WPI

Artificial Intelligence
CS 534

Week 8

Combining Classifiers

- The basic philosophy behind the combination of different classifiers lies in the fact that even the “best” classifier fails in some patterns that other classifiers may classify correctly
- Combining classifiers aims at exploiting this complementary information residing in the various classifiers

Combining Classifiers

- The basic philosophy behind the combination of different classifiers lies in the fact that even the "best" classifier fails in some patterns that other classifiers may classify correctly
- Combining classifiers aims at exploiting this complementary information residing in the various classifiers

Idea: one designs different optimal classifiers and then combines the results with a specific rule

- Assume that each of the, say, L designed classifiers provides at its output the posterior probabilities:

$$P(\omega_i \mid \boldsymbol{x}),\ i = 1, 2, ..., M$$

Basic Rules

- Product Rule: Assign $\boldsymbol{x}$ to the class ω_i:

$$i = \arg\max_k \prod_{j=1}^{L} P_j\left(\omega_k \mid \boldsymbol{x}\right)$$

where $P_j\left(\omega_k \mid \boldsymbol{x}\right)$ is the respective posterior probability of the j^{th} classifier.

Basic Rules

- Product Rule: Assign $\boldsymbol{x}$ to the class ω_i:

$$i = \arg\max_k \prod_{j=1}^{L} P_j\left(\omega_k \mid \boldsymbol{x}\right)$$

where $P_j\left(\omega_k \mid \boldsymbol{x}\right)$ is the respective posterior probability of the j^{th} classifier

- Sum Rule: Assign $\boldsymbol{x}$ to the class ω_i:

$$i = \arg\max_k \sum_{j=1}^{L} P_j\left(\omega_k \mid \boldsymbol{x}\right)$$

Majority Voting Rule

Majority Voting Rule: Assign $\boldsymbol{x}$ to the class for which there is a consensus or when at least ℓ_c of the classifiers agree on the class label of $\boldsymbol{x}$ where:

$$\ell_c = \begin{cases} \frac{L}{2}+1, & L \text{ is even} \\ \frac{L+1}{2}, & L \text{ is odd} \end{cases}$$

otherwise the decision is rejection, that is no decision is taken.

Majority Voting Rule

Majority Voting Rule: Assign $\boldsymbol{x}$ to the class for which there is a consensus or when at least ℓ_c of the classifiers agree on the class label of $\boldsymbol{x}$ where:

$$\ell_c = \begin{cases} \frac{L}{2}+1, & L \text{ is even} \\ \frac{L+1}{2}, & L \text{ is odd} \end{cases}$$

otherwise the decision is rejection, that is no decision is taken.

Thus, correct decision is made if the majority of the classifiers agree on the correct label, and wrong decision if the majority agrees in the wrong label

Dependent or not Dependent classifiers?

- Although there are not general theoretical results, experimental evidence has shown that the more independent in their decision the classifiers are, the higher the expectation should be for obtaining improved results after combination

- However, there is no guarantee that combining classifiers results in better performance compared to the "best" one among the classifiers

Towards Independence: A number of Scenarios

Train the individual classifiers using different training data points

To this end, we can choose among a number of possibilities:

- Bootstrapping: This is a popular technique to combine unstable classifiers such as decision trees (Bagging belongs to this category of combination)
- Stacking: Train the combiner with data points that have been excluded from the set used to train the individual classifiers
- Use different subspaces to train individual classifiers: According to the method, each individual classifier operates in a different feature subspace. That is, use different features for each classifier

Remarks

- The majority voting and the summation schemes rank among the most popular combination schemes
- Training individual classifiers in different subspaces seems to lead to substantially better improvements compared to classifiers operating in the same subspace
- Besides the above three rules, other alternatives are also possible, such as to use the median value of the outputs of individual classifiers

The Boosting Approach

The origins: Is it possible a weak learning algorithm (one that performs slightly better than a random guessing) to be boosted into a strong algorithm? (Villiant 1984)

—The procedure to achieve it:

- Adopt a weak classifier known as the base classifier.
- Employing the *base* classifier, design a series of classifiers, in a hierarchical fashion, each time employing a different weighting of the training samples. Emphasis in the weighting is given on the hardest samples, *i.e.*, the ones that keep "failing".
- Combine the hierarchically designed classifiers by a weighted average procedure

Decision Tree —>Random Forrest

Slides are adapted based on:

Foundations and Trends® in
Computer Graphics and Vision
Vol. 7, Nos. 2–3 (2011) 81–227

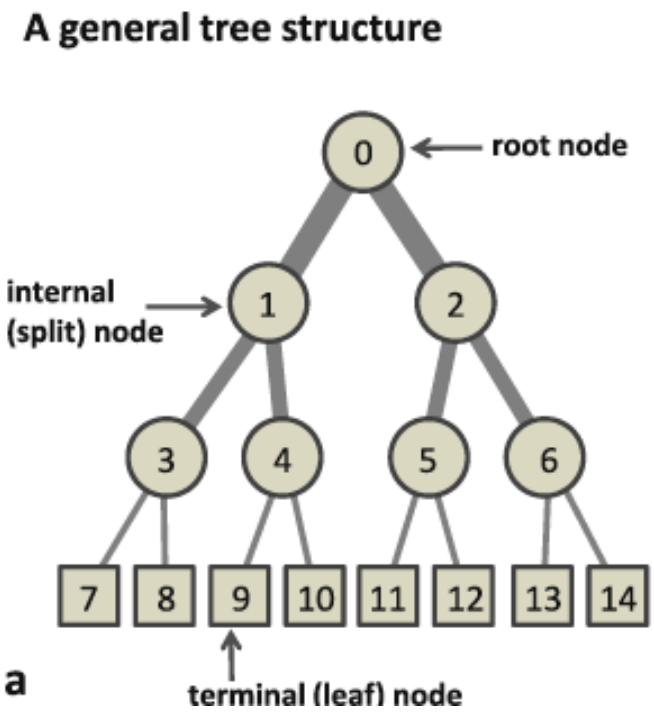
DOI: 10.1561/0600000035

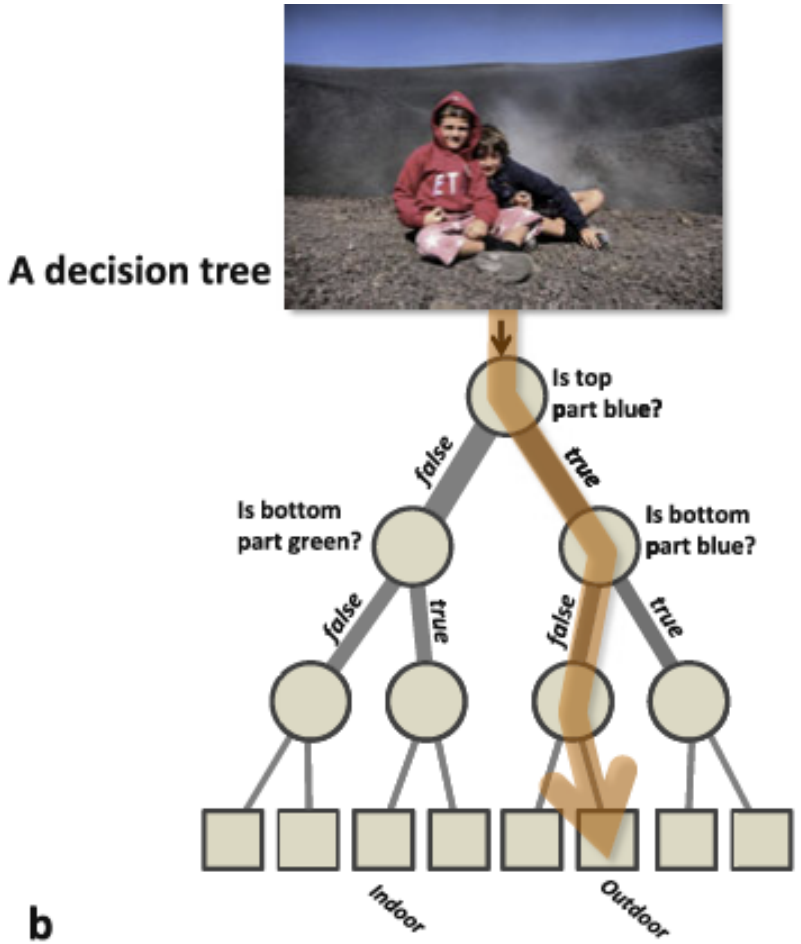
now
the essence of knowledge

Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning

By Antonio Criminisi, Jamie Shotton,
and Ender Konukoglu

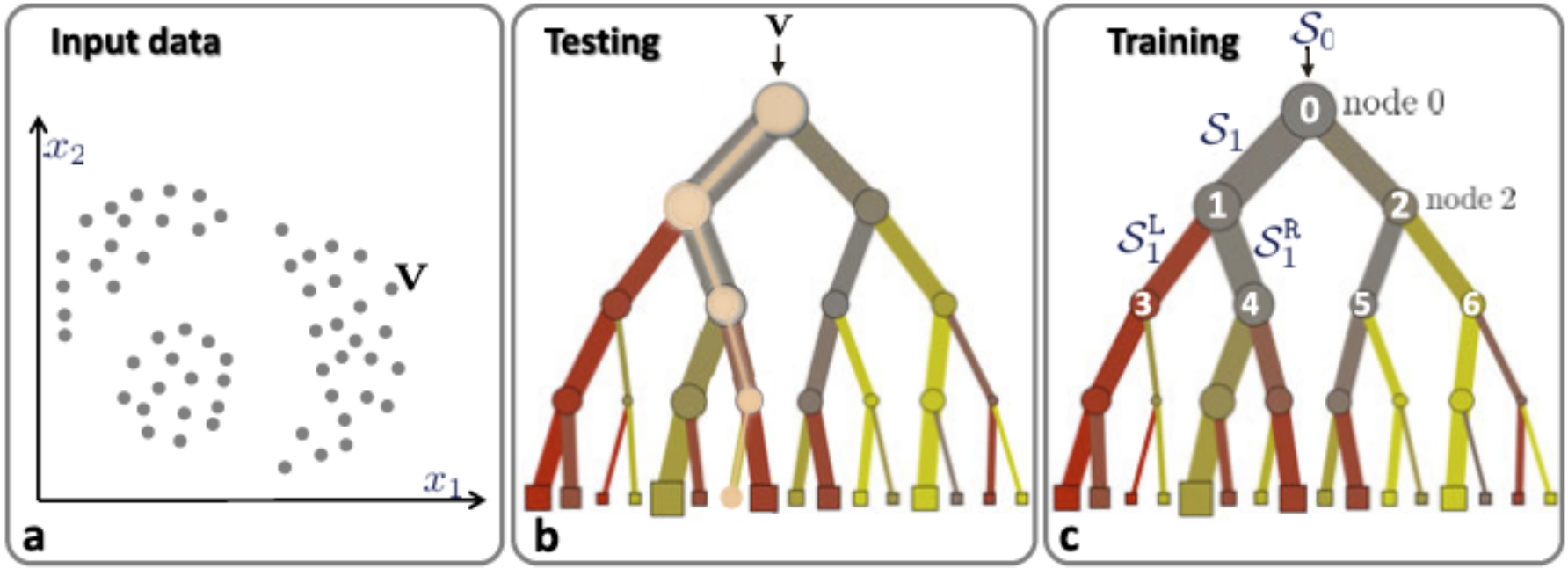
Decision tree





- No loops. Internal nodes circles. Terminal nodes with squares
- Each internal node stores a split (or test) function to be applied to the incoming data
- Each leaf stores the final answer (predictor)
- Example: figure out whether a photo represents an indoor or outdoor scene

Basic notation



(a) Input data are represented as a collection of points in the *d*-dimensional space defined by their feature responses (2D in this example)

(b) During testing, a split (internal) node applies a test to the input data *v* and sends it to the appropriate child. The process is repeated until a leaf (terminal) node is reached (beige path)

(c) Training a decision tree involves sending the entire training set S_0 into the tree and optimizing the parameters of the split nodes so as to optimize a chosen energy function

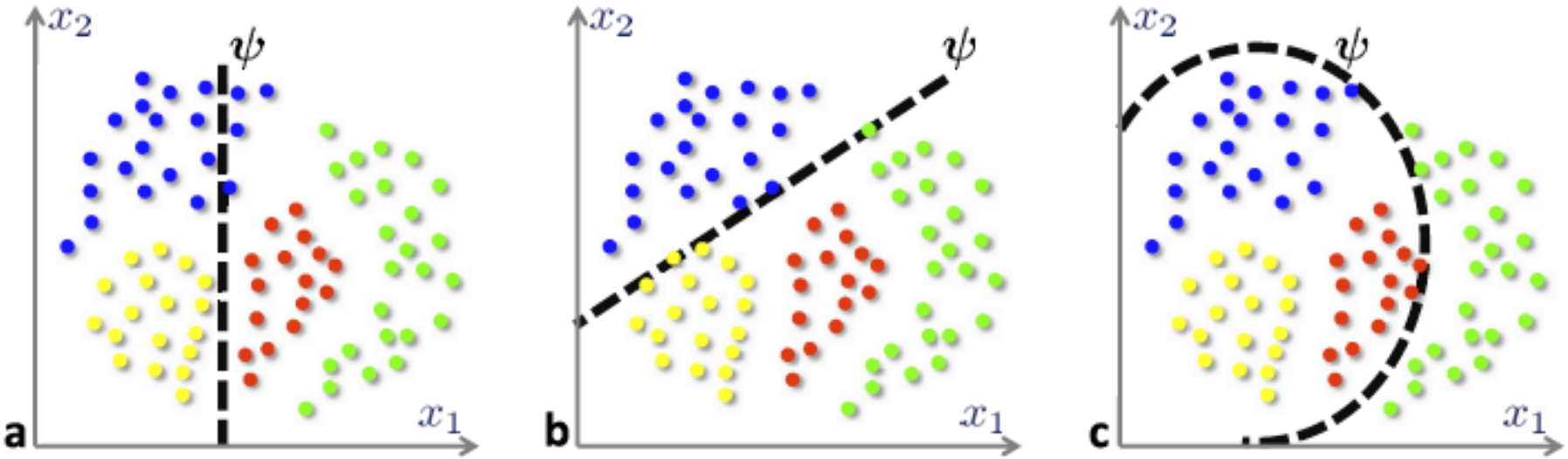
Learning tree structure

- The structure of the tree depends on how and when we decide to stop growing various branches of the tree

