$$

where the equality holds for support vectors only

Optimal Hyperplane

- For a support vector $\mathbf{x}^{(s)}$, its algebraic distance to the optimal hyperplane is:

$$r = \frac{g(\mathbf{x}^{(s)})}{\|\mathbf{w}_o\|} = \begin{cases} \frac{1}{\|\mathbf{w}_o\|} & \text{if } d^{(s)} = 1 \\ \frac{-1}{\|\mathbf{w}_o\|} & \text{if } d^{(s)} = -1 \end{cases}$$
$$\mathbf{w}_o^T \mathbf{x}_i + b_o \geq 1 \quad \text{for } d_i = 1$$
$$\mathbf{w}_o^T \mathbf{x}_i + b_o \leq -1 \quad \text{for } d_i = -1$$

- Thus, the margin of separation (from optimal hyperplane to support vector) is:

$$|r| = \frac{1}{\|\mathbf{w}_o\|}$$

In other words, maximizing the margin of separation is equivalent to minimizing $\|\mathbf{w}_o\|$

- The optimal margin of separation between support vectors from the two classes is:

$$\rho = 2|r| = \frac{2}{\|\mathbf{w}_o\|}$$

Primal Problem

- Therefore \mathbf{w}_o and b_o satisfy

$$d_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad \text{for } i = 1, \dots, N$$

and $\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ is minimized

- The above constrained minimization problem is called the **primal** problem

Lagrangian Formulation

- Using Lagrangian formulation, we construct the Lagrangian function:

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [d_i(\mathbf{w}^T \mathbf{x}_i + b) - 1] \quad \text{where } \alpha_i \geq 0$$

- where nonnegative variables, α_i 's are called Lagrange multipliers

Lagrangian Formulation

- The solution of the optimization problem is a saddle point, minimized with respect to \mathbf{w} and b , but maximized with respect to α

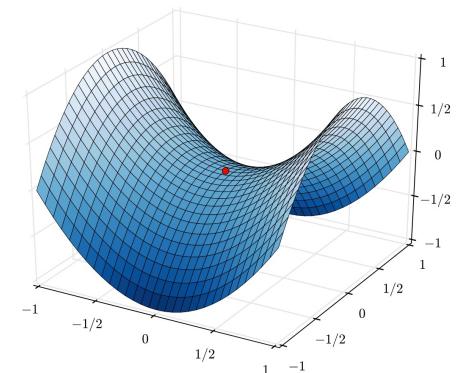
$$\arg \max_{\alpha} \left(\arg \min_{\mathbf{w}, b} J(\mathbf{w}, b, \alpha) \right)$$

- Lets start by solving: $\arg \min_{\mathbf{w}, b} J(\mathbf{w}, b, \alpha)$
 - Condition 1:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0$$

- Condition 2:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0$$



Saddle point (source: Wikipedia)

Lagrangian Formulation

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [d_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] \quad \text{where } \alpha_i \geq 0$$

- From condition 1:

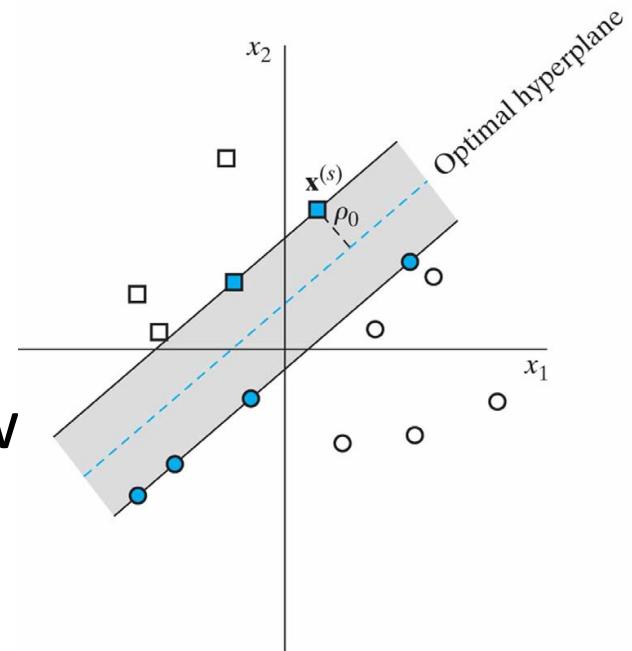
$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} - \sum_i \alpha_i d_i \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$$

- From condition 2:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0 \quad \Rightarrow \quad \sum_i \alpha_i d_i = 0$$

Finding Optimal Weights

- To solve: $\arg \min_{\mathbf{w}, b} J(\mathbf{w}, b, \alpha)$
 - We found that: $\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$
 - Under constraints $\alpha_i \geq 0$ and $\sum_i \alpha_i d_i$
- Still need to solve for α_i 's (will show later)



How to Find α ? Dual Problem

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [d_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

- The cost function can be first be expanded
- It can then be simplified using the solution for \mathbf{w}

$$\begin{aligned} \mathbf{w} &= \sum_i \alpha_i d_i \mathbf{x}_i & \Rightarrow & \mathbf{w}^T \mathbf{w} = \sum_i \alpha_i d_i \mathbf{w}^T \mathbf{x}_i \\ \Rightarrow J(\mathbf{w}, b, \alpha) &= -\frac{1}{2} \sum_{i=1}^N \alpha_i d_i \mathbf{w}^T \mathbf{x}_i - b \sum_{i=1}^N \alpha_i d_i + \sum_{i=1}^N \alpha_i \end{aligned}$$

How to Find α ? Dual Problem (Cont.)

$$J(\mathbf{w}, b, \alpha) = -\frac{1}{2} \sum_{i=1}^N \alpha_i d_i \mathbf{w}^T \mathbf{x}_i - b \sum_{i=1}^N \alpha_i d_i + \sum_{i=1}^N \alpha_i$$

- After, plugging in for \mathbf{w} and removing the 2nd term from our constraint $\sum_i \alpha_i d_i = 0$
$$\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$$
- Hence, $J(\mathbf{w}, b, \alpha)$ becomes

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$$

Dual Problem

- The solution to the Lagrangian function, is a saddle point, minimized w.r.t. \mathbf{w} and b , but maximized w.r.t. α
- The dual problem is stated as follows:
 - The Lagrange multipliers maximize

$$\arg \max_{\alpha} \left(\arg \min_{\mathbf{w}, b} J(\mathbf{w}, b, \alpha) \right)$$

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$$

– subject to (the prior constraints):

$$(1) \sum_i \alpha_i d_i = 0$$

$$(2) \alpha_i \geq 0$$

The dual problem can be solved as a quadratic optimization problem (Q is quadratic in terms of α)

Karush-Kuhn-Tucker Conditions

- **Remark:** The above constrained optimization problem satisfies:

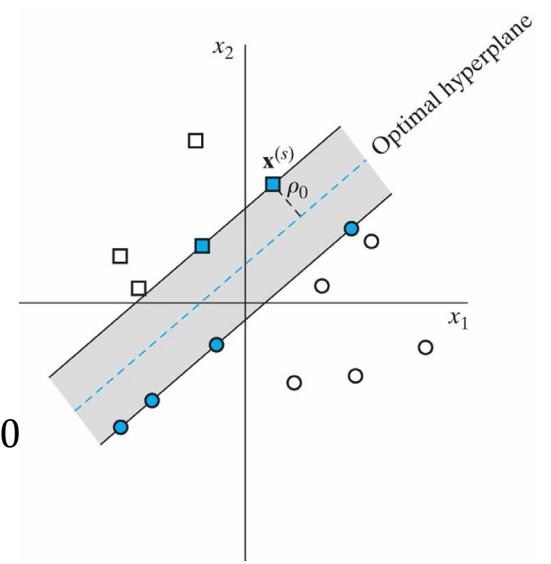
$$\alpha_i[d_i(\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0$$

called the Karush-Kuhn-Tucker conditions

- In other words, $\alpha_i = 0$ when $d_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 > 0$
- α_i can be greater than 0 only when $d_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$, that is, for support vectors

$$\begin{aligned}\mathbf{w}_o^T \mathbf{x}_i + b_o &\geq 1 && \text{for } d_i = 1 \\ \mathbf{w}_o^T \mathbf{x}_i + b_o &\leq -1 && \text{for } d_i = -1\end{aligned}$$

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [d_i(\mathbf{w}^T \mathbf{x}_i + b) - 1]$$



Solution

$$\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$$

- Having found optimal multipliers, $\alpha_{o,i}$

$$\mathbf{w}_o = \sum_{i=1}^{N_s} \alpha_{o,i} d_i \mathbf{x}_i$$

- where N_s is the number of support vectors

Finding b ?

- For any support vector $\mathbf{x}^{(s)}$, we have

$$d^{(s)} \left(\mathbf{w}_o^T \mathbf{x}^{(s)} + b_o \right) = 1$$

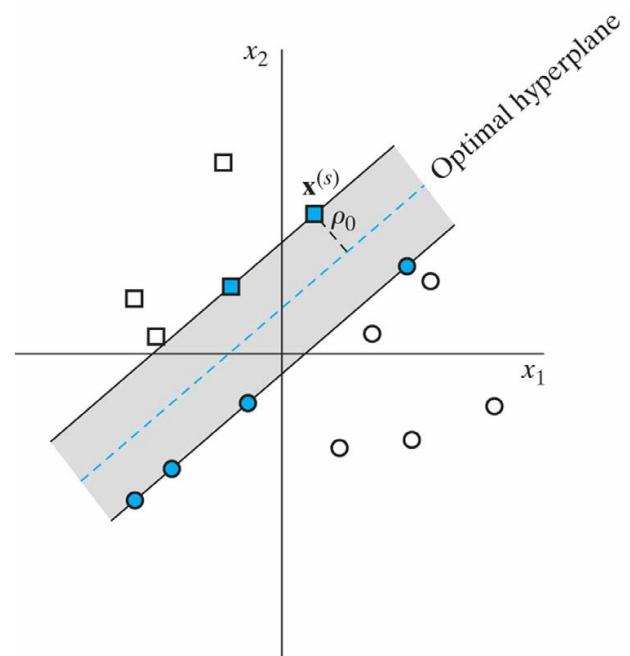
- Hence,

$$b_o = \frac{1}{d^{(s)}} - \mathbf{w}_o^T \mathbf{x}^{(s)} = \frac{1}{d^{(s)}} - \sum_{i=1}^{N_s} \alpha_{o,i} d_i \mathbf{x}_i^T \mathbf{x}^{(s)}$$

- Note that for robustness, one should average over all support vectors to compute b_o

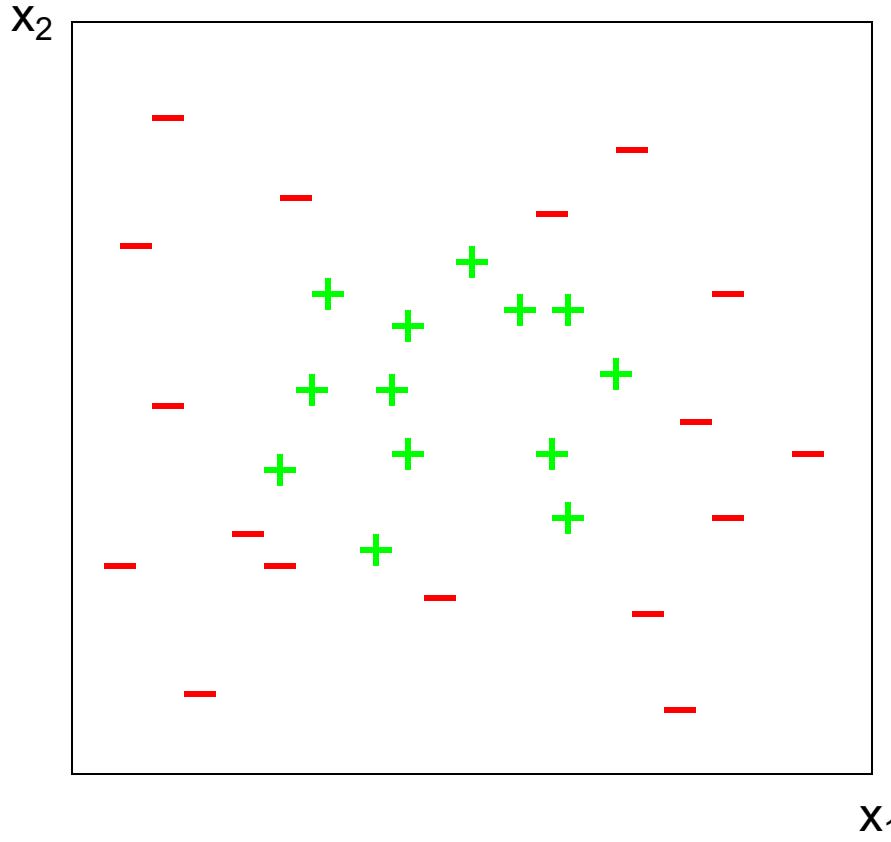
Interim summary

- Support vectors are the training patterns (i.e. inputs) closest to the optimal hyperplane
- To improve generalization, we want to place a decision boundary as far away from training classes as possible.
- Finding the maximum margin hyperplane has been formulated as a constrained quadratic problem
 - Convex problem, well studied, conceptually easy to solve
 - It can be solved in the primal or dual formulation
 - Only some data points contribute to the solution (i.e. support vectors)
 - So far, only applies to linearly separable data



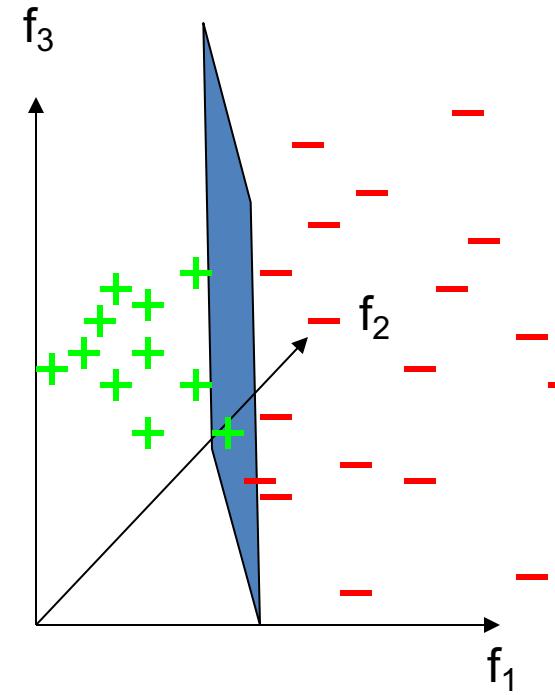
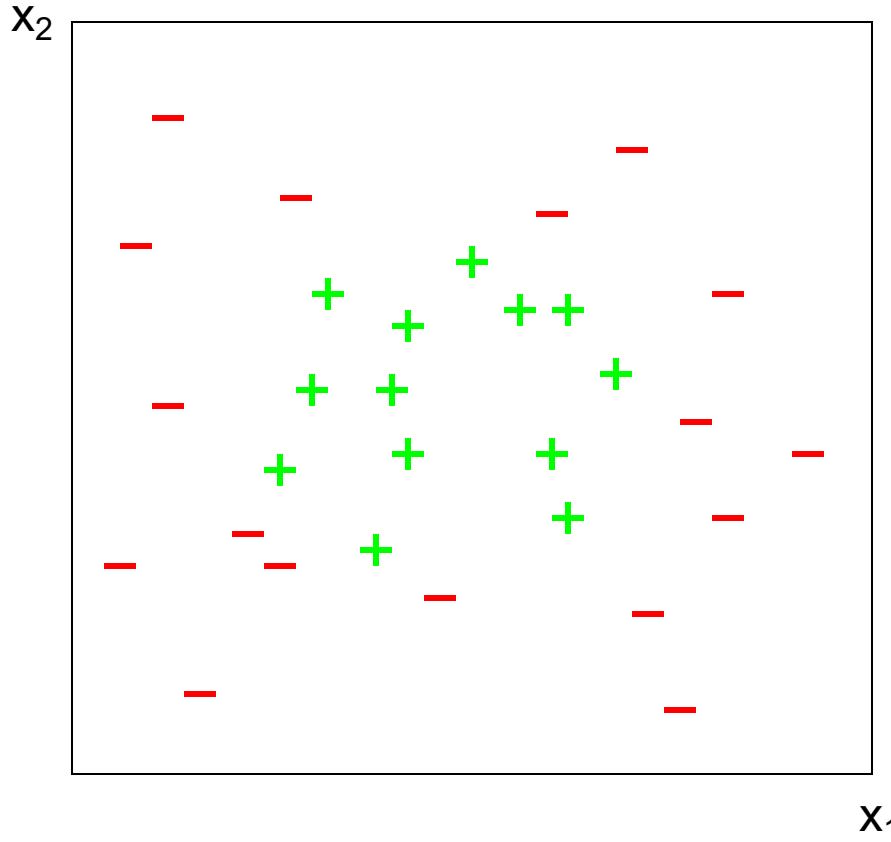
What if space is not linearly separable?

- Higher dimensions! And transformations! And robustness!



What if space is not linearly separable?

- Higher dimensions! And transformations! And robustness!
- Choose $f_1 = x_1^2$, $f_2 = x_2^2$, $f_3 = \sqrt{2} x_1 x_2$



The kernel trick

- **Cover's Theorem:** A complex classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linear separable than in the low-dimensional input space
- SVM for non-linear classification
 - Nonlinear mapping of the input space into a high-dimensional feature space
 - Constructing the optimal hyperplane for the feature space

The kernel trick

- Replace $Q(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$ with

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j d_i d_j \underline{K(\mathbf{x}_i, \mathbf{x}_j)}$$

Kernel function

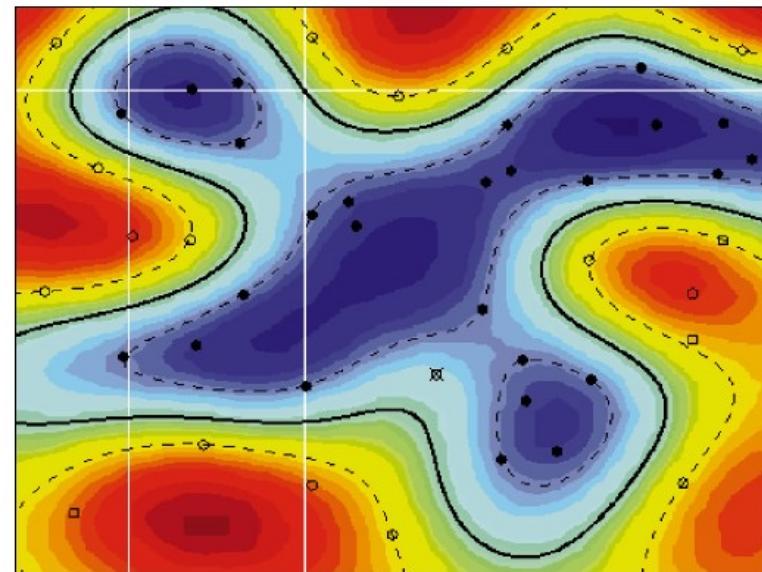
- Notice we can't necessarily precompute the decision boundary anymore; but since alphas are sparse, classification can still be efficient
- Kernels of the form $K(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i) \cdot f(\mathbf{x}_j)$, where $f()$ is a feature embedding function, work out particularly nicely

Kernel Functions

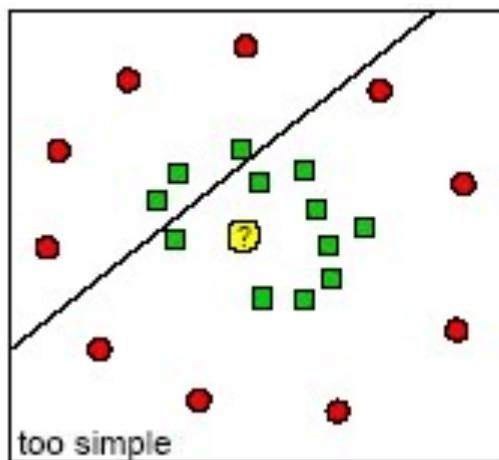
- Can implicitly compute a feature mapping to a high dimensional space, without having to construct the features! (all you need is the dot product)
- Example: $K(\mathbf{a}, \mathbf{b}) = (\mathbf{a} \bullet \mathbf{b})^2$
 - $(a_1 b_1 + a_2 b_2)^2$
 $= a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2$
 $= [a_1^2, a_2^2, \sqrt{2}a_1 a_2] \bullet [b_1^2, b_2^2, \sqrt{2}b_1 b_2]$

Types of Kernel

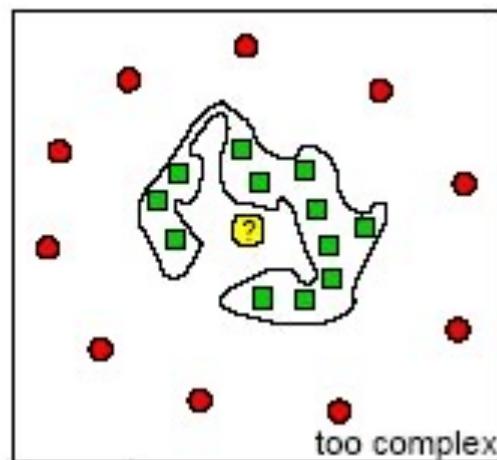
- Polynomial $K(a,b) = (a^T b + 1)^d$
- Gaussian $K(a,b) = \exp(-||a-b||^2/s^2)$
- Sigmoid, etc...
- Decision boundaries in feature space may be highly curved in original space!



