

Machine Learning

References

Text Books:

1. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education.

Reference Books:

1. Trevor Hastie, Robert Tibshirani, Jerome Friedman, h The Elements of Statistical Learning, 2nd edition, springer series in statistics.
2. Ethem Alpaydın, Introduction to machine learning, second edition, MIT press.

Prerequisites

For Machine Learning Course we recommend that students meet the following prerequisites:

- Basic programming skills (in Python)
- Algorithm design
- Basics of probability & statistics

Content

Module – 1	Introduction, Concept Learning, Decision Tree Learning
Module – 2	Artificial Neural Networks-1, Artificial Neural Networks-2, Evaluating Hypothesis,
Module – 3	Bayesian Learning, Computational learning theory, Instance Based Learning,
Module – 4	Genetic Algorithms, Learning Sets of Rules, Reinforcement Learning
Module – 5	Analytical Learning-1, Analytical Learning-2, Combining Inductive and Analytical Learning

MODULE -1

Machine Learning

Introduction

Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

- Imagine computers learning from medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper

Examples of Successful Applications of Machine Learning

- Learning to recognize spoken words
- Learning to drive an autonomous vehicle
- Learning to classify new astronomical structures
- Learning to play world-class backgammon

Why is Machine Learning Important?

- Some tasks cannot be defined well, except by examples (e.g., recognizing people).
- Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- Environments change over time.
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems “by hand”.

Areas of Influence for Machine Learning

- ***Statistics***: How best to use samples drawn from unknown probability distributions to help decide from which distribution some new sample is drawn?
- ***Brain Models***: Non-linear elements with weighted inputs (Artificial Neural Networks) have been suggested as simple models of biological neurons.
- ***Adaptive Control Theory***: How to deal with controlling a process having unknown parameters that must be estimated during operation?
- ***Psychology***: How to model human performance on various learning tasks?
- ***Artificial Intelligence***: How to write algorithms to acquire the knowledge humans are able to acquire, at least, as well as humans?
- ***Evolutionary Models***: How to model certain aspects of biological evolution to improve the performance of computer programs?

Machine Learning: A Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

Why “Learn”?

Learning is used when:

- Human expertise does not exist (navigating on Mars)
- Humans are unable to explain their expertise (speech recognition)
- Solution changes in time (routing on a computer network)
- Solution needs to be adapted to particular cases (user biometrics)

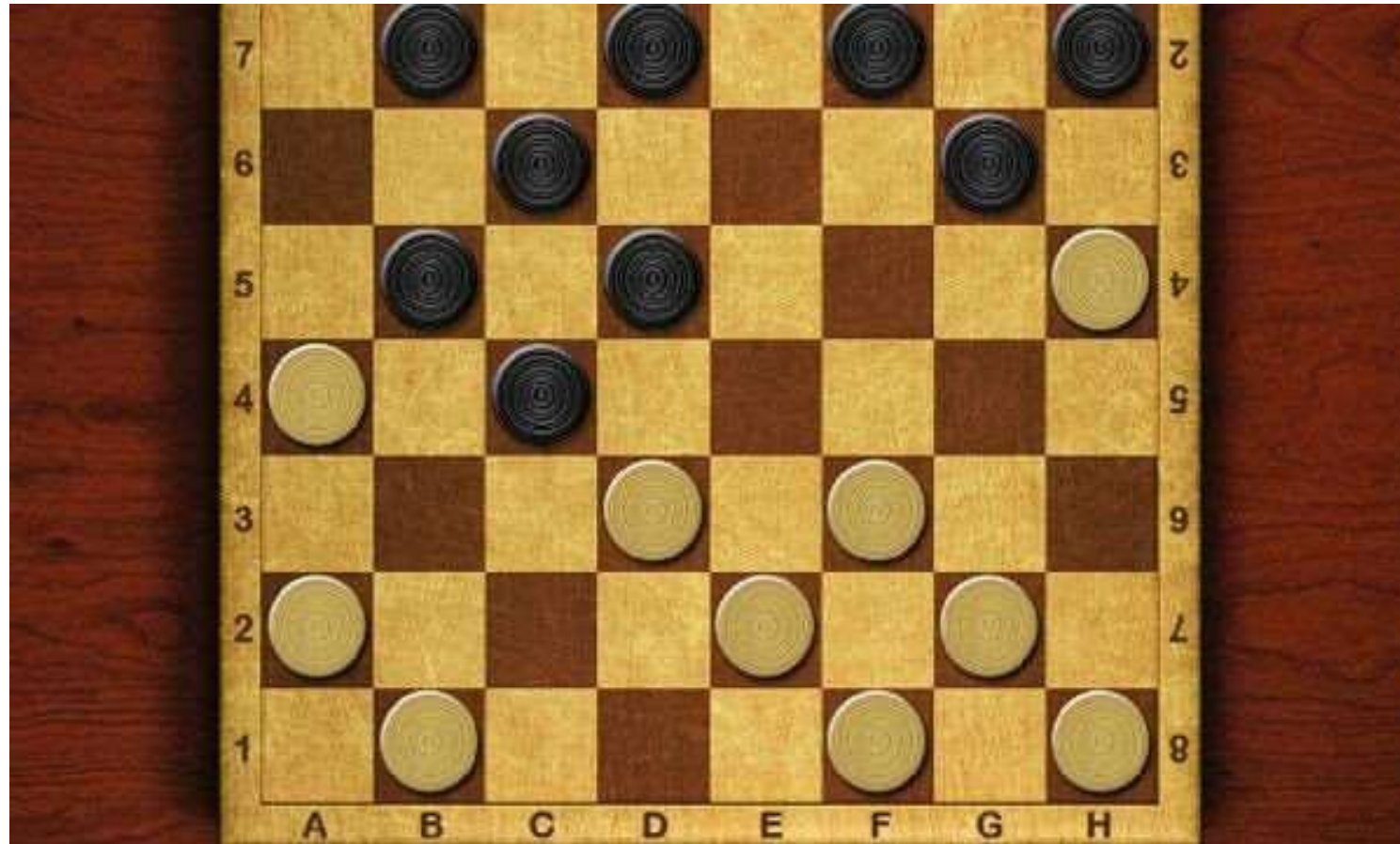
Well-Posed Learning Problem

Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

To have a well-defined learning problem, three features needs to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Checkers Game



Game Basics

- Checkers is played by two players. Each player begins the game with 12 colored discs. (One set of pieces is black and the other red.) Each player places his or her pieces on the 12 dark squares closest to him or her. Black moves first. Players then alternate moves.
- The board consists of 64 squares, alternating between 32 dark and 32 light squares.
- It is positioned so that each player has a light square on the right side corner closest to him or her.
- A player wins the game when the opponent cannot make a move. In most cases, this is because all of the opponent's pieces have been captured, but it could also be because all of his pieces are blocked in.

Rules of the Game

- Moves are allowed only on the dark squares, so pieces always move diagonally. Single pieces are always limited to forward moves (toward the opponent).
- A piece making a non-capturing move (not involving a jump) may move only one square.
- A piece making a capturing move (a jump) leaps over one of the opponent's pieces, landing in a straight diagonal line on the other side. Only one piece may be captured in a single jump; however, multiple jumps are allowed during a single turn.
- When a piece is captured, it is removed from the board.
- If a player is able to make a capture, there is no option; the jump must be made.
- If more than one capture is available, the player is free to choose whichever he or she prefers.

Rules of the Game Cont.

- When a piece reaches the furthest row from the player who controls that piece, it is crowned and becomes a king. One of the pieces which had been captured is placed on top of the king so that it is twice as high as a single piece.
- Kings are limited to moving diagonally but may move both forward and backward. (Remember that single pieces, i.e. non-kings, are always limited to forward moves.)
- Kings may combine jumps in several directions, forward and backward, on the same turn. Single pieces may shift direction diagonally during a multiple capture turn, but must always jump forward (toward the opponent).

Well-Defined Learning Problem

Checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

Handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

Arobot driving learning problem:

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance travelled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

Designing a Learning System

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
 - 1 Estimating training values
 - 2 Adjusting the weights
5. The Final Design

The basic design issues and approaches to machine learning is illustrated by considering designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

There are three attributes which impact on success or failure of the learner

1. Whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.
2. The degree to which the learner controls the sequence of training examples
3. How well it represents the distribution of examples over which the final system performance P must be measured.

1. Whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.

For example, in checkers game:

- In learning to play checkers, the system might learn from direct training examples consisting of individual *checkers board states and the correct move for each*.
- Indirect training examples consisting of the *move sequences and final outcomes of various games played*.
- The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.
- Here the learner faces an additional problem of *credit assignment*, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome.
- Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves.
- Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

2. A second important attribute of the training experience *is the degree to which the learner controls the sequence of training examples*

For example, in checkers game:

- The learner might depend on the teacher to select informative board states and to provide the correct move for each.
- Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.
- The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher present.
- Notice in this last case the learner may choose between experimenting with novel board states that it has not yet considered, or honing its skill by playing minor variations of lines of play it currently finds most promising.

3. A third attribute of the training experience is how well it represents *the distribution of examples* over which the final system performance P must be measured.

Learning is most reliable when the training examples follow a distribution similar to that of future test examples.

For example, in checkers game:

- In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.
- If its training experience E consists only of games played against itself, there is an danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested. For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion.
- It is necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be evaluated. Such situations are problematic because mastery of one distribution of examples will not necessary lead to strong performance over some other distribution.

2. Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

- Lets begin with a checkers-playing program that can generate the legal moves from any board state.
- The program needs only to learn how to choose the best move from among these legal moves. This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known.

Given this setting where we must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let *ChooseMove* be the target function and the notation is

$$\mathbf{ChooseMove : B \longrightarrow M}$$

which indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

ChooseMove is an choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

2. An alternative target function is an *evaluation function* that assigns a *numerical score* to any given board state

Let the target function V and the notation

$$V : B \longrightarrow R$$

which denote that V maps any legal board state from the set B to some real value

We intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V , then it can easily use it to select the best move from any current board position.

Let us define the target value $V(b)$ for an arbitrary board state b in B , as follows:

1. if b is a final board state that is won, then $V(b) = 100$
2. if b is a final board state that is lost, then $V(b) = -100$
3. if b is a final board state that is drawn, then $V(b) = 0$
4. if b is not a final state in the game, then $V(b) = V(b')$,

where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game

3. Choosing a Representation for the Target Function

let us choose a simple representation - for any given board state, the function c will be calculated as a linear combination of the following board features:

- x1: the number of black pieces on the board
- x2: the number of red pieces on the board
- x3: the number of black kings on the board
- x4: the number of red kings on the board
- x5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x6: the number of red pieces threatened by black

Thus, learning program will represent as a linear function of the form

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Where,

- w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w_1 through w_6 will determine the relative importance of the various board features in determining the value of the board
- The weight w_0 will provide an additive constant to the board value

Partial design of a checkers learning program:

- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself
- Target function: $V: \text{Board} \longrightarrow \mathbb{R}$
- Target function representation

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program.

4. Choosing a Function Approximation Algorithm

- In order to learn the target function f we require a set of training examples, each describing a specific board state b and the training value $V_{\text{train}}(b)$ for b .
- Each training example is an ordered pair of the form $(b, V_{\text{train}}(b))$.
- For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{\text{train}}(b)$ is therefore $+100$.

$$((x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0), +100)$$

Function Approximation Procedure

1. Derive training examples from the indirect training experience available to the learner
2. Adjusts the weights w_i to best fit these training examples

1. Estimating training values

A simple approach for estimating training values for intermediate board states is to assign the training value of $V_{\text{train}}(b)$ for any intermediate board state b to be $\hat{v}(\text{Successor}(b))$

Where ,

\hat{v} is the learner's current approximation to V

$\text{Successor}(b)$ denotes the next board state following b for which it is again the program's turn to move

Rule for estimating training values

$$V_{\text{train}}(b) \leftarrow \hat{v}(\text{Successor}(b))$$

2. Adjusting the weights

Specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{\text{train}}(b))\}$

A first step is to define what we mean by the bestfit to the training data.

- One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

$$E \equiv \sum_{\langle b, V_{\text{train}}(b) \rangle \in \text{training examples}} (V_{\text{train}}(b) - \hat{V}(b))^2$$

- Several algorithms are known for finding weights of a linear function that minimize E .

In our case, we require an *algorithm* that will incrementally refine the weights as new training examples become available and that will be robust to errors in these estimated training values

One such algorithm is called the *least mean squares*, or *LMS training rule*. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

LMS weight update rule :- For each training example (b, $V_{\text{train}}(b)$)

Use the current weights to calculate $\hat{v}(b)$

For each weight w_i , update it as

$$w_i \leftarrow w_i + \eta (V_{\text{train}}(b) - \hat{v}(b)) x_i$$

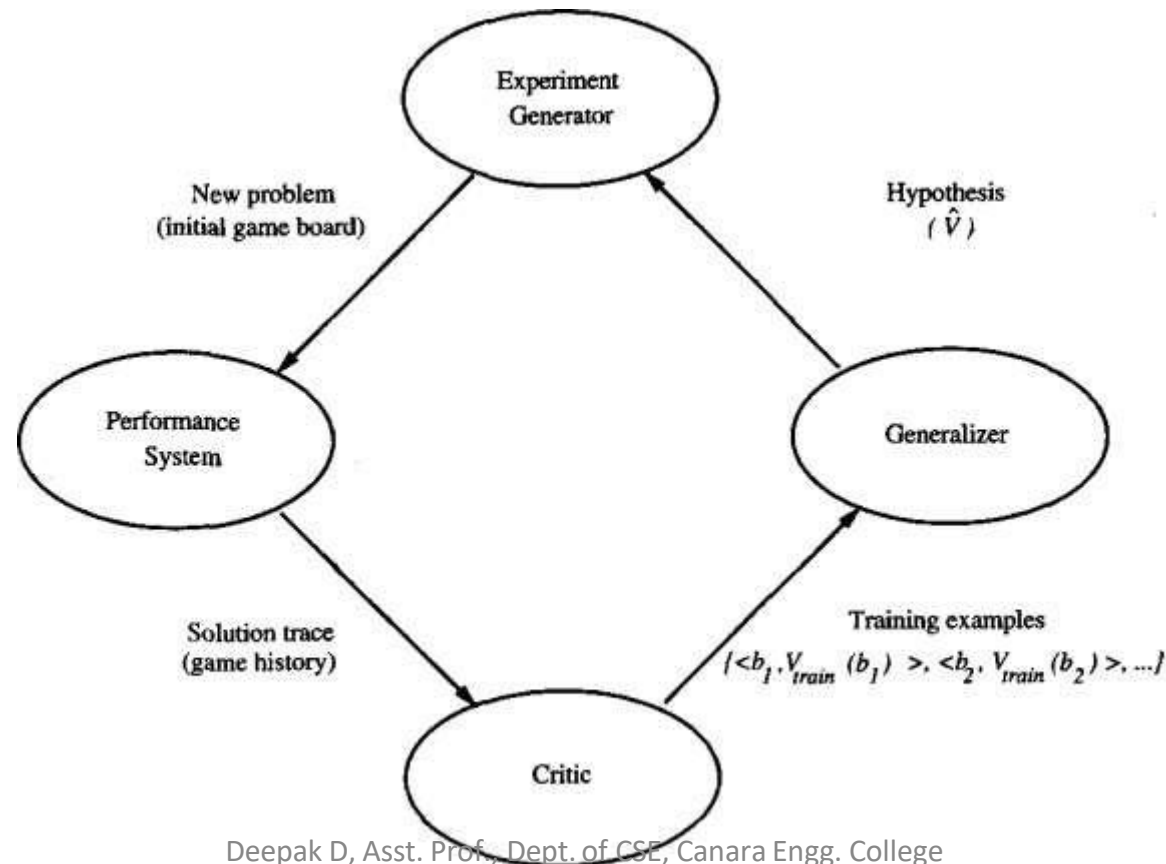
Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

- When the error ($V_{\text{train}}(b) - \hat{v}(b)$) is zero, no weights are changed.
- When ($V_{\text{train}}(b) - \hat{v}(b)$) is positive (i.e., when $\hat{v}(b)$ is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{v}(b)$, reducing the error.
- If the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

5. The Final Design

The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems



1. The Performance System is the module that must solve the given performance task by using the learned target function(s).

It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

In checkers game, the strategy used by the Performance System to select its next move at each step is determined by the learned \hat{V} evaluation function. Therefore, we expect its performance to improve as this evaluation function becomes increasingly accurate.

2. The Critic takes as input the history or trace of the game and produces as output a set of training examples of the target function. As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate V_{train} of the target function value for this example.

3. *The Generalizer* takes as input the training examples and produces an output hypothesis that is its estimate of the target function.

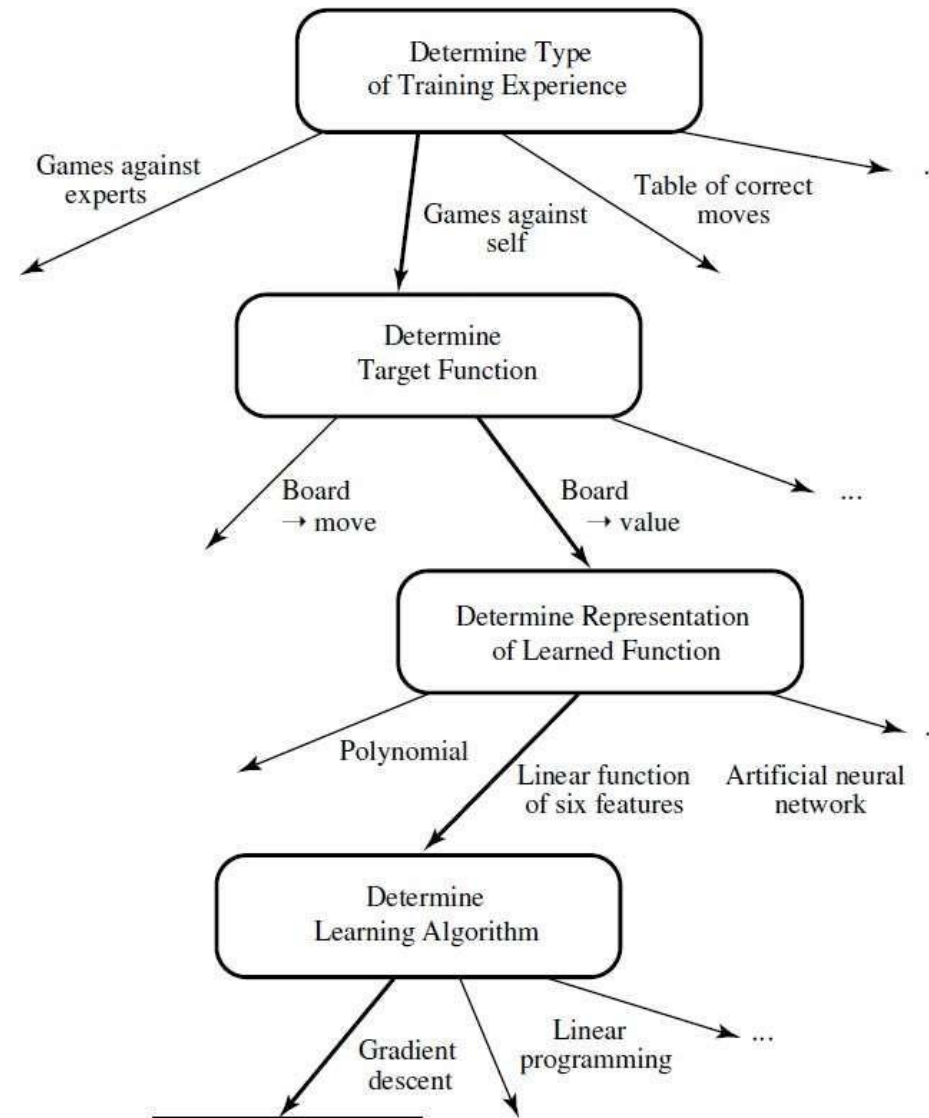
It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

In our example, the Generalizer corresponds to the LMS algorithm, and the output hypothesis is the function \hat{V} described by the learned weights w_0, \dots, W_6 .