- Diverse stopping criteria can be applied. For example it is common to stop the tree when a maximum number of levels D has been reached. Tree growing may also be stopped when a node contains too few training points

- At the end of the training phase we obtain:
  (i) the (greedily) optimum weak learners (split functions) associated with each node
  (ii) a learned tree structure,
  (iii) a different set of training points at each leaf

Weak learner



Example: colors of each data point (*circles*) indicate different classes

(a) Axis-aligned hyperplane weak learner

(b) General oriented hyperplane

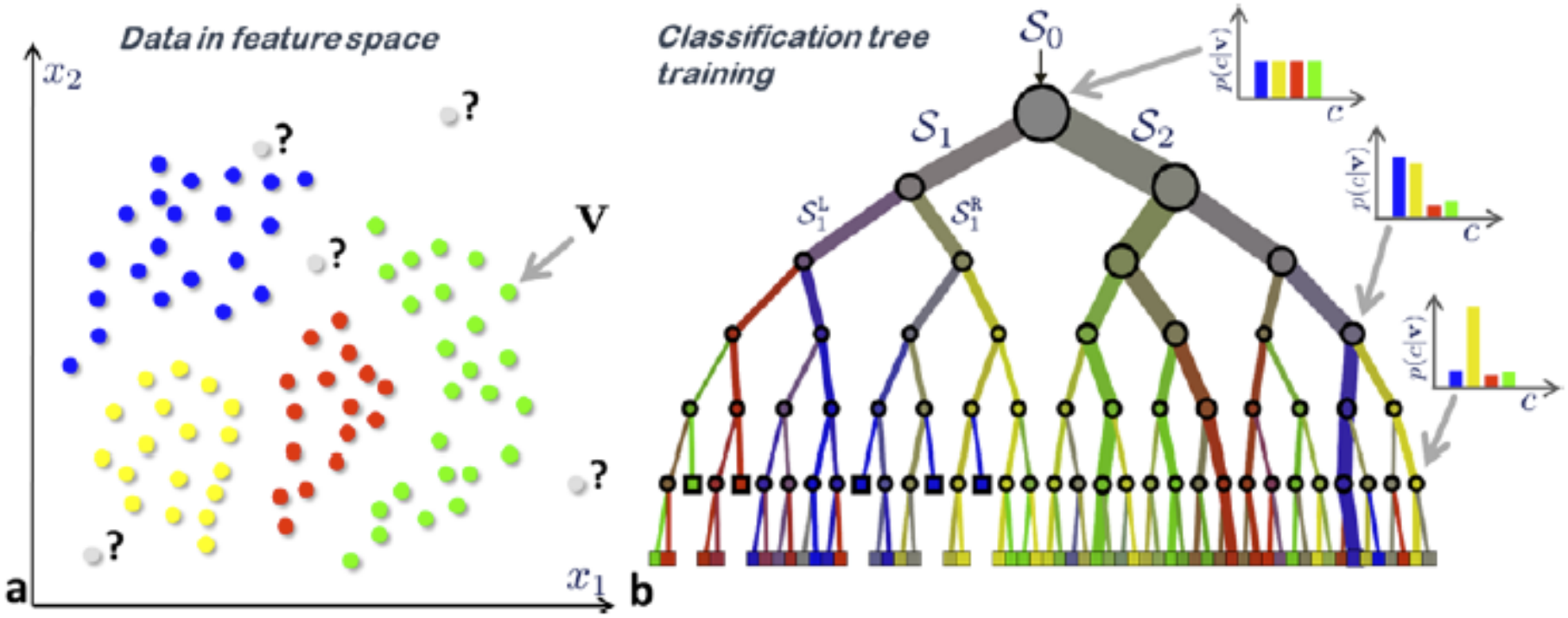
(c) Quadratic surface

In general, $\psi(v)$ has a dimensionality ≤ 2, irrespective of number of features

Random Forests: Basics and historical notes

- Random forests are an ensemble learning method for classification / regression
- Idea: generalization of decision trees
- Algorithm for inducing a random forest was developed by Leo Breiman and Adele Cutler in 2001
- "Random Forests" is their trademark
- The term came from random decision forests that was first proposed by Tin Kam Ho of Bell Labs in 1995
- Algorithm combines Breiman's "bagging" idea and the random selection of features, introduced by Ho, Amit and Geman for constructing decision trees with controlled variation.
- In addition, the idea of randomized node optimization, where the decision at each node is selected by a randomized procedure introduced by Dietterich
- More recently several major advances in this area have come from Microsoft Research

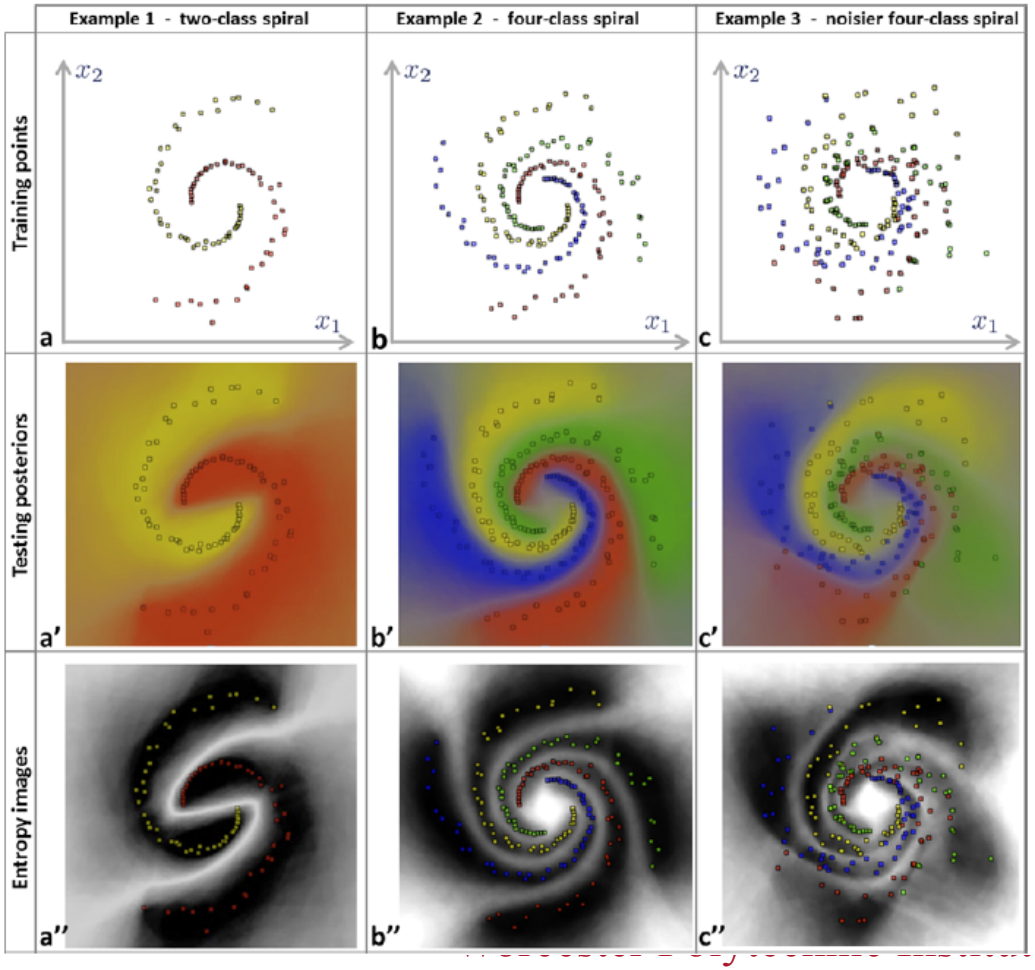
Classification training data and tree training



- *Gray circles* indicate unlabeled, previously unseen test data
- The edge thickness is proportional to the amount of training data going through it
- Edge colors are a mix of the colors of the four classes, weighted in proportion to the associated class probabilities
- Note the gray-ish color of the root node and the more distinct colors of the leaves

Multiple classification

- One major advantage of decision forests over *e.g.* support vector machines and boosting is that the **same** classification model can handle both binary and multi-class problems
- Shown are both two- and four-class examples, and different levels of noise in the training data