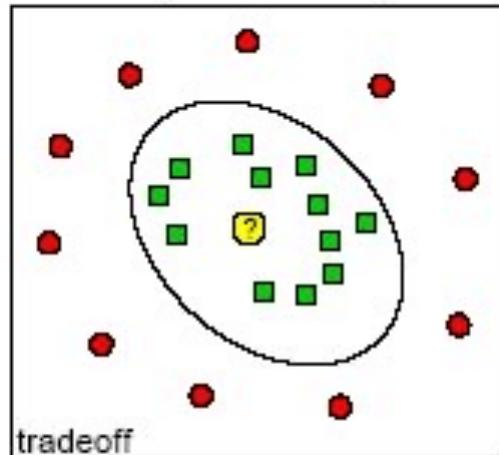
Overfitting / underfitting



too simple



too complex



tradeoff

- negative example
- positive example
- ? new patient

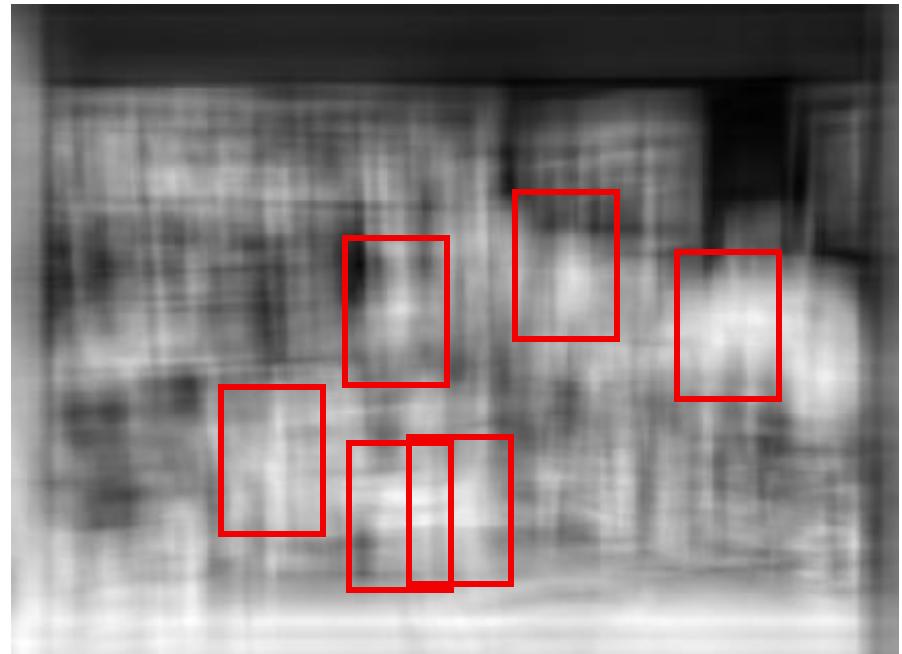
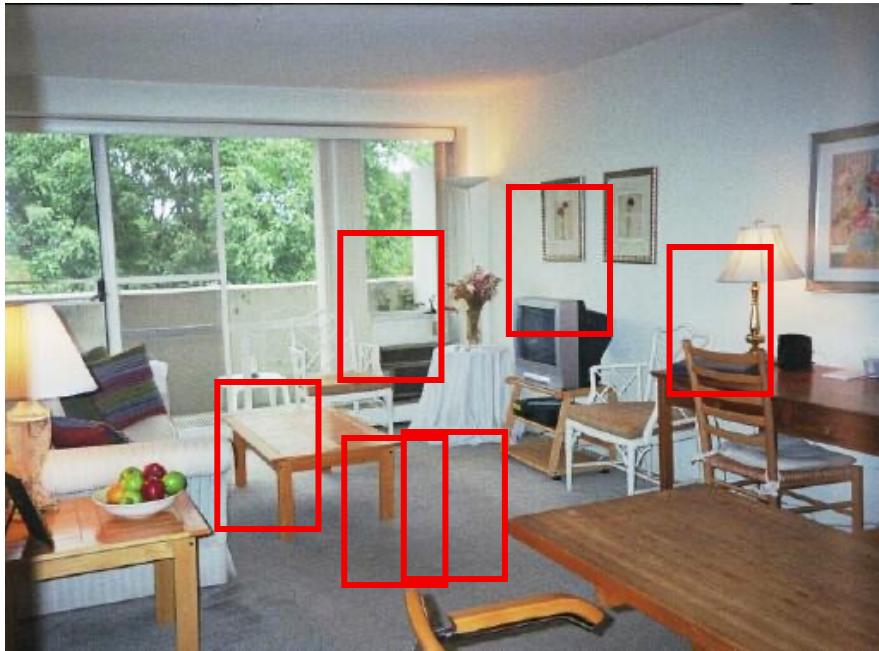
SVM practicalities

- SVMs often have very good performance
 - E.g., digit classification, face recognition, etc
- Still need parameter tweaking
 - Kernel type
 - Kernel parameters
 - Regularization weight
- Fast optimization for medium datasets ($\sim 100k$)
- Off-the-shelf libraries
 - E.g. SVM^{light}



Example: Object recognition

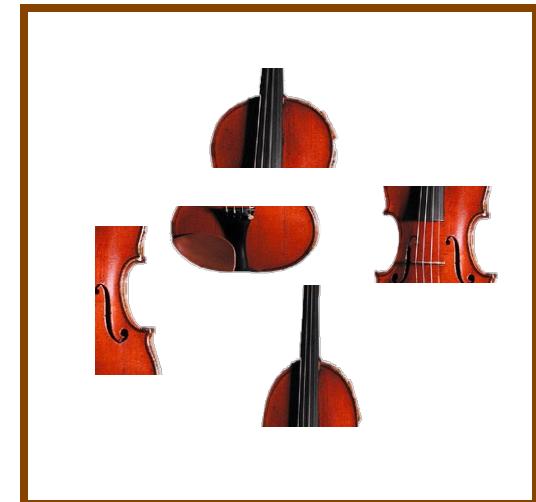
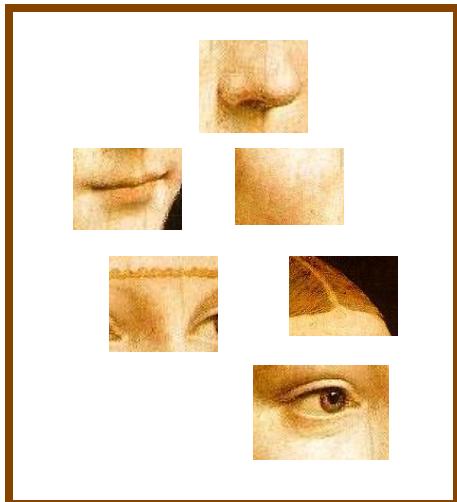
Find the chair in this image



Pretty much garbage
Simple template matching is not going to suffice

Bag of features: outline

1. Extract features



Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”

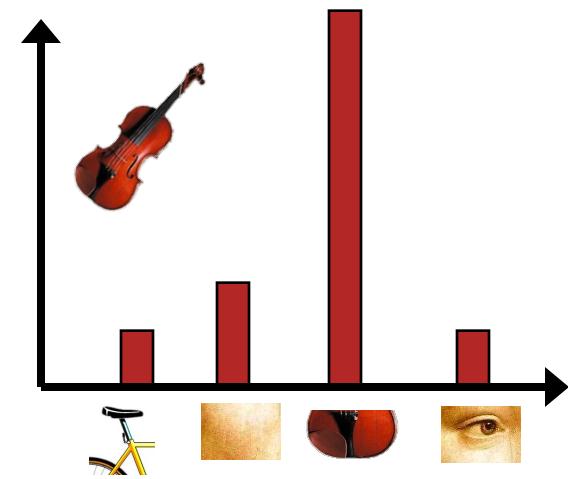
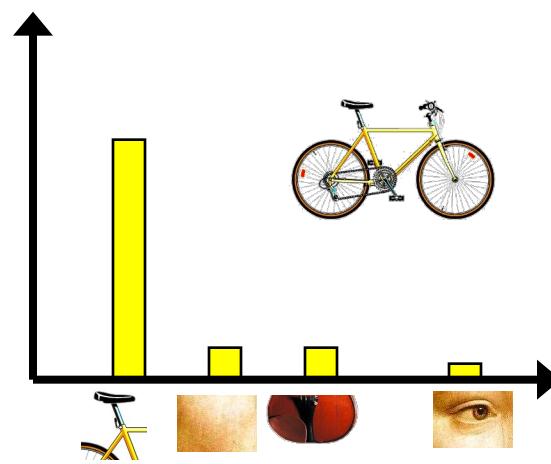
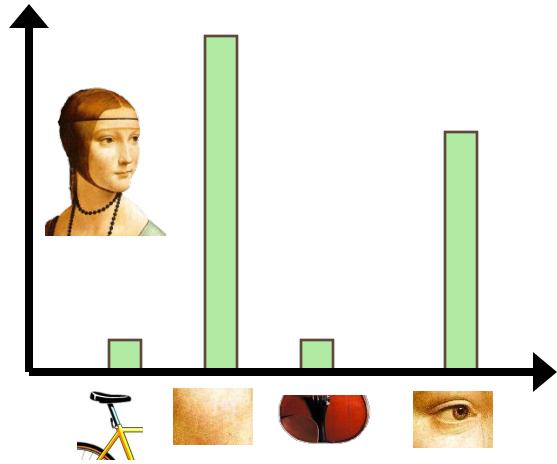


Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary

Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary
4. Represent images by frequencies of “visual words”



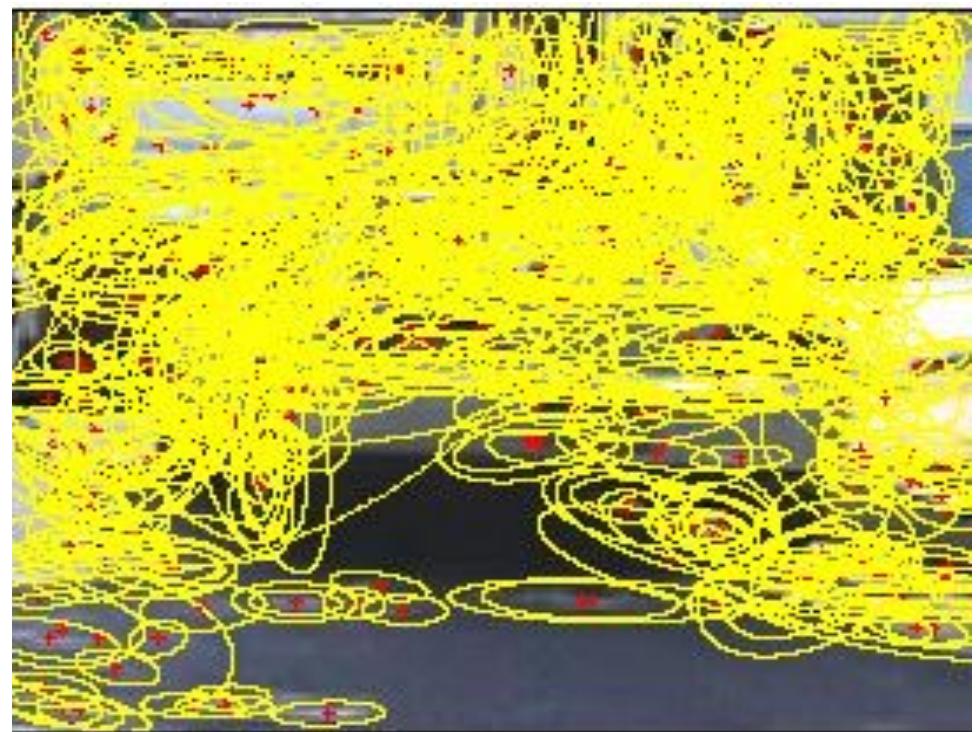
1. Feature extraction

- Regular grid
 - Vogel & Schiele, 2003
 - Fei-Fei & Perona, 2005



1. Feature extraction

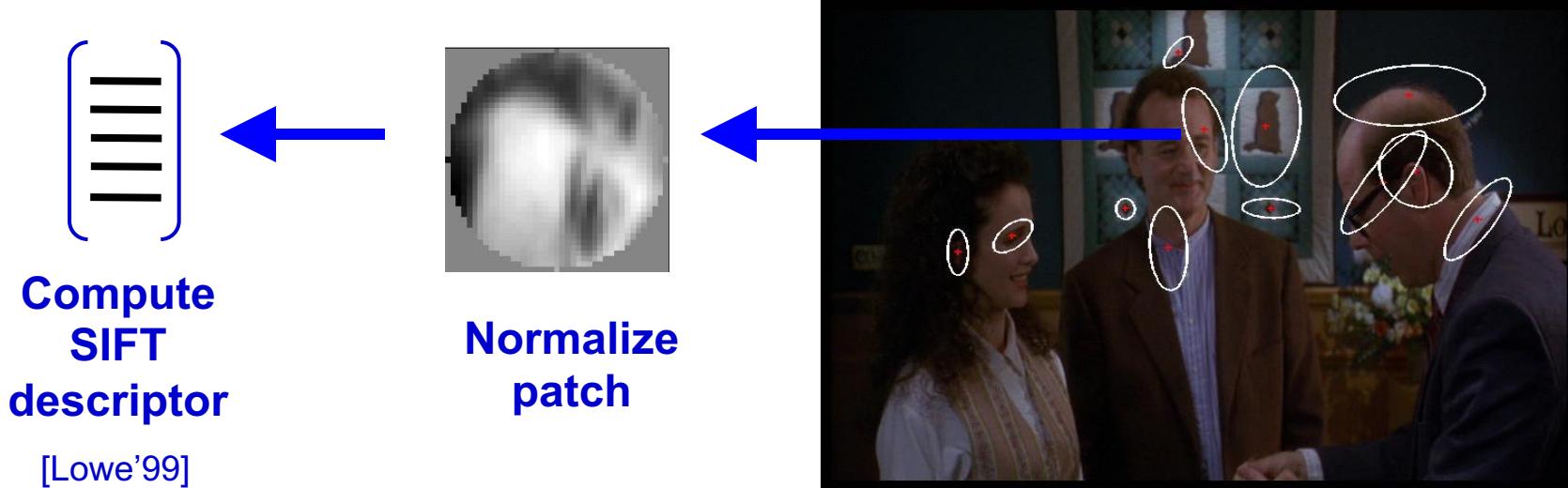
- Regular grid
 - Vogel & Schiele, 2003
 - Fei-Fei & Perona, 2005
- Interest point detector
 - Csurka et al. 2004
 - Fei-Fei & Perona, 2005
 - Sivic et al. 2005



1. Feature extraction

- Regular grid
 - Vogel & Schiele, 2003
 - Fei-Fei & Perona, 2005
- Interest point detector
 - Csurka et al. 2004
 - Fei-Fei & Perona, 2005
 - Sivic et al. 2005
- Other methods
 - Random sampling (Vidal-Naquet, 2002, Crandall 2006)
 - Segmentation-based patches (Barnard et al. 2003)

1. Feature extraction

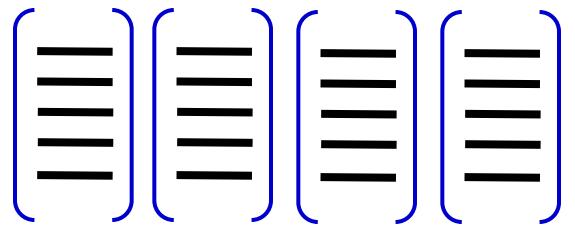


[Mikojaczyk and Schmid '02]

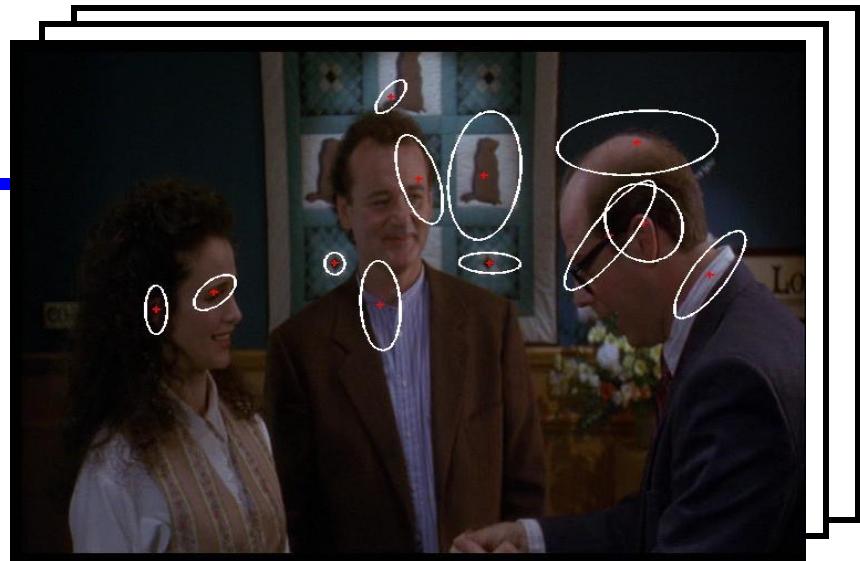
[Mata, Chum, Urban & Pajdla, '02]

[Sivic & Zisserman, '03]