4. *The Experiment Generator* takes as input the current hypothesis and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

In our example, the Experiment Generator always proposes the same initial game board to begin a new game.

The sequence of design choices made for the checkers program is summarized in below figure



Perspectives of Machine Learning

Perspective of machine learning involves searching very large space of possible hypothesis to determine one that best fits the observed data and any prior knowledge held by learner.

Issues in Machine Learning

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?

- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Concept Learning

- Learning involves acquiring general concepts from specific training examples. Example: People continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.
- Each such concept can be viewed as describing some subset of objects or events defined over a larger set
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

Concept learning - Inferring a Boolean-valued function from training examples of its input and output

A Concept Learning Task

Consider the example task of learning the target concept

"Days on which my friend Aldo enjoys his favorite water sport."

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Table- Describes a set of example days, each represented by a set of attributes

The attribute *EnjoySport* indicates whether or not a Person enjoys his favorite water sport on this day.

The task is to learn to predict the value of *EnjoySport* for an arbitrary day, based on the values of its other attributes ?

What hypothesis representation is provided to the learner?

Let's consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.

Let each hypothesis be a vector of six constraints, specifying the values of the six attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast.

For each attribute, the hypothesis will either

- Indicate by a "?" that any value is acceptable for this attribute,
- Specify a single required value (e.g., Warm) for the attribute, or
- Indicate by a " Φ " that no value is acceptable

If some instance x satisfies all the constraints of hypothesis h , then h classifies x as a positive example ($h(x) = 1$).

The hypothesis that PERSON enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes) is represented by the expression

$(?, \text{Cold}, \text{High}, ?, ?, ?)$

The most general hypothesis-that every day is a positive example-is represented by

$(?, ?, ?, ?, ?, ?)$

The most specific possible hypothesis-that no day is a positive example-is represented by

$(\Phi, \Phi, \Phi, \Phi, \Phi, \Phi)$

Notation

The set of items over which the concept is defined is called the set of *instances*, which we denote by X .

Example: X is the set of all possible days, each represented by the attributes: Sky, AirTemp, Humidity, Wind, Water, and Forecast

The concept or function to be learned is called the *target concept*, which we denote by c .

c can be any Boolean valued function defined over the instances X

$$c : X \longrightarrow \{0, 1\}$$

Example: The target concept corresponds to the value of the attribute *EnjoySport* (i.e., $c(x) = 1$ if *EnjoySport* = Yes, and $c(x) = 0$ if *EnjoySport* = No).

- Instances for which $c(x) = 1$ are called positive examples, or members of the target concept.
- Instances for which $c(x) = 0$ are called negative examples, or non-members of the target concept.
- The ordered pair $(x, c(x))$ to describe the training example consisting of the instance x and its target concept value $c(x)$.
- D to denote the set of available training examples
- The symbol H to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis h in H represents a Boolean-valued function defined over X

$$h : X \longrightarrow \{0, 1\}$$

- The goal of the learner is to find a hypothesis h such that $h(x) = c(x)$ for all x in X .

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

● **Given:**

- Instances X : Possible days, each described by the attributes
 - *Sky* (with possible values *Sunny*, *Cloudy*, and *Rainy*),
 - *AirTemp* (with values *Warm* and *Cold*),
 - *Humidity* (with values *Normal* and *High*),
 - *Wind* (with values *Strong* and *Weak*),
 - *Water* (with values *Warm* and *Cool*), and
 - *Forecast* (with values *Same* and *Change*).
- Hypotheses H : Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be “?” (any value is acceptable), “ \emptyset ” (no value is acceptable), or a specific value.
- Target concept c : $EnjoySport : X \rightarrow \{0, 1\}$
- Training examples D : Positive and negative examples of the target function (see Table 2.1).

● **Determine:**

- A hypothesis h in H such that $h(x) = c(x)$ for all x in X .
-

TABLE The *EnjoySport* concept learning task.

The Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

Concept learning as Search

- Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- The goal of this search is to find the hypothesis that best fits the training examples.

Example, the instances X and hypotheses H in the *EnjoySport* learning task.

The attribute *Sky* has three possible values, and *AirTemp*, *Humidity*, *Wind*, *Water Forecast* each have two possible values, the instance space X contains exactly

- $3 \times 2 \times 2 \times 2 \times 2 = 96$ Distinct instances
- $2^{16} = 65536$ Syntactically distinct hypotheses within H .

Every hypothesis containing one or more " Φ " symbols represents the empty set of instances; that is, it classifies every instance as *negative*.

- $1 + (2^{16} - 1) = 65537$. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

- Consider the two hypotheses

$h_1 = (\text{Sunny}, ?, ?, \text{Strong}, ?, ?)$

$h_2 = (\text{Sunny}, ?, ?, ?, ?, ?)$

- Consider the sets of instances that are classified positive by h_1 and by h_2 .
- h_2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h_1 will also be classified positive by h_2 . Therefore, h_2 is more general than h_1 .

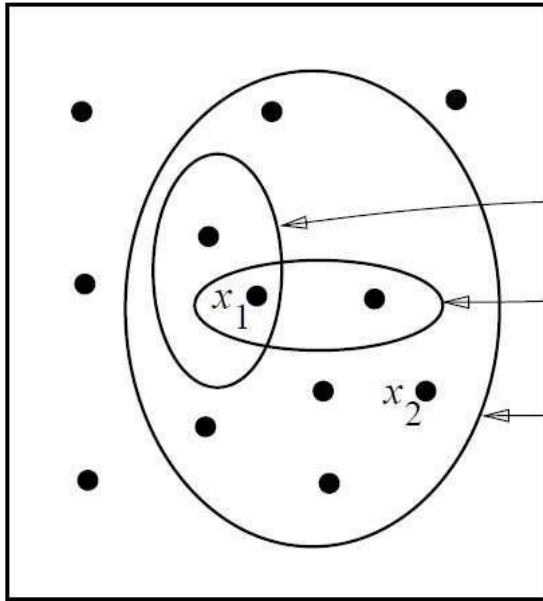
General-to-Specific Ordering of Hypotheses

- Given hypotheses h_j and h_k , h_j is more-general-than or- equal to h_k if and only if any instance that satisfies h_k also satisfies h_j

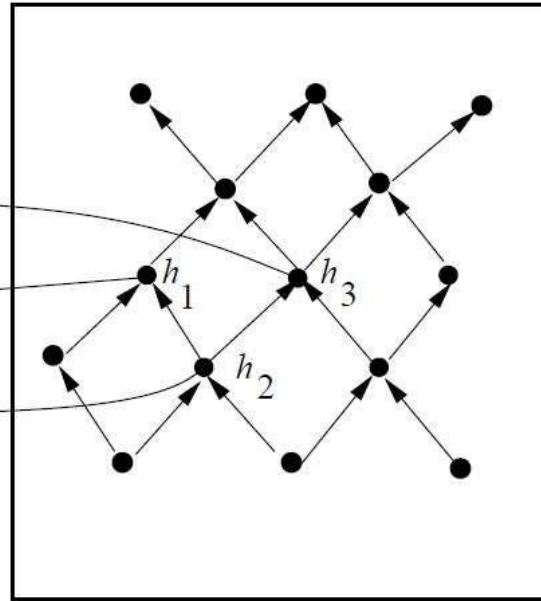
Definition: Let h_j and h_k be Boolean-valued functions defined over X . Then h_j is more general-than-or-equal-to h_k (written $h_j \geq h_k$) if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

Instances X



Hypotheses H



Specific
↑
↓
General

$x_1 = \langle \text{Sunny, Warm, High, Strong, Cool, Same} \rangle$
 $x_2 = \langle \text{Sunny, Warm, High, Light, Warm, Same} \rangle$

$h_1 = \langle \text{Sunny, ?, ?, Strong, ?, ?} \rangle$
 $h_2 = \langle \text{Sunny, ?, ?, ?, ?, ?} \rangle$
 $h_3 = \langle \text{Sunny, ?, ?, ?, Cool, ?} \rangle$

- In the figure, the box on the left represents the set X of all instances, the box on the right the set H of all hypotheses.
- Each hypothesis corresponds to some subset of X —the subset of instances that it classifies positive.
- The arrows connecting hypotheses represent the **more - general - than** relation, with the arrow pointing toward the less general hypothesis.
- Note the subset of instances characterized by h_2 subsumes the subset characterized by h_1 , hence h_2 is more - general— than h_1

FIND-S: Finding a Maximally Specific Hypothesis

FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
 - For each attribute constraint a_i in h
 - If the constraint a_i is satisfied by x
 - Then do nothing
 - Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

To illustrate this algorithm, assume the learner is given the sequence of training examples from the *EnjoySport* task

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

The first step of FIND-S is to initialize h to the most specific hypothesis in H

$$h - (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$$

$$x_1 = \langle \textit{Sunny Warm Normal Strong Warm Same} \rangle, +$$

Observing the first training example, it is clear that our hypothesis is too specific. In particular, none of the " \emptyset " constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example

$$h_1 = \langle \textit{Sunny Warm Normal Strong Warm Same} \rangle$$

This h is still very specific; it asserts that all instances are negative except for the single positive training example

$$x_2 = \langle \textit{Sunny, Warm, High, Strong, Warm, Same} \rangle, +$$

The second training example forces the algorithm to further generalize h , this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example

$$h_2 = \langle \textit{Sunny Warm ? Strong Warm Same} \rangle$$

$x_3 = \langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle, -$

Upon encountering the third training the algorithm makes no change to h . The FIND-S algorithm simply ignores every negative example.

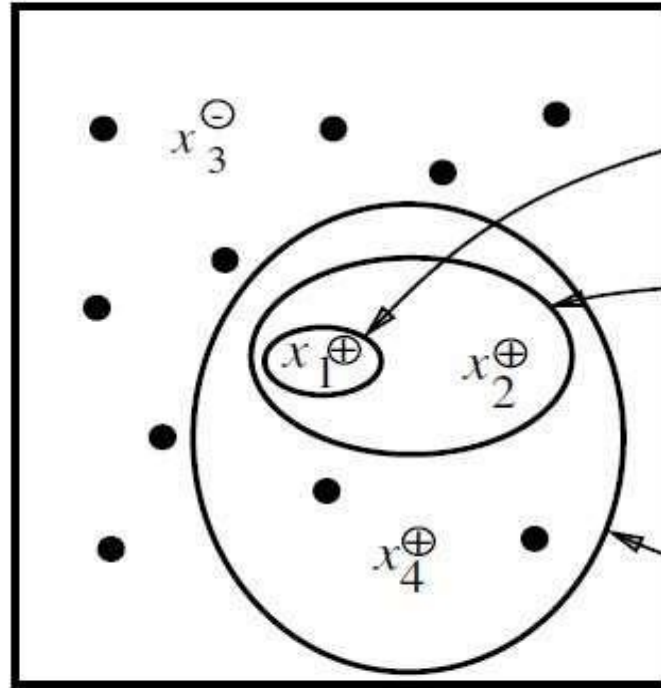
$h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$

$x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

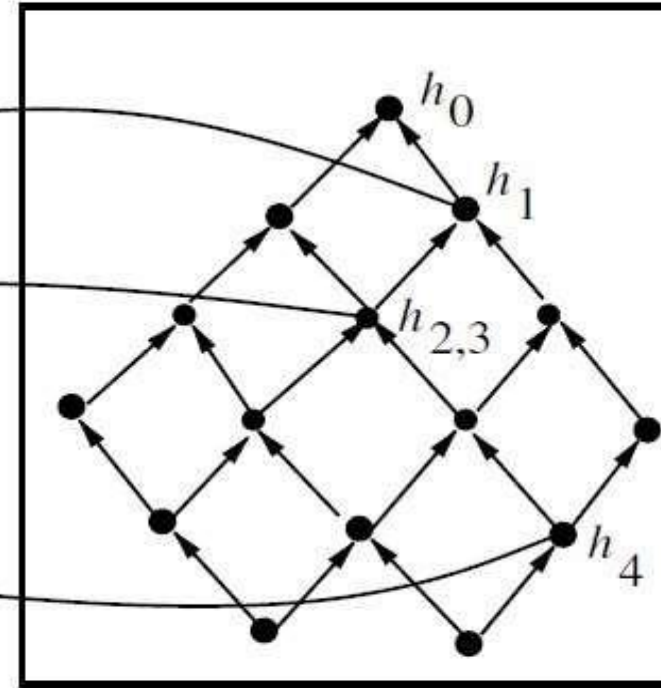
The fourth example leads to a further generalization of h

$h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

Instances X



Hypotheses H



Specific

General

$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$
 $x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle, +$
 $x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle, -$
 $x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$h_1 = \langle \text{Sunny Warm Normal Strong Warm Sam} \rangle$

$h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$

$h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$

$h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

The key property of the FIND-S algorithm is

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples
- FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H , and provided the training examples are correct.

Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

Version Space and CANDIDATE ELIMINATION Algorithm

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all *hypotheses consistent with the training examples*

Representation

- **Definition:** A hypothesis h is **consistent** with a set of training examples D if **and** only if $h(x) = c(x)$ for each example $(x, c(x))$ in D .

$$\text{Consistent}(h, D) \equiv (\forall \langle x, c(x) \rangle \in D) h(x) = c(x))$$

Note difference between definitions of *consistent* and *satisfies*

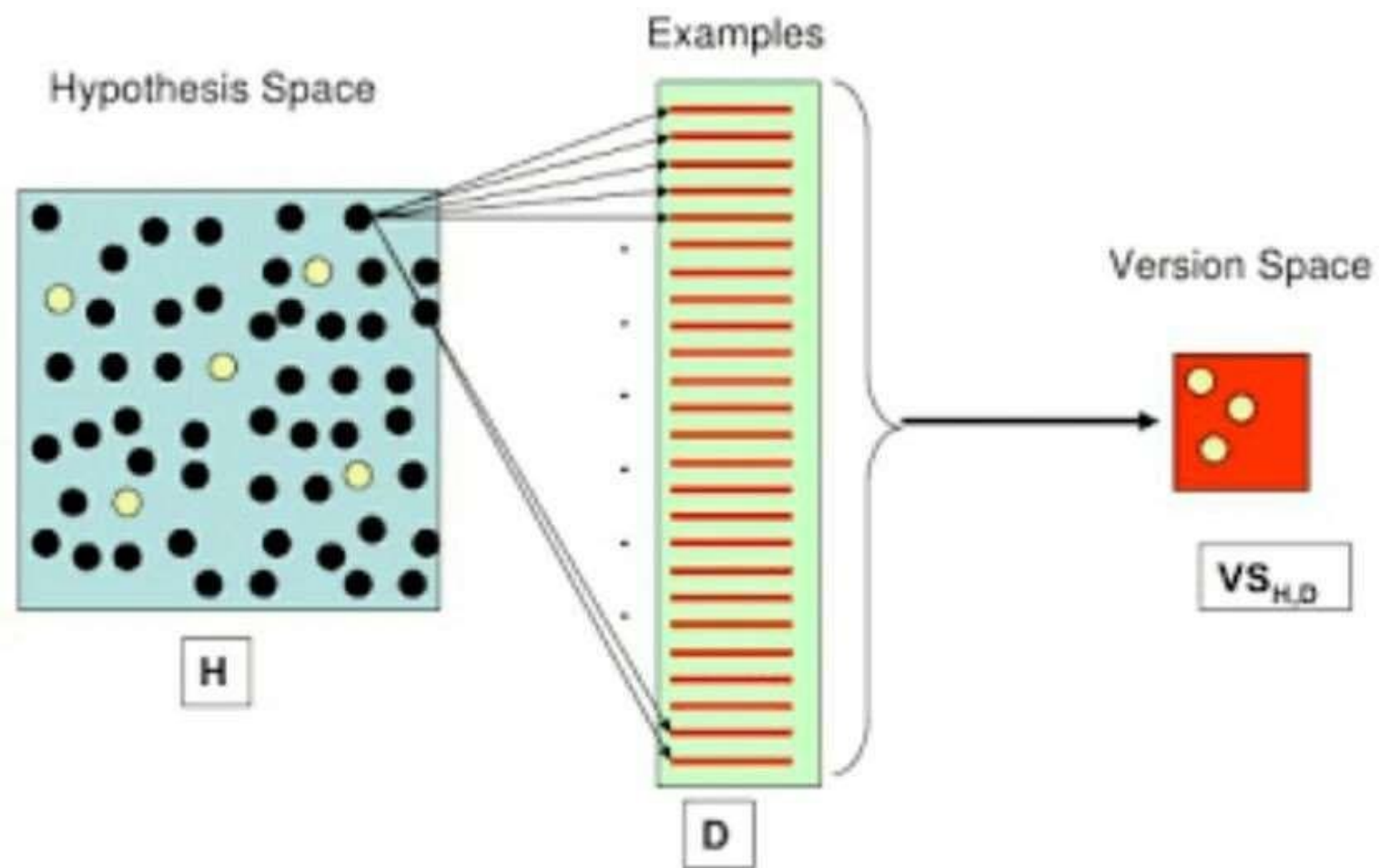
- an example x is said to *satisfy* hypothesis h when $h(x) = 1$, regardless of whether x is a positive or negative example of the target concept.
- an example x is said to *consistent* with hypothesis h iff $h(x) = c(x)$

Version Space

A representation of the set of all hypotheses which are consistent with D

Definition: The **version space**, denoted $VS_{H,D}$ with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with the training examples in D

$$VS_{H,D} \equiv \{h \in H \mid \text{Consistent}(h, D)\}$$



The LIST-THEN-ELIMINATE Algorithm

The LIST-THEN-ELIMINATE algorithm first initializes the version space to contain all hypotheses in H and then eliminates any hypothesis found inconsistent with any training example.

The LIST-THEN-ELIMINATE Algorithm

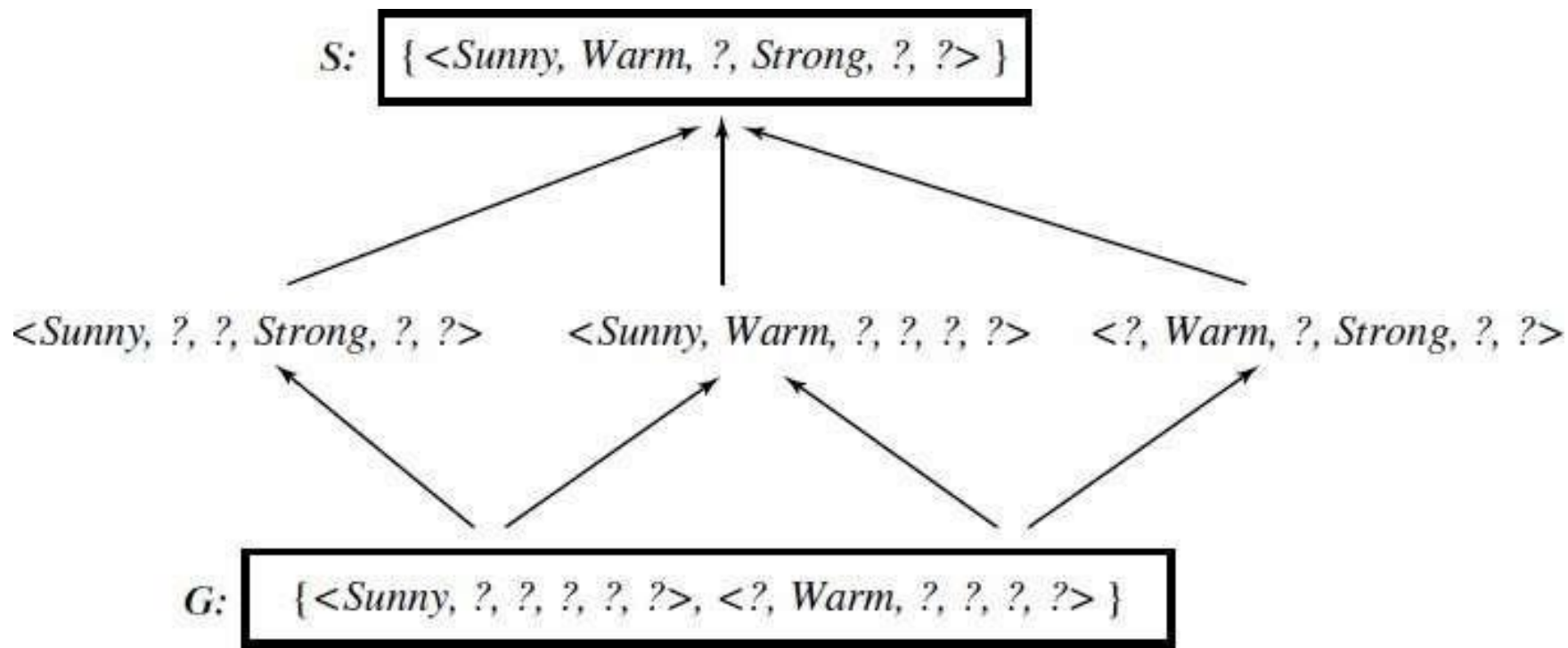
1. *VersionSpace* c a list containing every hypothesis in H
 2. For each training example, $(x, c(x))$
remove from *VersionSpace* any hypothesis h for which $h(x) \neq c(x)$
 3. Output the list of hypotheses in *VersionSpace*
-

The LIST-THEN-ELIMINATE Algorithm

- **List-Then-Eliminate** works in principle, so long as version space is finite.
- However, since it requires exhaustive enumeration of all hypotheses in practice it is not feasible.