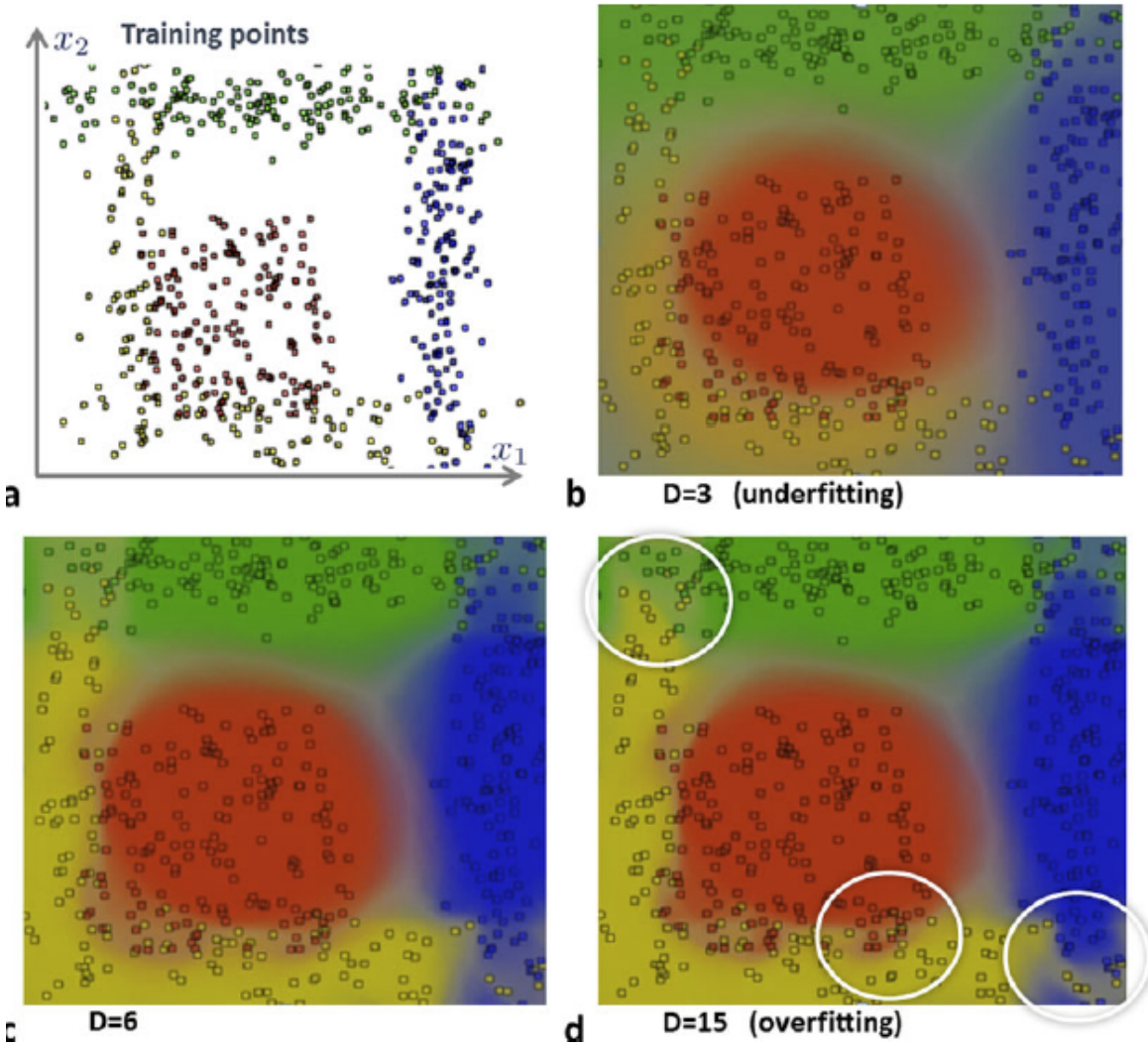


The effect of tree depth

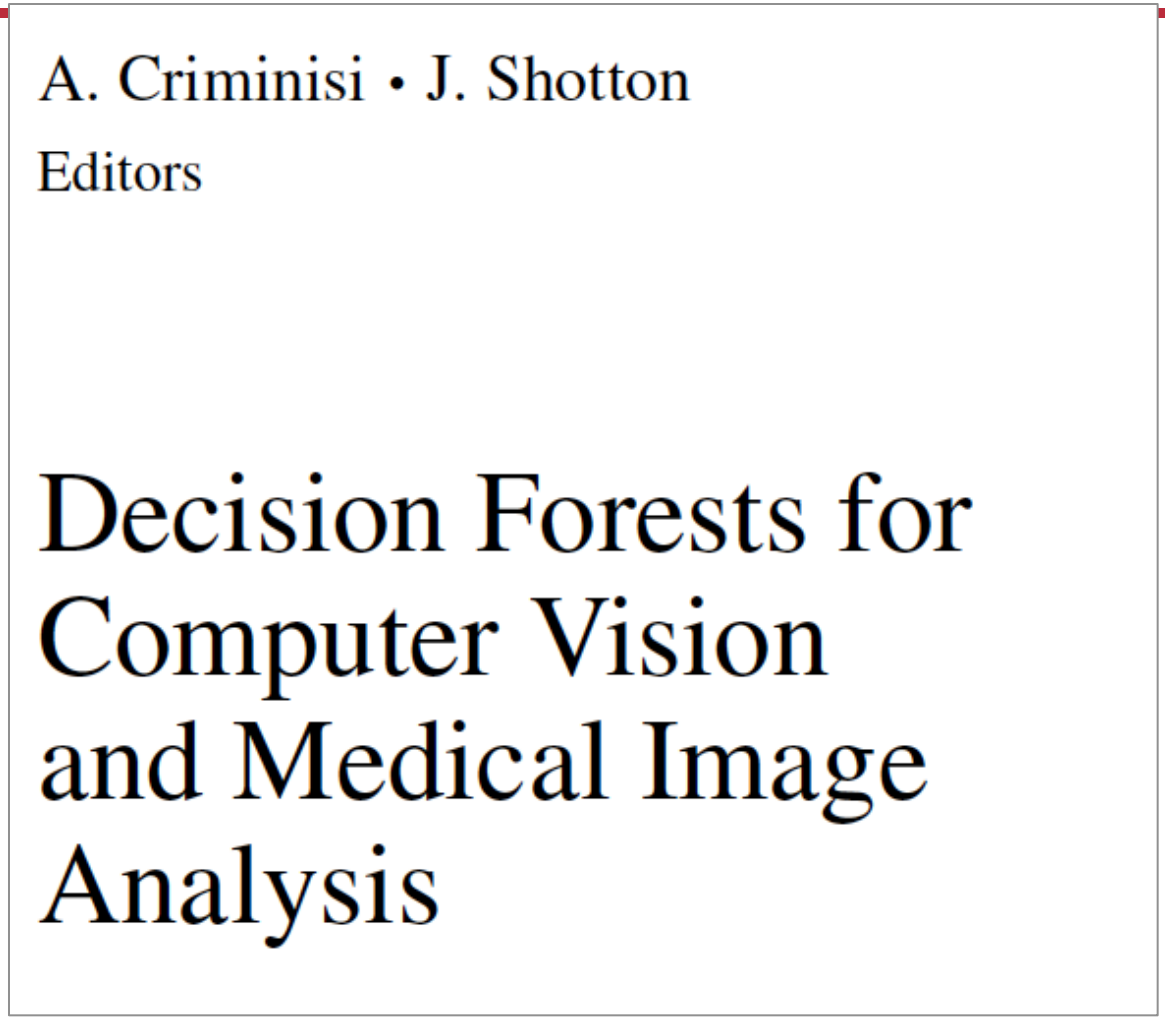
- A four-class problem with both mixing of training labels and large gaps
- The tree depth is a crucial parameter in avoiding under- or overfitting

(**a**) Training points

(**b**, **c**, **d**): Different tree depths D



More information



Semi-Supervised Learning

Supervised vs. semi-supervised learning

Consider an easier 2-class problem:

Supervised vs. semi-supervised learning

Consider an easier 2-class problem:

Supervised learning

Positive training set:

Negative training set:

Supervised vs. semi-supervised learning


Consider an easier 2-class problem:

Supervised learning

Positive training set:

Negative training set:

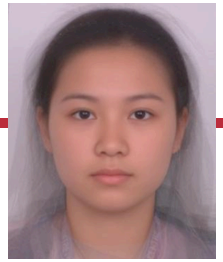
Semi-supervised learning

Same labeled set

Formalisms:

- Logistic regression
- Support Vector Machine (SVM)
- Random Forrest (RF)

Supervised vs. semi-supervised learning




Consider an easier 2-class problem:

Supervised learning

Positive training set:

Negative training set:

Formalisms:

- Logistic regression
- Support Vector Machine (SVM)
- Random Forrest (RF)

Semi-supervised learning

Same labeled set

+

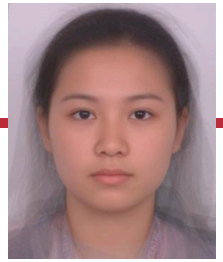
(c) AMP Lab, Cornell

Large unlabeled set

Formalisms:

- Low-density separation using SVM
- Self-Learning using RF

Supervised vs. semi-supervised learning




Consider an easier 2-class problem:

Supervised learning

Positive training set:

Negative training set:

Semi-supervised learning

Same labeled set

+

(c) AMP Lab, Cornell

Large unlabeled set

Formalisms:

- Logistic regression
- Support Vector Machine (SVM)
- Random Forrest (RF)

Semi-Supervised Learning: Basics

- In addition to unlabeled data, the algorithm is provided with some supervision information
  - but not necessarily for all examples
- Data set $X = (x_i)_{i \in [n]}$ is thus split into two subsets:
  - Labeled set $X_l := (x_1, \ldots, x_l)$ $Y_l := (y_1, \ldots, y_l)$
  - Unlabeled set $X_u := (x_{l+1}, \ldots, x_{l+u})$
-
  -
  -
  -
-

Semi-Supervised Learning: Basics

- In addition to unlabeled data, the algorithm is provided with some supervision information
  - but not necessarily for all examples
- Data set $X = (x_i)_{i \in [n]}$ is thus split into two subsets:
  - Labeled set $X_l := (x_1, \ldots, x_l)$ $Y_l := (y_1, \ldots, y_l)$
  - Unlabeled set $X_u := (x_{l+1}, \ldots, x_{l+u})$
- A problem related to SSL was introduced by Vapnik already several decades ago: *transductive learning*
  - In this setting, one is also given a (labeled) training set and an (unlabeled) test set
  - The idea of transduction is to perform predictions **only** for the test inductive learning points
  - This is in contrast to *inductive learning*, where the goal is to output a prediction function which is defined on the entire space
- Many SSL algorithms are transductive learning algorithms

When Can SSL work?

- One of the most common criteria: Smoothness assumption

- Supervised learning: If two points x_1; x_2 are close, then so should be the corresponding outputs y_1; y_2
  - Strictly speaking, this assumption only refers to continuity rather than smoothness; however, the term smoothness is commonly used, possibly because in regression estimation y is often modeled in practice as a smooth function of x

-
  -
  -

When Can SSL work?

- One of the most common criteria: Smoothness assumption

- Supervised learning: If two points x_1; x_2 are close, then so should be the corresponding outputs y_1; y_2
  - Strictly speaking, this assumption only refers to continuity rather than smoothness; however, the term smoothness is commonly used, possibly because in regression estimation y is often modeled in practice as a smooth function of x

- SSL: If two points x_1; x_2 in a **high-density region** are close, then so should be the corresponding outputs y_1; y_2
  - Note that by transitivity, this assumption implies that if two points are linked by a path of high density (e.g., if they belong to the same cluster), then their outputs are likely to be close
  - If, on the other hand, they are separated by a **low-density region**, then their outputs need not be close

The cluster assumption

- Suppose we knew that the points of each class tended to form a cluster
- Then the unlabeled data could aid in finding the boundary of each cluster more accurately: one could run a clustering algorithm and use the labeled points to assign a class to each cluster
-
  -
  -

Classes of SSL algorithms

1. Self-learning algorithms

2. Generative models
   - Mixture models
   - Data-dependent priors

3. Low-density separation (LDS)

4. Graph-based methods

5. Change of representation methods

A basic approach: Self-learning algorithm

- Also known as self-training, self-labeling, or decision-directed learning: probably the earliest idea about using unlabeled data in
-
  -
  -

Self-learning algorithms

- Also known as self-training, self-labeling, or decision-directed learning: probably the earliest idea about using unlabeled data in

- Idea: a wrapper-algorithm that repeatedly uses a supervised learning method
  - It starts by training on the labeled data only. In each step a part of the unlabeled points is labeled according to the current decision function
  - then the supervised method is retrained using its own predictions as additional labeled points

Self-learning algorithms

- Also known as self-training, self-labeling, or decision-directed learning: probably the earliest idea about using unlabeled data in

- Idea: a wrapper-algorithm that repeatedly uses a supervised learning method
  - It starts by training on the labeled data only. In each step a part of the unlabeled points is labeled according to the current decision function
  - then the supervised method is retrained using its own predictions as additional labeled points

- An unsatisfactory aspect of self-learning is that the effect of the wrapper depends on the supervised method used inside it
  - Sometimes it is unclear what the self-learning is really doing, and which assumption it corresponds to

More information

Semi-Supervised Learning

Olivier Chapelle
Bernhard Schölkopf
Alexander Zien

LDS approach

- Idea: place boundaries in regions where there are few data points (labeled or unlabeled)
- Transductive support vector machine (TSVM) is most commonly used algorithm
- While SVM (SL) seek a decision boundary with maximal margin over the labeled data, the goal of TSVM is a labeling of the unlabeled data such that the decision boundary has maximal margin over all of the data

Semi-Supervised Classification by Low Density Separation

Olivier Chapelle, Alexander Zien
Max Planck Institute for Biological Cybernetics
72076 Tübingen, Germany

Semi-Supervised Classification by Low Density Separation

Olivier Chapelle, Alexander Zien
Max Planck Institute for Biological Cybernetics
72076 Tübingen, Germany

Different types of SSL algorithms proposed

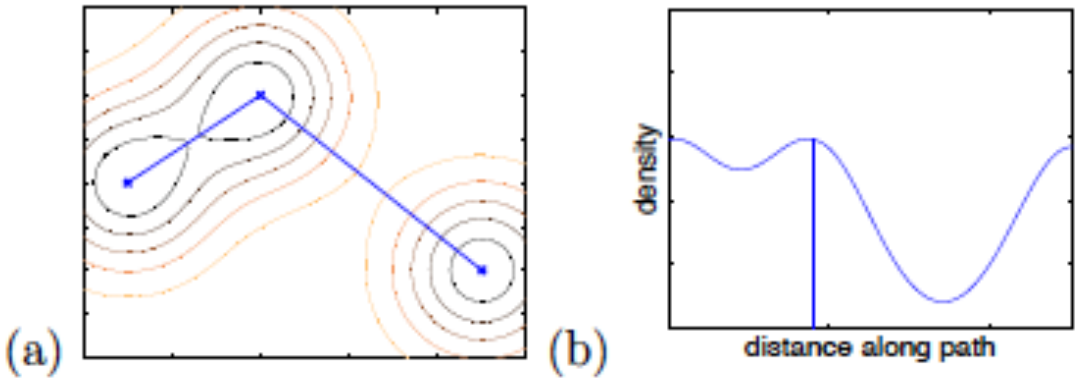
1. Deriving graph-based distances that emphasize low density regions between clusters, followed by training a standard SVM

2. Optimizing the Transductive SVM objective function, which places the decision boundary in low density regions, by gradient descent

3. combining the first two algorithms to make maximum use of the cluster assumption

LDS idea

- If two points are in the same cluster, it means there exists a continuous connecting curve that only goes through regions of high density
- If two points are in different clusters, every such curve has to traverse a density valley
- We can thus define the similarity of two points by maximizing over all continuous connecting curves the minimum density along the connection, but this is hard to compute
- Two observations allow to approximate the above similarity with paths on a graph:



a) An optimal connecting curve can be well approximated by conjoining short line segments that directly connect points

b) The minimum density is assumed at the middle of a line segment, and dominated by the closest points.

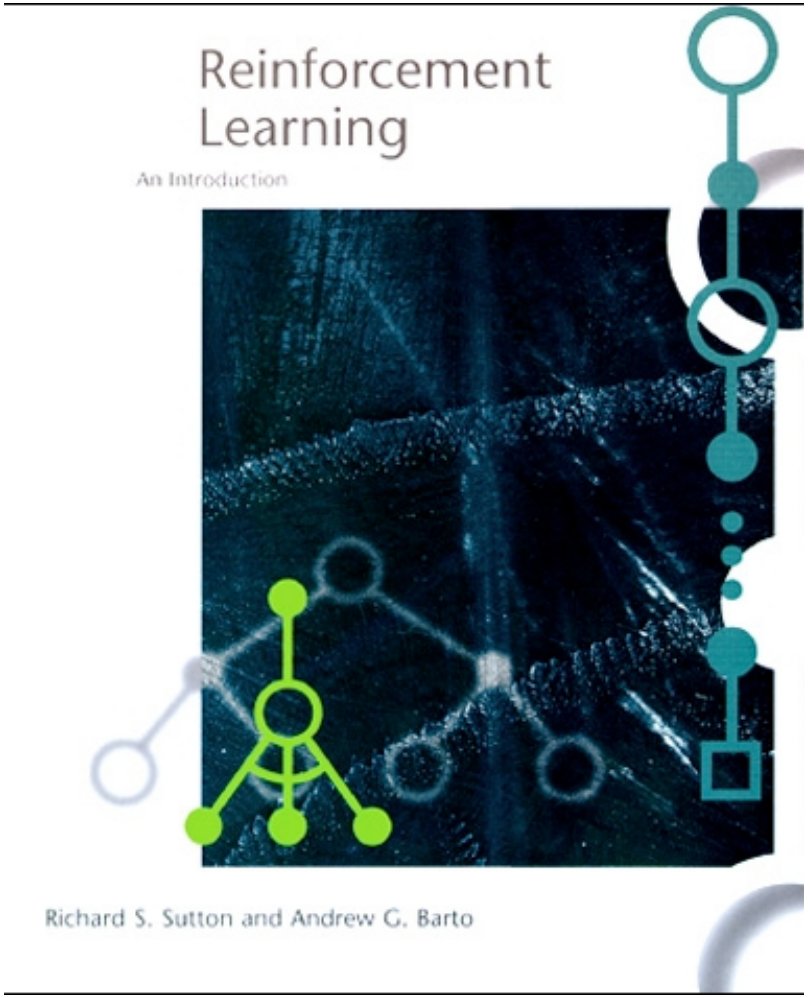
A basic INTRO to Reinforcement Learning

Reinforcement Learning

- Based on "Reinforcement Learning – An Introduction" by Richard Sutton and Andrew Barto

- Slides are based on:
  - the course material provided by the authors
  - Peter Bodík

- Online draft of the second editio is available for free!