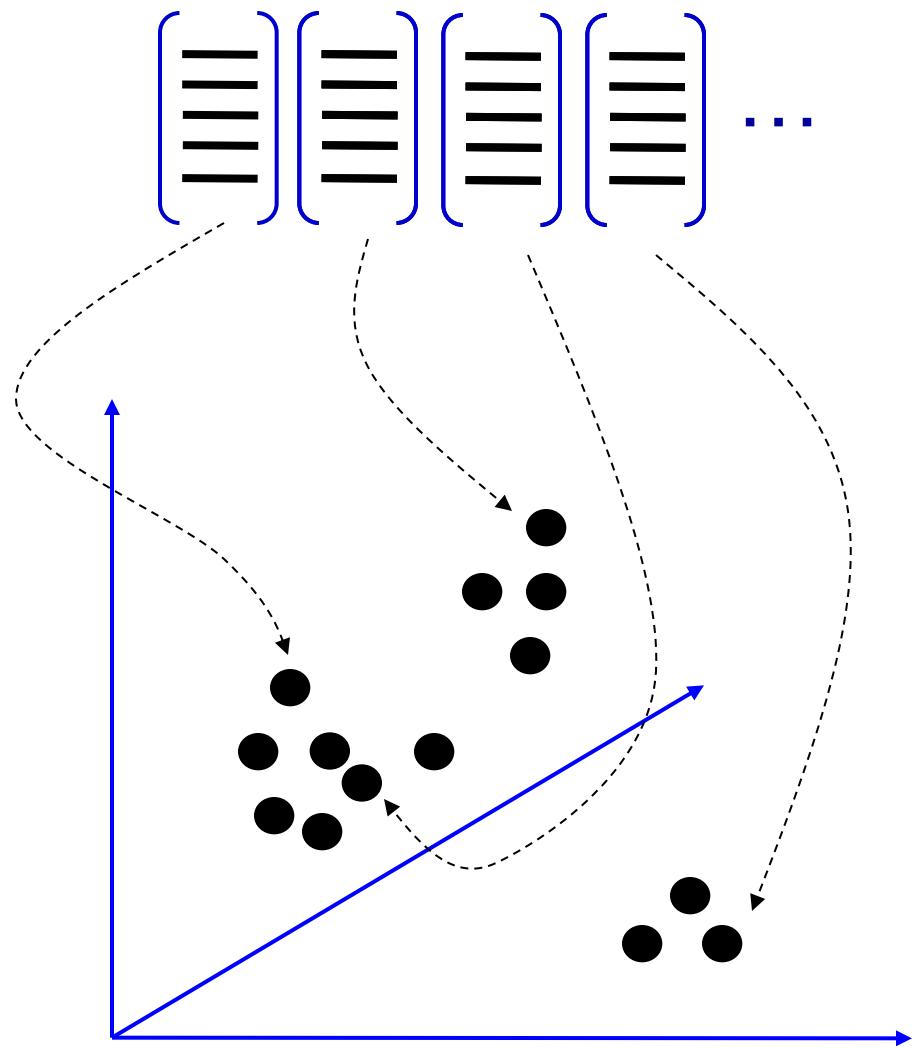
1. Feature extraction



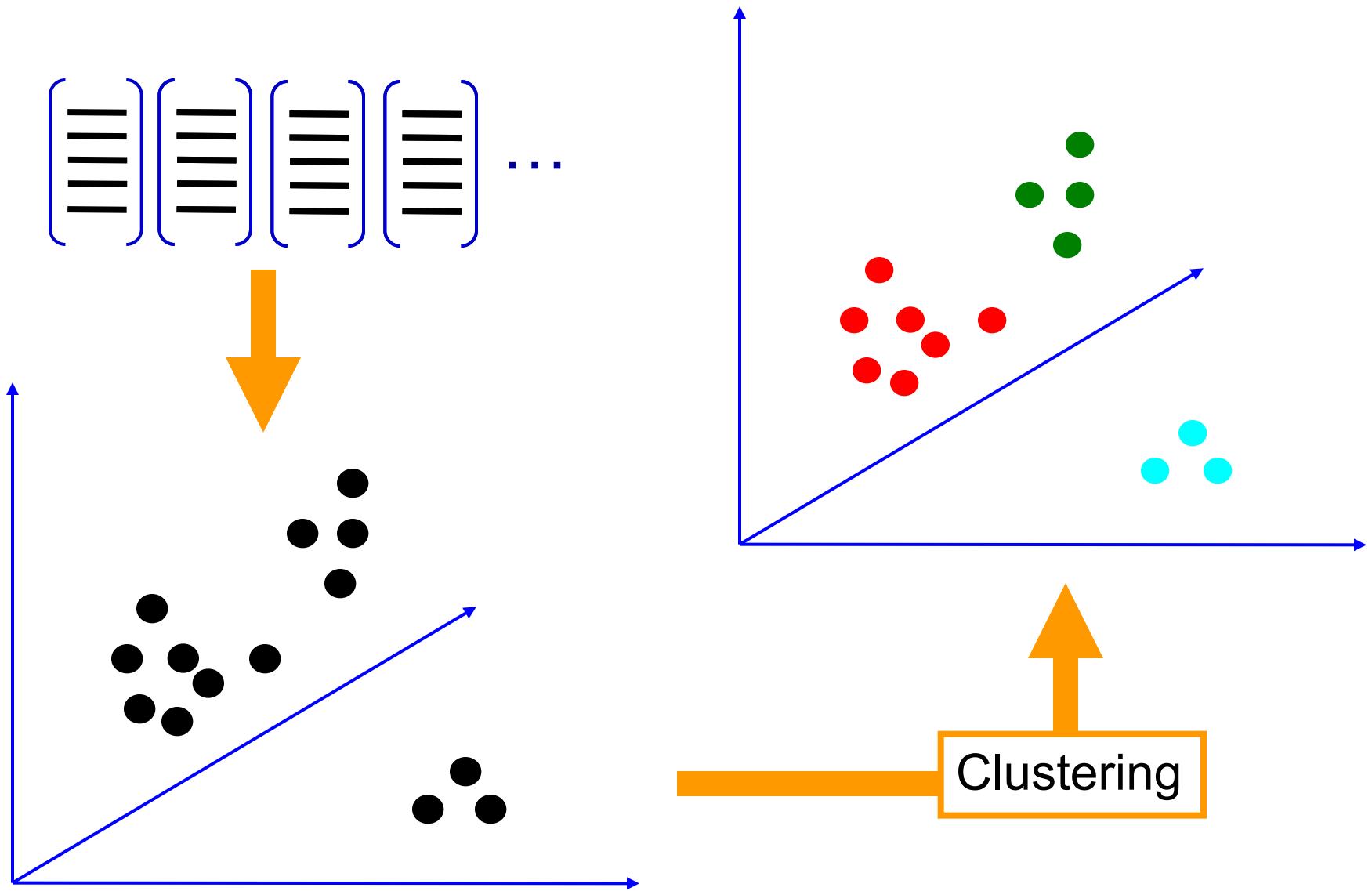
... ←



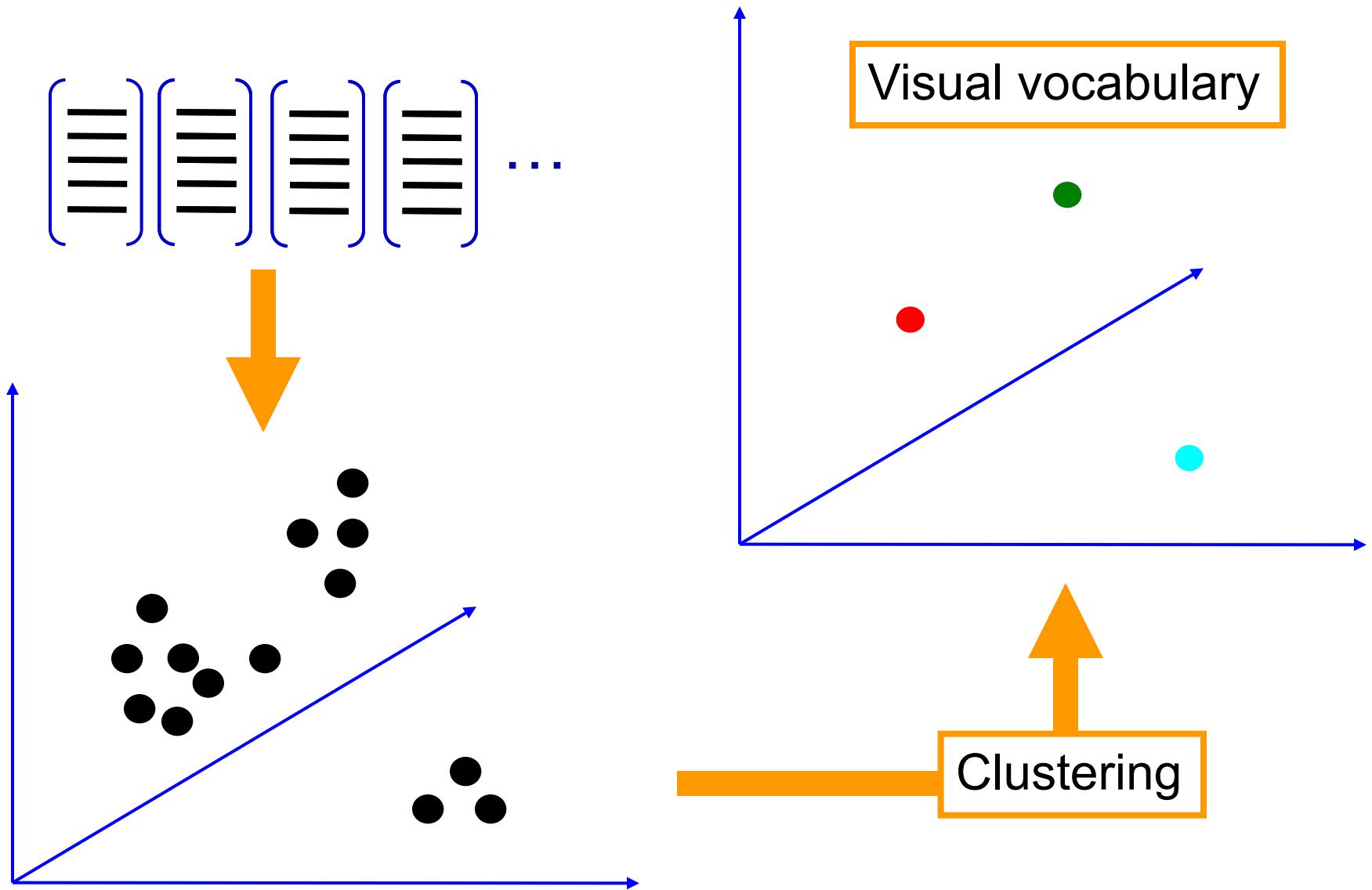
2. Learning the visual vocabulary



2. Learning the visual vocabulary

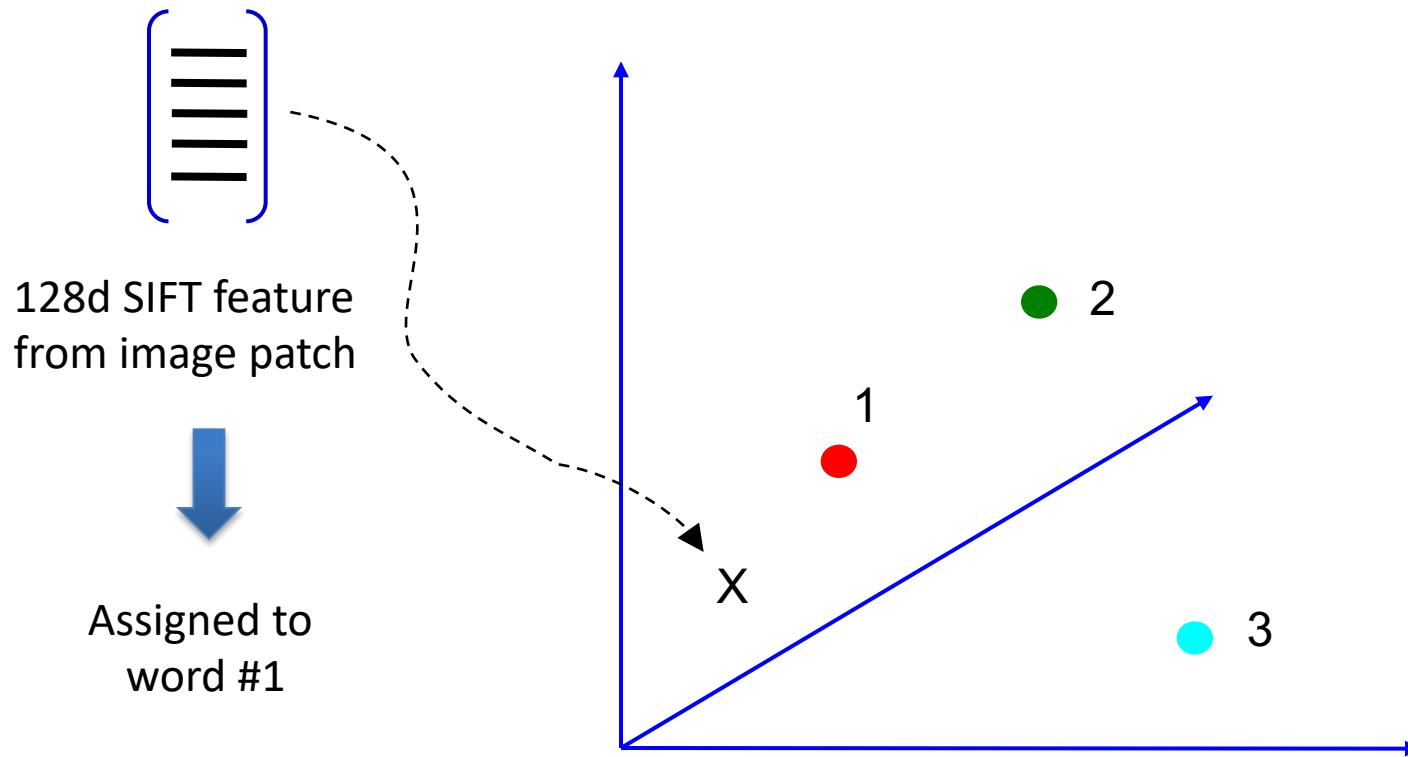


2. Learning the visual vocabulary

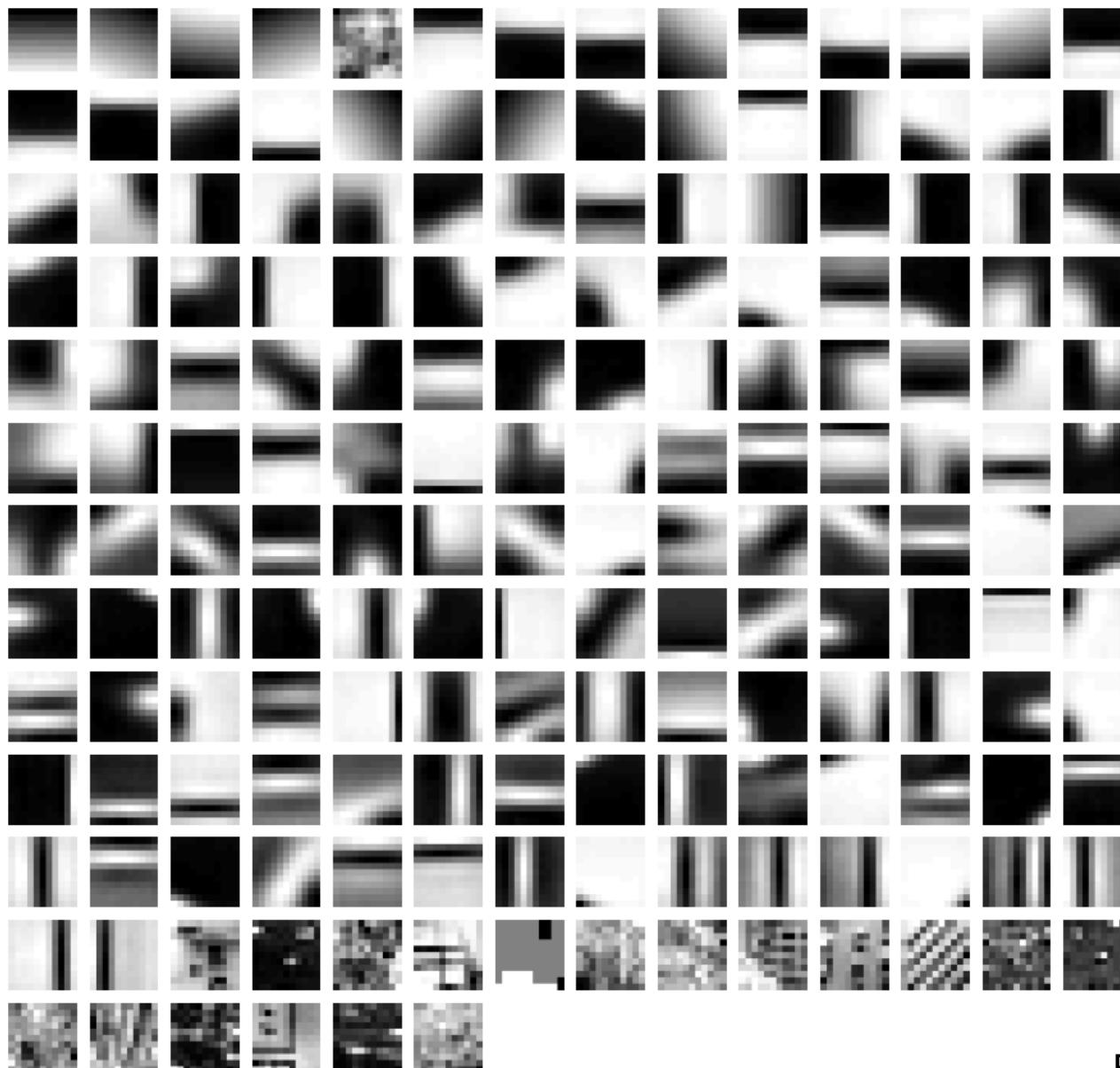


Vector quantization

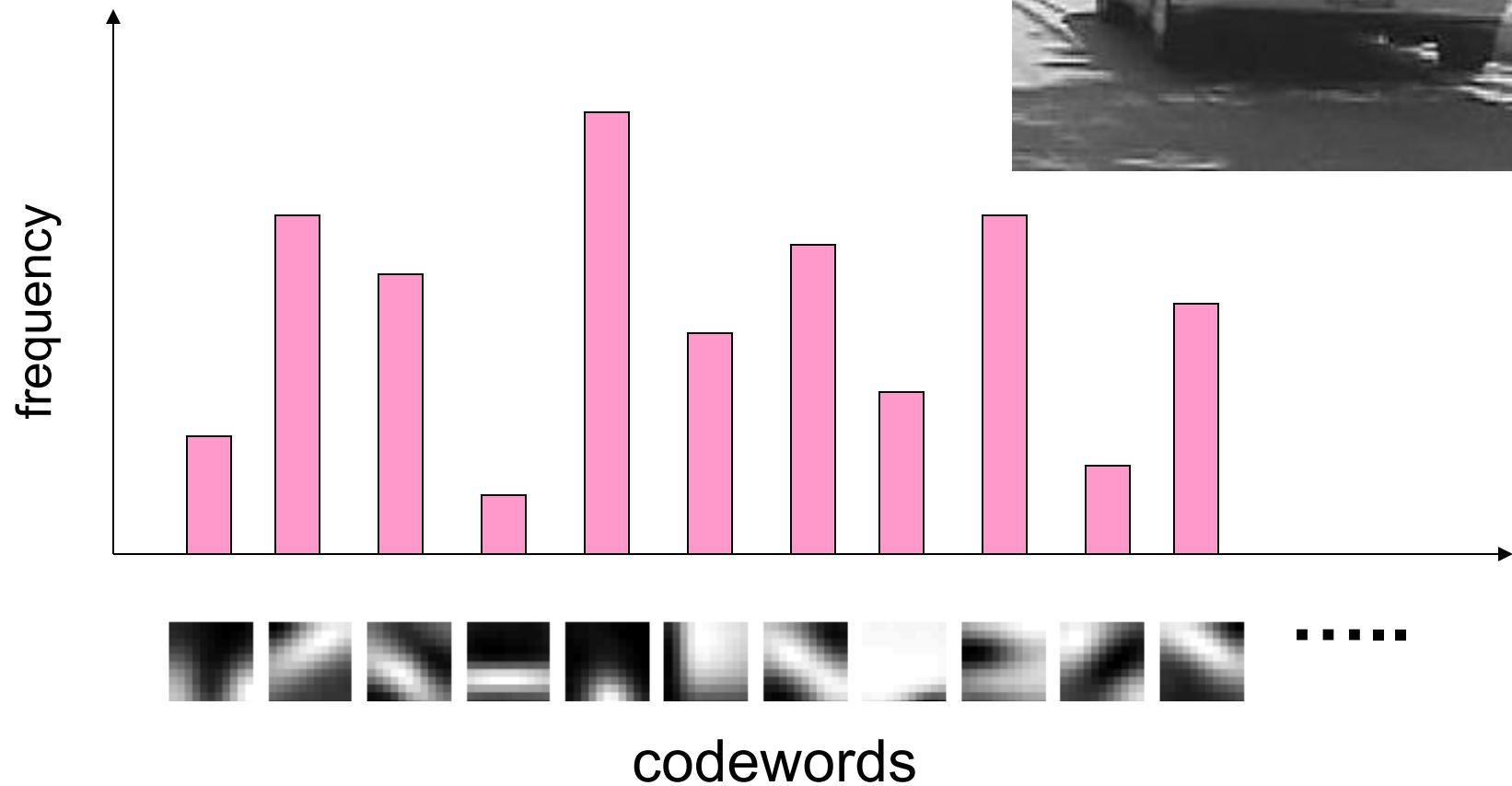
- Given a new feature vector, find the closest codevector, and assign it the nearest cluster centroid



Example visual vocabulary



3. Image representation



4. Learning a classifier

- SVMs are de facto standard technique
 - RBF kernel usually works slightly (2-3%) better, but at significantly higher computational cost
 - Why such a modest improvement?