A More Compact Representation for Version Spaces

- The version space is represented by its most general and least general members.
- These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.



- A version space with its general and specific boundary sets.
- The version space includes all six hypotheses shown here, but can be represented more simply by S and G .
- Arrows indicate instance of the *more-general-than* relation. This is the version space for the *EnjoySport* concept learning
- problem and training examples described in below table

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Definition: The **general boundary** G , with respect to hypothesis space H and training data D , is the set of maximally general members of H consistent with D

$$G \equiv \{g \in H \mid \text{Consistent}(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge \text{Consistent}(g', D)]\}$$

Definition: The **specific boundary** S , with respect to hypothesis space H and training data D , is the set of minimally general (i.e., maximally specific) members of H consistent with D .

$$S \equiv \{s \in H \mid \text{Consistent}(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge \text{Consistent}(s', D)]\}$$

Version Space representation theorem

Theorem: Let X be an arbitrary set of instances and Let H be a set of Boolean-valued hypotheses defined over X . Let $c : X \rightarrow \{0, 1\}$ be an arbitrary target concept defined over X , and let D be an arbitrary set of training examples $\{(x, c(x))\}$. For all X, H, c , and D such that S and G are well defined,

$$VS_{H,D} = \{h \in H \mid (\exists s \in S) (\exists g \in G) (g \geq_g h \geq_g s)\}$$

$$VS_{H,D} = \{h \in H \mid (\exists s \in S) (\exists g \in G) (g \geq_g h \geq_g s)\}$$

To Prove:

1. Every h satisfying the right hand side of the above expression is in $VS_{H,D}$
2. Every member of $VS_{H,D}$ satisfies the right-hand side of the expression

Sketch of proof:

1. let g, h, s be arbitrary members of G, H, S respectively with $g \geq_g h \geq_g s$

By the definition of S , s must be satisfied by all positive examples in D . Because $h \geq_g s$, h must also be satisfied by all positive examples in D .

By the definition of G , g cannot be satisfied by any negative example in D , and because $g \geq_g h$ h cannot be satisfied by any negative example in D . Because h is satisfied by all positive examples in D and by no negative examples in D , h is consistent with D , and therefore h is a member of $VS_{H,D}$

2. It can be proven by assuming some h in $VS_{H,D}$, that does not satisfy the right-hand side of the expression, then showing that this leads to an inconsistency

The CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-ELIMINATION algorithm computes the *version space* containing all hypotheses from H that are consistent with an observed sequence of training examples.

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training example d , do

- If d is a positive example
 - Remove from G any hypothesis inconsistent with d
 - For each hypothesis s in S that is not consistent with d
 - Remove s from S
 - Add to S all minimal generalizations h of s such that
 - h is consistent with d , and some member of G is more general than h
 - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
 - Remove from S any hypothesis inconsistent with d
 - For each hypothesis g in G that is not consistent with d
 - Remove g from G
 - Add to G all minimal specializations h of g such that
 - h is consistent with d , and some member of S is more specific than h
 - Remove from G any hypothesis that is less general than another hypothesis in G

An Illustrative Example

The boundary sets are first initialized to G_0 and S_0 , the most general and most specific hypotheses in H .

S_0

$\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

G_0

$\langle ?, ?, ?, ?, ?, ? \rangle$

For training example d,

$\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle +$

S₀

$\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$



S₁

$\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle$

G₀, G₁

$\langle ?, ?, ?, ?, ?, ? \rangle$

For training example d,

$\langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle +$

S₁

$\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle$



S₂

$\langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$

G₁, G₂

$\langle ?, ?, ?, ?, ?, ? \rangle$

For training example d,

$\langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle -$

$\mathbf{S}_2, \mathbf{S}_3$

$\langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$

\mathbf{G}_3

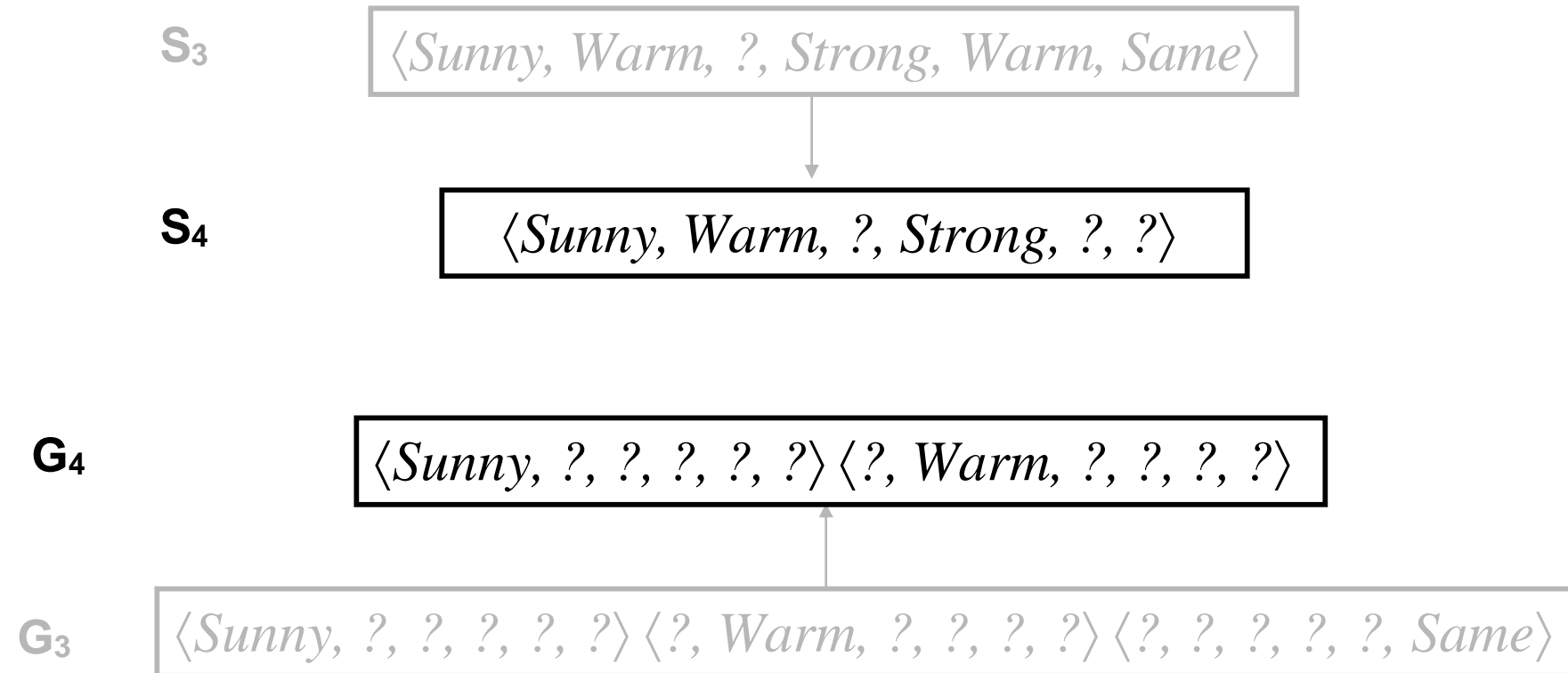
$\langle \text{Sunny, ?, ?, ?, ?, ?} \rangle \langle \text{?, Warm, ?, ?, ?, ?} \rangle \langle \text{?, ?, ?, ?, ?, Same} \rangle$

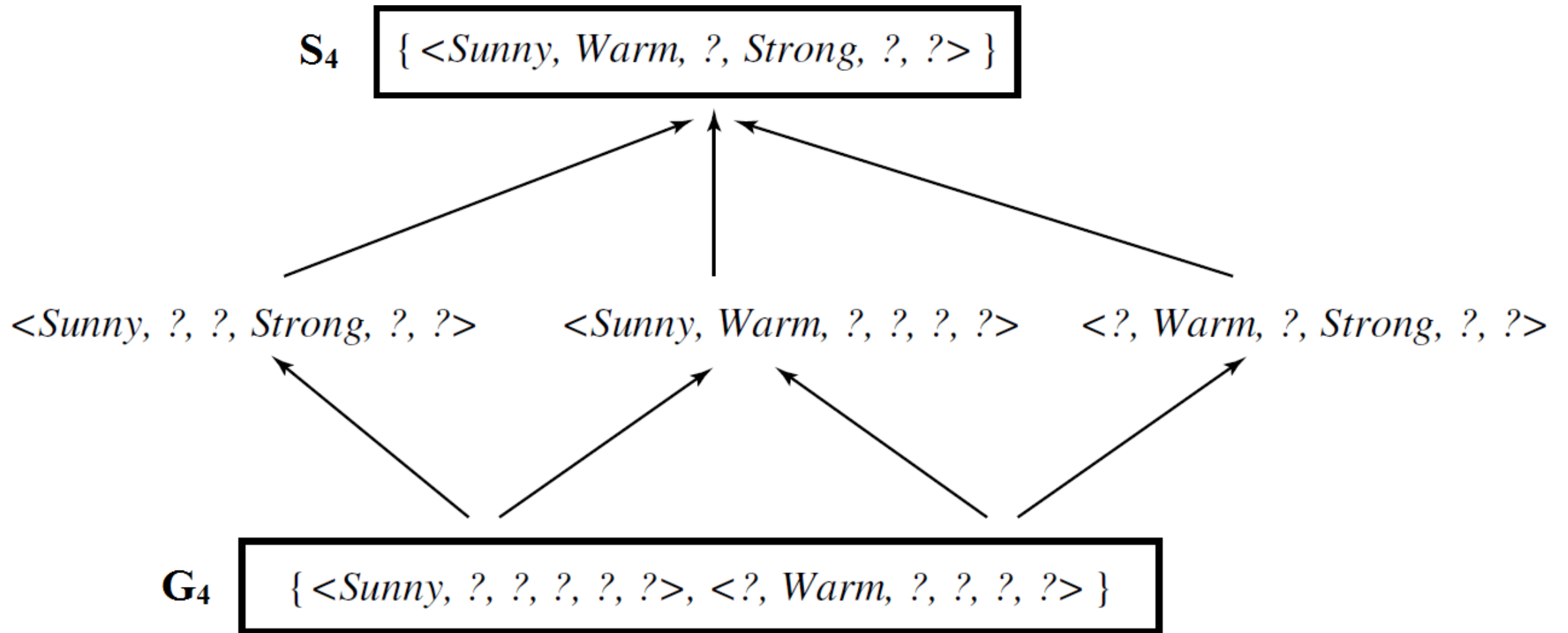
\mathbf{G}_2

$\langle \text{?, ?, ?, ?, ?, ?} \rangle$

For training example d,

$\langle \text{Sunny, Warm, High, Strong, Cool Change} \rangle +$





The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

Inductive Bias

The fundamental questions for inductive inference

- What if the target concept is not contained in the hypothesis space?
- Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?
- How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?
- How does the size of the hypothesis space influence the number of training examples that must be observed?

Effect of incomplete hypothesis space

Preceding algorithms work if target function is in H

Will generally not work if target function not in H

Consider following examples which represent target function

“sky = sunny or sky = cloudy”:

⟨Sunny Warm Normal Strong Cool Change⟩	Y
⟨Cloudy Warm Normal Strong Cool Change⟩	Y
⟨Rainy Warm Normal Strong Cool Change⟩	N

If apply Candidate Elimination algorithm as before, end up with empty Version Space

After first two training example

S= ⟨? Warm Normal Strong Cool Change⟩

New hypothesis is overly general and it covers the third negative training example!

Our H does not include the appropriate c

An Unbiased Learner

Incomplete hypothesis space

- If c not in H , then consider generalizing representation of H to contain c
- The size of the instance space X of days described by the six available attributes is 96. The number of distinct subsets that can be defined over a set X containing $|X|$ elements (i.e., the size of the power set of X) is $2^{|X|}$
- Recall that there are 96 instances in *EnjoySport*; hence there are 2^{96} possible hypotheses in full space H
- Can do this by using full propositional calculus with AND, OR, NOT
- Hence H defined only by conjunctions of attributes is biased (containing only 973 h 's)

- Let us reformulate the *Enjoysport* learning task in an unbiased way by defining a new hypothesis space H' that can represent every subset of instances; that is, let H' correspond to the power set of X .
- One way to define such an H' is to allow arbitrary disjunctions, conjunctions, and negations of our earlier hypotheses.

For instance, the target concept "*Sky = Sunny or Sky = Cloudy*" could then be described as

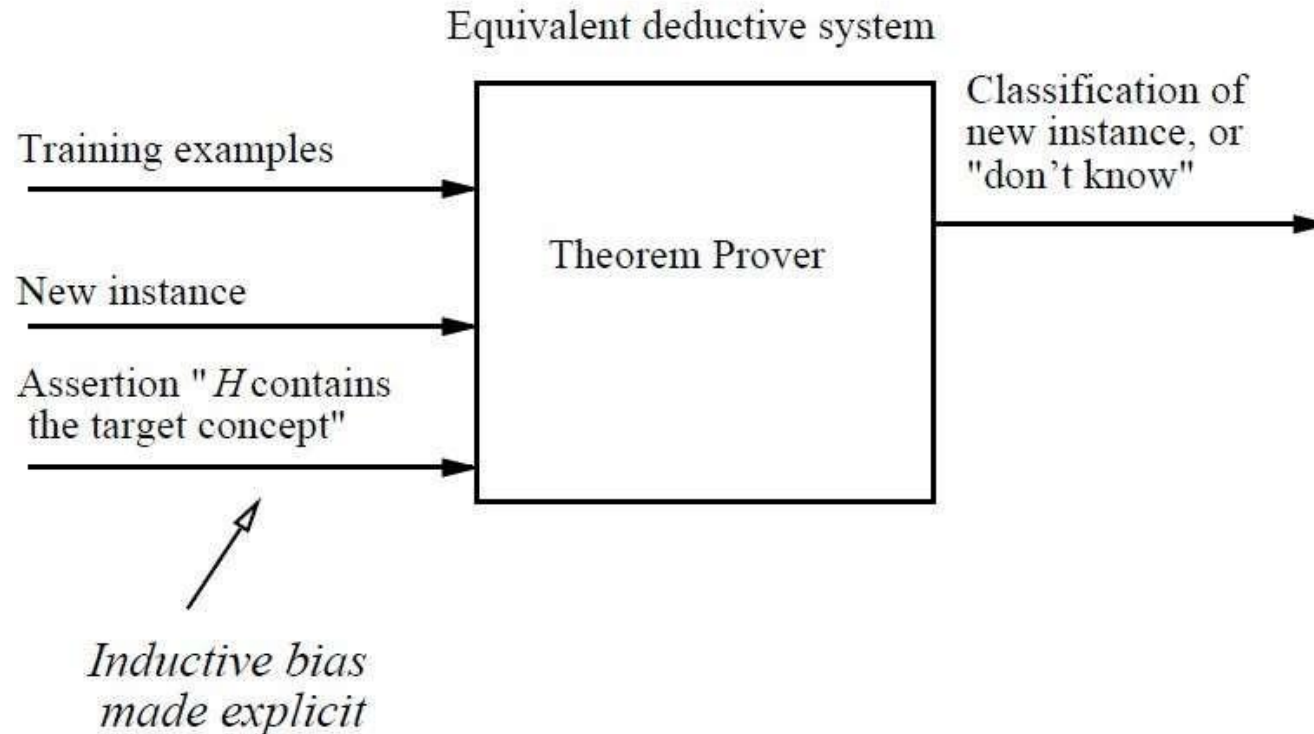
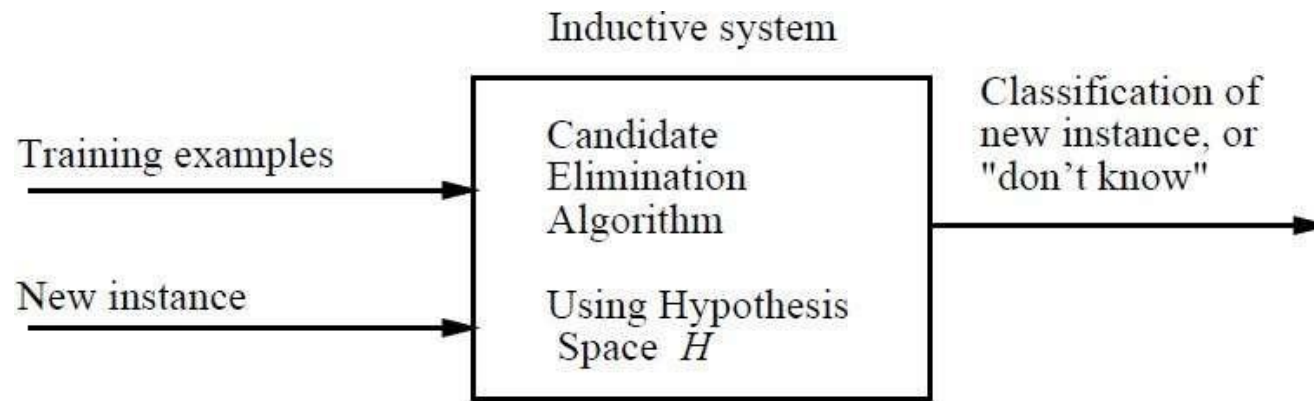
$$(\textit{Sunny}, ?, ?, ?, ?, ?) \vee (\textit{Cloudy}, ?, ?, ?, ?, ?)$$

Definition:

Consider a concept learning algorithm L for the set of instances X .

- Let c be an arbitrary concept defined over X
- Let $D_c = \{(x, c(x))\}$ be an arbitrary set of training examples of c .
- Let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on the data D_c .
- The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

$$(\forall \langle x_i \in X \rangle [(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)])$$



Modelling inductive systems by equivalent deductive systems. The input-output behavior of the CANDIDATE-ELIMINATION algorithm using a hypothesis space H is identical to that of a deductive theorem prover utilizing the assertion " H contains the target concept." This assertion is therefore called the **inductive bias** of the CANDIDATE-ELIMINATION algorithm. characterizing inductive systems by their inductive bias allows modelling them by their equivalent deductive systems. This provides a way to compare inductive systems according to their policies for generalizing beyond the observed training data.