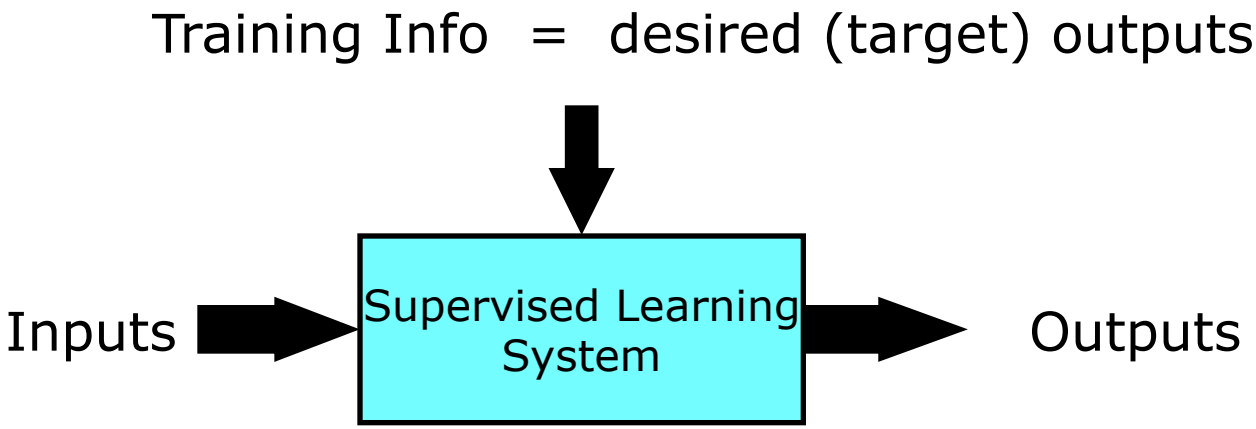


http://incompleteideas.net/sutton/book/the-book-2nd.html

What is Reinforcement Learning?

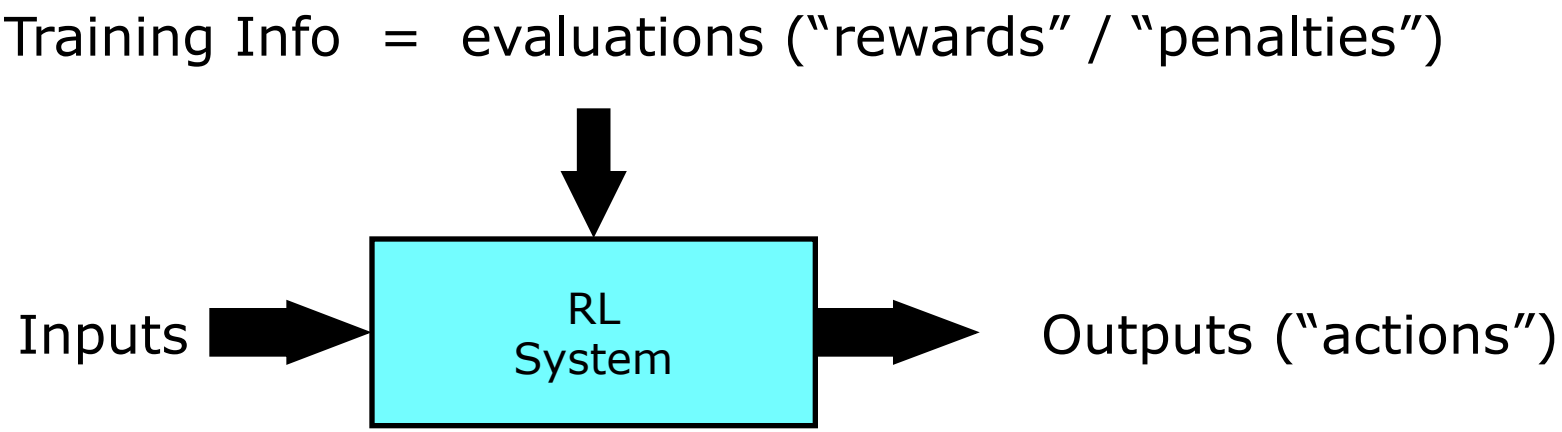
- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions —so as to maximize a numerical reward signal

Supervised Learning



Error = (target output – actual output)

Reinforcement Learning



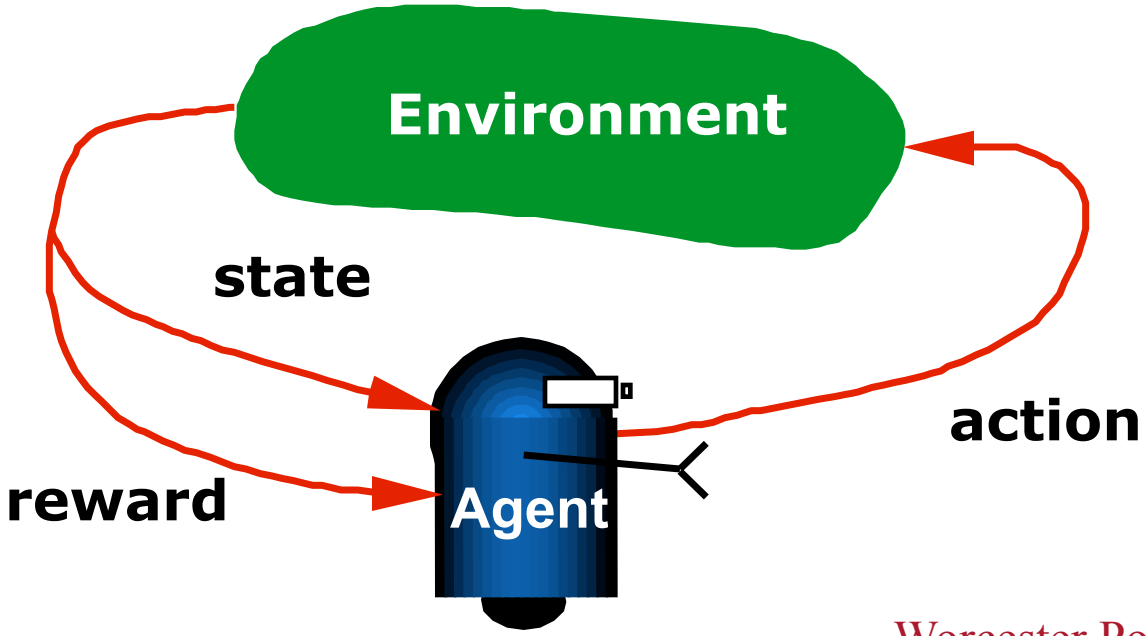
Objective: get as much reward as possible

Key Features of RL

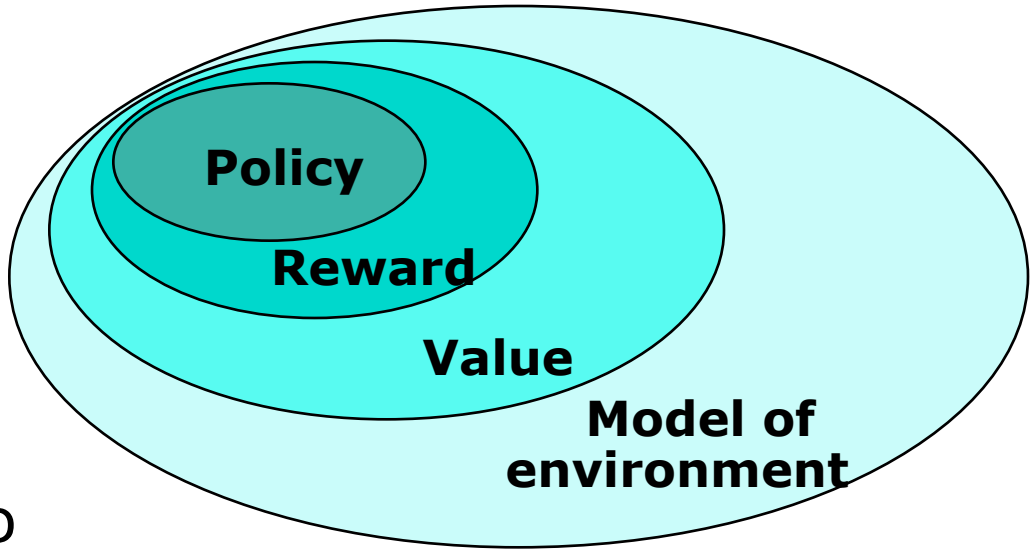
- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward (sacrifice short-term gains for greater long-term gains)
- The need to ***explore*** and ***exploit***
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to ***affect*** the environment
- Environment is stochastic and uncertain

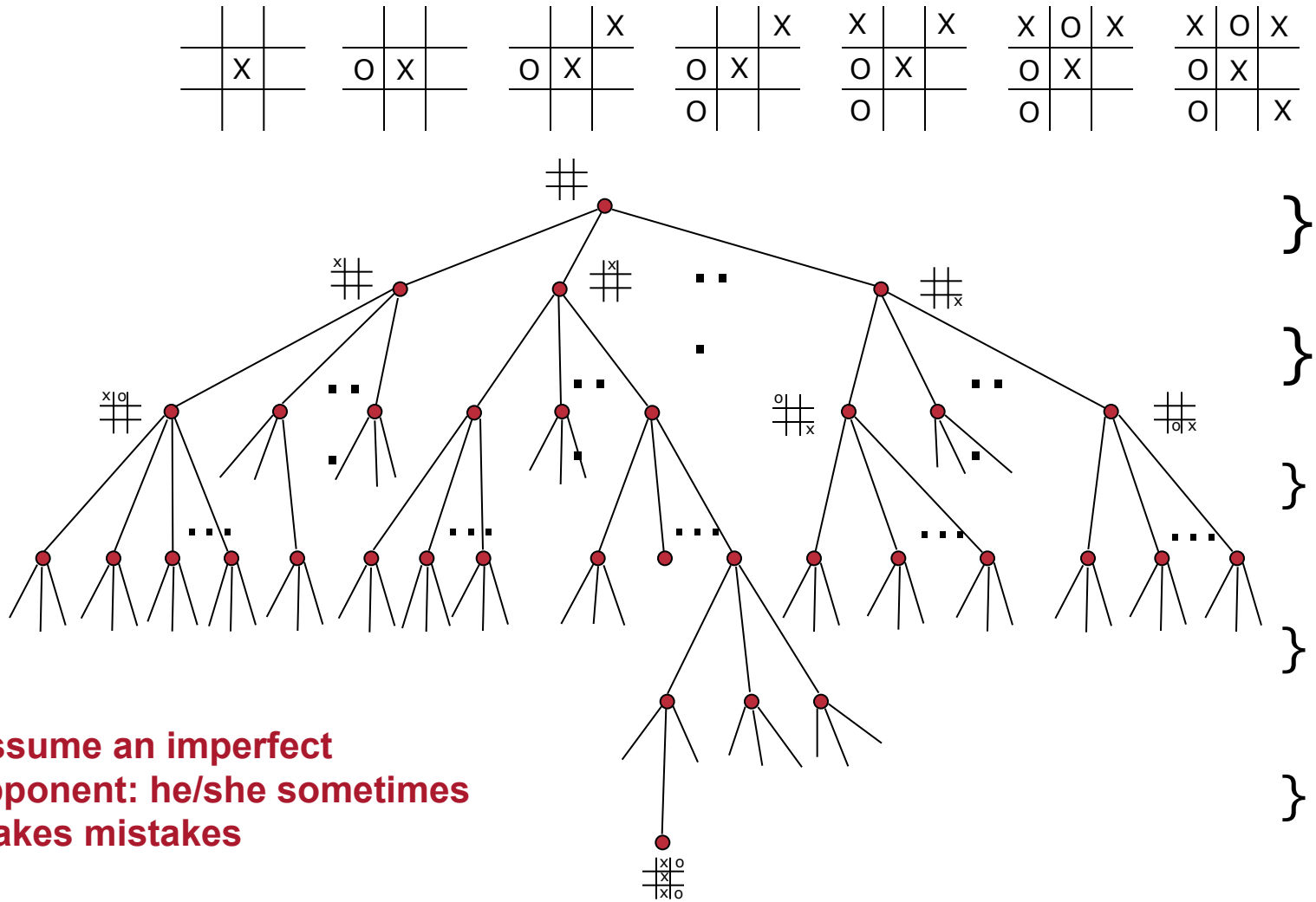


Elements of RL



- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it *predicts* reward
- **Model**: what follows what

An Extended Example: Tic-Tac-Toe



} x's move

} o's move

} x's move

} o's move

} x's move

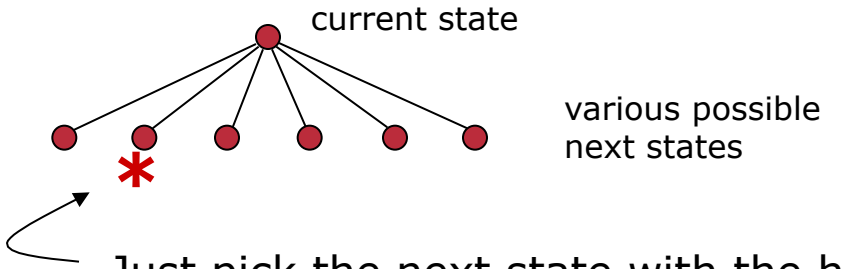
Assume an imperfect opponent: he/she sometimes makes mistakes

An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

State	$V(s)$ – estimated probability of winning	
	.5	?
	.5	?
⋮	⋮	
	1	win
⋮	⋮	
	0	loss
⋮	⋮	
	0	draw