Secret of SVM popularity

- At their core, SVMs are simple linear classifiers, but:
 - Soft-margin learning allows for outliers – not “brittle” like perceptron learning
 - Well-principled, easy to understand learning – less likely to fall into local minima
 - Many applications have high dimensional feature spaces with redundancy – linear classification works okay
 - Fast – classification is just a dot product, basic learning also fast (but gets complicated with kernels, slack variables, huge datasets, high dimensionality)
 - Kernel trick – allows trade-off between strength of model and danger of overfitting

Next class

- Convolutional neural networks

Deep learning: Convolutional Neural Networks (CNN)

Announcements

- A2 grades will be uploaded tonight
- A3 due on December 2nd
- Optional A4 will be released this week (it will be due during finals week)
- Final exam is on December 16
- Let us know if there is any outstanding regrade request
- Bump in points for A0 readme

22.87 / 33

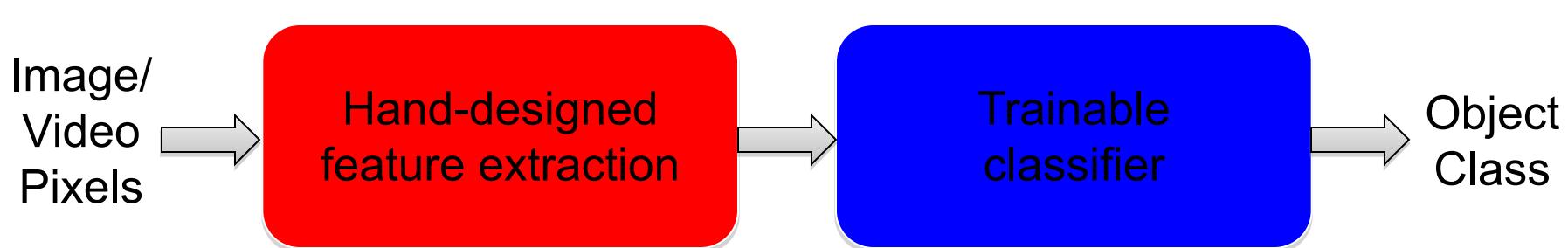
31.50	/33
31.00	/33
30.50	/33
30.00	/33
30.00	/33
30.00	/33
29.50	/33
29.50	/33
29.50	/33
29.00	/33
29.00	/33
28.50	/33
28.50	/33
28.00	/33
28.00	/33
28.00	/33
28.00	/33
28.00	/33
28.00	/33
28.00	/33
27.50	/33
27.50	/33
27.50	/33
27.50	/33
27.00	/33

First place: automatic A+

Second and third place: bump in the grade (e.g. A to A+)

“Shallow” vs. “deep” learning

“Shallow” architecture



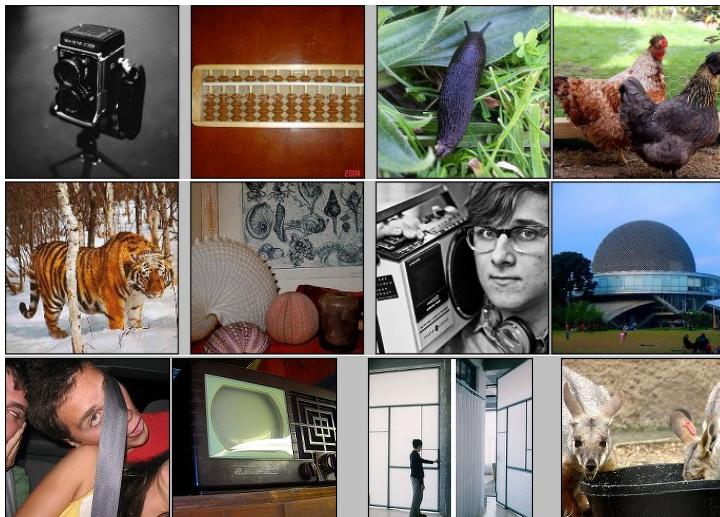
Deep learning: “Deep” architecture



Regularization and optimization in Neural Networks

- Regularization and optimization in Neural Networks
 - Parameter sharing: e.g. Convolutional Neural Networks (CNNs)
 - Weight decay
 - Dropout
 - Early stopping
 - Change of learning rate during learning
 - Adding momentum
 - Dataset augmentation
 - Activation function selection
- Based on:
 - Deep learning book (<http://www.deeplearningbook.org/>), chapter 9 (CNN)

ImageNet Challenge 2012



[Deng et al. CVPR 2009]

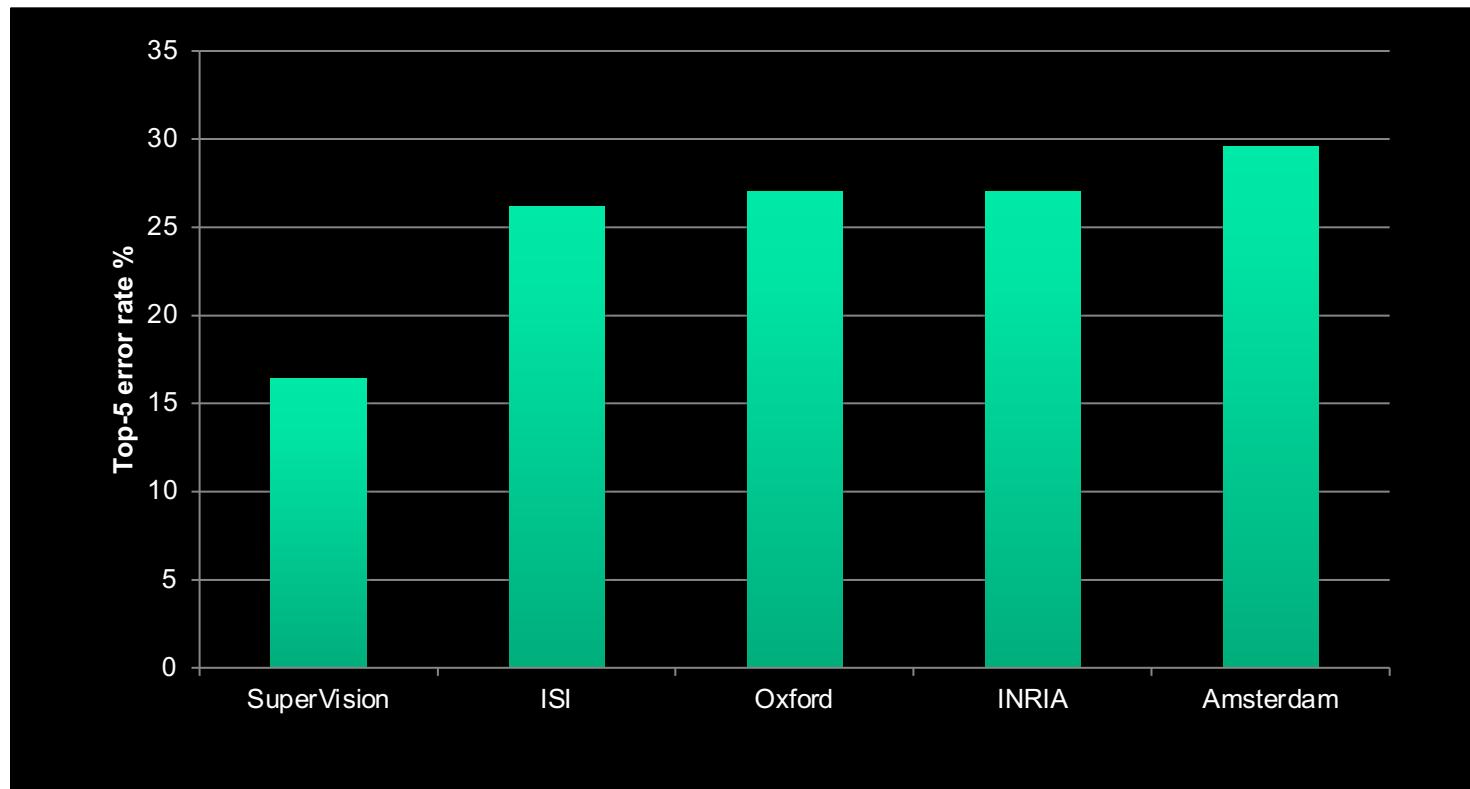
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes

A. Krizhevsky, I. Sutskever, and G. Hinton, [ImageNet Classification with Deep Convolutional Neural Networks](#), NIPS 2012

Slide credit: Rob Fergus

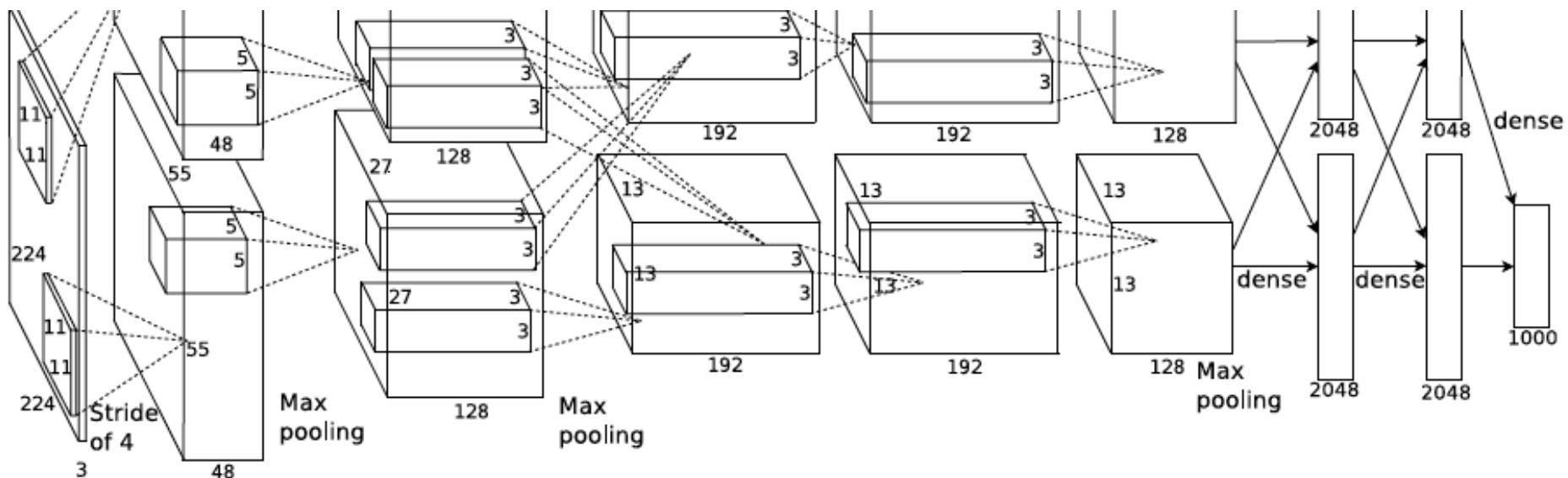
ImageNet Challenge 2012

- Krizhevsky et al. -- **16.4% error (top-5)**
- Next best (non-convnet) – **26.2% error**



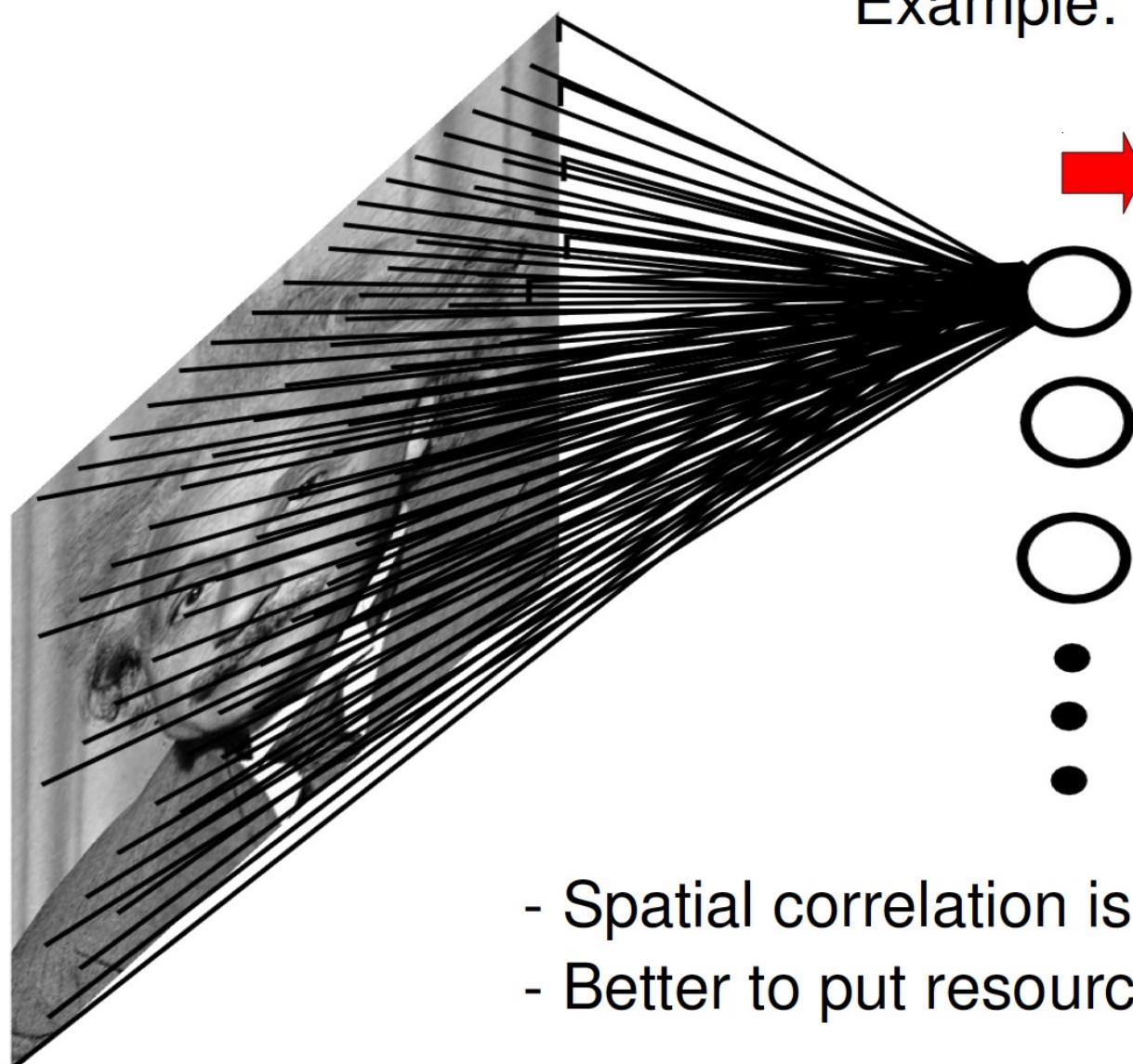
ImageNet Challenge 2012

- Similar framework to LeCun'98 but:
 - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
 - More data (10^6 vs. 10^3 images)
 - GPU implementation (50x speedup over CPU)
 - Better regularization for training (DropOut)



A. Krizhevsky, I. Sutskever, and G. Hinton, [ImageNet Classification with Deep Convolutional Neural Networks](#), NIPS 2012

FULLY CONNECTED NEURAL NET



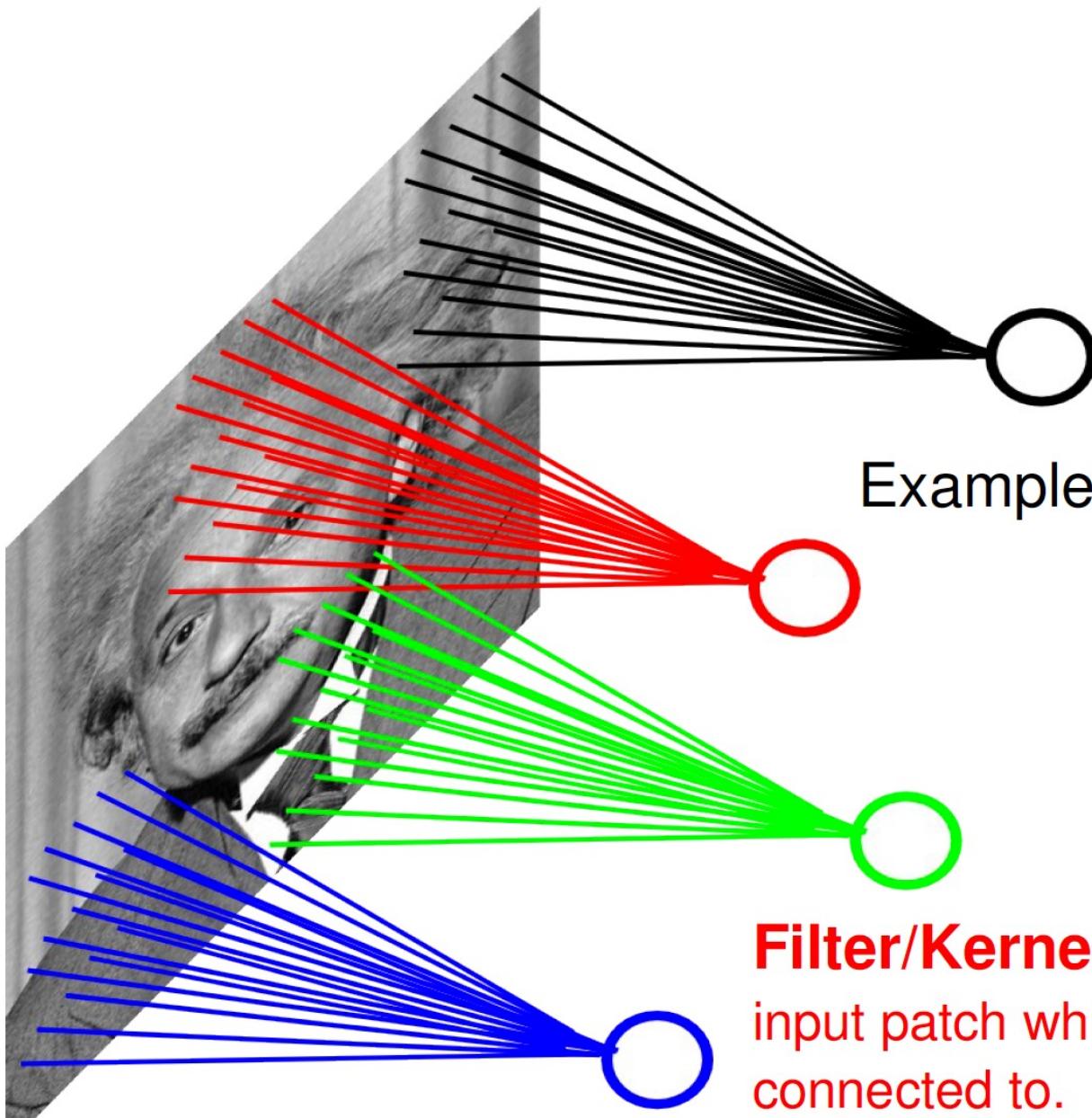
Example: 1000x1000 image

1M hidden units

10¹² parameters!!!

- Spatial correlation is local
- Better to put resources elsewhere!