DECISION TREE LEARNING

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.

DECISION TREE REPRESENTATION

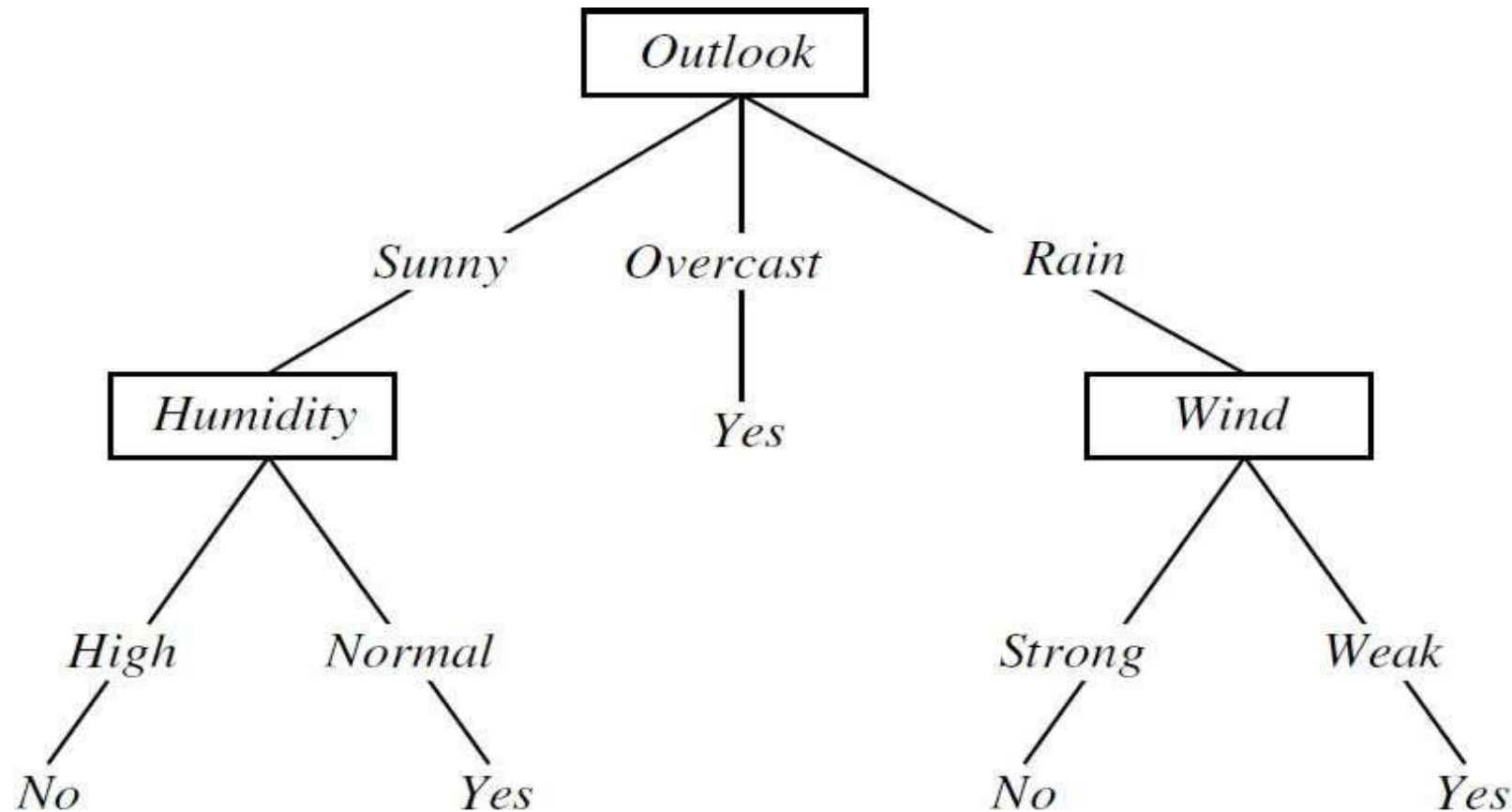


FIGURE: A decision tree for the concept *PlayTennis*. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions

For example,

The decision tree shown in above figure corresponds to the expression

(Outlook = Sunny A Humidity = Normal)

(Outlook = Overcast)

(Outlook = Rain A Wind = Weak)

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

Decision tree learning is generally best suited to problems with the following characteristics:

1. *Instances are represented by attribute-value pairs* – Instances are described by a fixed set of attributes and their values
2. *The target function has discrete output values* – The decision tree assigns a Boolean classification (e.g., yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.
3. *Disjunctive descriptions may be required*

4. *The training data may contain errors* – Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.
5. *The training data may contain missing attribute values* – Decision tree methods can be used even when some training examples have unknown values
- Decision tree learning has been applied to problems such as learning to classify *medical patients by their disease, equipment malfunctions by their cause, and loan applicants by their likelihood of defaulting on payments.*
 - Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as *classification problems.*

THE BASIC DECISION TREE LEARNING ALGORITHM

- Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees. This approach is exemplified by the ID3 algorithm and its successor C4.5

What is the ID3 algorithm?

- ID3 stands for Iterative Dichotomiser 3
- ID3 is a precursor to the C4.5 Algorithm.
- The ID3 algorithm was invented by Ross Quinlan in 1975
- Used to generate a decision tree from a given data set by employing a top-down, greedy search, to test each attribute at every node of the tree.
- The resulting tree is used to classify future samples.

ID3 algorithm

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- 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 Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

- Otherwise Begin
 - $A \leftarrow$ the attribute from Attributes that best* classifies Examples
 - The decision attribute for Root $\leftarrow A$
 - For each possible value, v_i , of A,
 - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
 - Let $Examples_{v_i}$, be the subset of Examples that have value v_i for A
 - If $Examples_{v_i}$, is empty
 - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
 - Else below this new branch add the subtree
 $ID3(Examples_{v_i}, \text{Target_attribute}, \text{Attributes} - \{A\})$
- End
- Return Root

* The best attribute is the one with highest information gain

Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- A statistical property called *information gain* that measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses *information gain* measure to select among the candidate attributes at each step while growing the tree.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

- To define information gain, we begin by defining a measure called entropy.
Entropy measures the impurity of a collection of examples.
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

$$\text{Entropy}(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where,

p_{+} is the proportion of positive examples in S

p_{-} is the proportion of negative examples in S.

Example: Entropy

- Suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this boolean classification is

$$\begin{aligned} \text{Entropy}([9+, 5-]) &= -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) \\ &= 0.940 \end{aligned}$$

- The entropy is 0 if all members of S belong to the same class
- The entropy is 1 when the collection contains an equal number of positive and negative examples
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1

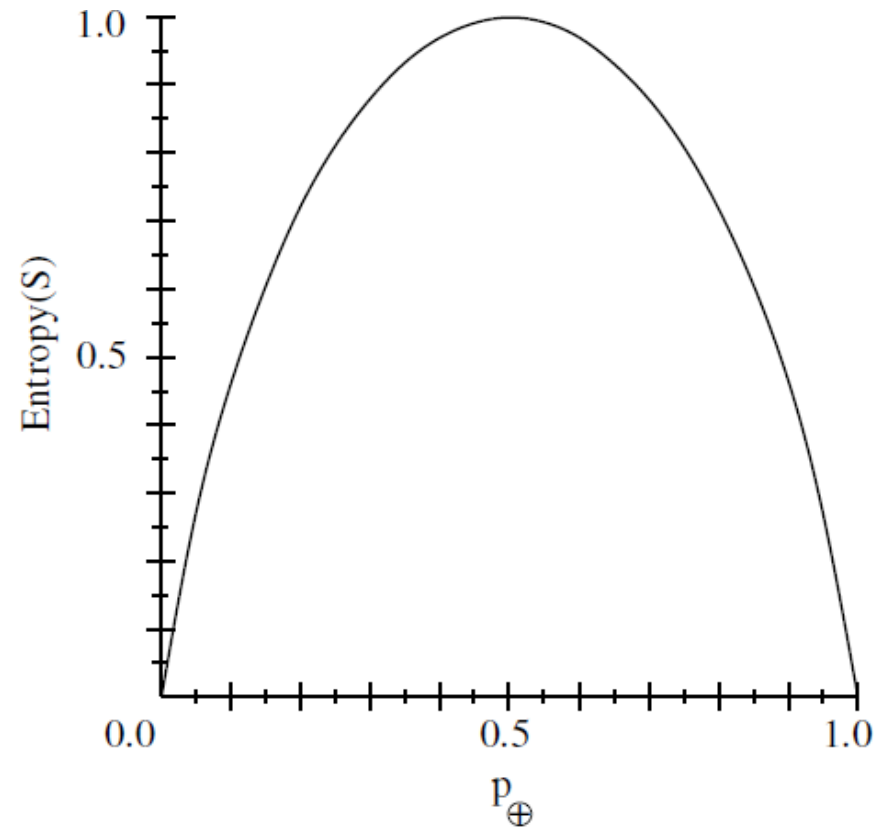


FIGURE The entropy function relative to a boolean classification, as the proportion, p_{\oplus} , of positive examples varies between 0 and 1.

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

- *Information gain*, is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, $\text{Gain}(S, A)$ of an attribute A , relative to a collection of examples S , is defined as

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Example: Information gain

Let, $Values(Wind) = \{Weak, Strong\}$

$$S = [9+, 5-]$$

$$S_{Weak} = [6+, 2-]$$

$$S_{Strong} = [3+, 3-]$$

Information gain of attribute *Wind*:

$$\begin{aligned} Gain(S, Wind) &= Entropy(S) - 8/14 Entropy(S_{Weak}) - 6/14 Entropy(S_{Strong}) \\ &= 0.94 - (8/14) * 0.811 - (6/14) * 1.00 \\ &= 0.048 \end{aligned}$$

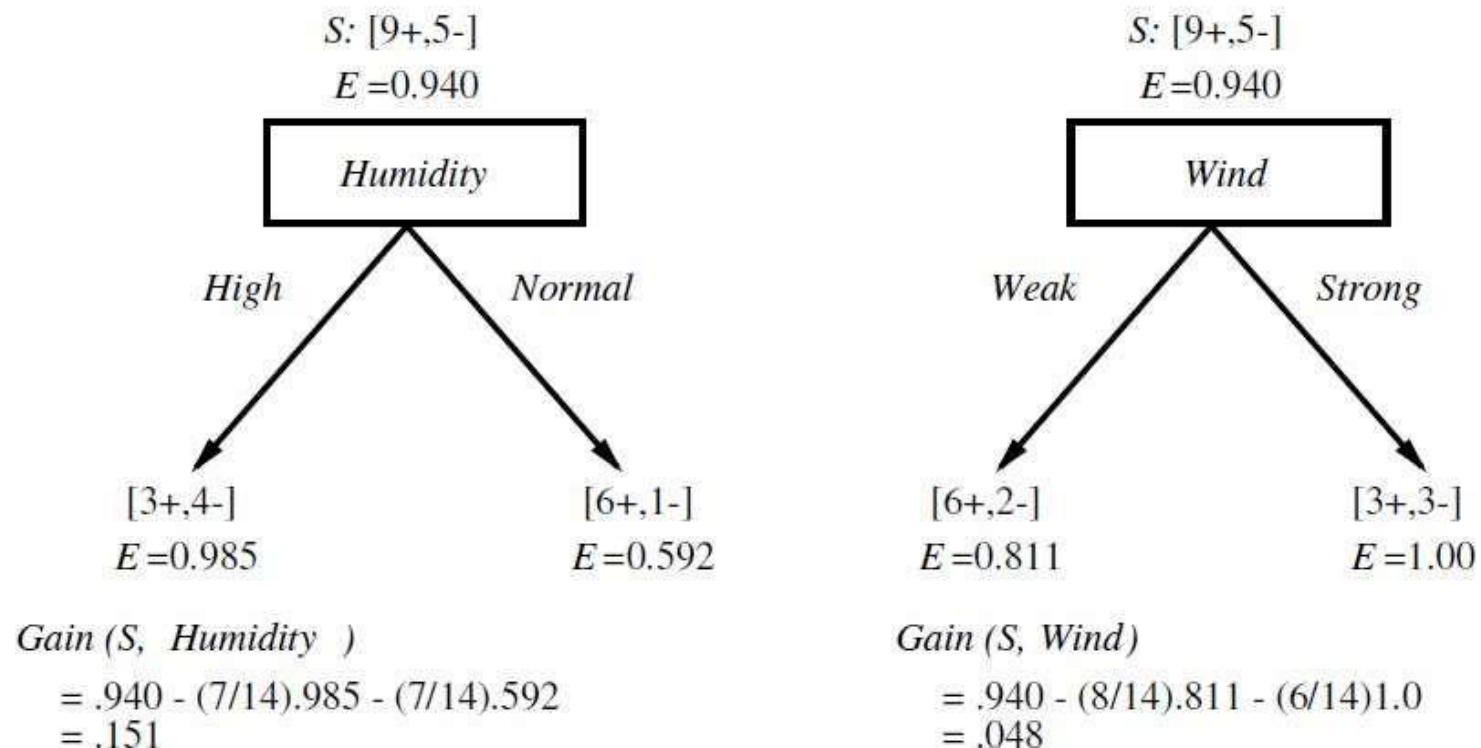
An Illustrative Example

- To illustrate the operation of ID3, consider the learning task represented by the training examples of below table.
- Here the target attribute ***PlayTennis***, which can have values ***yes*** or ***no*** for different days.
- Consider the first step through the algorithm, in which the topmost node of the decision tree is created.

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

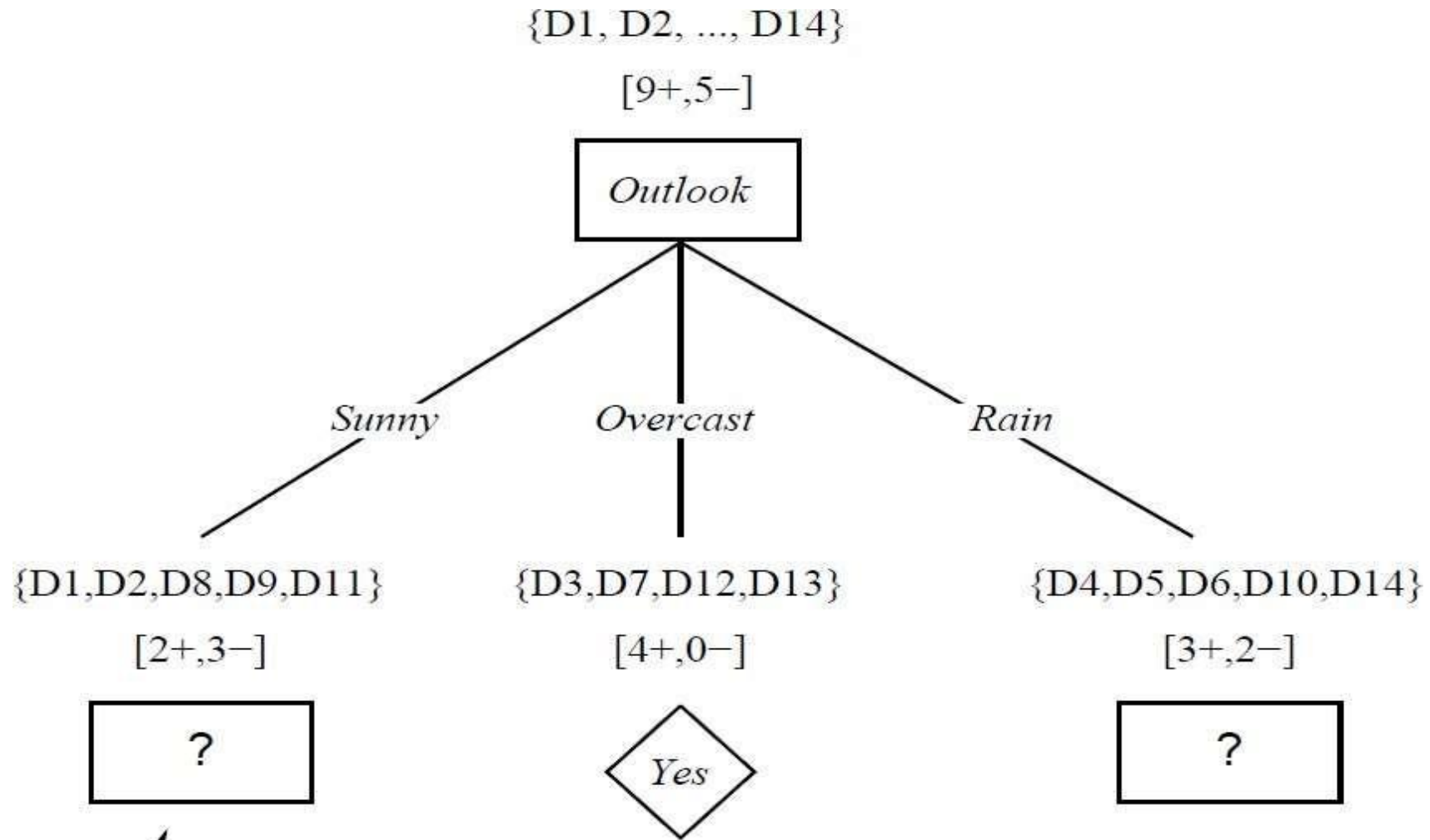
ID3 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain

Which attribute is the best classifier?



The information gain values for all four attributes are

- $\text{Gain}(S, \text{Outlook}) = 0.246$
 - $\text{Gain}(S, \text{Humidity}) = 0.151$
 - $\text{Gain}(S, \text{Wind}) = 0.048$
 - $\text{Gain}(S, \text{Temperature}) = 0.029$
- According to the information gain measure, the ***Outlook*** attribute provides the best prediction of the target attribute, ***PlayTennis***, over the training examples. Therefore, ***Outlook*** is selected as the decision attribute for the root node, and branches are created below the root for each of its possible values i.e., Sunny, Overcast, and Rain.