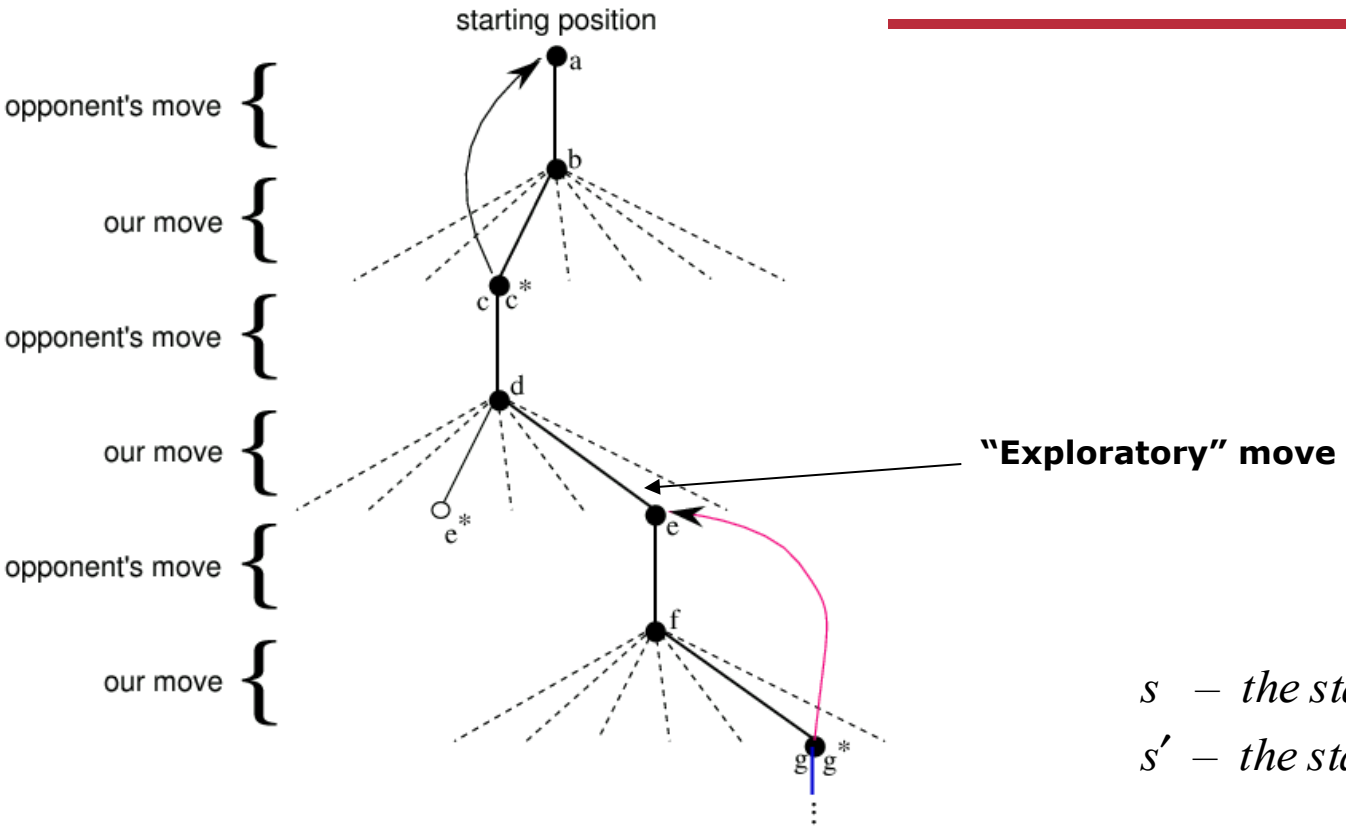
2. Now play lots of games. To pick our moves, look ahead one step:



Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random;an ***exploratory move***.

RL Learning Rule for Tic-Tac-Toe



s – *the state before our greedy move*

s' – *the state after our greedy move*

We increment each $V(s)$ *toward* $V(s')$ – *a* ***backup*** *:*

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)]$$

a small positive fraction, e.g., $\alpha = .1$

the ***step - size parameter***

How can we improve this T.T.T. player?

- Take advantage of symmetries
  - representation/generalization
  - How might this backfire?
- Do we need “random” moves? Why?
  - Do we always need a full 10%?
- Can we learn from “random” moves?
- Can we learn offline?
  - Pre-training from self play?
  - Using learned models of opponent?
- …

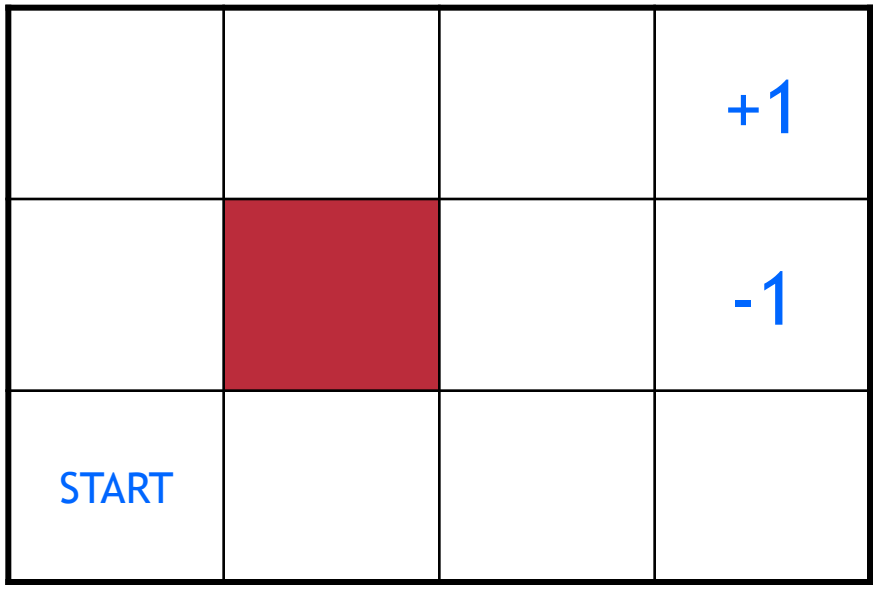
How is Tic-Tac-Toe Too Easy?

- Finite, small number of states
- One-step look-ahead is always possible
- State completely observable
- …

Some Notable RL Applications

- **TD-Gammon**: Tesauro
  - world's best backgammon program
- **Elevator Control**: Crites & Barto
  - high performance down-peak elevator controller
- **Dynamic Channel Assignment**: Singh & Bertsekas, Nie & Haykin
  - high performance assignment of radio channels to mobile telephone calls
- **Stanford Autonomous Helicopter**: Abbeel, Coates, and Ng

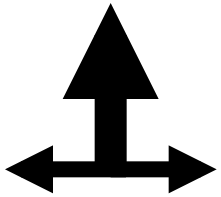
Robot in a room



actions: UP, DOWN, LEFT, RIGHT

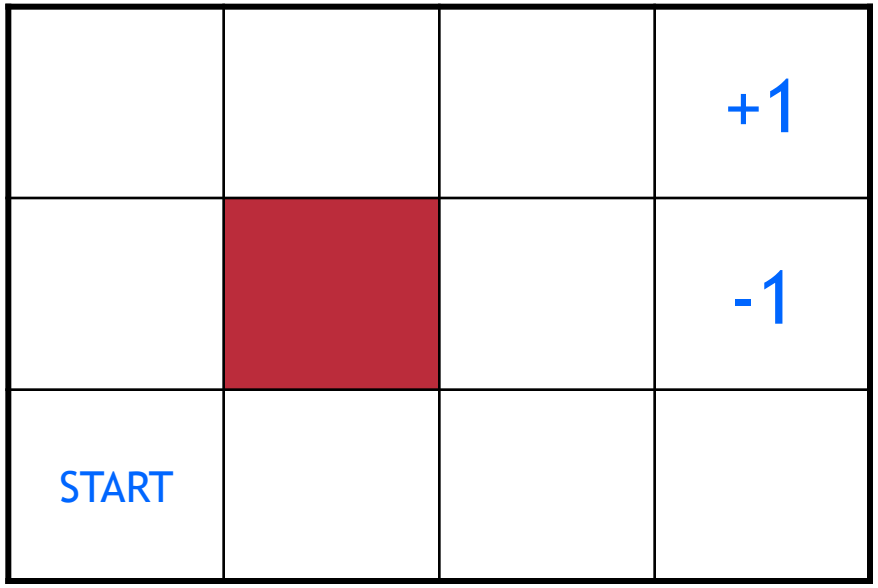
UP

80% move UP
10% move LEFT
10% move RIGHT



- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step
- What's the strategy to achieve max reward?
- What if the actions were deterministic?

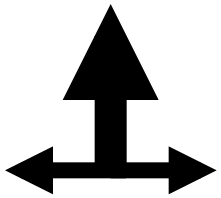
Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

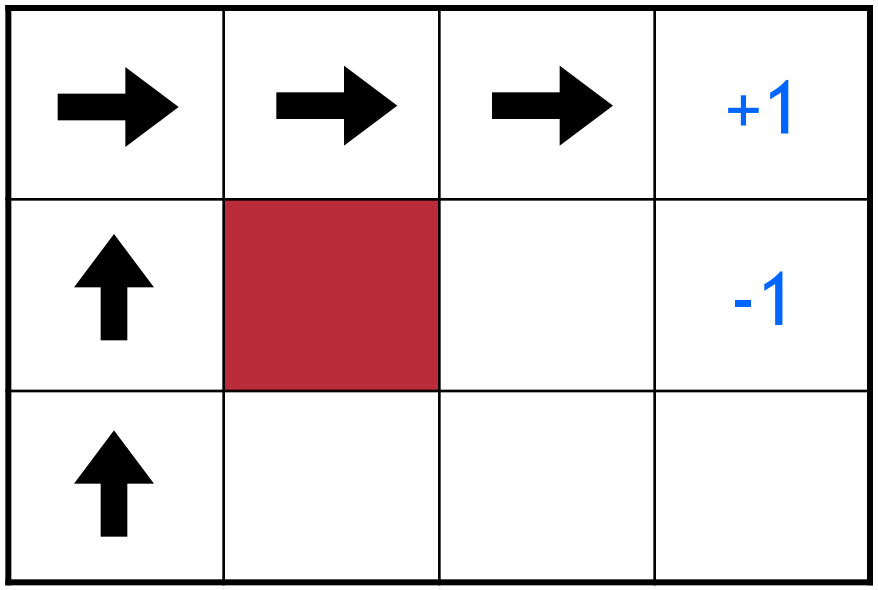


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

- what is the solution?

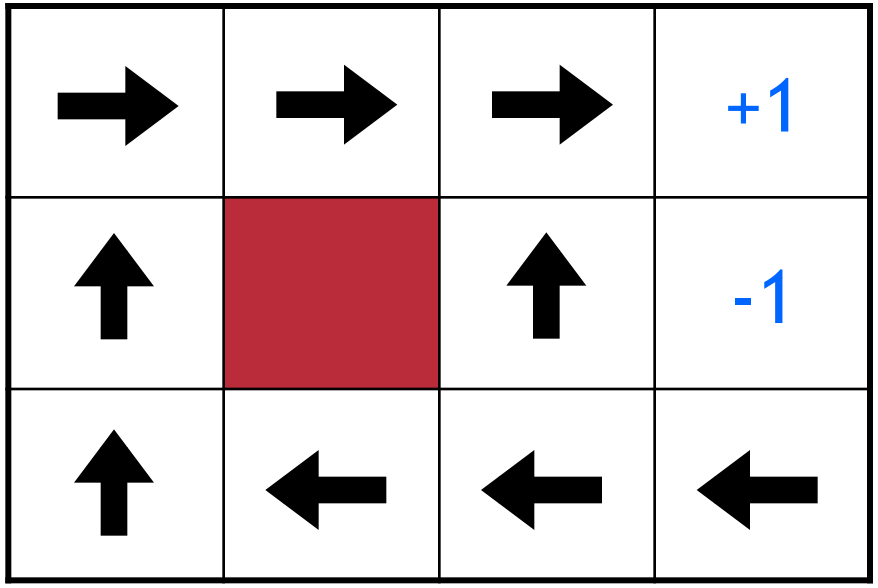
Is this a solution?