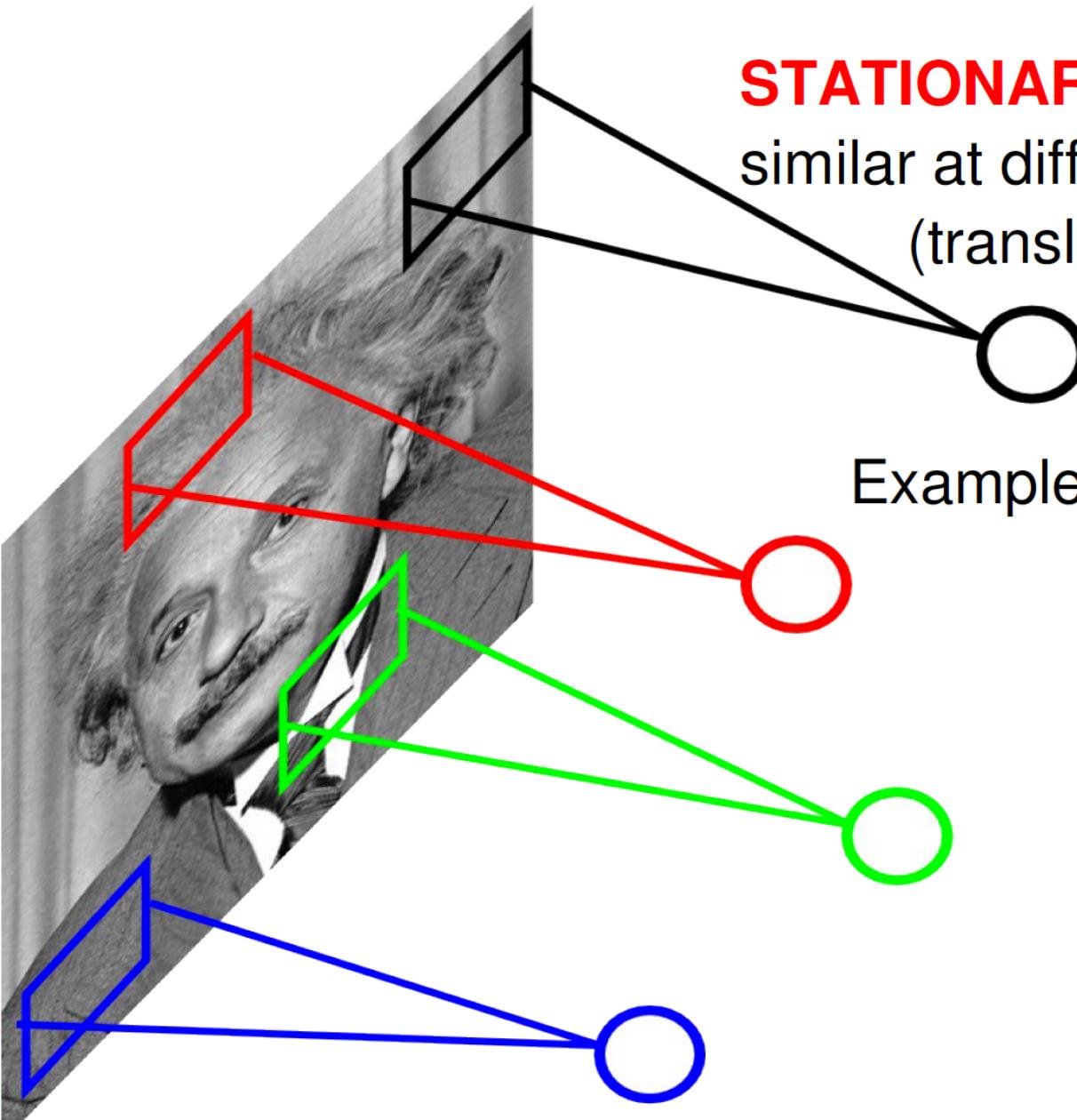
LOCALLY CONNECTED NEURAL NET



Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Filter/Kernel/Receptive field:
input patch which the hidden unit is
connected to.

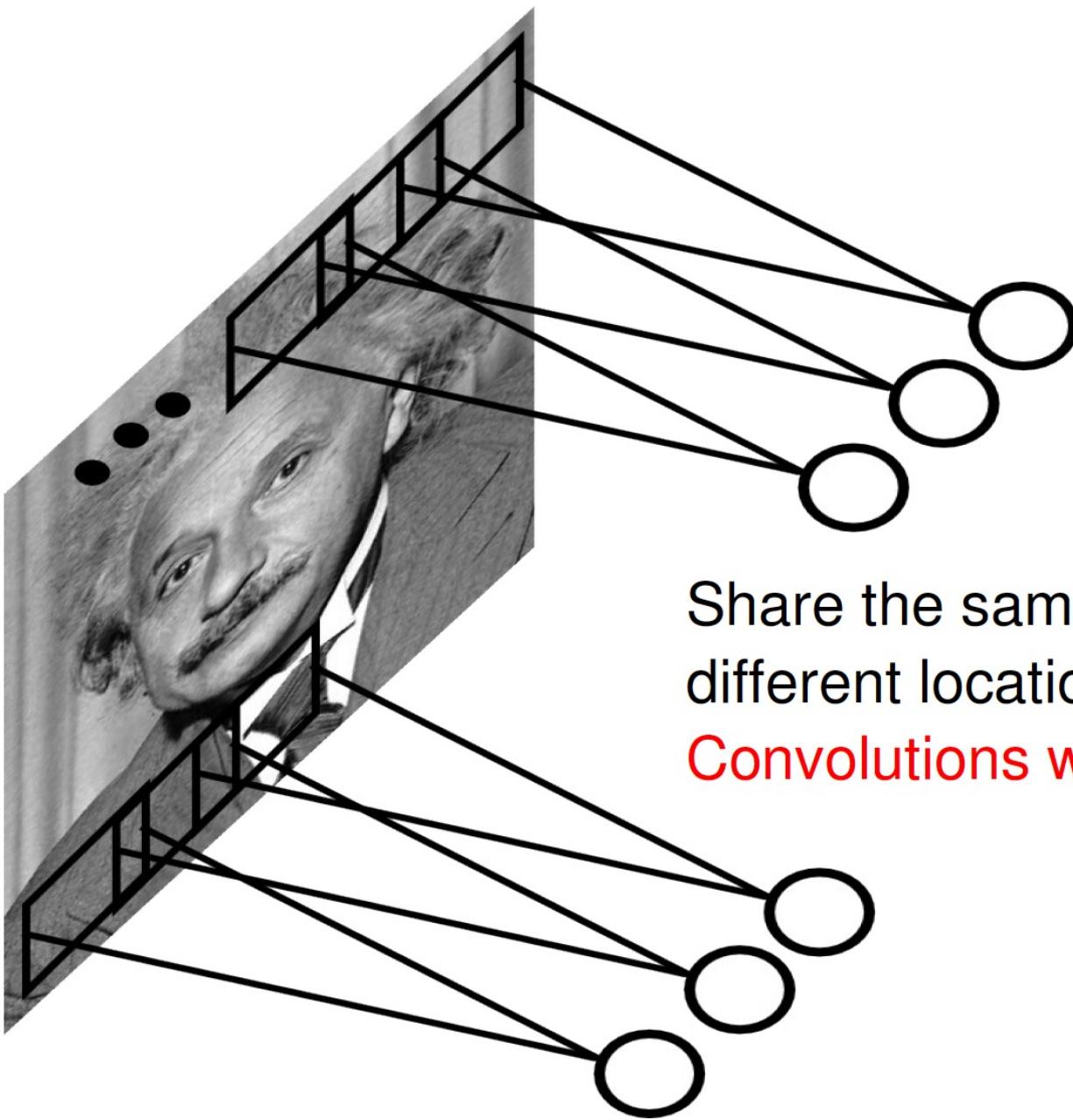
LOCALLY CONNECTED NEURAL NET



STATIONARITY? Statistics are similar at different locations
(translation invariance)

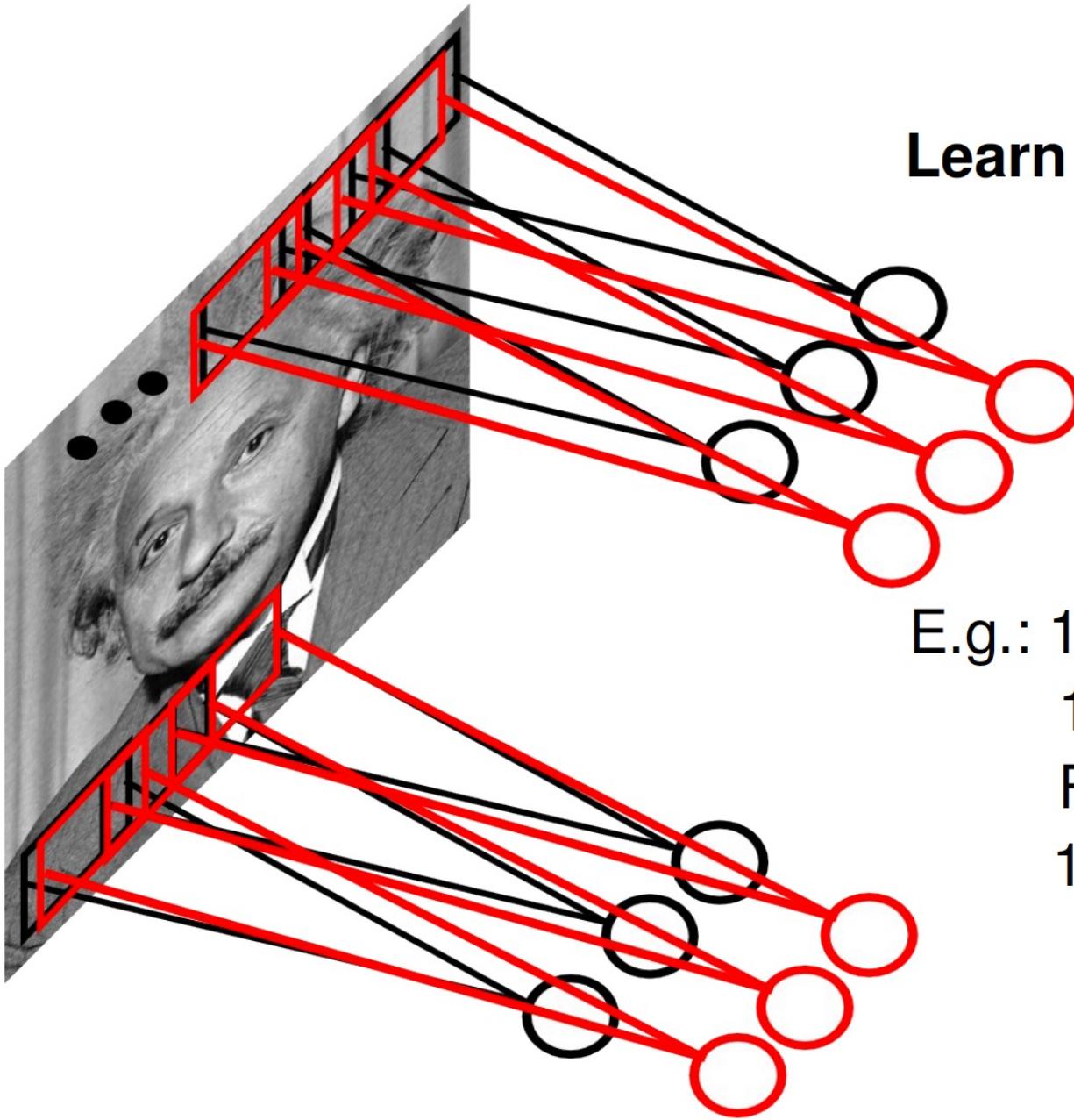
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

CONVOLUTIONAL NET



Share the same parameters across
different locations:
Convolutions with learned kernels

CONVOLUTIONAL NET

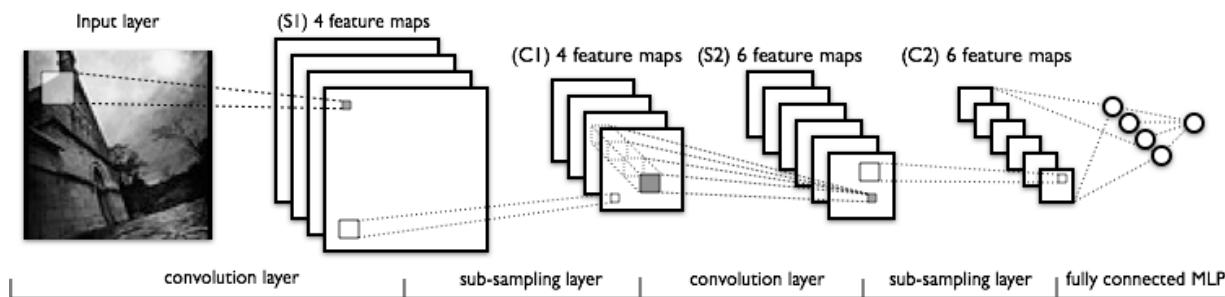


Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

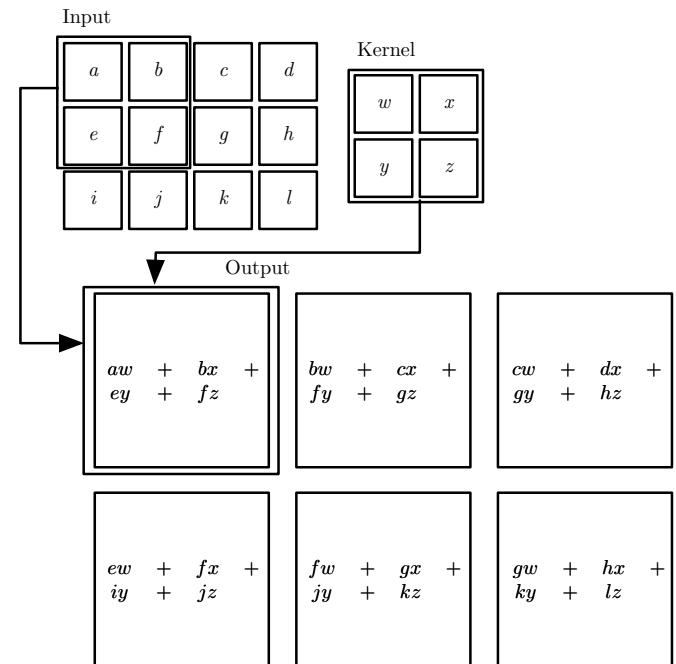
Convolutional Neural Networks

- Three principles:
 - local receptive fields
 - weight sharing
 - subsampling
- Multiple layers of Convolution, Activation, and Pooling may be used in a CNN
- These layers act as feature extraction, to find useful features from the input
- Generally, a final Fully Connected layer is added via a MLP for classification or regression purposes



Convolution Operation

- A mathematical depiction of the convolution operation on an input image
- A CNN learns the values of the filter (or kernel) on its own during the training process
- It outputs feature maps
- Most of the slides based on
 - <http://www.deeplearningbook.org/contents/convnets.html>



Goodfellow 2016

Example: Convolution on an Image

- Suppose you are given the following binary image, X

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Feature Map

4		

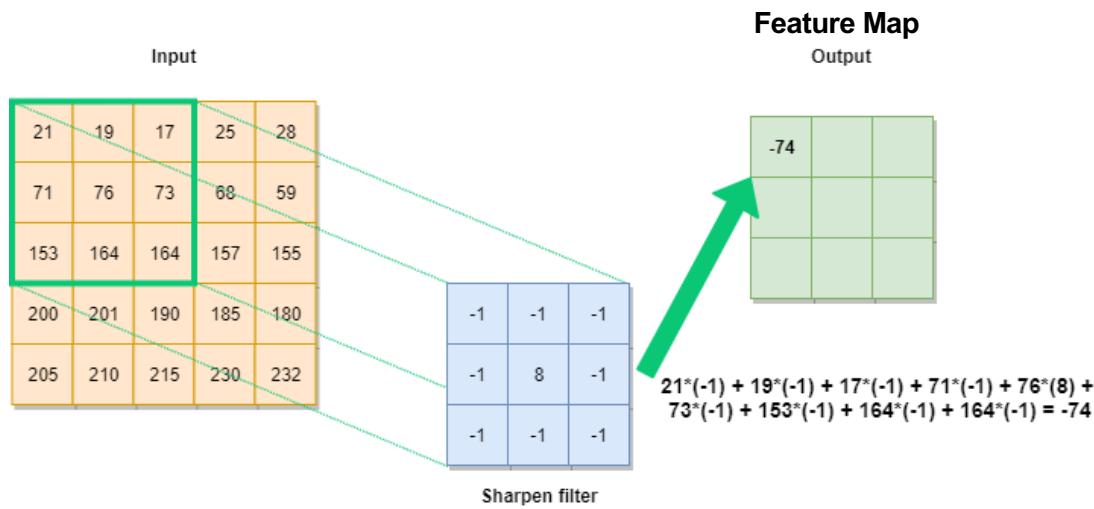
Convolved Feature

- You want to convolve this image with matrix, W (below). W is called ‘filter’ or ‘kernel’ or ‘feature detector’

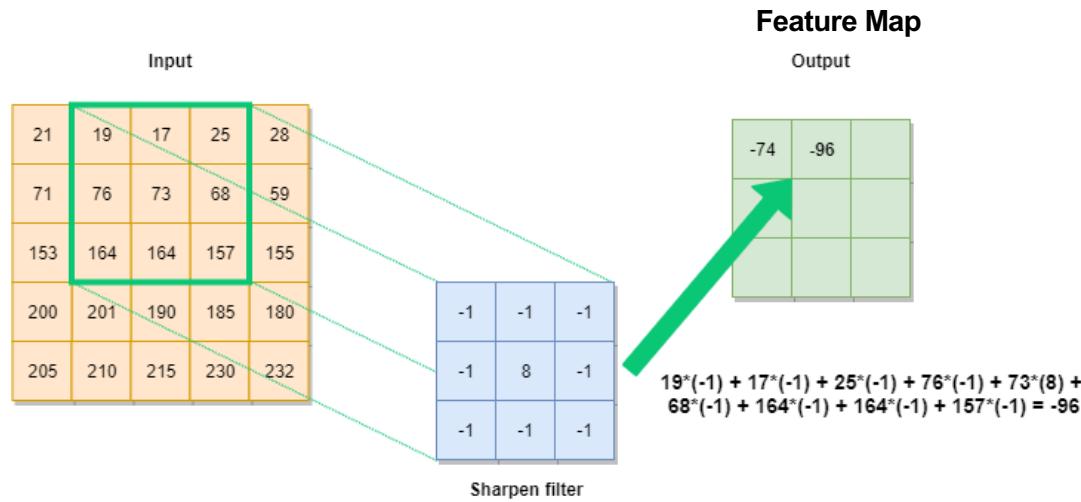
1	0	1
0	1	0
1	0	1

[the data science blog](http://the-data-science-blog.com)

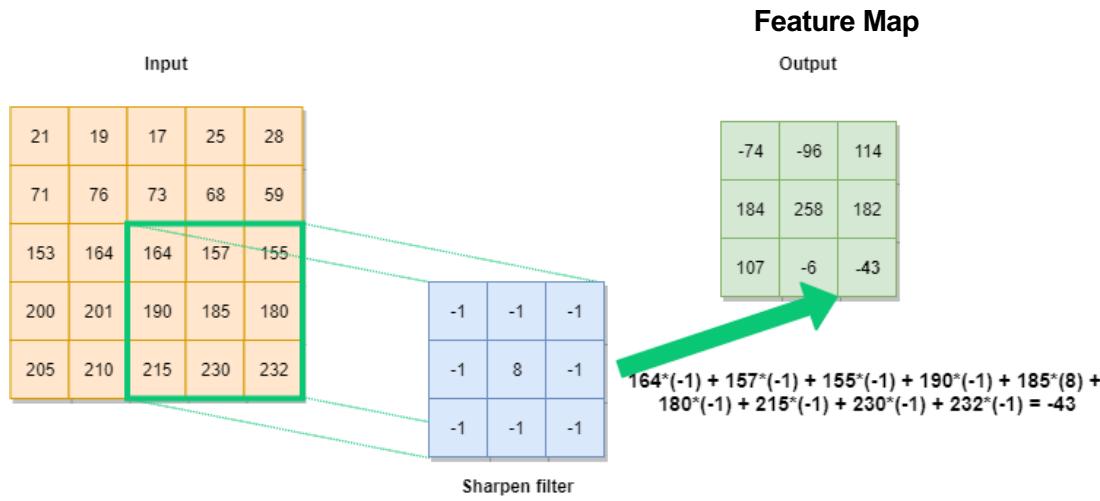
Example: Convolution on an Image



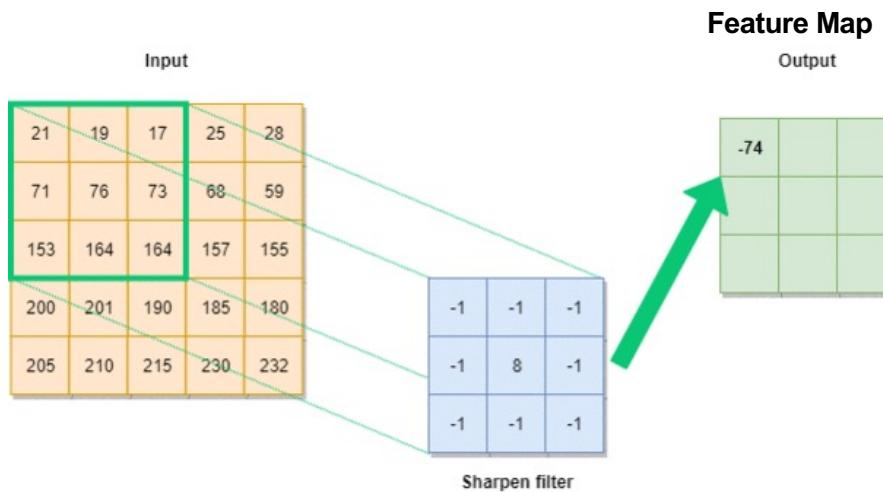
Example: Convolution on an Image



Example: Convolution on an Image



Example: Convolution on an Image



Convolution

- ▶ Convolution is a linear mathematical operation on two functions
- ▶ Convolution of functions $f(t)$ and $g(t)$ is:

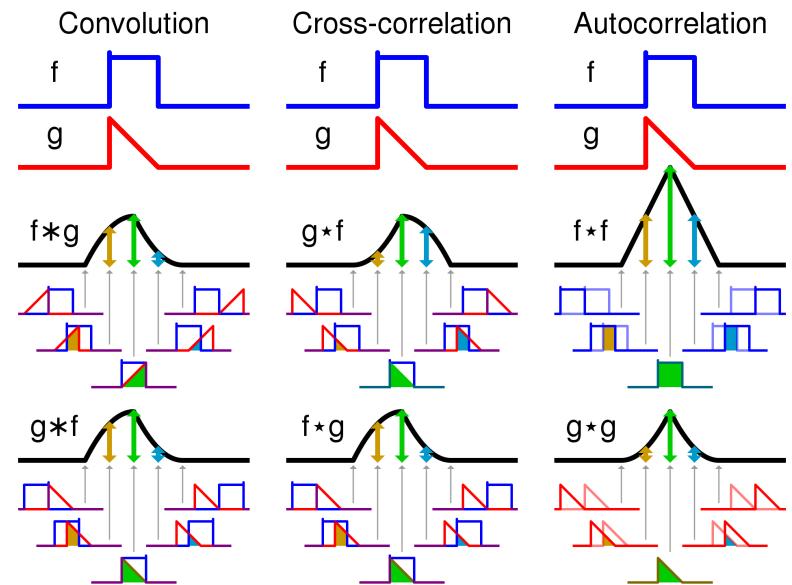
▶ 2D: $f(t) * g(t) = \sum_{a=-\infty}^{\infty} f(a)g(t-a)$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

- ▶ Cross-correlation of functions $f(t)$ and $g(t)$ is:

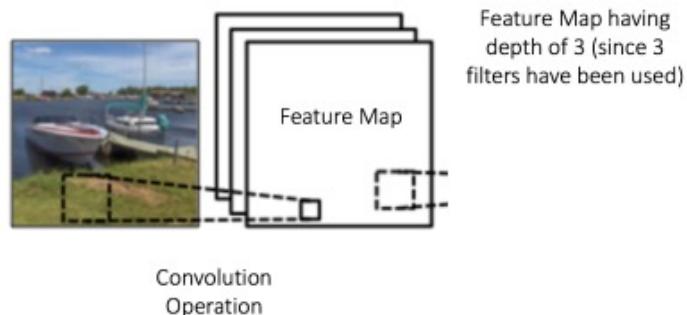
$$f(t) \star g(t) = \sum_{a=-\infty}^{\infty} f(a)g(t+a)$$

▶ 2D: $S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$



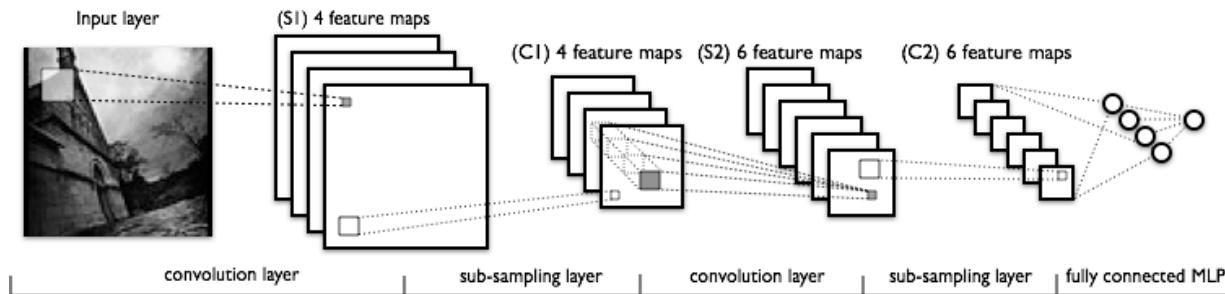
Feature Map

- The size of the Feature Map (resulting image after convolution) is controlled by three parameters
 - Depth: Number of different filters to use for the convolution operation
 - Stride: is the amount by which the kernel is moved by as the kernel is passed over the input
 - Zero-padding: May pad the input with zeros around the border



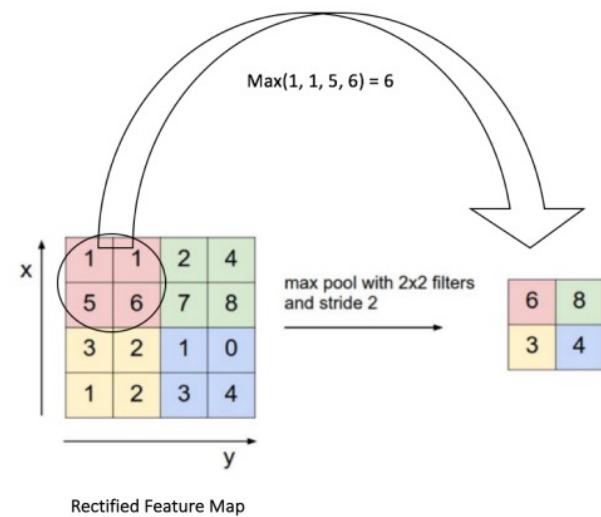
Convolutional Neural Networks

- Three principles:
 - local receptive fields
 - weight sharing
 - subsampling
- Multiple layers of Convolution, Activation, and Pooling may be used in a CNN
- These layers act as feature extraction, to find useful features from the input
- Generally, a final Fully Connected layer is added via a MLP for classification or regression purposes



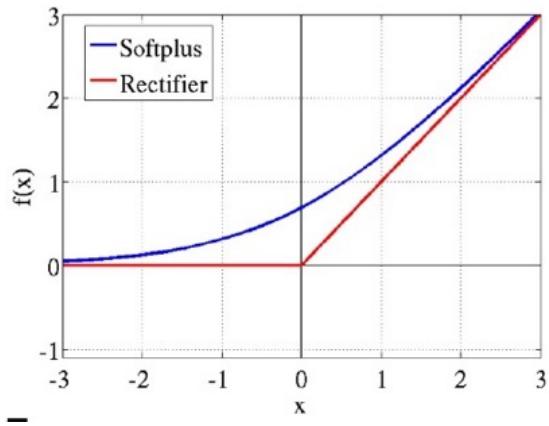
Pooling

- Pooling (aka spatial pooling, subsampling, or downsampling) is used to reduce the dimensionality of the feature map
- Different types of pooling include: Max, Average, Sum, etc.
- A window is defined, and the pooling operation is performed over the elements within that window
- The pooling window slides over the feature map by the stride amount
- It is applied to each feature map



Rectified Linear (ReLU) Activation

- A rectified linear (ReLU) activation operation may be applied after the convolution operation
 - Introduces nonlinearity to the network
- $\text{ReLU}(x) = \max(0, x)$
- ReLU is applied to every element (pixel)
 - Negative values are replaced by 0
- Other nonlinear activation functions may be used instead

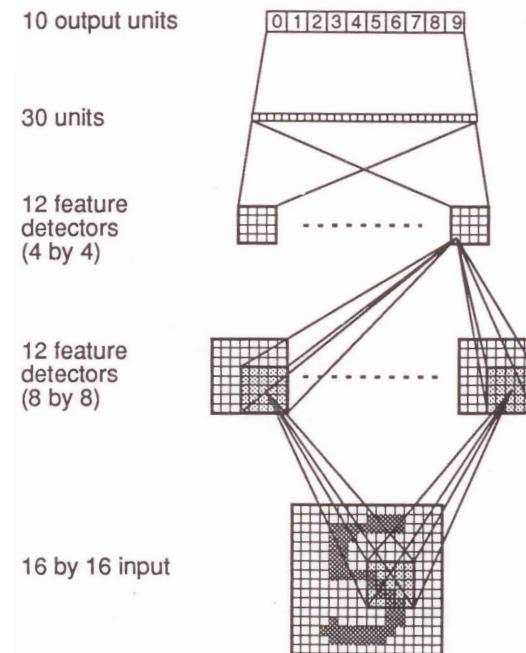


CNN Training

- The Backpropagation algorithm is used to train the parameters of a CNN
- Basic steps:
 - Randomly initialize all filters (or kernels) and weights
 - Propagate the input forward through the layers of the CNN (convolution, activation, pooling, MLP) to get an output(s)
 - Calculate the error between the actual output(s) and the desired output(s)
 - Use backpropagation to calculate the gradients and deltas, and then update the filters and weights accordingly

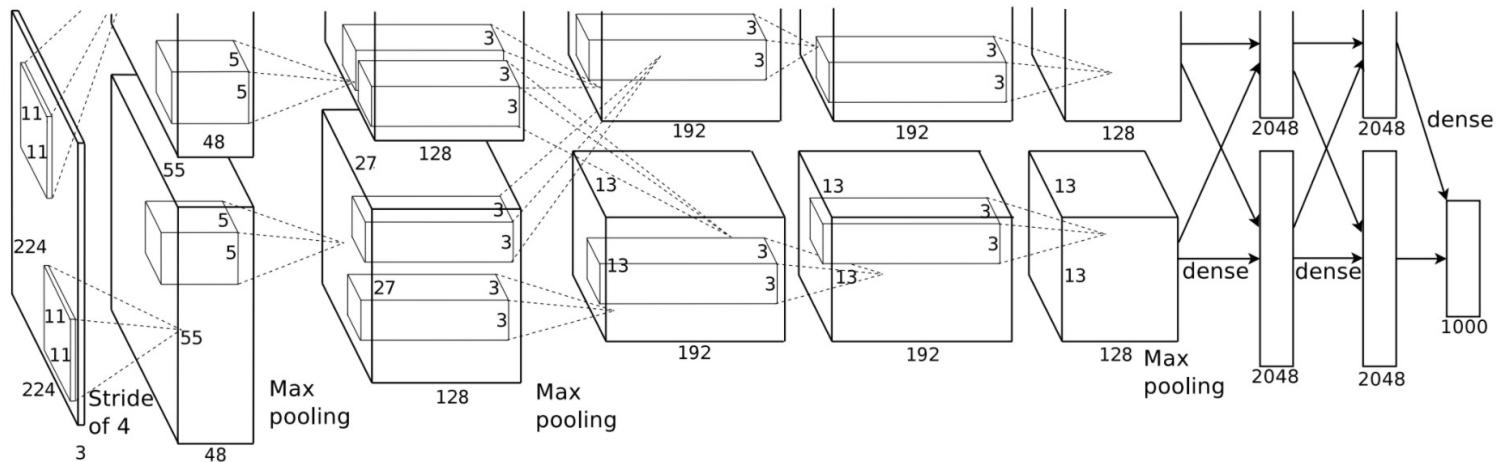
An Early CNN Application

- **Task:** Handwritten Zip code Recognition (1989)
- **Network Description**
 - Input: binary pixels for each digit
 - Output: 10 digits
 - Architecture: 4 layers ($16 \times 16 - 12 \times 8 \times 8 - 12 \times 4 \times 4 - 30 - 10$)
- **Performance:** Trained on 7300 digits and tested on 2000 new ones
 - Achieved 1% error on the training set and 5% error on the test set
 - If allowing rejection (no decision), 1% error on the test set
 - This task is not easy

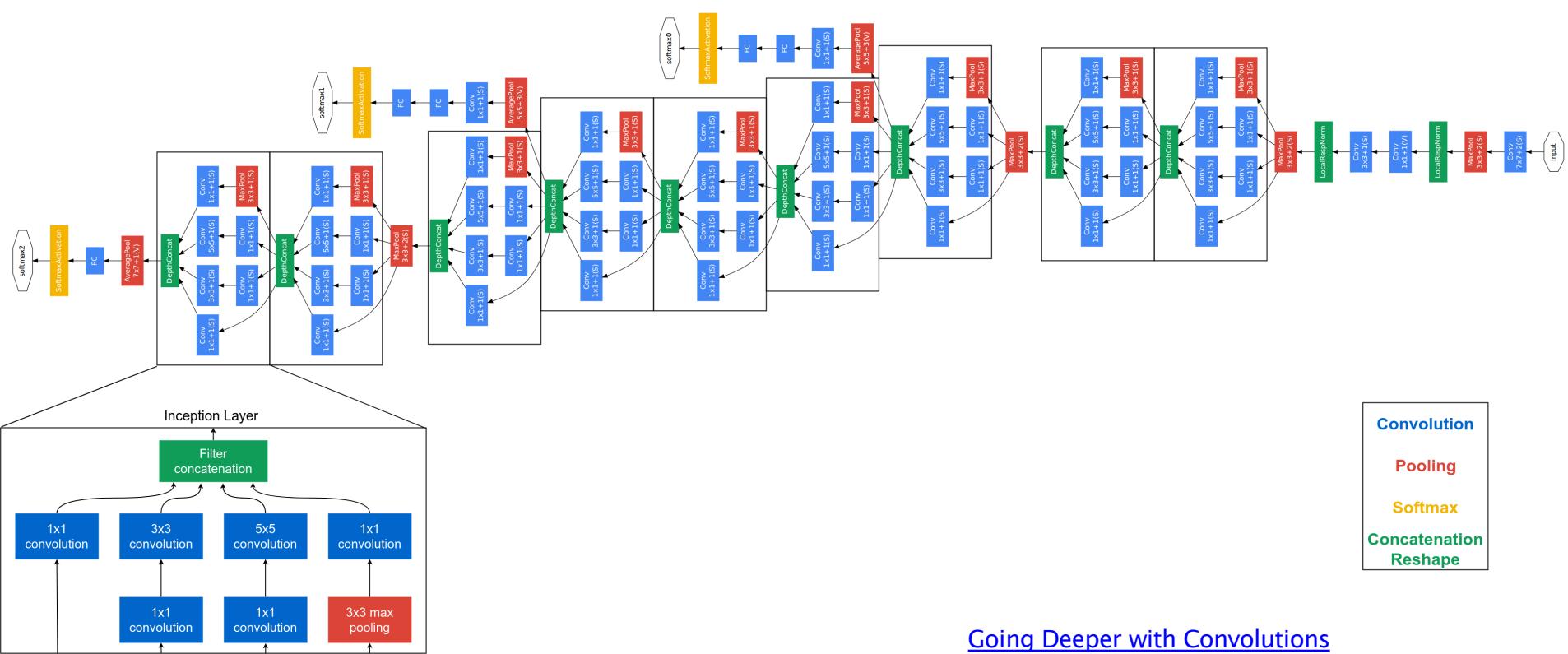


AlexNet

- Krizhevsky, NeurIPS 2012 (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton)



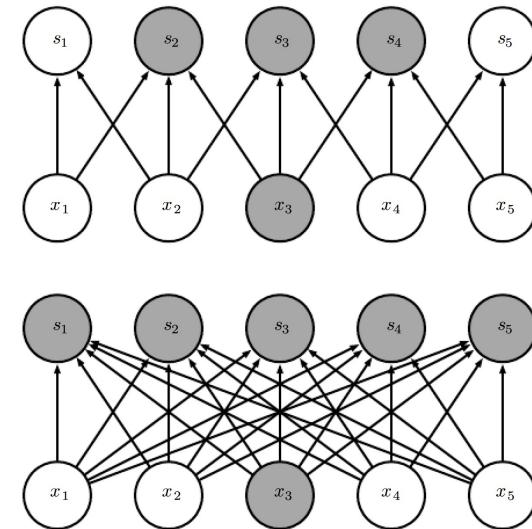
GoogLeNet



[Going Deeper with Convolutions](#)

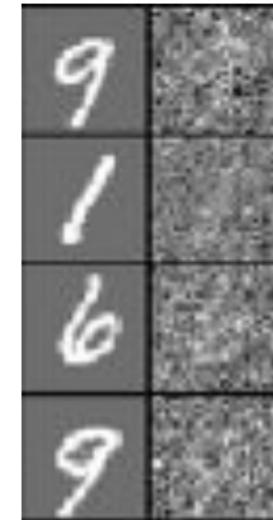
Important properties of CNNs

- Sparse interactions (sparse connectivity or sparse weights)
 - Kernel is smaller than the input (top plot)
- Parameter sharing (tied weights)
 - using the same parameter for more than one function in a model
- Equivariance to translation
 - If we move the object in the input, its representation will move the same amount in the output



CNN mysteries

CNNs can sometimes outperform humans. Even after adding extreme noise, all these digits are recognized correctly!

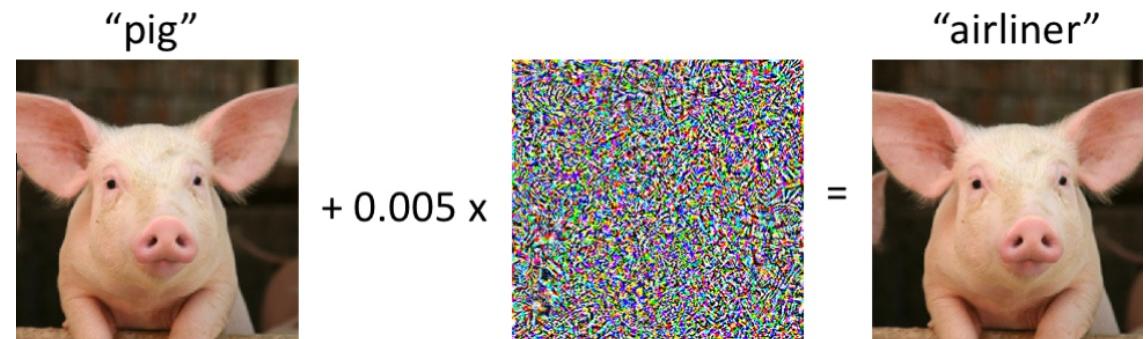


But, only first column of these are recognized correctly. Second column digits are all wrong!