↗
Which attribute should be tested here?

$$S_{\text{sunny}} = \{D1, D2, D8, D9, D11\}$$

$$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = .970 - (3/5) 0.0 - (2/5) 0.0 = .970$$

$$\text{Gain}(S_{\text{sunny}}, \text{Temperature}) = .970 - (2/5) 0.0 - (2/5) 1.0 - (1/5) 0.0 = .570$$

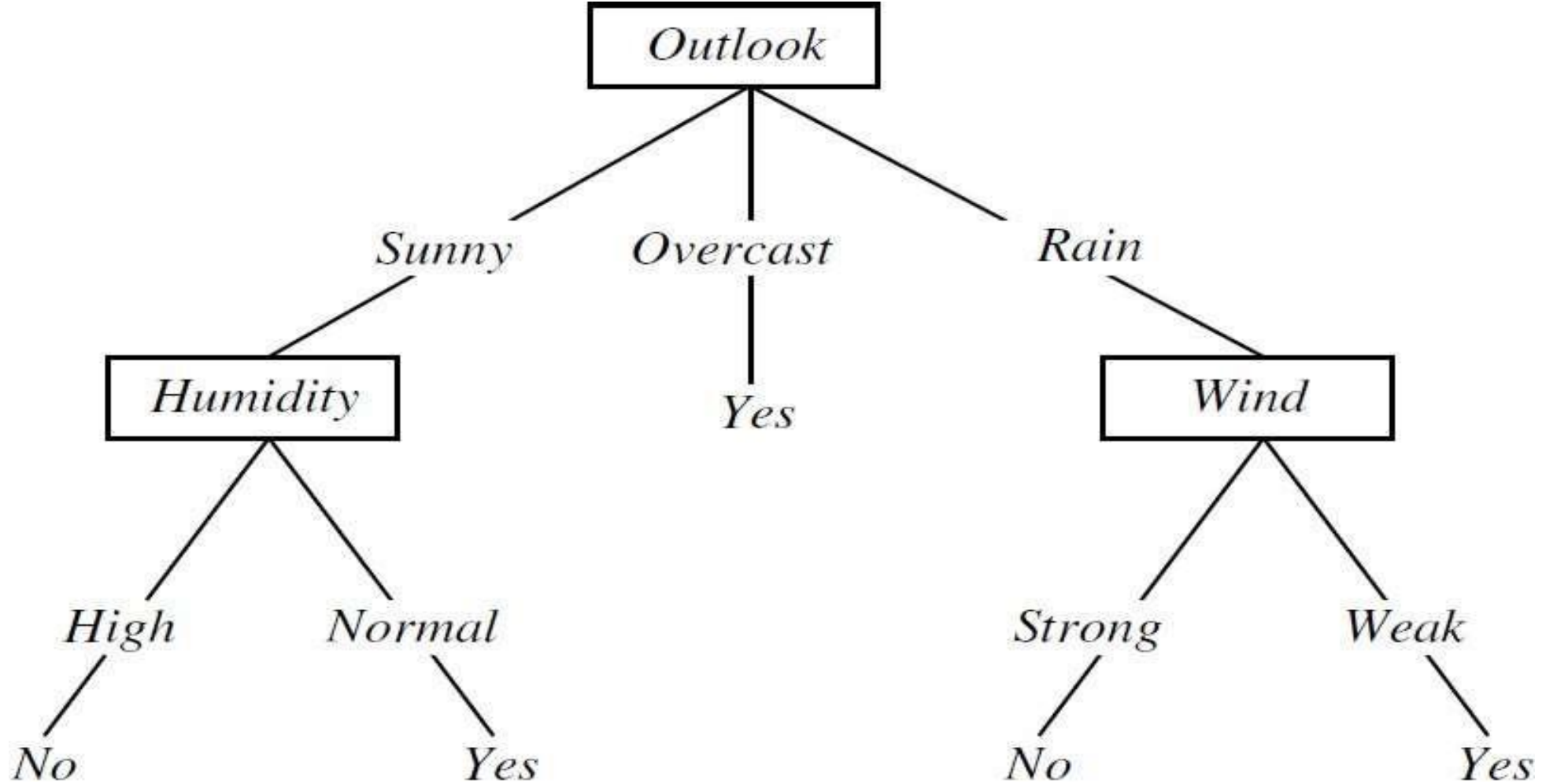
$$\text{Gain}(S_{\text{sunny}}, \text{Wind}) = .970 - (2/5) 1.0 - (3/5) .918 = .019$$

$$S_{\text{Rain}} = \{ D4, D5, D6, D10, D14 \}$$

$$\text{Gain}(S_{\text{Rain}}, \text{Humidity}) = 0.970 - (2/5)1.0 - (3/5)0.917 = 0.019$$

$$\text{Gain}(S_{\text{Rain}}, \text{Temperature}) = 0.970 - (0/5)0.0 - (3/5)0.918 - (2/5)1.0 = 0.019$$

$$\text{Gain}(S_{\text{Rain}}, \text{Wind}) = 0.970 - (3/5)0.0 - (2/5)0.0 = 0.970$$



HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.
- The hypothesis space searched by ID3 is the set of possible decision trees.
- ID3 performs a *simple-to complex, hill-climbing search* through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data

By viewing ID3 in terms of its search space and search strategy, we can get some insight into its capabilities and limitations

1. ID3's hypothesis space of all decision trees is a *complete* space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree
- ID3 avoids one of the major risks of methods that *search incomplete hypothesis spaces* : that the hypothesis space might not contain the target function.

2. ID3 maintains *only a single current hypothesis* as it searches through the space of decision trees.

For example, with the earlier version space candidate elimination method, which maintains the set of *all* hypotheses consistent with the available training examples.

By determining only a single hypothesis, ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.

For example, it does not have the ability to determine how many alternative decision trees are consistent with the available training data, or to pose new instance queries that optimally resolve among these competing hypotheses

3. **ID3** in its pure form performs *no backtracking in its search*. Once it selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice.

- In the case of **ID3**, a locally optimal solution corresponds to the decision tree it selects along the single search path it explores. However, this locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search.

4. **ID3** *uses all training examples at each step* in the search to make statistically based decisions regarding how to refine its current hypothesis.

- One advantage of using statistical properties of all the examples is that the resulting search is much *less sensitive to errors* in individual training examples.
- **ID3** can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

INDUCTIVE BIAS IN DECISION TREE LEARNING

Inductive bias is the set of assumptions that, together with the training data, deductively justify the classifications assigned by the learner to future instances

Given a collection of training examples, there are typically many decision trees consistent with these examples. Which of these decision trees does ID3 choose?

ID3 search strategy

- (a) selects in favour of shorter trees over longer ones
- (b) selects trees that place the attributes with highest information gain closest to the root.

Approximate inductive bias of ID3: Shorter trees are preferred over larger trees

- Consider an algorithm that begins with the empty tree and searches *breadth first* through progressively more complex trees.
- First considering all trees of depth 1, then all trees of depth 2, etc.
- Once it finds a decision tree consistent with the training data, it returns the smallest consistent tree at that search depth (e.g., the tree with the fewest nodes).
- Let us call this breadth-first search algorithm BFS-ID3.
- BFS-ID3 finds a shortest decision tree and thus exhibits the bias "shorter trees are preferred over longer trees."

A closer approximation to the inductive bias of ID3: Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not.

- ID3 can be viewed as an efficient approximation to BFS-ID3, using a greedy heuristic search to attempt to find the shortest tree without conducting the entire breadth-first search through the hypothesis space.
- Because ID3 uses the information gain heuristic and a hill climbing strategy, it exhibits a more complex bias than BFS-ID3.
- In particular, it does not always find the shortest consistent tree, and it is biased to favour trees that place attributes with high information gain closest to the root.

Restriction Biases and Preference Biases

Difference between the types of inductive bias exhibited by ID3 and by the CANDIDATE-ELIMINATION Algorithm.

ID3

- ID3 searches a complete hypothesis space
- It searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met
- Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy. Its hypothesis space introduces no additional bias

CANDIDATE-ELIMINATION Algorithm

- The version space CANDIDATE-ELIMINATION Algorithm searches an incomplete hypothesis space
- It searches this space completely, finding every hypothesis consistent with the training data.
- Its inductive bias is solely a consequence of the expressive power of its hypothesis representation. Its search strategy introduces no additional bias

Restriction Biases and Preference Biases

- The inductive bias of ID3 is a *preference* for certain hypotheses over others (e.g., preference for shorter hypotheses over larger hypotheses), with no hard restriction on the hypotheses that can be eventually enumerated. This form of bias is called a *preference bias* or a *search bias*.
- The bias of the CANDIDATE ELIMINATION algorithm is in the form of a *categorical* restriction on the set of hypotheses considered. This form of bias is typically called a *restriction bias* or a *language bias*.

Which type of inductive bias is preferred in order to generalize beyond the training data, a preference bias or restriction bias?

- A preference bias is more desirable than a restriction bias, because it allows the learner to work within a complete hypothesis space that is assured to contain the unknown target function.
- In contrast, a restriction bias that strictly limits the set of potential hypotheses is generally less desirable, because it introduces the possibility of excluding the unknown target function altogether.

Occam's razor

Occam's razor: is the problem-solving principle that the simplest solution tends to be the right one. When presented with competing hypotheses to solve a problem, one should select the solution with the fewest assumptions.

Occam's razor: “*Prefer the simplest hypothesis that fits the data*”.

Why Prefer Short Hypotheses ?

Argument in favour:

Fewer short hypotheses than long ones:

- Short hypotheses fits the training data which are *less likely* to be *coincident*
- Longer hypotheses fits the training data might be *coincident*.

Many complex hypotheses that fit the current training data but fail to generalize correctly to subsequent data.

Argument opposed:

- There are few small trees, and our priori chance of finding one consistent with an arbitrary set of data is therefore small. The difficulty here is that there are very many small sets of hypotheses that one can define *but understood by fewer learner*.
- The size of a hypothesis is determined by the representation used *internally* by the learner. Occam's razor will produce *two different hypotheses from the same training examples when it is applied by two learners*, both justifying their contradictory conclusions by Occam's razor. On this basis we might be tempted to reject Occam's razor altogether.

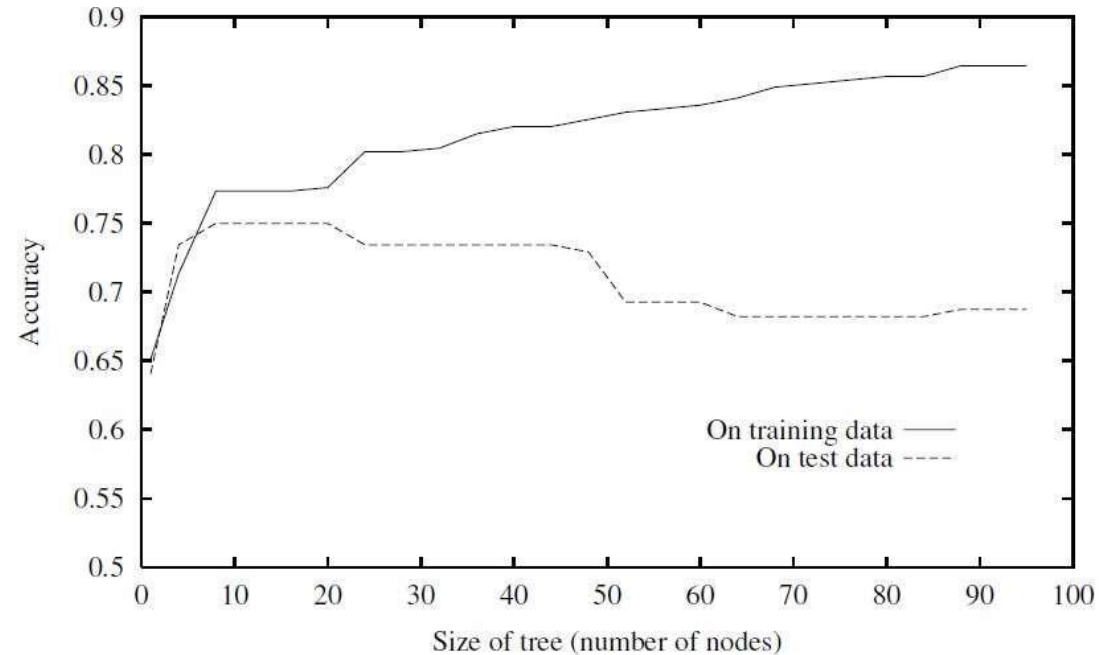
ISSUES IN DECISION TREE LEARNING

1. Avoiding Overfitting the Data
 - Reduced error pruning Rule
 - post-pruning
2. Incorporating Continuous-Valued Attributes
3. Alternative Measures for Selecting Attributes
4. Handling Training Examples with Missing Attribute Values
5. Handling Attributes with Differing Costs

1. Avoiding Overfitting the Data

- The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that *overfit* the training examples.
- **Definition - Overfit:** Given a hypothesis space H , a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

- The below figure illustrates the impact of overfitting in a typical application of decision tree learning.



- The **horizontal axis** of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The **vertical axis** indicates the accuracy of predictions made by the tree.
- The **solid line** shows the accuracy of the decision tree over the training examples. The **broken line** shows accuracy measured over an independent set of test example
- The **accuracy** of the tree over the **training examples increases** monotonically as the tree is grown. The **accuracy** measured over the independent test examples **first increases, then decreases**.

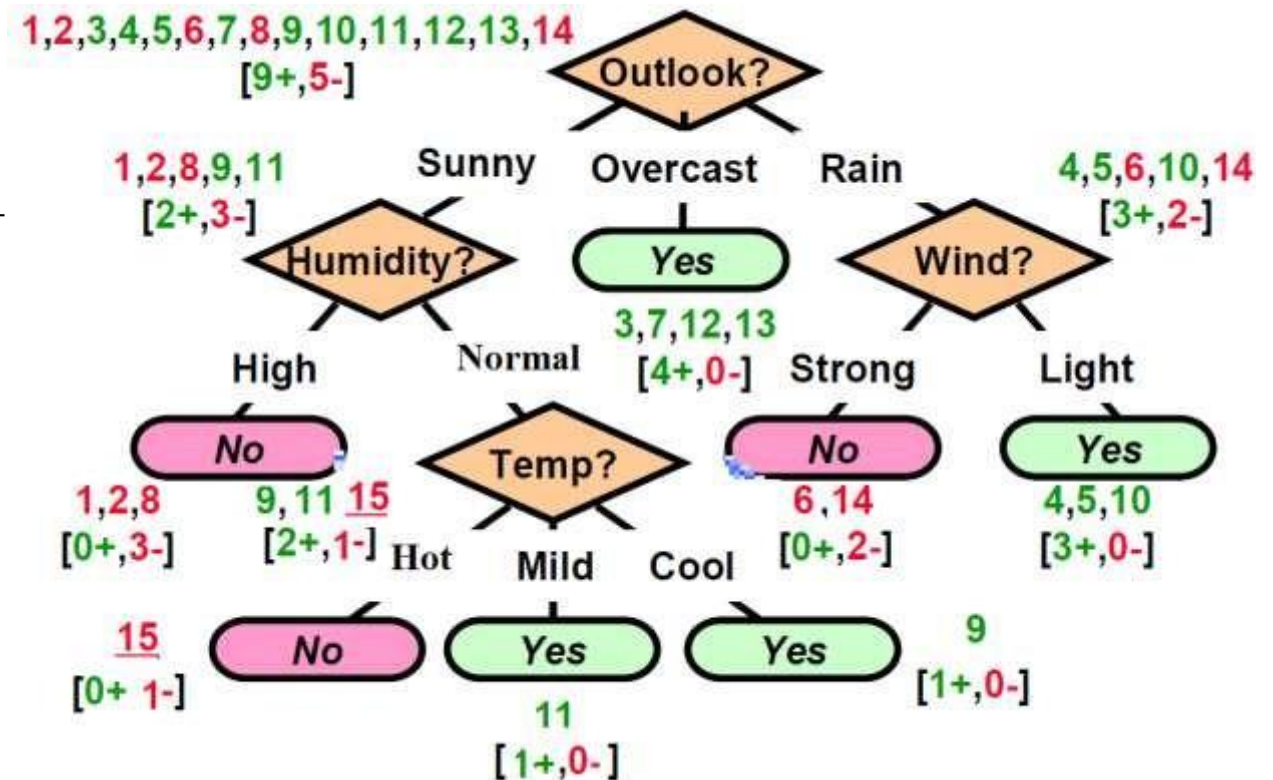
How can it be possible for tree h to fit the training examples better than h' , but for it to perform more poorly over subsequent examples?

1. Overfitting can occur when the training examples contain random errors or noise
2. When small numbers of examples are associated with leaf nodes.

Noisy Training Example

Example 15: $\langle \text{Sunny}, \text{Hot}, \text{Normal}, \text{Strong}, - \rangle$

- Example is noisy because the correct label is +
- Previously constructed tree misclassifies it



Approaches to avoiding overfitting in decision tree learning

- **Pre-pruning (avoidance):** Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- **Post-pruning (recovery):** Allow the tree to overfit the data, and then post-prune the tree

Criterion used to determine the correct final tree size

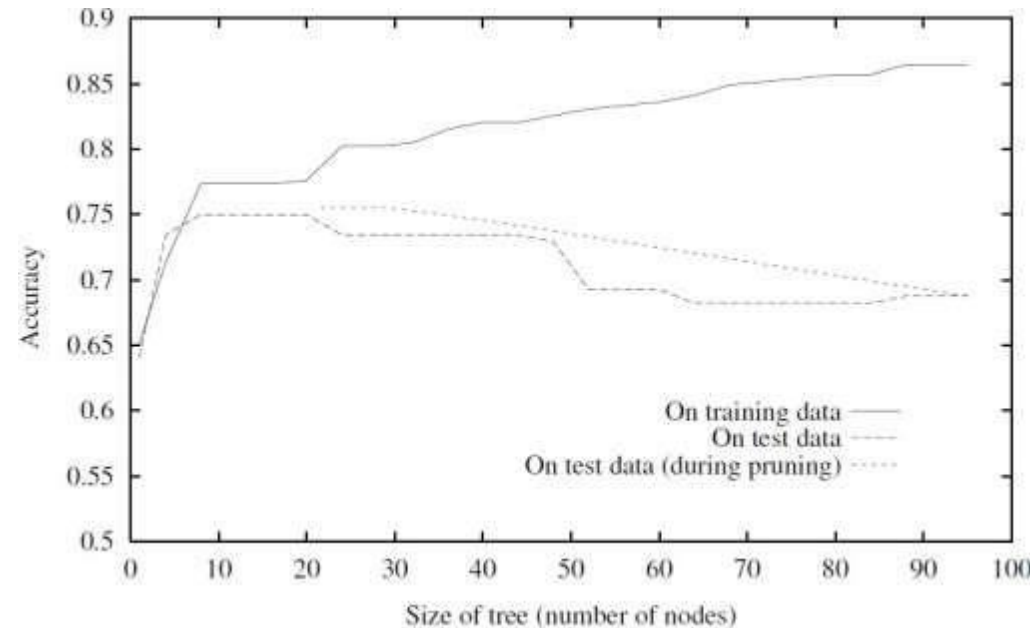
- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree
- Use all the available data for training, but apply *a statistical test* to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set
- Use measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is called the Minimum Description Length

$$MDL - Minimize : size(tree) + size(misclassifications(tree))$$

Reduced-Error Pruning

- ***Reduced-error pruning***, is to consider each of the decision nodes in the tree to be candidates for pruning
- ***Pruning*** a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.
- Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set

The impact of reduced-error pruning on the accuracy of the decision tree is illustrated in below figure



- The additional line in figure shows **accuracy over the test examples** as the tree is pruned. When pruning begins, the tree is at its maximum size and lowest accuracy over the test set. As pruning proceeds, the number of nodes is reduced and accuracy over the test set increases.
- The available data has been split into three subsets: the training examples, the validation examples used for pruning the tree, and a set of test examples used to provide an unbiased estimate of accuracy over future unseen examples. The plot shows accuracy over the training and test sets.