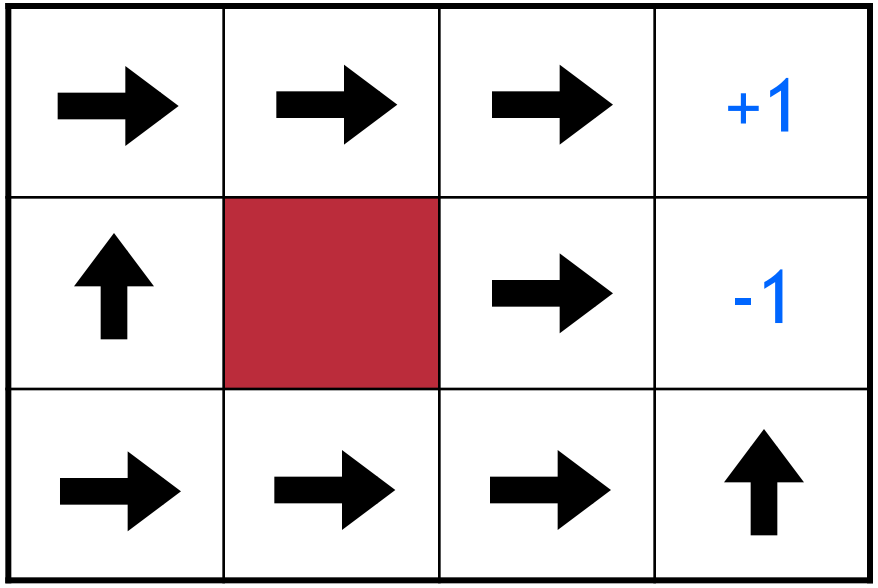
- only if actions deterministic
  - not in this case (actions are stochastic)

- solution/policy:
  - mapping from each state to an action

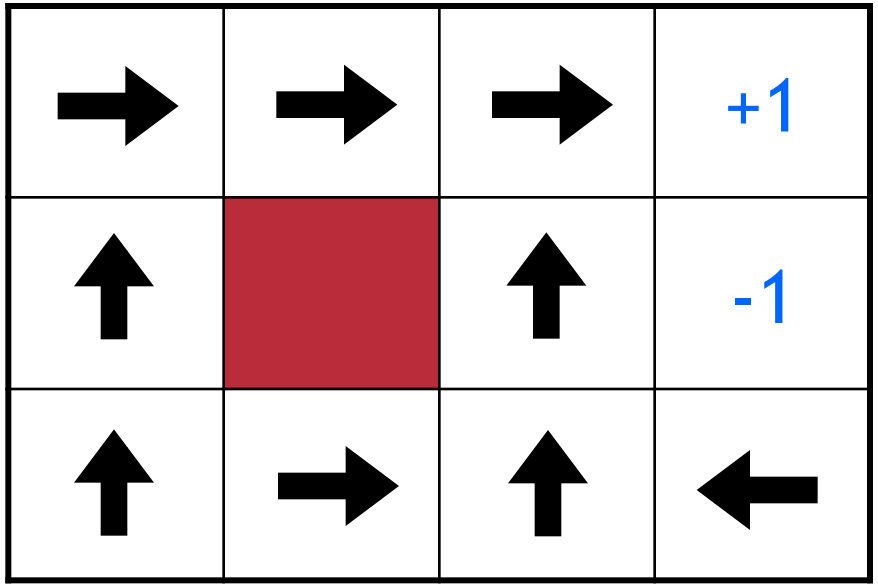
Optimal policy



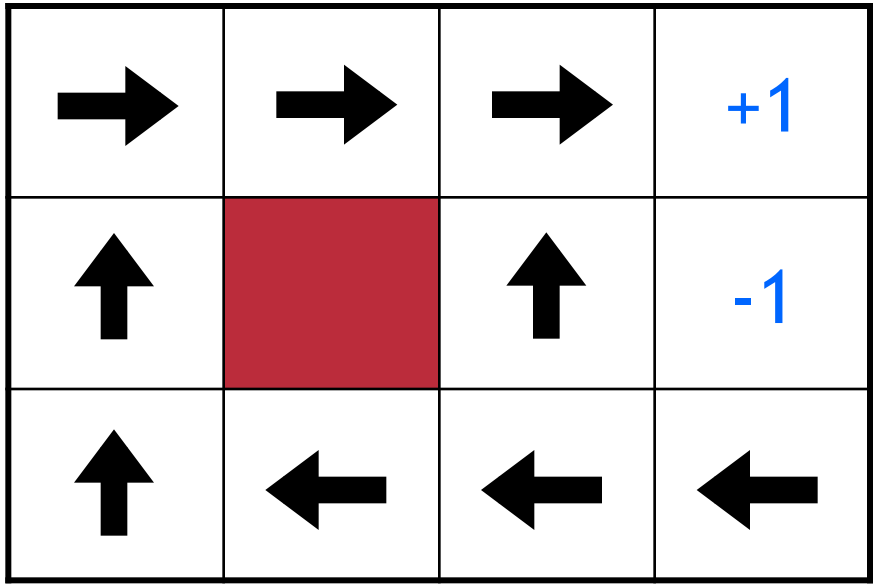
Reward for each step -2



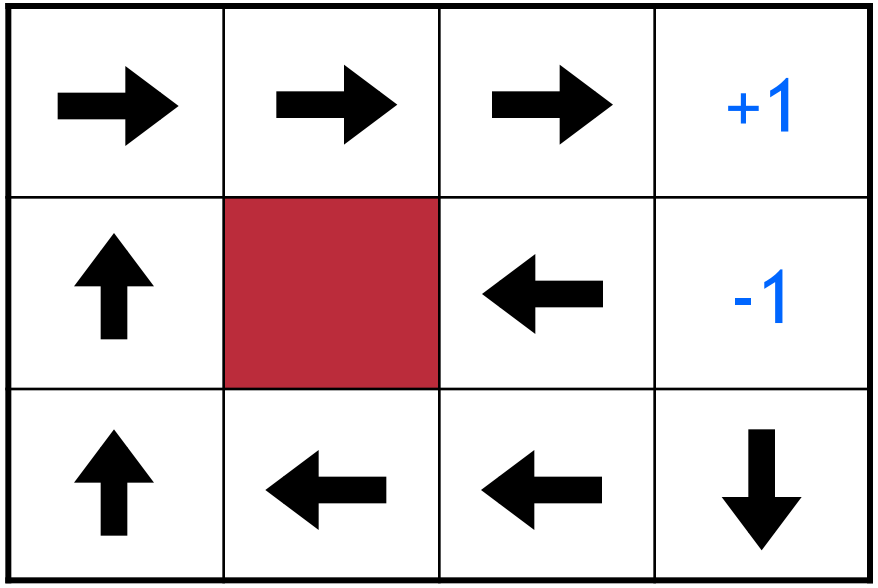
Reward for each step: -0.1



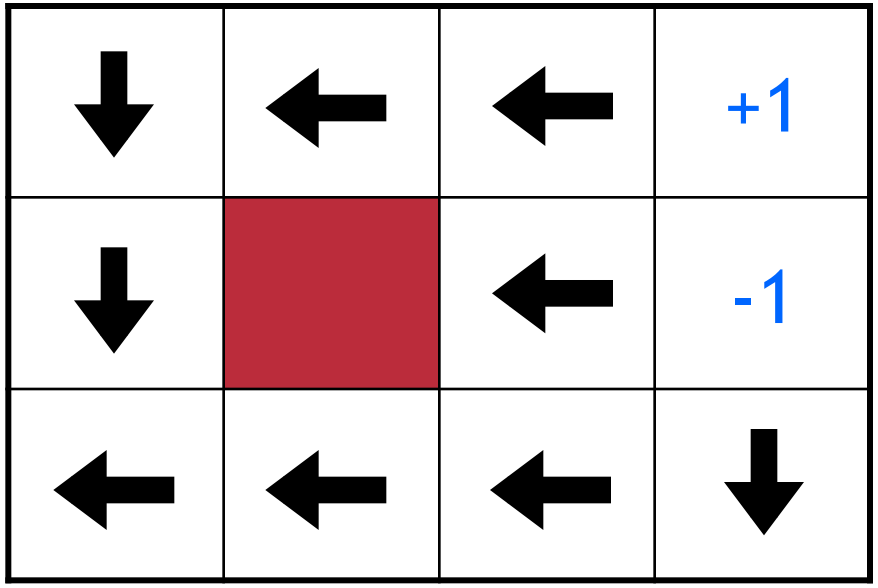
Reward for each step: -0.04



Reward for each step: -0.01



Reward for each step: +0.01



Computing return from rewards (a.k.a. returns)

- episodic (vs. continuing) tasks
  - "game over" after N steps
  - optimal policy depends on N; harder to analyze

- additive rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - infinite value for continuing tasks

- discounted rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  - value bounded if rewards bounded

Markov Property

- "The state" at step *t*, refers to whatever information is available to the agent at step *t* about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:

$$\Pr\left\{ s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0 \right\} =$$

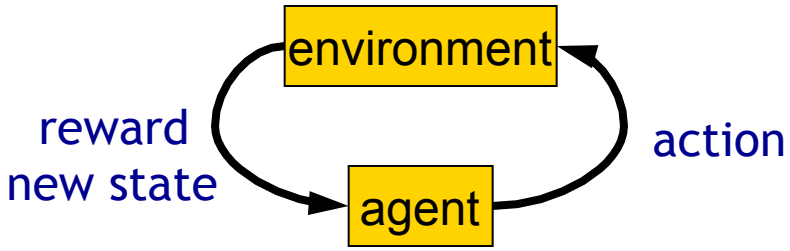
$$\Pr\left\{ s_{t+1} = s', r_{t+1} = r \mid s_t, a_t \right\}$$

for all s', r, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Even when the state signal is non-Markov, it is still appropriate to think of the state in reinforcement learning as an approximation to a Markov state.

Markov Decision Process (MDP)

- set of states S, set of actions A, initial state S_0
- transition model P(s'| s, a)
  - P([1,2] | [1,1], up) = 0.8
  - Markov property
- reward function r(s)
  - r([4,3]) = +1



- goal: maximize cumulative reward in the long run
- policy: mapping from S to A
  - $\pi(s)$ or $\pi(s,a)$
- reinforcement learning
  - transitions and rewards usually not available
  - how to change the policy based on experience
  - how to explore the environment

Markov Decision Process (MDP)

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$\mathbf{P}^a_{ss'} = \Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\} \quad \text{for all } s, s' \in S, a \in A(s).$$

  - **reward probabilities (expectation)**:

$$\mathbf{R}^a_{ss'} = E\left\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right\} \quad \text{for all } s, s' \in S, a \in A(s).$$

deterministic:

$$P^a_{ss'} = \begin{cases} 1 & \text{if } s' = s(a) \\ 0 & \text{else} \end{cases} \qquad R^a_{ss'} = r(s,a)$$

Examples of a finite MDP

Recycling Robot:

- At each step, robot has to decide whether it should
  - (1) actively search for a can,
  - (2) wait for someone to bring it a can, or
  - (3) go to home base and recharge.
- Searching is better but runs down the battery; if the robot runs out of power while searching, he has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Recycling Robot MDP

$S = \{\texttt{high}, \texttt{low}\}$

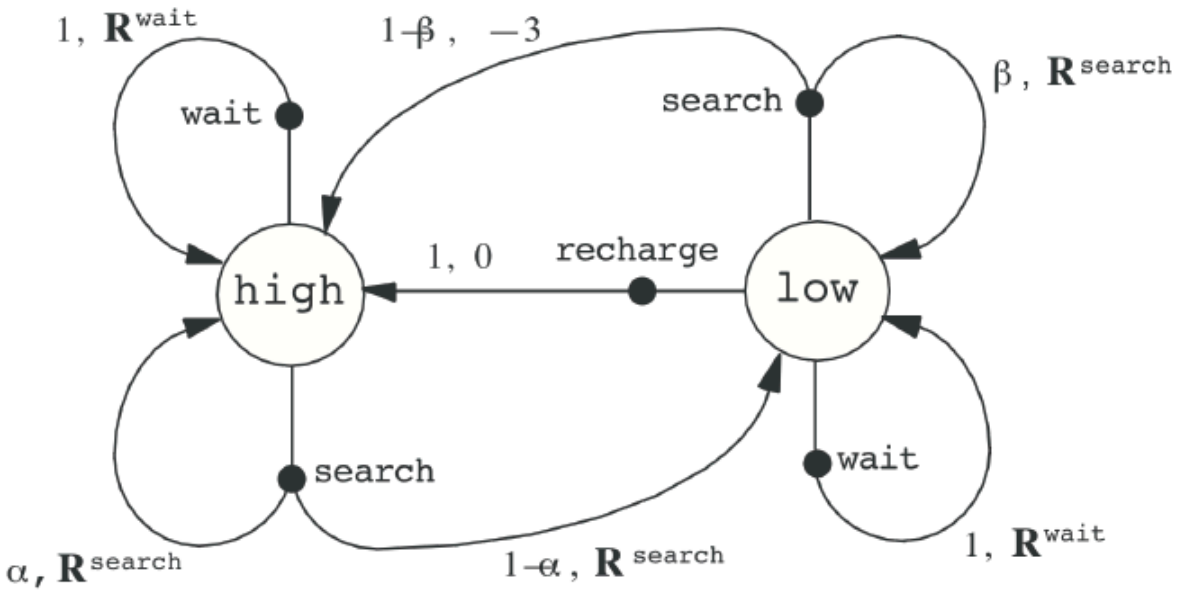
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$R^{\texttt{search}}$ = expected no. of cans while searching

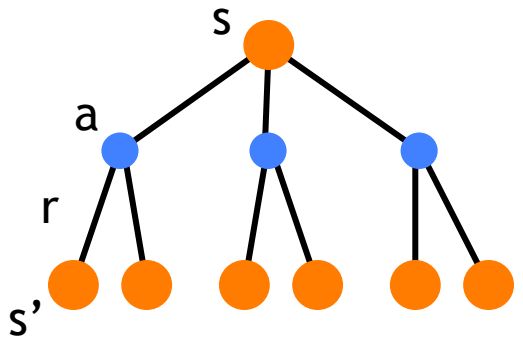
$R^{\texttt{wait}}$ = expected no. of cans while waiting

$R^{\texttt{search}} > R^{\texttt{wait}}$



Value functions

- state value function: $V^{\pi}(s)$
  - expected return when starting in *s* and following π
- state-action value function: $Q^{\pi}(s,a)$
  - expected return when starting in *s*, performing *a,* and following π
- useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^{\pi}(s,a)$
- Bellman equation



$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{\pi}(s') \right] = \sum_{a} \pi(s,a) Q^{\pi}(s,a)$$

Value functions

The value functions can be estimated from experience.

- If an agent follows policy and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, $V^{\pi}(s)$, as the number of times that state is encountered approaches infinity.
- If separate averages are kept for each action taken in a state, then these averages will similarly converge to the action values, $Q^{\pi}(s,a)$.

→ Monte Carlo methods because they involve averaging over many random samples of actual returns

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy particular recursive relationships.