Problems with CNNs



Sample of how an object in a noncanonical orientation and context fools many current object classification systems (Alcorn et al., 2018)



The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence

Visualizing Convnets

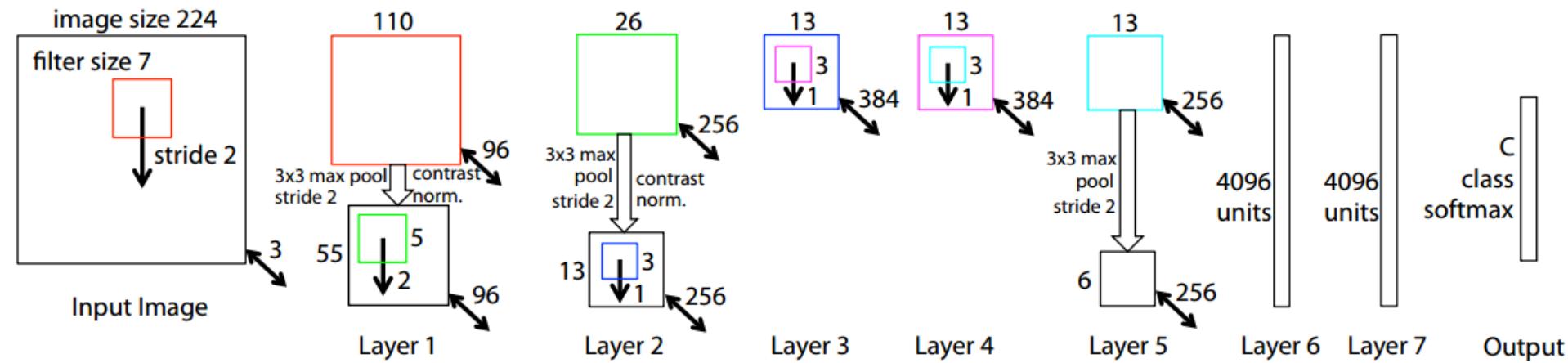
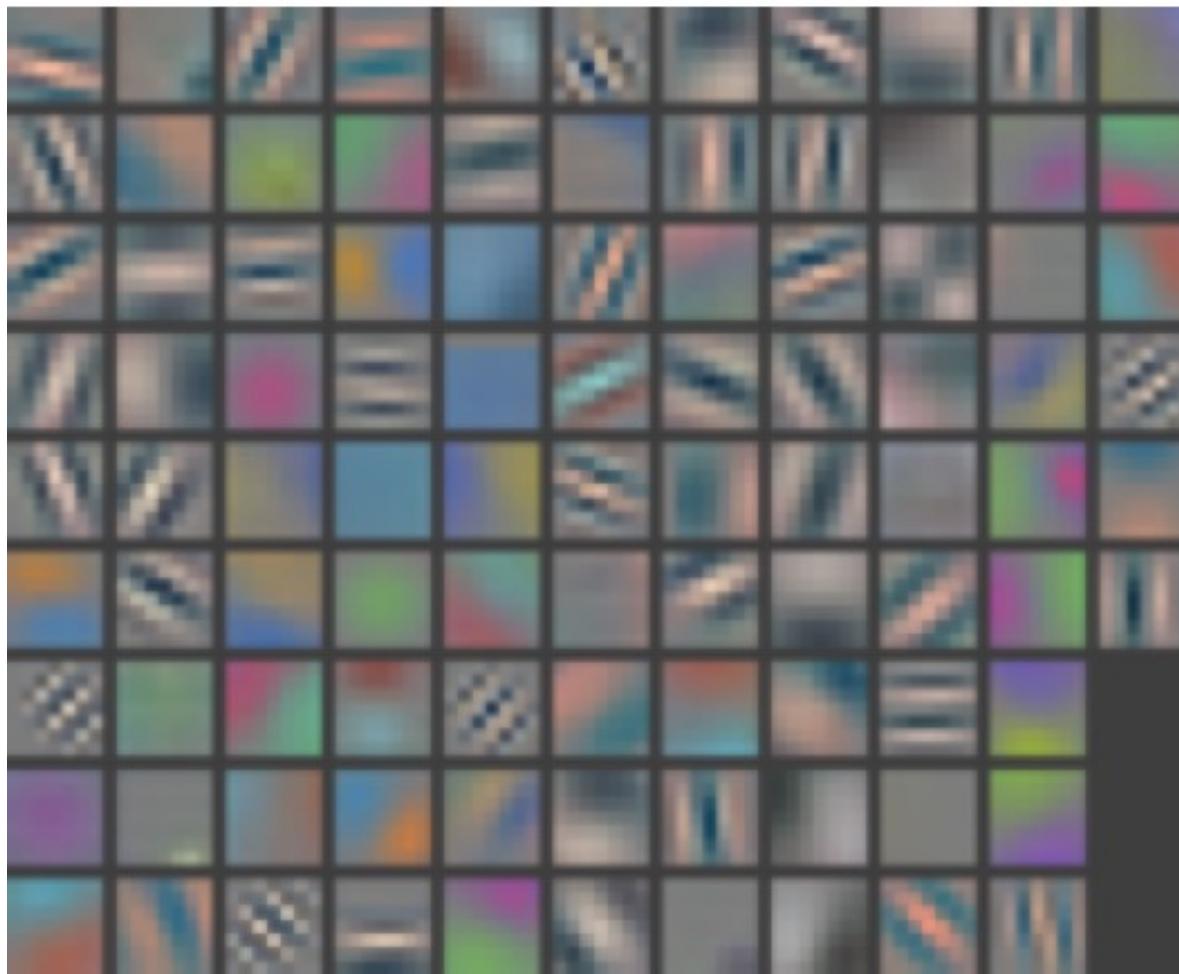
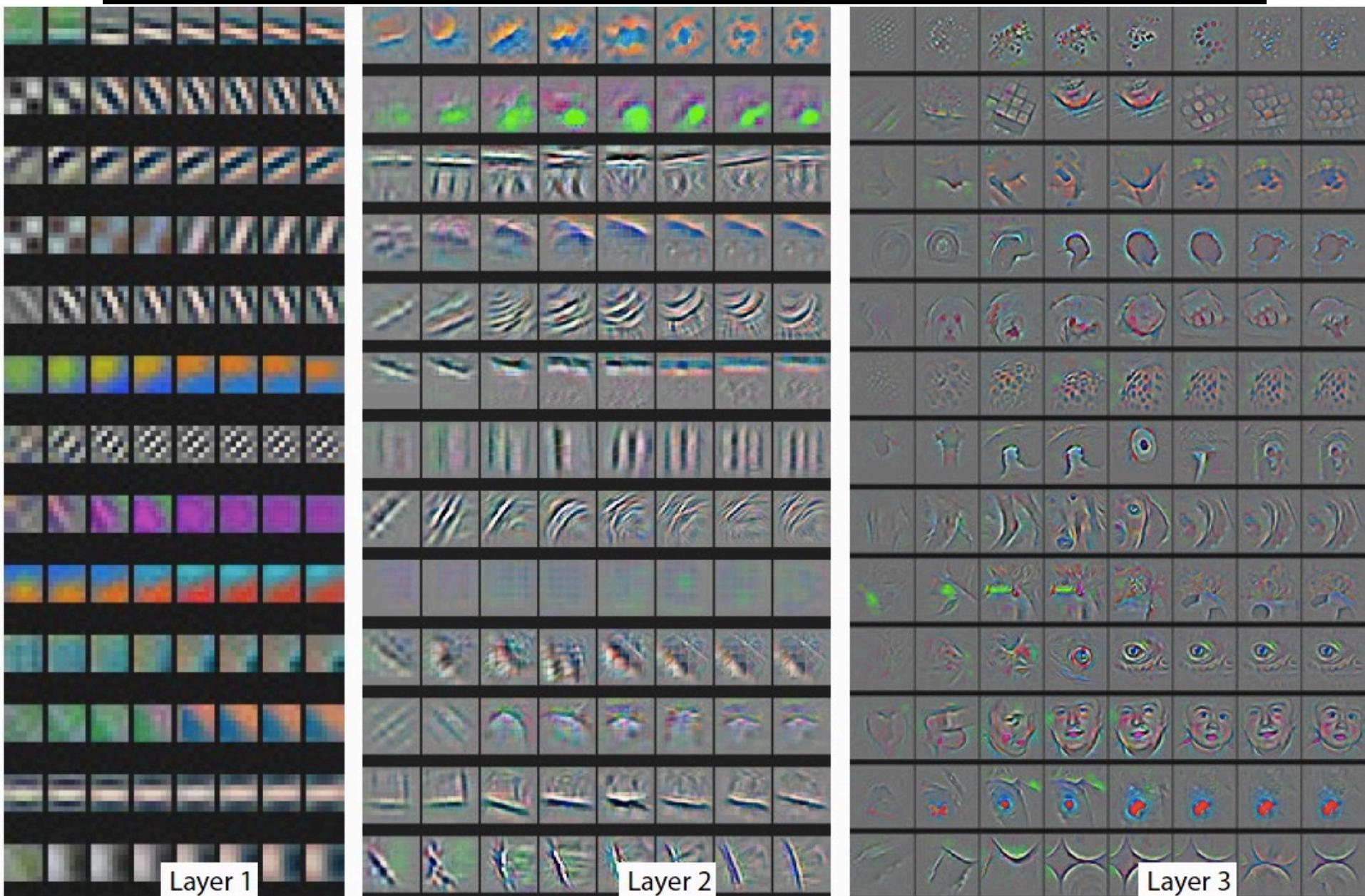


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax function, C being the number of classes. All filters and feature maps are square in shape.

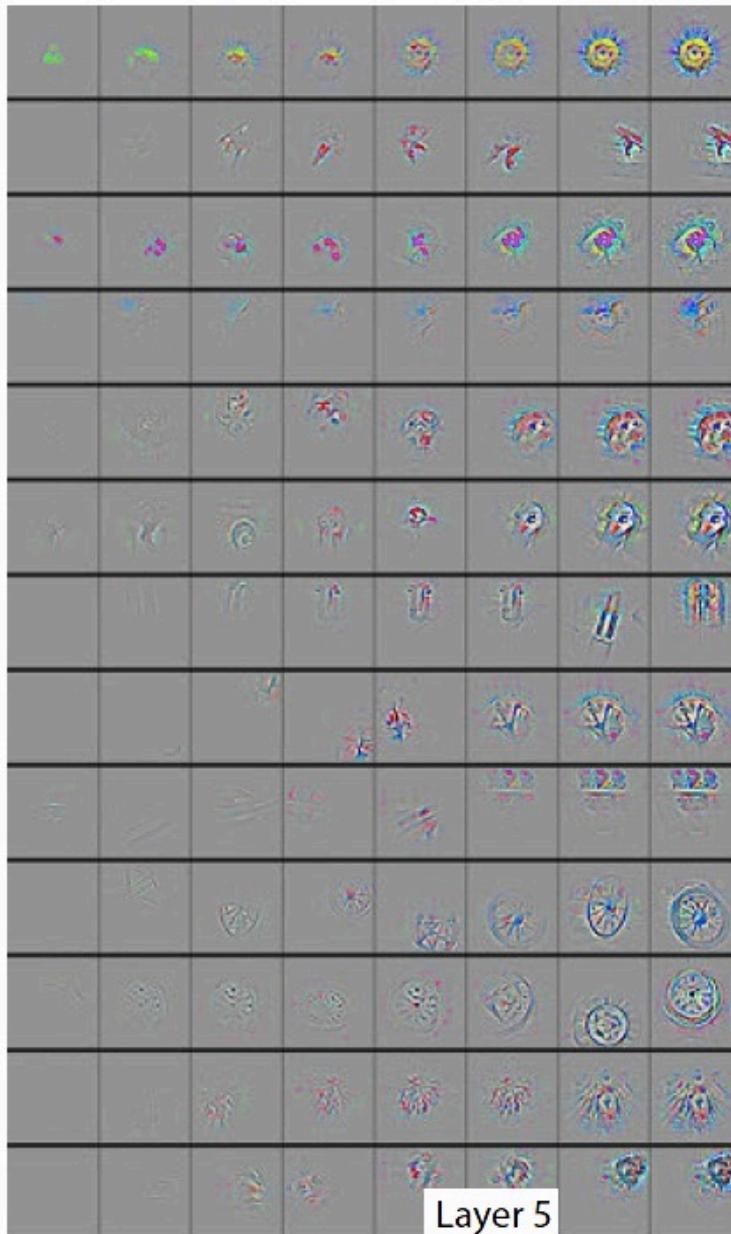
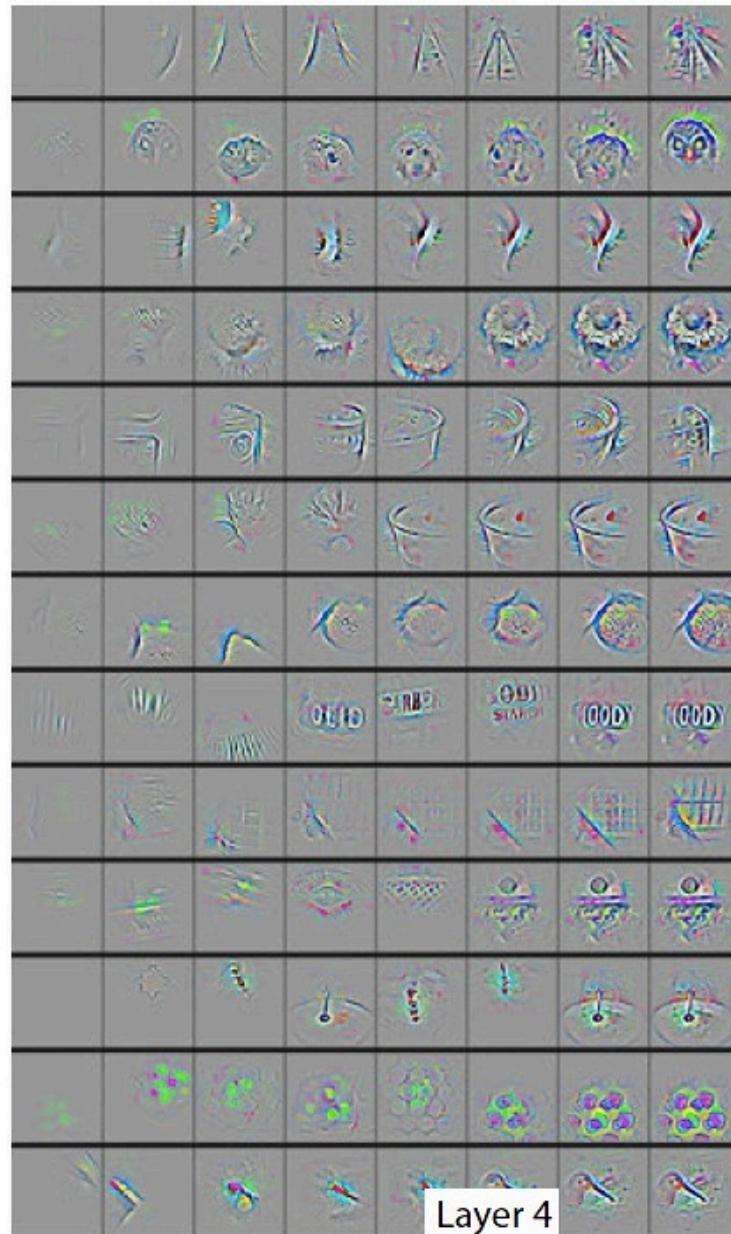
Layer 1 Filters



Evolution of Features During Training



Evolution of Features During Training



Other optimization and regularization techniques

Weight decay

Weight decay:

- Penalizes complexity and prevent overfitting

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

Weight decay

Weight decay

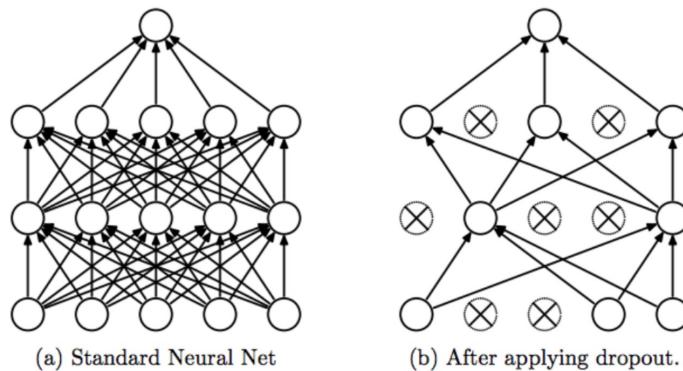
- Penalizes complexity and prevent overfitting

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2$$

Dropout

Dropout

- Dropping out units with (both hidden and visible) in a neural network. There is some probability that a unit will be dropped out (ignored in both forward and backward pass) at each training iteration.



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Early stopping

Early stopping

- Stopping the training of a neural network once the model performance stops improving on a hold out validation dataset.

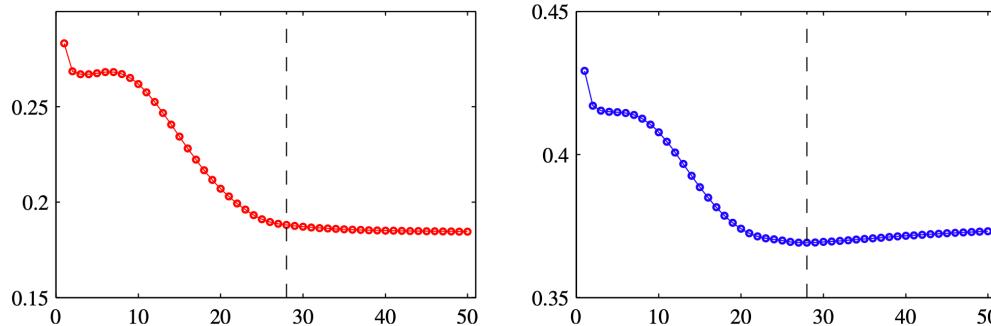


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

Change learning rate during learning

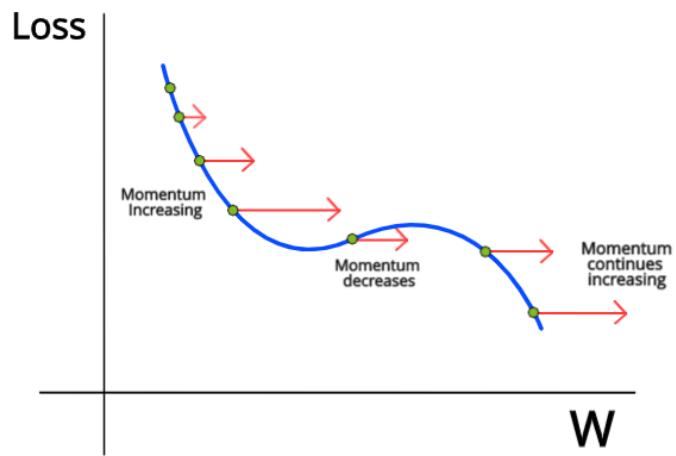
Change learning rate during learning:

- Make η large at the beginning of learning and decrease it later
- Simulated annealing: cleverly adjust learning rate using physics-inspired parameter called temperature

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

Momentum

Momentum: a way to escape small gradients



$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment learning rate weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

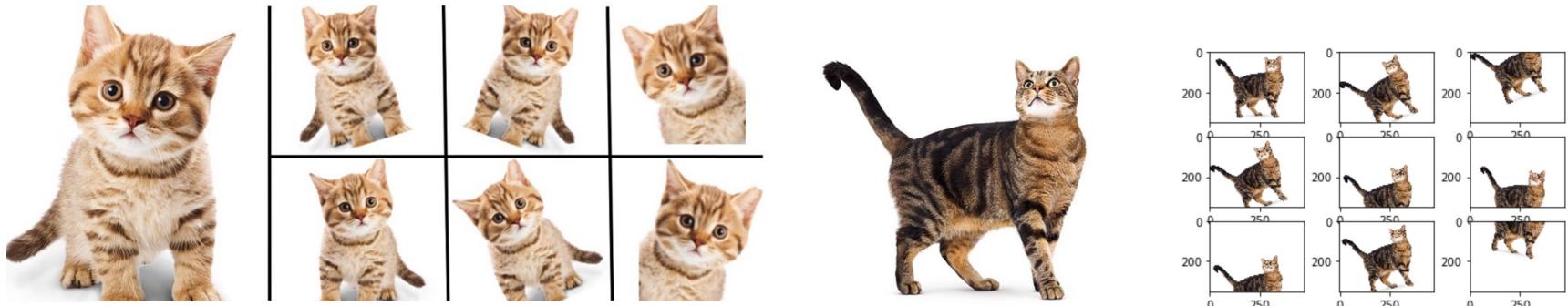
momentum factor weight increment, previous iteration

<https://deeplearningdemystified.com/article/fdl-4>

Dataset augmentation

Dataset augmentation

- E.g. adding noise to the input, applying spatial transformations to the input



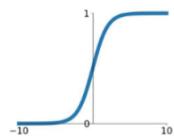
Activation function selection

In deep (many layers) networks, to avoid small gradient we can use activations functions such as ReLU, which have a large gradient for a wide range of values.