Pros and Cons

Pro: Produces smallest version of most accurate T (subtree of T)

Con: Uses less data to construct T

Can afford to hold out $D_{validation}$?. If not (data is too limited), may make error worse (insufficient D_{train})

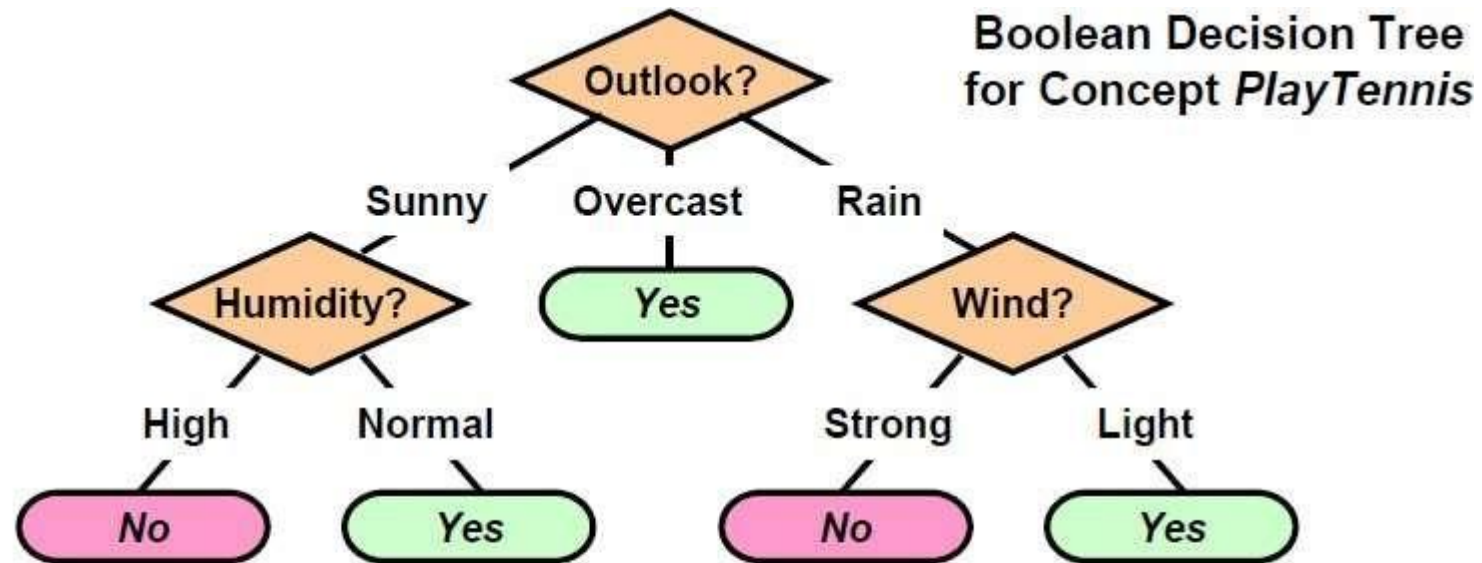
Rule Post-Pruning

Rule post-pruning is successful method for finding high accuracy hypotheses

Rule post-pruning involves the following steps:

1. Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.
2. Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.
3. Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
4. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

Converting a Decision Tree into Rules



Example

- IF (*Outlook* = *Sunny*) \wedge (*Humidity* = *High*) THEN *PlayTennis* = *No*
- IF (*Outlook* = *Sunny*) \wedge (*Humidity* = *Normal*) THEN *PlayTennis* = *Yes*
- ...

For example, consider the decision tree. The leftmost path of the tree in below figure is translated into the rule.

IF (Outlook = Sunny) ^ (Humidity = High)
THEN *PlayTennis* = No

Given the above rule, rule post-pruning would consider removing the preconditions
(Outlook = Sunny) and (Humidity = High)

- It would select whichever of these pruning steps produced the greatest improvement in estimated rule accuracy, then consider pruning the second precondition as a further pruning step.
- No pruning step is performed if it reduces the estimated rule accuracy.

There are three main advantages by converting the decision tree to rules before pruning

- Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path.
- Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves. Thus, it avoids messy bookkeeping issues such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.
- Converting to rules improves readability. Rules are often easier for to understand.

2. Incorporating Continuous-Valued Attributes

Continuous-valued decision attributes can be incorporated into the learned tree.

There are two methods for Handling Continuous Attributes

1. Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.

E.g., {high \equiv Temp $> 35^{\circ}$ C, med $\equiv 10^{\circ}$ C $<$ Temp $\leq 35^{\circ}$ C, low \equiv Temp $\leq 10^{\circ}$ C}

2. Using thresholds for splitting nodes

e.g., $A \leq a$ produces subsets $A \leq a$ and $A > a$

What threshold-based boolean attribute should be defined based on Temperature?

<i>Temperature:</i>	40	48	60	72	80	90
<i>PlayTennis:</i>	No	No	Yes	Yes	Yes	No

Pick a threshold, c , that produces the greatest information gain

- In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of *PlayTennis* changes: $(48 + 60)/2$, and $(80 + 90)/2$. The information gain can then be computed for each of the candidate attributes, Temperature $>_{54}$, and Temperature $>_{85}$ and the best can be selected (Temperature $>_{54}$)

3. Alternative Measures for Selecting Attributes

The problem is if attributes with many values, *Gain* will select it ?

Example: consider the attribute *Date*, which has a very large number of possible values. (e.g., March 4, 1979).

- If this attribute is added to the *PlayTennis* data, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a tree of depth one, which perfectly classifies the training data.
- This decision tree with root node *Date* is not a useful predictor because it perfectly separates the training data, but poorly predict on subsequent examples.

One Approach: Use *GainRatio* instead of *Gain*

- The gain ratio measure penalizes attributes by incorporating a split information, that is sensitive to how broadly and uniformly the attribute splits the data

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv - \sum_{i=1}^c \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- where S_i is subset of S , for which attribute A has value v_i

4. Handling Training Examples with Missing Attribute Values

The data which is available may contain missing values for some attributes

Example: Medical diagnosis

- $\langle \textit{Fever} = \textit{true}, \textit{Blood-Pressure} = \textit{normal}, \dots, \textit{Blood-Test} = ?, \dots \rangle$
- Sometimes values truly unknown, sometimes low priority (or cost too high)

Example : PlayTennis

Day	Outlook	Temperature	Humidity	Wind	PlayTennis?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

Strategies for dealing with the missing attribute value

- If node n test A , assign most common value of A among other training examples sorted to node n
- Assign most common value of A among other training examples with same target value
- Assign a probability p_i to each of the possible values v_i of A rather than simply assigning the most common value to $A(x)$

5. Handling Attributes with Differing Costs

In some learning tasks the instance attributes may have associated costs.

For example:

- In learning to classify medical diseases, the patients described in terms of attributes such as Temperature, BiopsyResult, Pulse, BloodTestResults, etc.
- These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort
- Decision trees use low-cost attributes where possible, depends only on high-cost attributes only when needed to produce reliable classifications

How to Learn A Consistent Tree with Low Expected Cost?

One approach is replace Gain by *Cost-Normalized-Gain*

Examples of normalization functions

- Tan and Schlimmer

$$\frac{Gain^2(S, A)}{Cost(A)}.$$

- Nunez

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

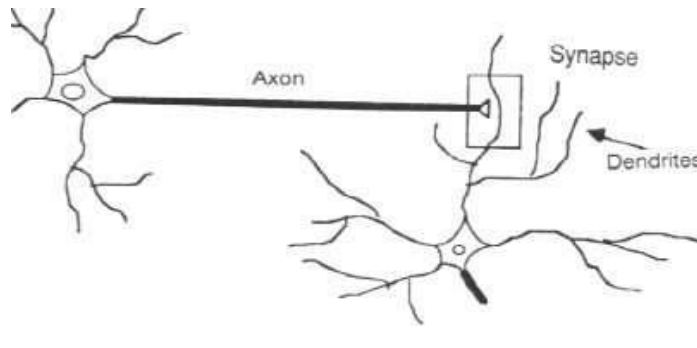
where $w \in [0, 1]$ determines importance of cost

Artificial Neural Networks

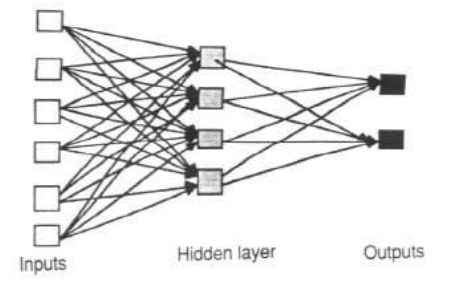
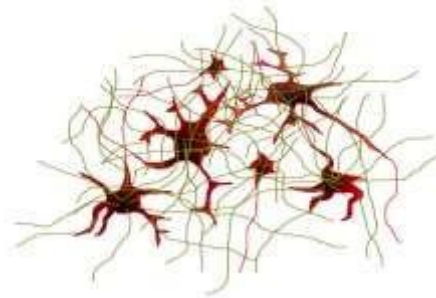
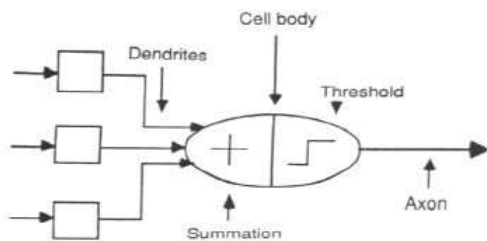
Overview

1. Introduction
2. ANN representations
3. Perceptron Training
4. Gradient Descent and Delta Rule
5. Multilayer networks and Backpropagation algorithm
6. Remarks on the backpropagation algorithm
7. An illustrative example: face recognition
8. Advanced topics in artificial neural networks

Introduction



- Human brain : densely interconnected network of 10^{11} neurons each connected to 10^4 others (neuron switching time : approx. 10^{-3} sec.)



- Properties of artificial neural nets (ANN's):
 - Many neuron-like threshold switching units
 - Many weighted interconnections among units
 - Highly parallel, distributed process

Appropriate problems for neural network learning

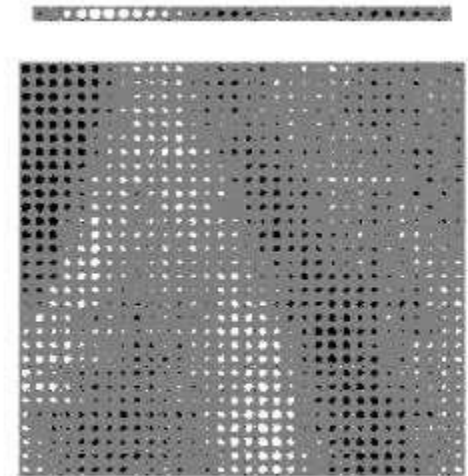
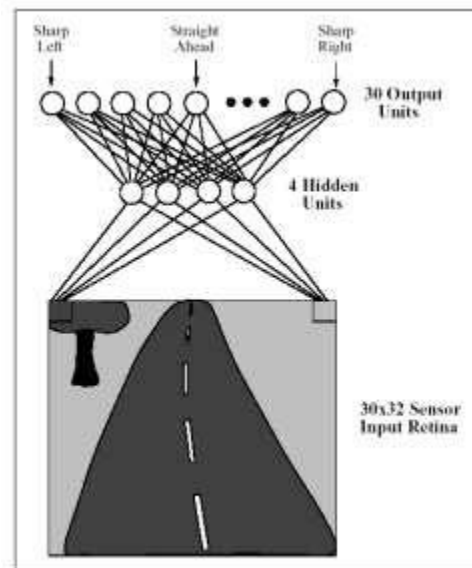
- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Long training times accepted
- Fast evaluation of the learned function required.
- Not important for humans to understand the weights

Examples:

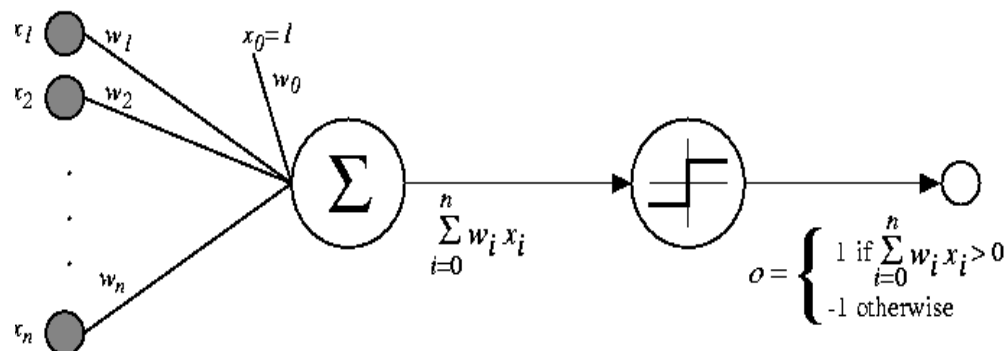
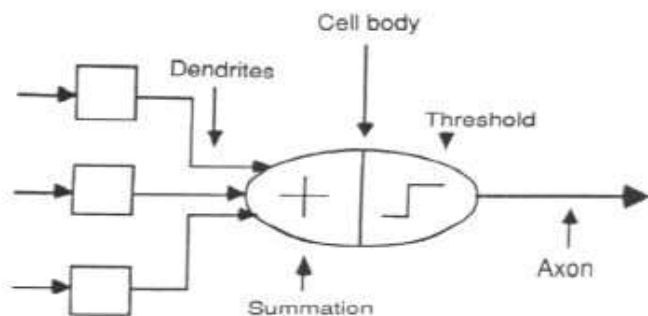
- Speech phoneme recognition
- Image classification
- Financial prediction

Appropriate problems for neural network learning

- ALVINN drives 70 mph on highways
- The ALVINN system uses backpropagation algorithm to learn to steer an autonomous vehicle driving at speeds up to 70 miles per hour



Perceptron



- Input values \rightarrow Linear weighted sum \rightarrow Threshold

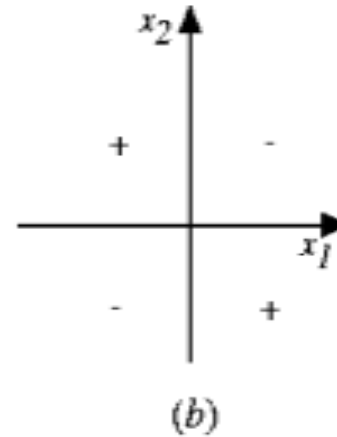
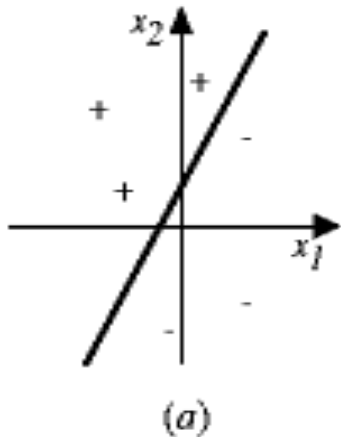
$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Decision surface of a perceptron

- Representational power of perceptrons
 - Linearly separable case like (a) :
possible to classify by hyperplane,
 - Linearly inseparable case like (b) :
impossible to classify



Perceptron training rule (delta rule)

$$w_i \leftarrow w_i + \Delta w_i$$
$$\text{where } \Delta w_i = \eta (t - o) x_i$$

Where:

- $t = c(x)$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

Can prove it will converge

- If training data is linearly separable

Gradient descent

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1x_1 + \cdots + w_nx_n$$

Let's learn w_i 's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Derivation of gradient descent

◆ Gradient descent

- Error (for all training examples.): $E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$

- the *gradient* of E (**partial differentiating**) :

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- direction : steepest increase in E.

- Thus, training rule is as follows.

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}] \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

(The negative sign : the direction that *decreases* E)

Derivation of gradient descent

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)\end{aligned}$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

where x_{id} denotes the single input components x_i for training example d

- The weight update rule for gradient descent

$$\therefore \Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{i,d}$$

Gradient descent and delta rule

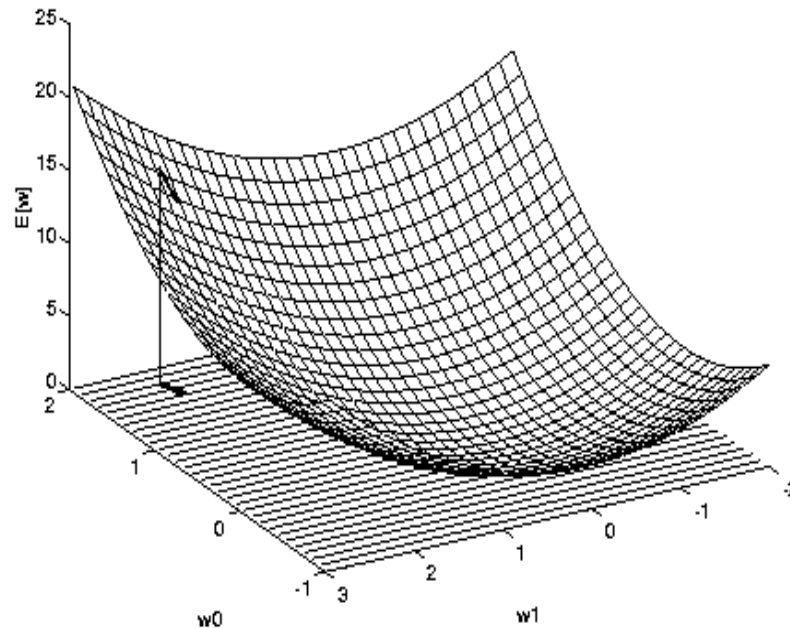
GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - * Input the instance \vec{x} to the unit and compute the output o
 - * For each linear unit weight w_i , Do
$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$
 - For each linear unit weight w_i , Do
$$w_i \leftarrow w_i + \Delta w_i$$

◆ Because the error surface contains only a single global minimum, this algorithm will converge to a weight vector with minimum error, given a sufficiently small η is used

Hypothesis Space



- Error of different hypotheses
- For a linear unit with two weights, the hypothesis space H is the w_0, w_1 plane.
- This error surface must be parabolic with a single global minimum (we desire a hypothesis with minimum error).

Stochastic approximation to gradient descent

Batch mode Gradient Descent:

Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

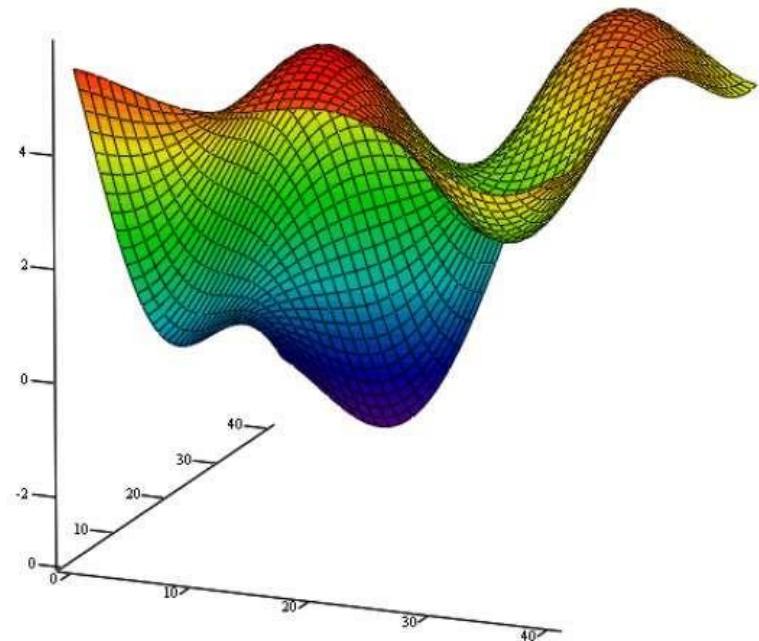
Incremental mode Gradient Descent:

Do until satisfied

- For each training example d in D
 1. Compute the gradient $\nabla E_d[\vec{w}]$
 2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$



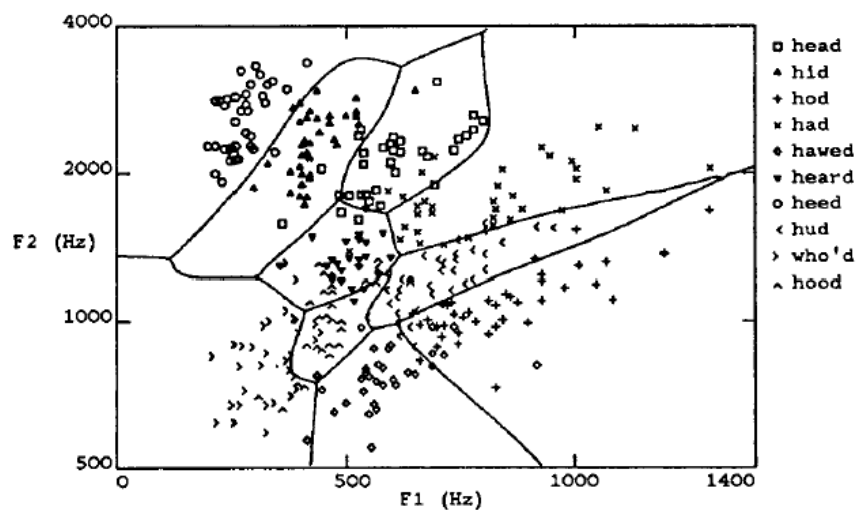
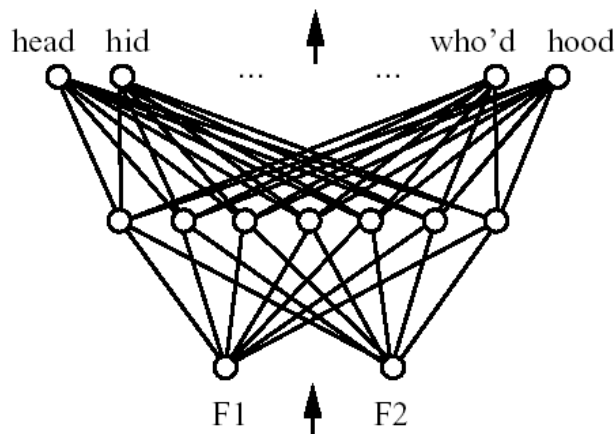
- Stochastic gradient descent (i.e. incremental mode) can sometimes avoid falling into local minima because it uses the various gradient of E rather than overall gradient of E .