Bellman equation:

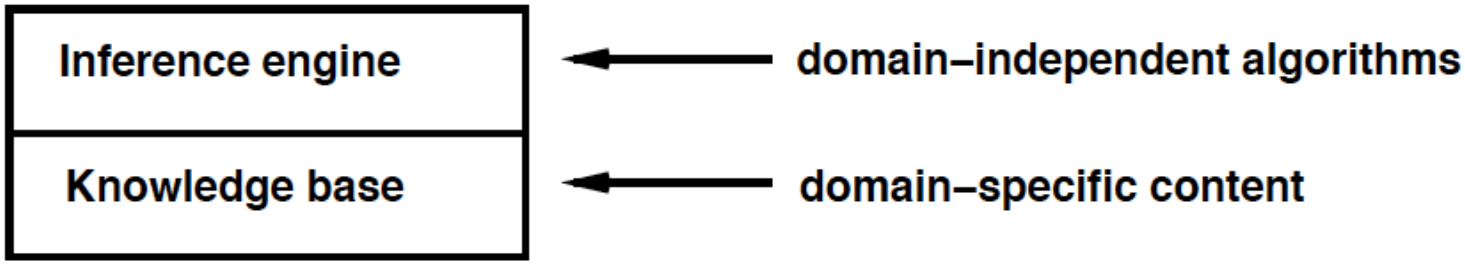
$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{\pi}(s') \right] = \sum_{a} \pi(s,a) Q^{\pi}(s,a)$$

Further reading

- Dynamic Programming methods
  - use value functions to structure the search for good policies
  - need a perfect model of the environment
- Monte Carlo methods
  - policy evaluation, policy improvement (similar to DP)
  - don't need full knowledge of environment; just experience, or simulated experience
- Temporal difference methods
  - Q-Learning
- Direct policy search methods
  - Stochastic optimizations

Knowledge-based reasoning

Basic concepts: Knowledge base



- Knowledge base = set of "knowledge units" = sentences in a formal language
  - Formal language = knowledge representation language
  - Inference = Sentences are typically derived from other sentences
  - Axioms = Sentences that are not derived from other sentences
- Declarative approach to building an agent:
  - `Tell` it what it needs to know
  - `Ask` itself what to do: answers should follow from the KB (what has been `Tell`-ed previously)

A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

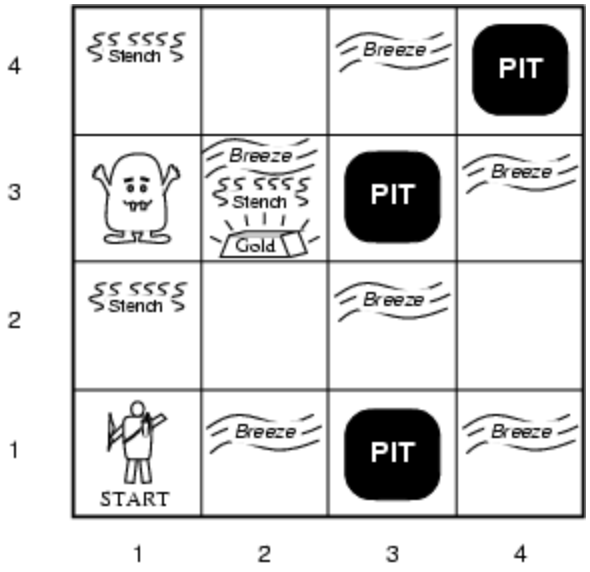
- The agent does 3 things:
  1. `Tells` the knowledge base what it perceives
  2. `Asks` the knowledge base what action it should perform
     - In the process, extensive reasoning may be done about the current state of the world, outcomes of possible action sequences, etc.
  3. `Tells` the knowledge base which action was chosen

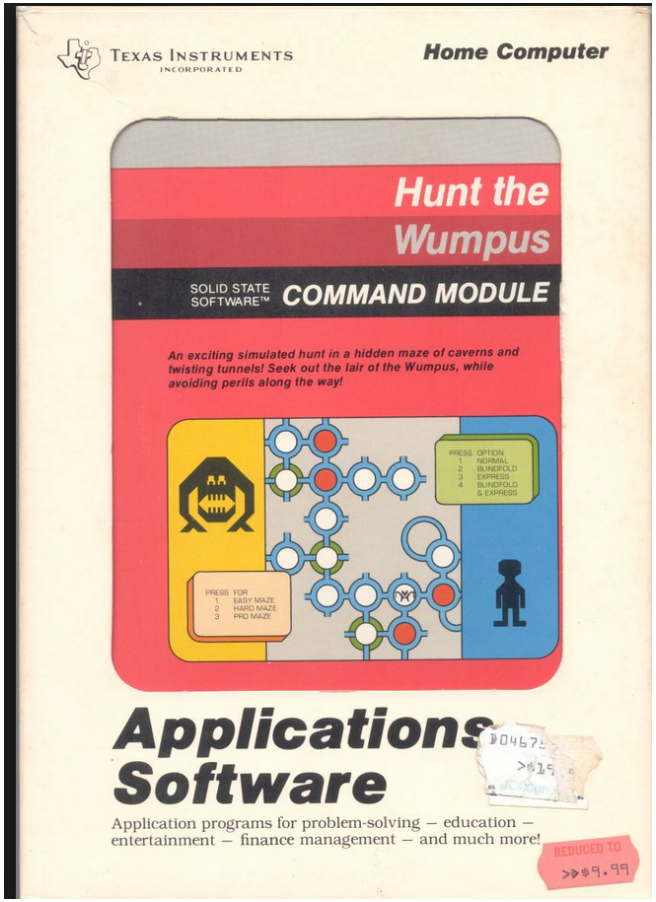
Knowledge bases

- Declarative *vs* procedural approach to build an agent:
  - Designer starts with an empty knowledge base
  - Designers creates a sequence of sentences
  - The sentences are `Tell`-ed to Agent, one-by-one
  - The step is repeated until the Agent knows how to operate in its environment
  - *Procedural approach* encodes desired behavior directly
- Agents can be viewed at the knowledge level
  - *i.e.*, what they know an what its goals are, regardless of how implemented
- Or at the implementation level
  - *i.e.*, data structures in KB and algorithms that manipulate them

Wumpus World PEAS description

- A cave consisting of rooms connected by passageways
- Wumpus is a beast that eats anyone who enters its room
- It can be shot by the agent
- Agent has only one arrow
- Pit: bottomless trap for the agent
- Agent can dig some gold





Logic in general

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines the sentences in the language
- Semantics define the "meaning" of sentences;
  - i.e., define truth of a sentence in a world

- *E.g.*, the language of arithmetic
  - x+2 ≥ y is a sentence; x2+y > {} is not a sentence
  - x+2 ≥ y is true **iff** the number x+2 is no less than the number y
  - x+2 ≥ y is true in a world where x = 7, y = 1
  - x+2 ≥ y is false in a world where x = 0, y = 6

Entailment

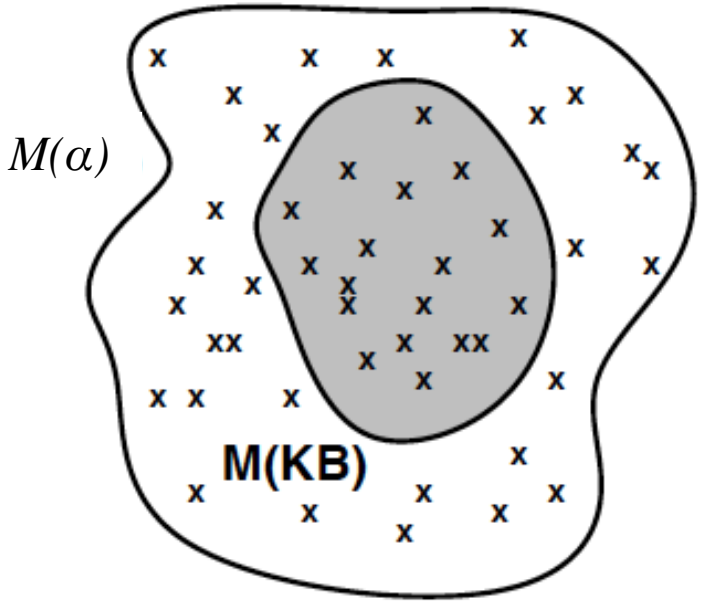
- Entailment means that one thing logically follows from another:

$$KB \models \alpha$$

- Knowledge base *KB* entails sentence α if and only if α is true in all worlds where *KB* is true
  - *E.g.*, the KB containing "the Giants won" and "the Red Sox won" entails "Either the Giants won or the Red Sox won"
  - *E.g.*, x+y = 4 entails 4 = x+y
  - Entailment is a relationship between sentences (*i.e.*, syntax) that is based on semantics

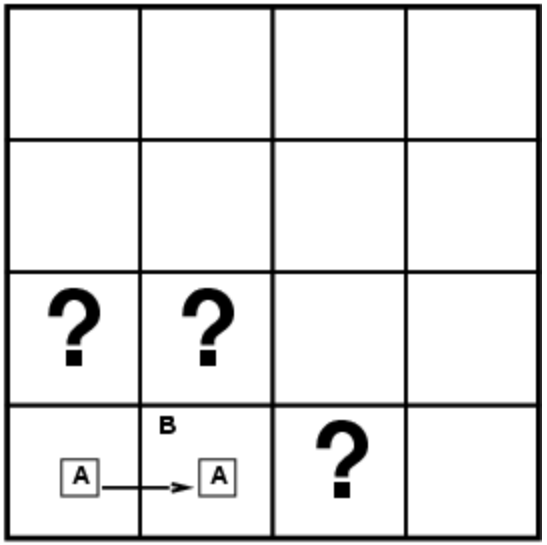
Models

- Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated
- We say m is a model of a sentence α if α is true in m
- $M(\alpha)$ is the set of all models of α
- Then $\mathrm{KB} \models \alpha$ **iff** $M(KB) \subseteq M(\alpha)$
  - E.g. *KB* = Giants won and Red Sox won
    α = Giants won
- Likewise, $\alpha \models \beta$ **iff** $M(\alpha) \subseteq M(\beta)$
  - *E.g.* x= 0 entails xy=0



Entailment in the wumpus world

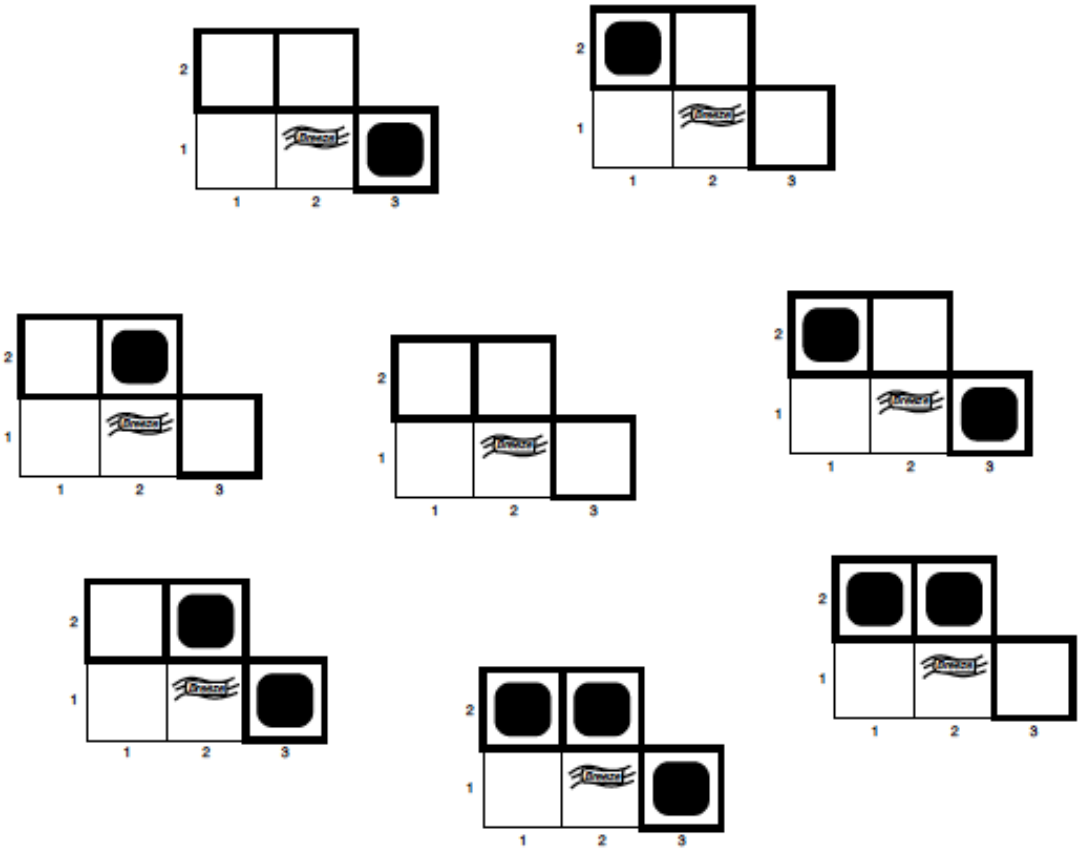
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]



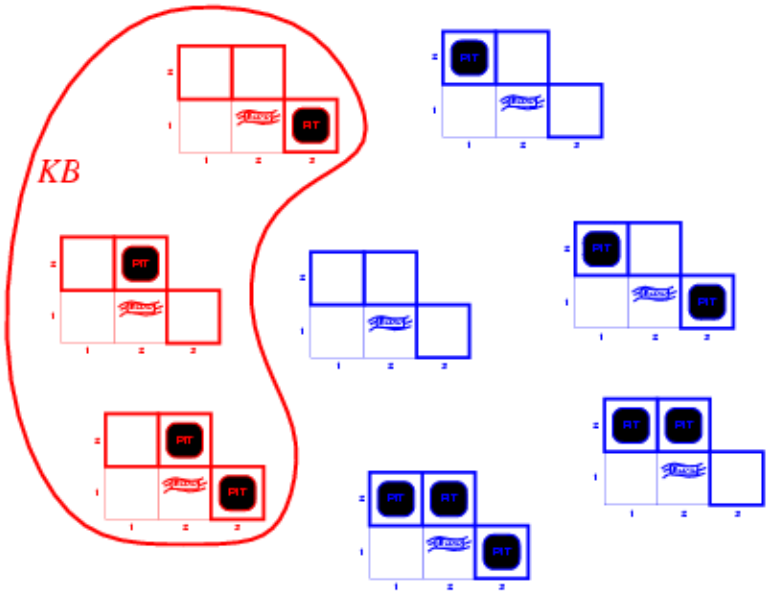
Consider possible models for *KB* assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models