Activation Functions

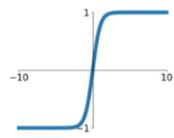
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



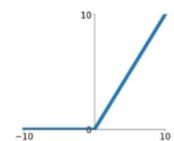
tanh

$$\tanh(x)$$



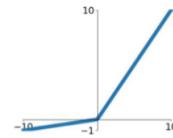
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

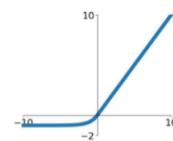


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Network References

For additional examples and references, check out the following:

- The Data Science Blog - <https://ujjwalkarn.me/blog/>
- Deep Learning Book - <http://www.deeplearningbook.org>
 - See also video lectures on the same page

Video lectures:

- <https://www.youtube.com/watch?v=Xogn6veSyxA> (Ian Goodfellow)
- <https://www.youtube.com/watch?v=H-HVZJ7kGI0&t=1057s> (Ava Soleimany)

Practical perspective:

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition
- See here for the code:
 - <https://github.com/ageron/handson-ml2>
 - https://github.com/ageron/handson-ml2/blob/master/14_deep_computer_vision_with_cnns.ipynb

Next class

More deep learning (Recurrent Neural Networks, Attention in Neural Networks, Reinforcement Learning)

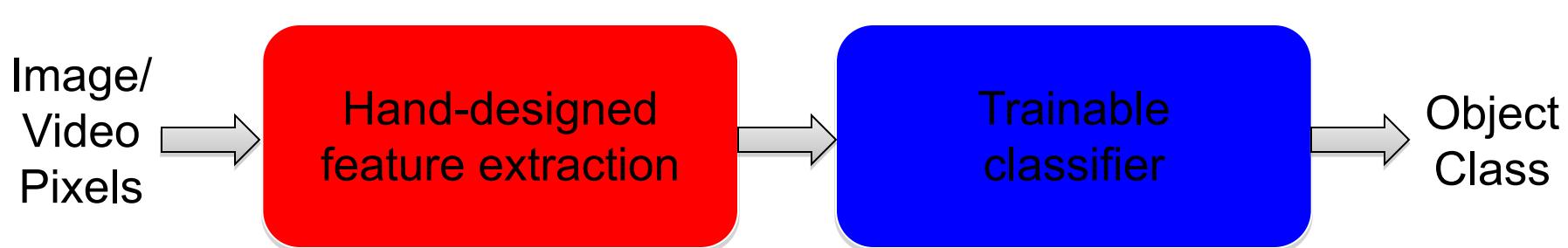
More about regularization and architectures in DNNs

Announcements

- A2 grades and comments posted
 - The three agents that we used for grading are hosted on sharks.luddy.indiana.edu, on ports 4996, 4997 and 4998 so you test your agents directly if you want.
- Optional A4 due on Sunday.
- Let us know if there is any outstanding regrade request
- Bump in points for A0 readme in the process

“Shallow” vs. “deep” learning

“Shallow” architecture



Deep learning: “Deep” architecture

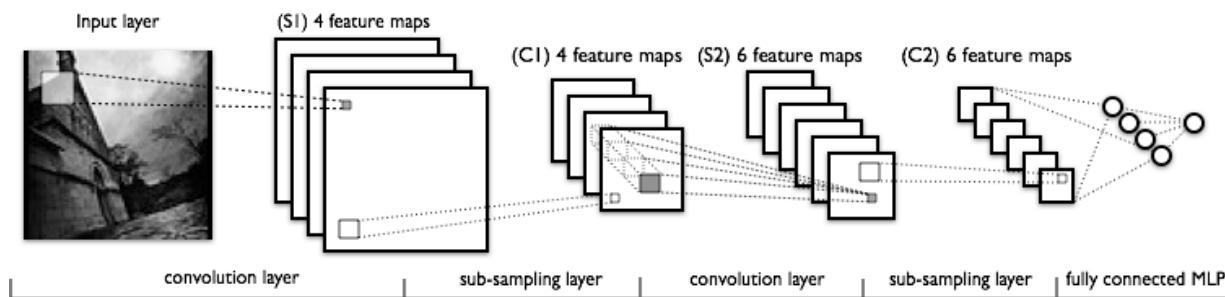


Regularization and optimization in Neural Networks

- Regularization and optimization in Neural Networks
 - Parameter sharing: e.g. Convolutional Neural Networks (CNNs)
 - Weight decay
 - Dropout
 - Early stopping
 - Change of learning rate during learning
 - Adding momentum
 - Dataset augmentation
 - Activation function selection
- Based on:
 - Deep learning book (<http://www.deeplearningbook.org/>), chapter 9 (CNN)

Convolutional Neural Networks

- Three principles:
 - local receptive fields
 - weight sharing
 - subsampling
- Multiple layers of Convolution, Activation, and Pooling may be used in a CNN
- These layers act as feature extraction, to find useful features from the input
- Generally, a final Fully Connected layer is added via a MLP for classification or regression purposes



Other optimization and regularization techniques

Weight decay

Weight decay:

- Penalizes complexity and prevent overfitting

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

Weight decay

Weight decay

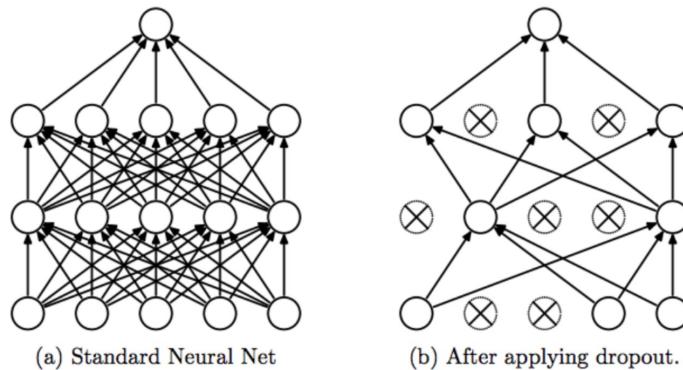
- Penalizes complexity and prevent overfitting

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2$$

Dropout

Dropout

- Dropping out units with (both hidden and visible) in a neural network. There is some probability that a unit will be dropped out (ignored in both forward and backward pass) at each training iteration.



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Early stopping

Early stopping

- Stopping the training of a neural network once the model performance stops improving on a hold out validation dataset.

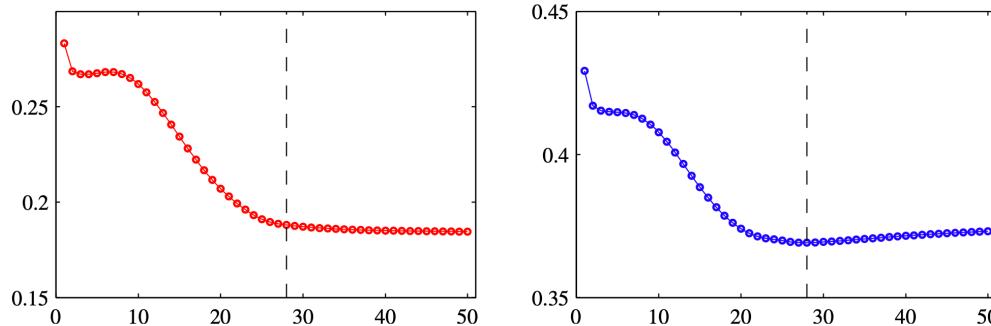


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

Change learning rate during learning

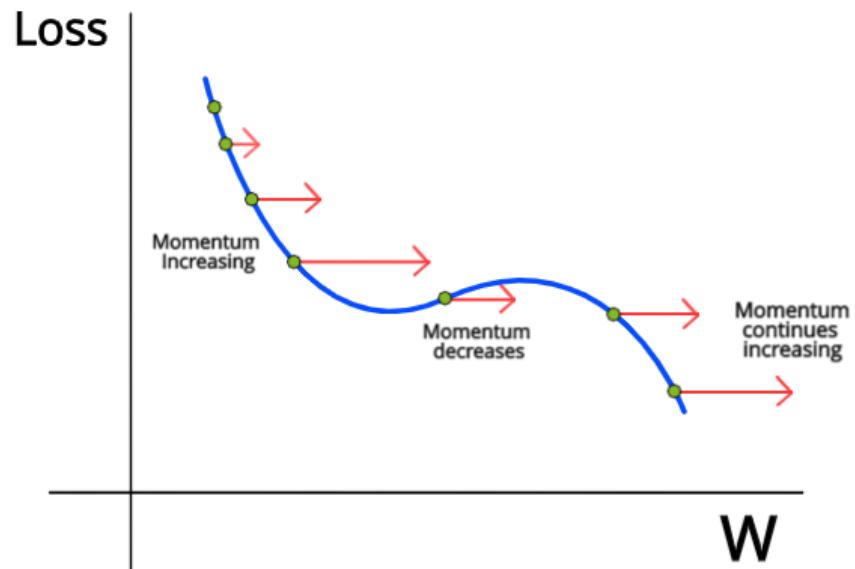
Change learning rate during learning:

- Make η large at the beginning of learning and decrease it later
- Simulated annealing: cleverly adjust learning rate using physics-inspired parameter called temperature

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

Momentum

Momentum: a way to escape small gradients



$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment learning rate weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

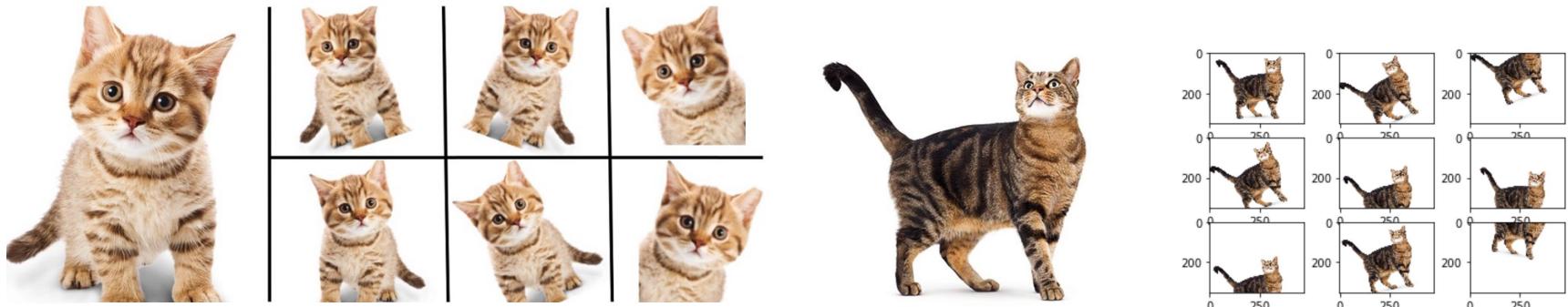
momentum factor weight increment, previous iteration

<https://deeplearningdemystified.com/article/fdl-4>

Dataset augmentation

Dataset augmentation

- E.g. adding noise to the input, applying spatial transformations to the input



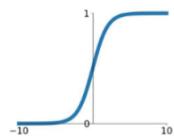
Activation function selection

In deep (many layers) networks, to avoid small gradient we can use activations functions such as ReLU, which have a large gradient for a wide range of values.

Activation Functions

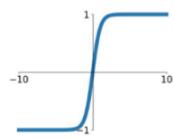
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



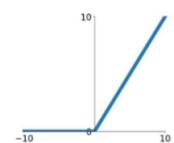
tanh

$$\tanh(x)$$



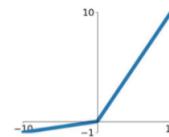
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

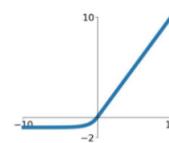


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

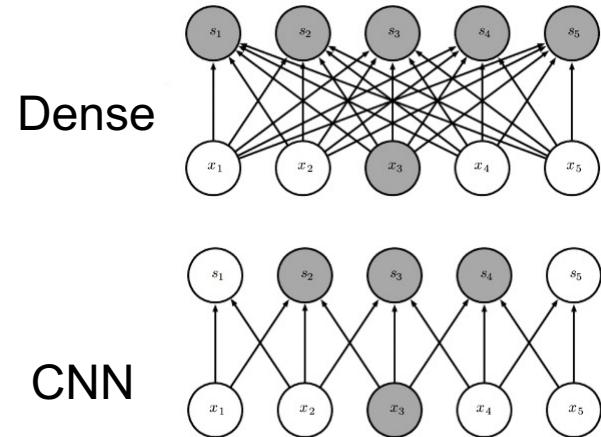


Recurrent neural networks: motivation

Consider time-series data \mathbf{x} (x_1 is the first time point, x_2 is the second time point...)

For long-time sequences, dense networks require a large number of parameters (all to all connections).

Convolutional networks share weights. This reduces the number of parameters, but can only capture local temporal relationships (within the width of the filter).



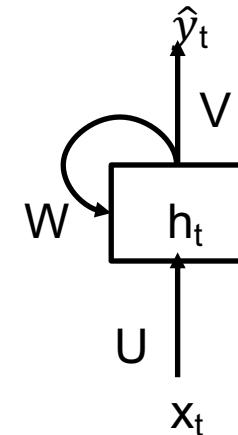
Introduction to RNNs

Recurrent neural networks or RNNs (Rumelhart et al., 1986) are a family of neural networks for processing sequential data.

Similar to CNNs, they use **parameter sharing (weight sharing)** – this makes it possible to extend and apply the model to examples of different forms (different lengths) and generalize across them.

A traditional fully connected feedforward network would have separate parameters for each input feature, so it would need to learn all of the rules of the sequence (e.g. language) separately at each position in the sequence.

A recurrent neural network shares the same weights across several time steps.



$$h_t = g(Wh_{t-1} + Ux_t) = g(z_t)$$

$$\hat{y}_t = g^*(Vh_t)$$

Training RNNs

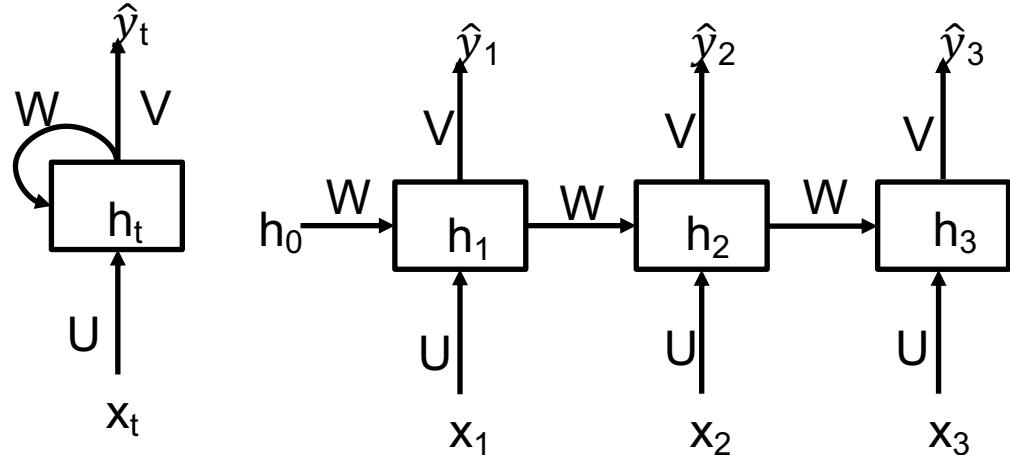
The total loss for a given sequence of x values paired with a sequence of y values is the sum of the losses over all the time steps.

We need to compute the gradient of the error with respect to all of the weights and then use any gradient-based techniques to train the network.

The back-propagation algorithm applied to the unrolled graph is called **back-propagation through time**.

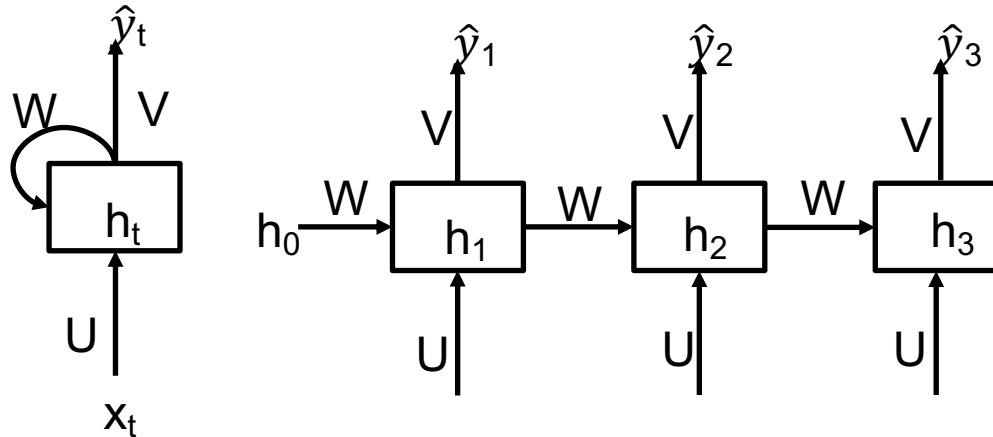
$$h_t = g(Wh_{t-1} + Ux_t) = g(z_t)$$

$$\hat{y}_t = g^*(Vh_t)$$



“Unfolded in time” RNN

Computing the gradient in an RNN using back-propagation through time



$$h_t = g(Wh_{t-1} + Ux_t) = g(z_t)$$

$$\hat{y}_t = g^*(Vh_t)$$

We assume g^* is identity function: $\hat{y}_t = Vh_t$.

$$L = \sum_t L_t$$

$$L_3 = \frac{1}{2}(y_3 - \hat{y}_3)^2$$

$$\frac{\partial L_3}{\partial V} = \frac{\partial L_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V} = -(y_3 - \hat{y}_3)h_3$$

$$\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial W} = -(y_3 - \hat{y}_3)V \frac{\partial g(z_3)}{\partial W} = -(y_3 - \hat{y}_3)V \frac{\partial g(z_3)}{\partial z_3} (h_2 + W \frac{\partial h_2}{\partial W})$$

$$\frac{\partial L_3}{\partial U} = \frac{\partial L_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial U} = -(y_3 - \hat{y}_3)V \frac{\partial g(z_3)}{\partial U} = -(y_3 - \hat{y}_3)V \frac{\partial g(z_3)}{\partial z_3} (W \frac{\partial h_2}{\partial U} + x_3)$$

Product rule:

$$\frac{d}{dx}[f(x)g(x)] = f(x)g'(x) + f'(x)g(x)$$

The Challenge of Long-Term Dependencies: vanishing and exploding gradient

The basic problem is that gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization), making it hard to learn long-term dependencies.

For simplicity, assume that activation functions are linear and that we are unfolding RNN over T steps, then we have:

$$\frac{\partial h_T}{\partial h_1} = \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \frac{\partial h_{T-2}}{\partial h_{T-3}} \cdots \frac{\partial h_2}{\partial h_1} = W^{T-1}$$

- So the gradient can easily explode or vanish if W is not very close to 1.
- Good resources for more details on math behind vanishing and exploding gradient:
 - <https://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-gradient-problems/>
 - http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf

Solutions for vanishing/exploding gradient problem: Gated RNNs

Gated RNNs:

- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

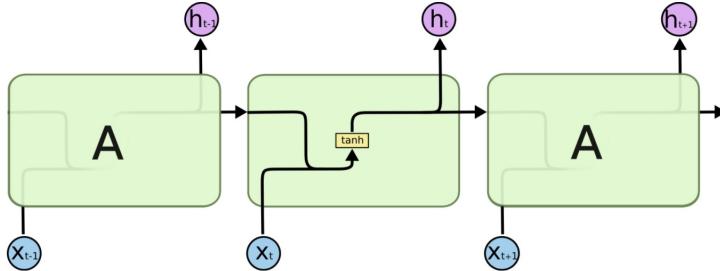
Gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode.

Example: If a sequence is made of sub-sequences and we want a leaky unit to accumulate evidence inside each sub-subsequence, we need a mechanism to forget the old state by setting it to zero.

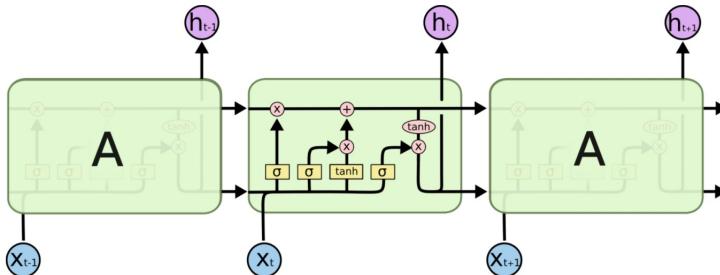
Instead of manually deciding when to clear the state, we want the neural network to learn to decide when to do it. This is what gated RNNs do.

This is related to my own research, see here for more resources

Long Short-Term Memory (LSTM)



The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Neural Network References

For additional examples and references, check out the following:

- The Data Science Blog - <https://ujjwalkarn.me/blog/>
- Deep Learning Book - <http://www.deeplearningbook.org>
 - See also video lectures on the same page

Video lectures:

- <https://www.youtube.com/watch?v=Xogn6veSyxA> (Ian Goodfellow)
- <https://www.youtube.com/watch?v=H-HVZJ7kGI0&t=1057s> (Ava Soleimany)

Practical perspective:

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition
- See here for the code:
 - <https://github.com/ageron/handson-ml2>
 - https://github.com/ageron/handson-ml2/blob/master/14_deep_computer_vision_with_cnns.ipynb

More Resources about RNNs

<http://www.deeplearningbook.org/contents/rnn.html>

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

<https://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-gradient-problems/>

Video lectures

- <https://www.youtube.com/watch?v=ZVN14xYm7JA&feature=youtu.be>
- <https://www.youtube.com/watch?v=o2QuErsWp6k&t=1s>