Summary

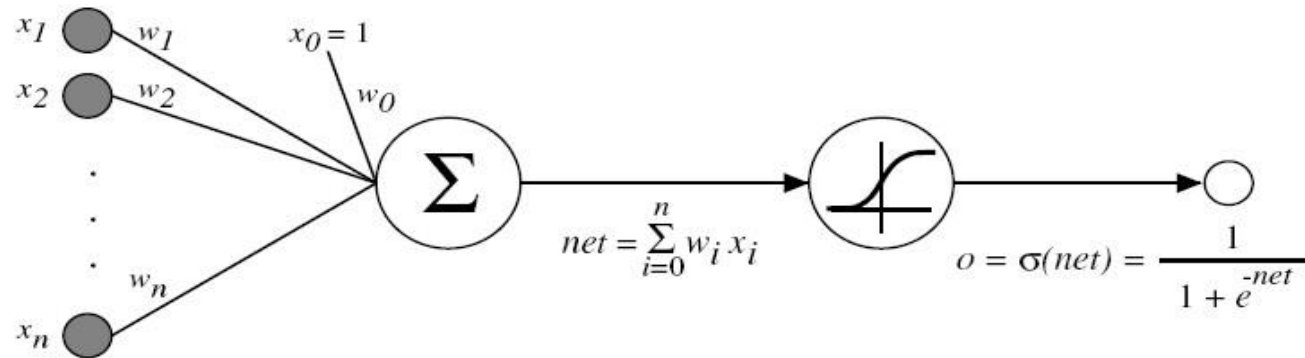
- Perceptron training rule guaranteed to succeed if
 - training examples are linearly separable
 - Sufficiently small learning rate η
- Linear unit training rule using gradient descent
 - Converge asymptotically to min. error hypothesis
(Guaranteed to converge to hypothesis with minimum squared error)

Multilayer networks and the backpropagation algorithm

- ◆ Speech recognition example of multilayer networks learned by the backpropagation algorithm
- ◆ **Highly nonlinear decision surfaces**



Sigmoid Threshold Unit



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units \rightarrow Backpropagation

The Backpropagation algorithm

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do

1. Input the training example to the network and compute the network outputs

2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h

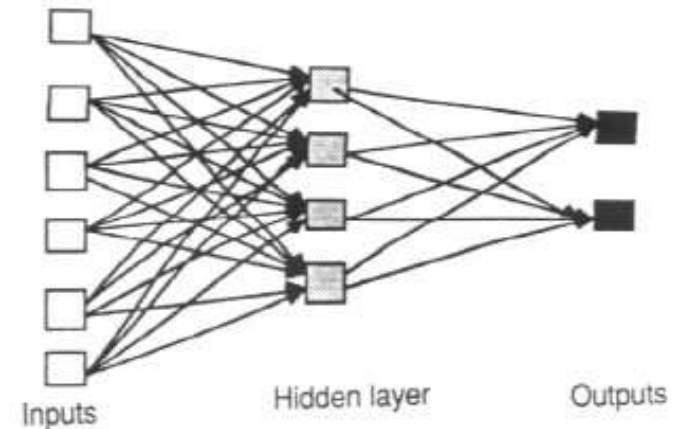
$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k$$

4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$



Adding Momentum

- ◆ Often include weight *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- n^{th} iteration update depend on $(n-1)^{\text{th}}$ iteration
- α : constant between 0 and 1 (momentum)

□ Roles of momentum term

- ◆ The effect of keeping the ball rolling through small local minima in the error surface
- ◆ The effect of gradually increasing the step size of the search in regions (greatly improves the speed of learning)

Convergence and Local Minima

- ◆ Gradient descent to some local minimum
 - Perhaps not global minimum...
 - Add momentum
 - Stochastic gradient descent

Expressive Capabilities of ANNs

Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

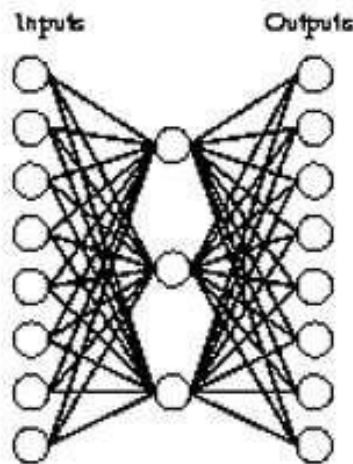
Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Hidden layer representations

Hidden layer representations

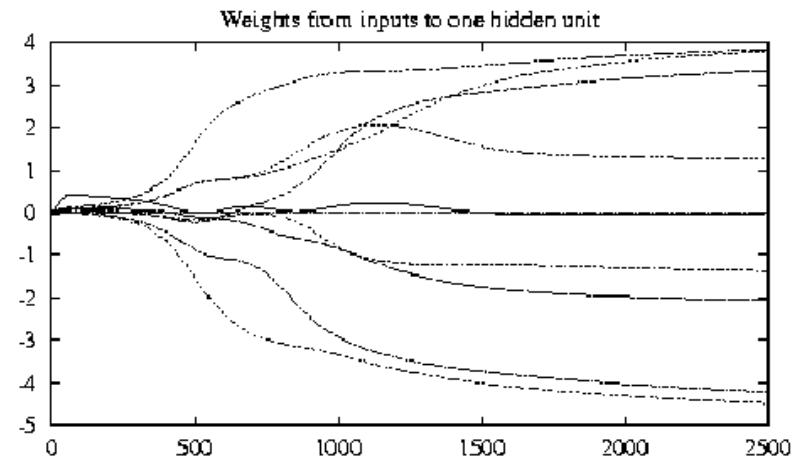
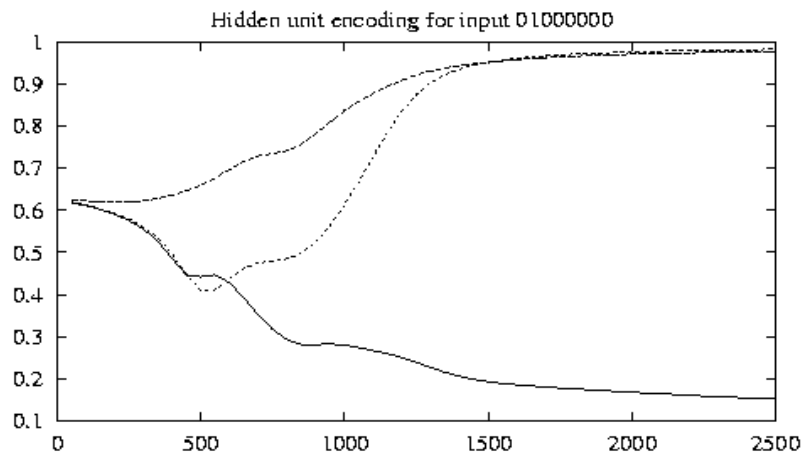
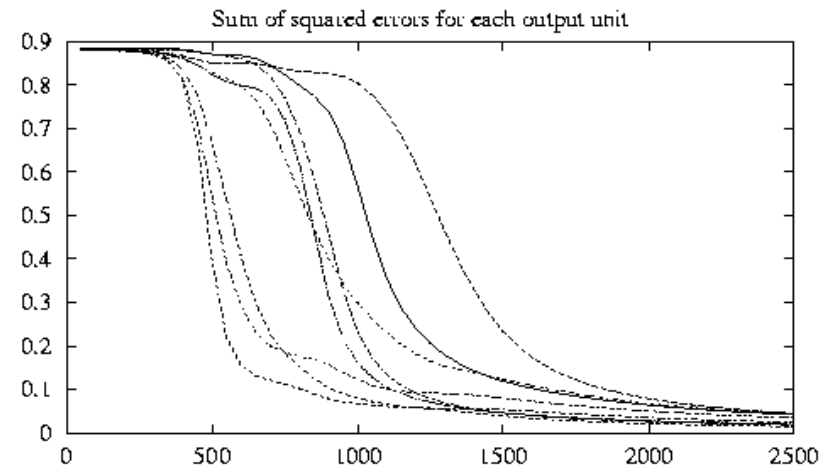
- This 8x3x8 network was trained to learn the identity function.
- 8 training examples are used.
- After 5000 training iterations, the three hidden unit values encode the eight distinct inputs using the encoding shown on the right.



Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001

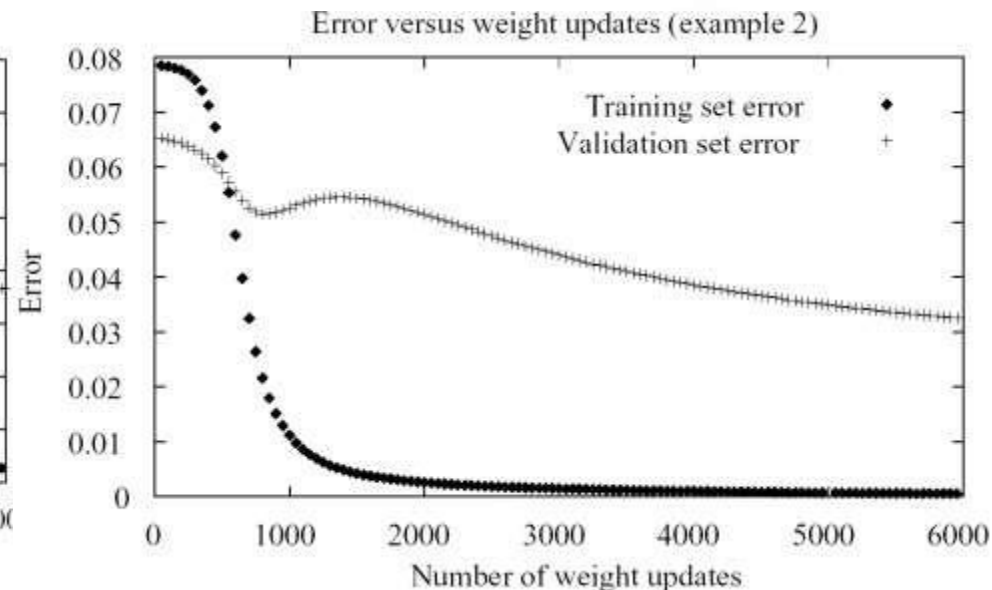
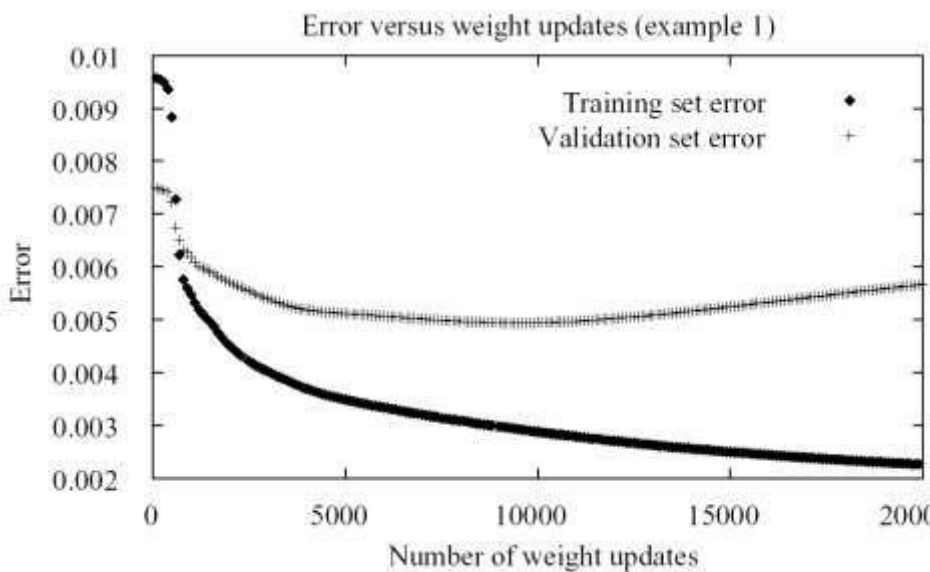
Learning the 8x3x8 network

- Most of the interesting weight changes occurred during the first 2500 iterations.



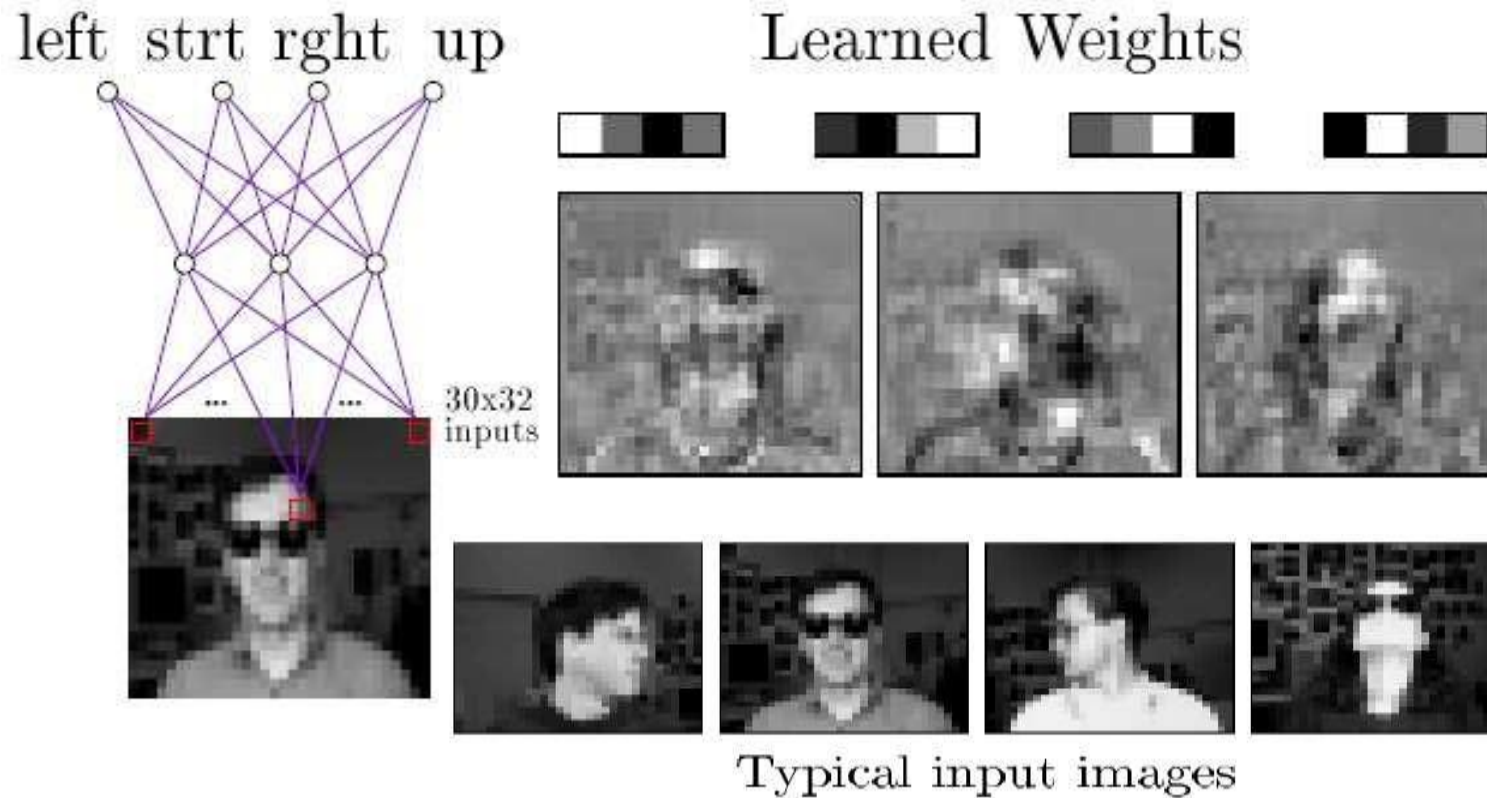
Generalization, Overfitting, and Stopping Criterion

- Termination condition
 - Until the error E falls below some predetermined threshold
- Techniques to address the overfitting problem
 - Weight decay : Decrease each weight by some small factor during each iteration.
 - Cross-validation (k-fold cross-validation)



Neural Nets for Face Recognition

(<http://www.cs.cmu.edu/tom/faces.html>)



- Training images : 20 different persons with 32 images per person.
- After 260 training images, the network achieves an accuracy of 90% over test set.
- Algorithm parameters : $\eta=0.3$, $\alpha=0.3$

Alternative Error Functions

- Penalize large weights: (weight decay) : Reducing the risk of overfitting

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

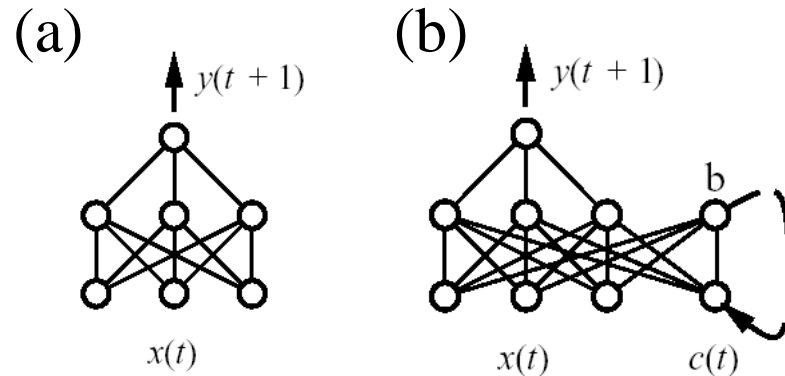
- Train on target slopes as well as values:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} \left[(t_{kd} - o_{kd})^2 + \mu \sum_{j \in \text{inputs}} \left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

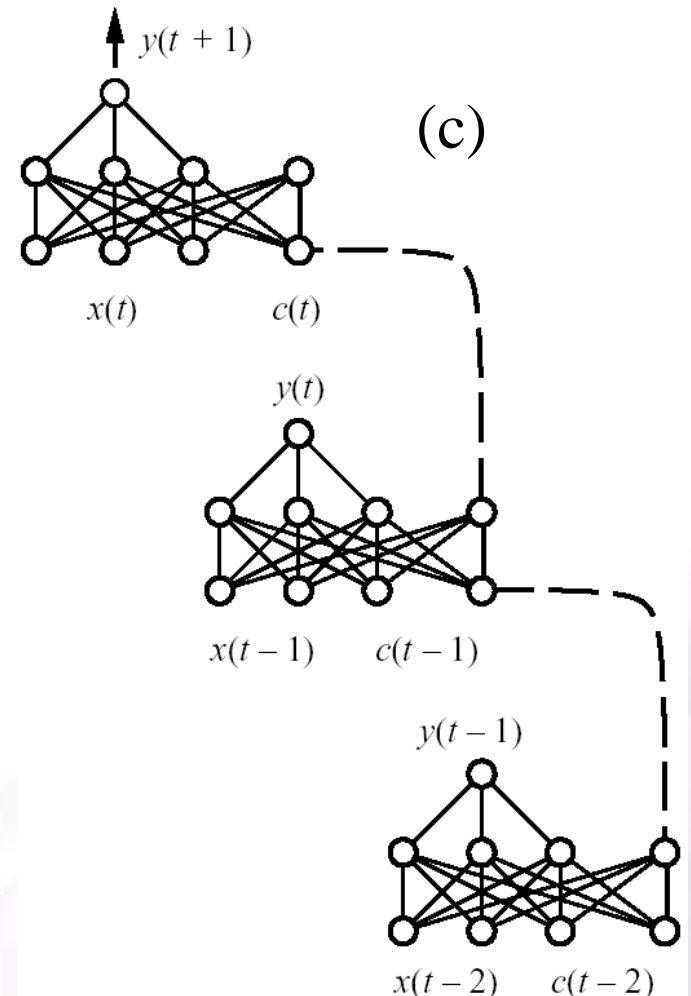
- Minimizing the cross entropy : Learning a probabilistic output function (chapter 6)

$$- \sum_{d \in D} t_d \log o_d + (1 - t_d) \log(1 - o_d)$$

Recurrent Networks



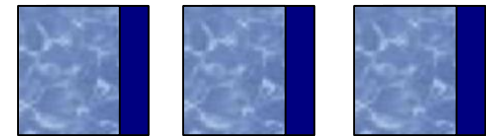
- (a) Feedforward network
- (b) Recurrent network
- (c) Recurrent network unfolded in time



Dynamically Modifying Network Structure

- To improve generalization accuracy and training efficiency
- Cascade-Correlation algorithm (Fahlman and Lebiere 1990)
 - Start with the simplest possible network (no hidden units) and add complexity
- Lecun *et al.* 1990
 - Start with the complex network and prune it as we find that certain connectives are inessential.