Wumpus models

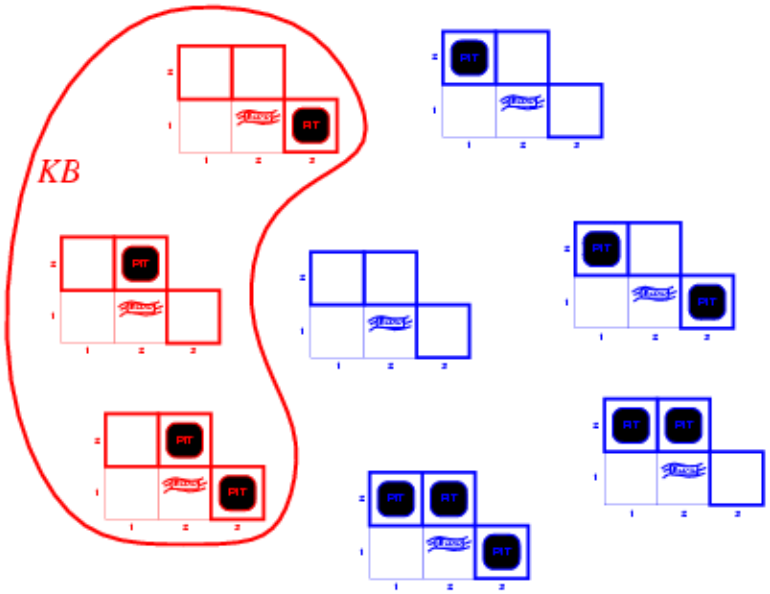


Wumpus models



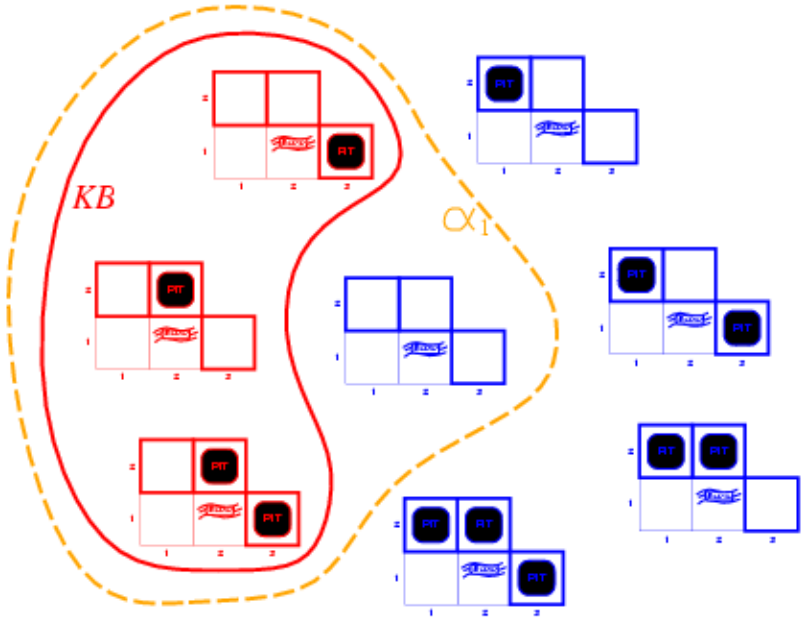
- *KB* = wumpus-world rules + observations
- The KB is false in models that contradict what the agent knows
  - Example: KB is false in any model in which [1,2] contains a pit, because there is no breeze in [1,1]

Wumpus models



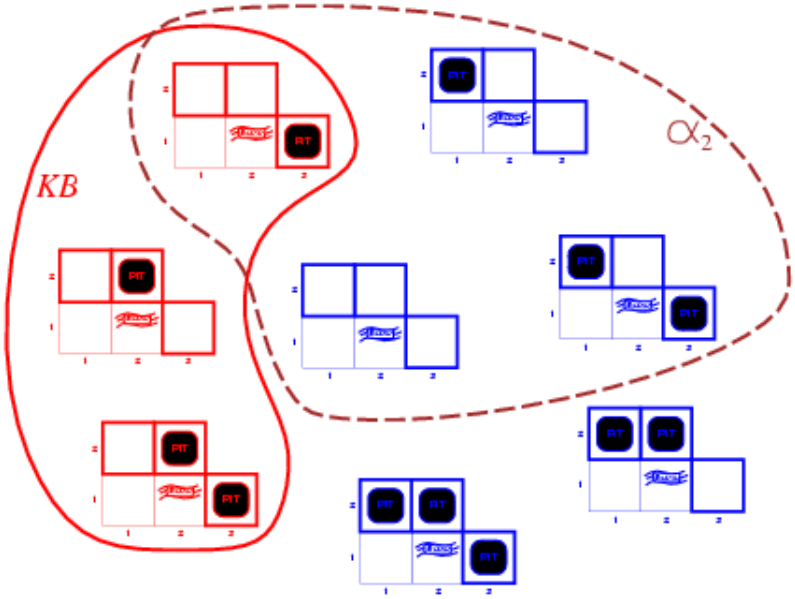
- *KB* = wumpus-world rules + observations

Wumpus models



- *KB* = wumpus-world rules + observations
- α_1 = "[1,2] is safe", $\mathrm{KB} \models \alpha_1$, proved by model checking

Wumpus models



- *KB* = wumpus-world rules + observations
- α_2 = "[2,2] is safe", $KB \not\models \alpha_2$ (that is, cannot conclude that no pit in [2,2])
  - in some models in which KB is true, α_2 is false

Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i
- Soundness: i is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
  - Also known as truth-preserving
  - An unsound inference "makes things up"
- Completeness: i is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

- Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

- That is, the procedure will answer any question whose answer follows from what is known by the *KB*

- *If KB is true in real world, then any sentences derived from KB by a sound inference procedure is also true in the real world*

Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas. But it is also powerful.

- Atomic sentences = single proposition symbols P_1, P_2 , *etc*
  - *True* is the always-true proposition
  - *False* is the always- false proposition
  - If S is a sentence, ¬S is a sentence (negation, NET)
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction, AND)
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction, OR)
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication, IMPLIES)
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional, IFF)

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
$true$ $true$ $false$

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false			
$S_1 \wedge S_2$	is true iff	S_1	is true **and**	S_2	is true	
$S_1 \vee S_2$	is true iff	S_1	is true **or**	S_2	is true	
$S_1 \Rightarrow S_2$	is true iff	S_1	is false **or**	S_2	is true	
i.e.,	is false iff	S_1	is true **and**	S_2	is false	
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true **and**	$S_2 \Rightarrow S_1$	is true	

Simple recursive process evaluates an arbitrary sentence, e.g.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Wumpus world sentences

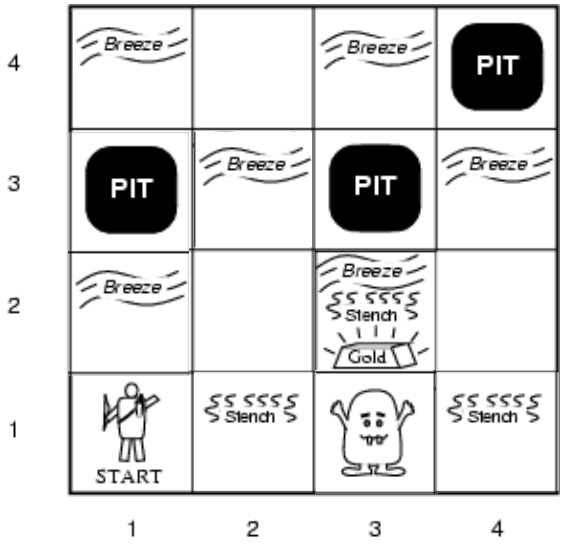
Let $P_{i,j}$ be true if there is a pit in $[i, j]$.

Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$\neg P_{1,1}$

$\neg B_{1,1}$

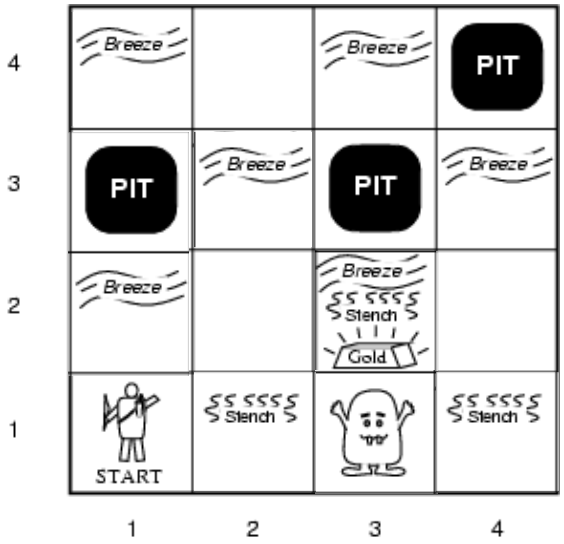
$B_{2,1}$



Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$$\neg P_{1,1}$$
$$\neg B_{1,1}$$
$$B_{2,1}$$



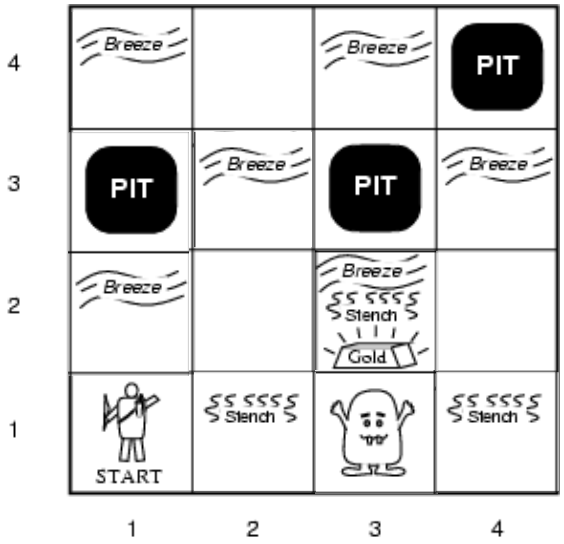
- "Pits cause breezes in adjacent squares"

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$$\neg P_{1,1}$$
$$\neg B_{1,1}$$
$$B_{2,1}$$



- "Pits cause breezes in adjacent squares"

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

"A square is breezy if and only if there is an adjacent pit"

Inference procedure

- Our goal is decide whether $KB \models \alpha$ for some sentence α
- A simple algorithm is a model-checking approach that implements directly the definition of entailment
- Idea: enumerate the models and check that α is true in every model KB is true
- Note: models are assignment of **true** or **false** to every propositional symbol
- For *N* symbols there are 2^N possible models

Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

Inference by enumeration

- Depth-first enumeration of all models is sound and complete

```
function TT-ENTAILS?(KB, α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic

    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [])

---

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
    if EMPTY?(symbols) then
        if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
        else return true
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
               TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

$O(2^n)$ for n symbols; problem is **co-NP-complete**

Logical equivalence

- Two sentences are logically equivalent **iff** true in same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validity and satisfiability

A sentence is valid if it is true in **all** models,

e.g., $True$, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in **some** model

e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in **no** models

e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

i.e., prove α by $reductio\ ad\ absurdum$

Validity and satisfiability

A sentence is valid if it is true in **all** models,

e.g., $True$, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in **some** model

e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in **no** models

e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

i.e., prove α by $reductio\ ad\ absurdum$

* A common form of argument which seeks to demonstrate that a statement is true by showing that a false, untenable, or absurd result follows from its denial

Proof methods

Proof methods divide into (roughly) two kinds:

Application of inference rules

- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.
- Typically require translation of sentences into a normal form

Model checking

truth table enumeration (always exponential in n)
improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
heuristic search in model space (sound but incomplete)
  e.g., min-conflicts-like hill-climbing algorithms

Resolution

Conjunctive Normal Form (CNF—universal)

conjunction of $\underbrace{\text{disjunctions of literals}}_{\text{clauses}}$

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

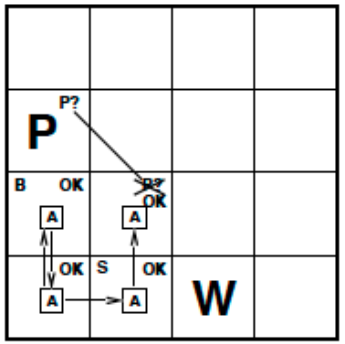
Resolution inference rule (for CNF): complete for propositional logic

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where ℓ_i and m_j are complementary literals. E.g.,

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic



Resolution

Conjunctive Normal Form (CNF—universal)

conjunction of disjunctions of literals

clauses

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

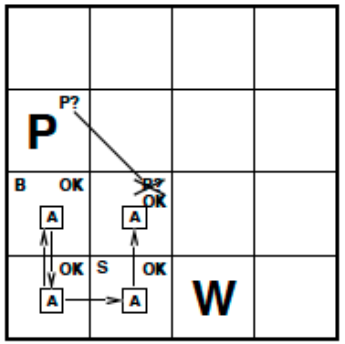
Resolution inference rule (for CNF): complete for propositional logic

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where ℓ_i and m_j are complementary literals. E.g.,

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic



Factoring: The removal of multiple copies of literals

Proof methods

Proof methods divide into (roughly) two kinds:

Application of inference rules

- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.
- Typically require translation of sentences into a normal form

Model checking

truth table enumeration (always exponential in n)
improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
heuristic search in model space (sound but incomplete)
  e.g., min-conflicts-like hill-climbing algorithms

Logical equivalence

- Two sentences are logically equivalent **iff** true in same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Acknowledgements

- Lecture materials are based on:
  - The textbook
  - Lecture materials by Stuart Russell (the co-author of the textbook) in Berkeley