Evaluating Hypotheses



Context

→ Motivation

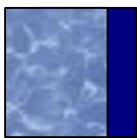
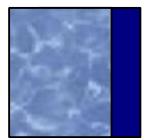
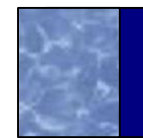
' Estimating Hypothesis Accuracy

' Basics of Sampling Theory

' Difference in Error of Two Hypotheses

' Comparing Learning Algorithms

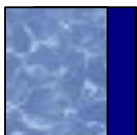
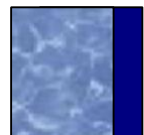
' Summary



Motivation

Goal: Introduction to statistical methods for estimating hypothesis accuracy, focusing on the followings:

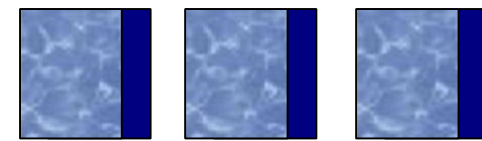
- ✓ Given the observed accuracy of a hypothesis over a limited sample of data, how well does this estimate its accuracy over additional examples?
- ✓ Given that one hypothesis outperforms another over some sample of data, how probable is it that this hypothesis is more accurate in general?
- ✓ When data is limited what is the best way to use this data to both learn a hypothesis and estimate its accuracy?



Motivation 2

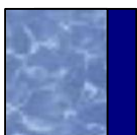
It is important to evaluate the performance of the learned hypotheses as precisely as possible:

- ✓ To understand whether to use the hypothesis
 - ✗ Example: Learning from limited-size database indicating the effectiveness of different medical treatments
- ✓ Evaluating hypotheses is an integral component of many learning methods
 - ✗ Example: in post-pruning decision trees to avoid overfitting
- ✓ Methods for comparing the accuracy of two hypotheses
- ✓ Methods for comparing two learning algorithms when only limited data is available



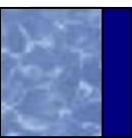
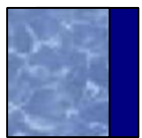
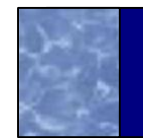
Motivation 3

- ' Estimating the accuracy of hypothesis is relatively straightforward when data is plentiful.
- ' Given only a limited set of data, two key difficulties arise:
 - ✓ *Bias in the estimate:*
 - ✗ Observed accuracy of the learned hypothesis over the training examples is often a poor estimator of its accuracy over future examples.
 - ✗ To obtain an unbiased estimate of future accuracy, we typically test the hypothesis on some set of test examples chosen independently of training examples and the hypothesis.
 - ✓ *Variance in the estimate:*
 - ✗ The measured accuracy can still vary from the true accuracy, depending on the makeup of the particular set of test examples.



Context

- ' Motivation
- Estimating Hypothesis Accuracy
 - ✓ Sample Error and True Error
- ' Basics of Sampling Theory
- ' Difference in Error of Two Hypotheses
- ' Comparing Learning Algorithms
- ' Summary



Estimating Hypothesis Accuracy

Setting:

- ✓ Some set of possible instances X over which various target functions may be defined
- ✓ Different instances in X may be encountered with different frequencies:
 - ✗ Unknown the probability distribution D that defines the probability of encountering each instance in X
 - ✗ D says nothing about whether x is a positive or a negative example
- ✓ Learning task: Learn target concept or target function f by considering a space H of possible hypotheses
- ✓ Training examples are provided to the learner by a trainer
 - ✗ who gives each instance independently
 - ✗ according to the distribution D ,
 - ✗ then forwards the instance x along with its correct target value $f(x)$ to the learner

Sample Error and True Error

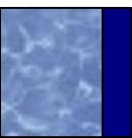
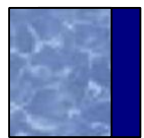
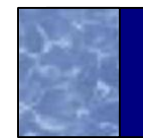
' The sample error of a hypothesis with respect to some sample S of instances given from X is the fraction of S that it misclassifies:

✓ **Def:** The sample error of a hypothesis h with respect to the target function f and data sample S is

$$error_S(h) = \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where

- ' n is the number of examples in S ,
- ' the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$ and 0 otherwise



Sample Error and True Error 2

The true error of a hypothesis is the probability that it will misclassify a single randomly given instance from the distribution D .

- ✓ **Def:** The true error of hypothesis h with respect to target function f and distribution D , is the probability that h will misclassify an instance drawn at random according to D

$$error_D(h) = Pr_{x \sim D}(f(x) \neq h(x))$$

Here the notation $Pr_{x \sim D}$ denotes that the probability is taken over the instance distribution D .

To wish to know is the true error $error_D(h)$.

Main question: How good is an estimate of $error_D(h)$ provided by $error_S(h)$?



Context

- ' Motivation
- ' Estimating Hypothesis Accuracy
- Basics of Sampling Theory
 - ✓ Error Estimation and Estimating Binomial Proportions
 - ✓ The Binomial Distribution
 - ✓ Mean and Variance
 - ✓ Confidence Intervals
 - ✓ Two-Sided and One-Sided Bounds
- ' Difference in Error of Two Hypotheses
- ' Comparing Learning Algorithms
- ' Summary



Basics of Sampling Theory

- ' **Question:** How does the derivation between sample error and true error depend on the size of the data sample?
- ' Equal with the statistical problem: The problem of estimating the proportion of a population that exhibits some property, given the observed proportion over some random sample of the population .
- ' **Here:** The property of interest is that h *misclassifies the example*
- ' **Answer:**
 - ✓ When measuring the sample error we are performing an experiment with a random outcome.
 - ✓ Repeating this experiment many times, each time drawing a different random sample set S_i of size n , we would expect to observe different values for the various $error_{S_i}(h)$ depending on random differences in the makeup of the various S_i
 - ✓ In such cases $error_{S_i}(h)$ the outcome of the i^{th} such experiment is a **random variable**

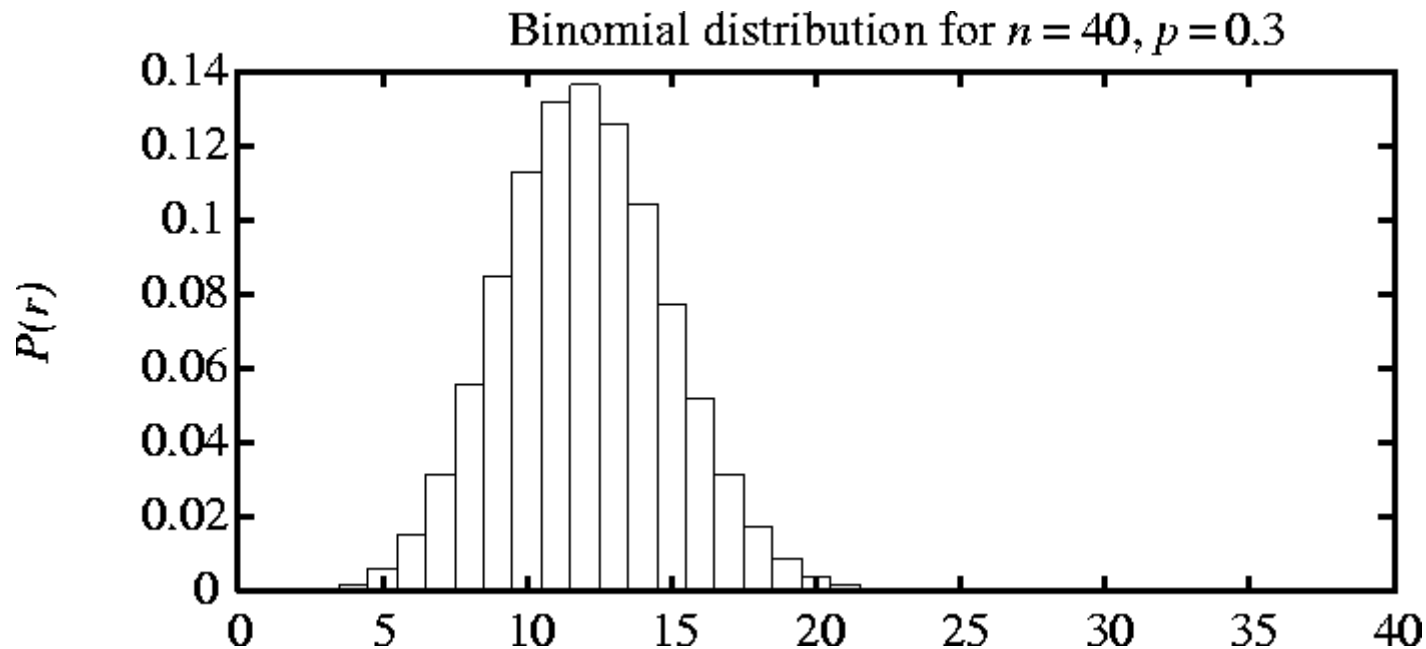


Error Estimation and Estimating Binomial Proportions 2

Imagine:

- ✓ Run k random experiments,
- ✓ Measuring the random variables $error_{S_1}(h), error_{S_2}(h) \dots error_{S_k}(h)$
- ✓ Plot a histogram displaying the frequency with which we observed each possible error value

Result: histogram



The Binomial Distribution

General setting to which the Binomial distribution applies:

- ✓ There is a base or underlying experiment whose outcome can be described by a random variable, say Y . It can take on two possible values.
- ✓ The probability that $Y=1$ on any single trial of the underlying experiment is given by some constant p , independent of the outcome of any other experiment.

The probability that $Y=0$ is therefore $1-p$.

Typically, p is not known in advance, and the problem is to estimate it.

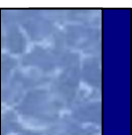
- ✓ A series of n independent trials of the underlying experiment is performed, producing the sequence of independent, identically distributed random variables Y_1, Y_2, \dots, Y_n

Let R denote the number of trials for which $Y_i = 1$ in this series of n experiments

$$R = \sum_{i=1}^n Y_i$$

- ✓ The probability that R will take on a specific value r is given by the Binomial distribution:

$$Pr(R=r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$



Mean and Variance

Def: Consider $Y = \{y_1, y_2, \dots, y_n\}$ The **expected value** of Y , $E[Y]$, is

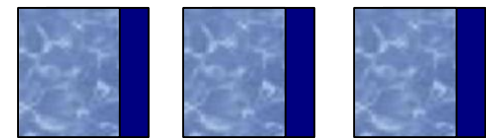
$$E[Y] = \sum_{i=1}^n$$

Example: If Y takes on the value 1 with probability 0.7 and the value 2 with probability 0.3 then its expected value is

$$1 \cdot 0.7 + 2 \cdot 0.3 = 1.3$$

In case of a random variable Y governed by a Binomial distribution the expected value is:

$$E[Y] = n \cdot p$$



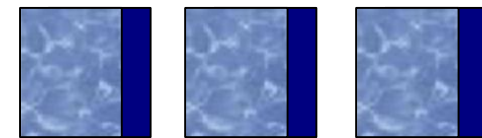
Mean and Variance 2

- Variance captures the „width“ or „spread“ of the probability distribution; that is it captures how far the random variable is expected to vary from its mean value
- Def:** The **variance** of Y , $Var[Y]$, is

$$Var[Y] = E[(Y - E[Y])^2]$$
- The square root of the variance is called the standard deviation of Y , denoted by σ_Y
- Def:** The **standard deviation** of a random variable Y , σ_Y is

$$\sigma_Y = \sqrt{E[(Y - E[Y])^2]}$$
- In case of a random variable Y governed by Binomial distribution the variance and the standard deviation are defined as follows:

$$\sigma_Y = \sqrt{n p (1 - p)}$$



Confidence Intervals

Describe:

- ✓ Give an interval within which the true value is expected to fall, along with the probability with which it is expected to fall into this interval

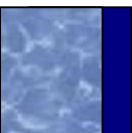
Def: An $N\%$ confidence interval for some parameters p is an interval that is expected with probability $N\%$ to contain p .

How confidence intervals for $error_D\{h\}$ can be derived:

- ✓ Binomial probability distribution governing the estimator $error_S\{h\}$
- ✓ The mean value of distribution is $error_D\{h\}$
- ✓ Standard deviation is $\sigma_{error_S\{h\}} \approx \sqrt{\frac{error_S\{h\}(1 - error_S\{h\})}{n}}$

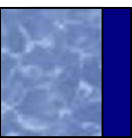
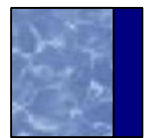
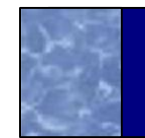
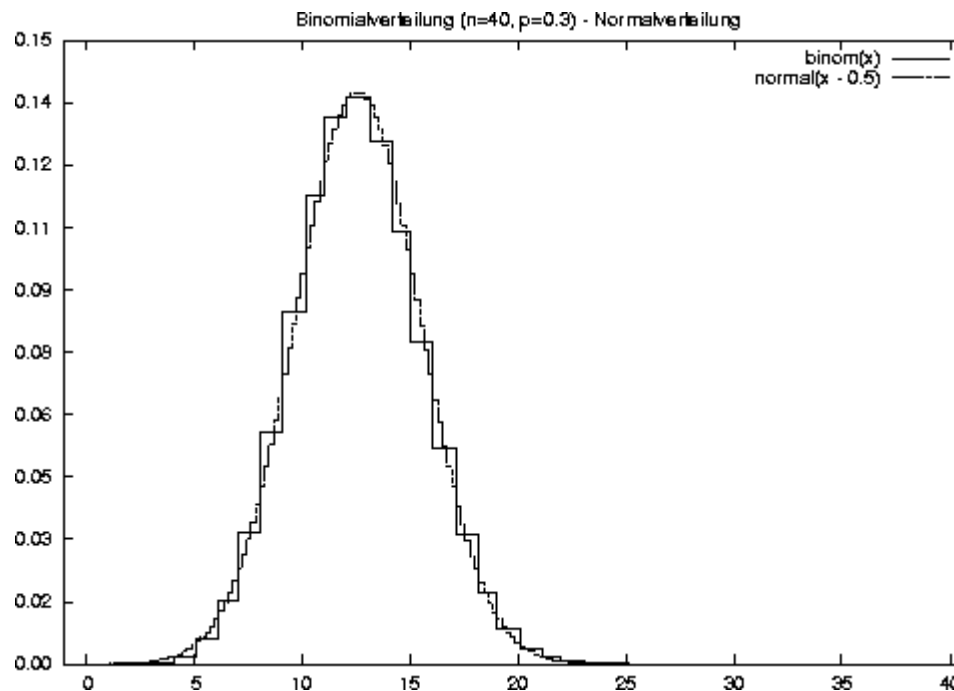
Goal: Derive a 95% confidence interval \Rightarrow

find the interval centered around the mean value $error_D\{h\}$, which is wide enough to contain 95% of total probability under this distribution



Confidence Intervals 2

- **Question:** How can the size of interval that contains $N\%$ of the probability mass be found for given N ?
- **Problem:** Unfortunately for the Binomial distribution this calculation can be quite tedious.
- **But:** Binomial distribution can be closely approximated by Normal distribution



Confidence Intervals 3

Normal or gaussian distribution is a bell-shaped distribution defined by the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

If the random variable X follows a normal distribution then:

✓ The probability that X will fall into the interval (a,b) is given by

$$\int_a^b p(X) dx$$

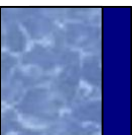
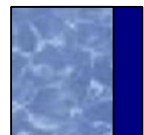
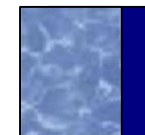
✓ The expected, or mean value of X , $E[X]$, is

$$E[X] = \mu$$

✓ The variance of X , $\text{Var}(X)$ is

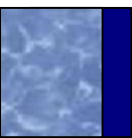
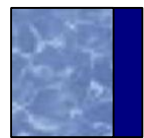
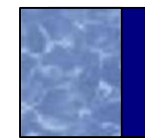
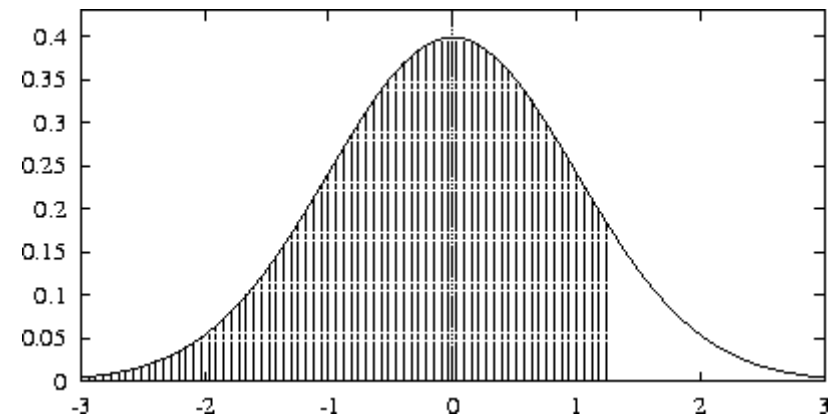
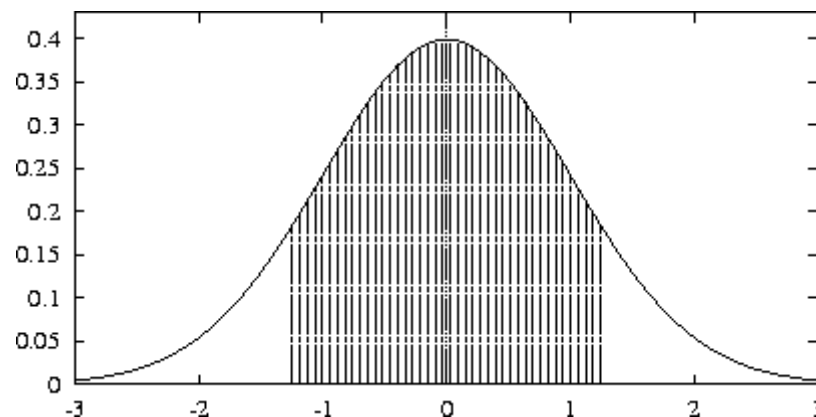
$$\text{Var}(X) = \sigma^2$$

✓ The standard deviation of X ,

$$\sigma_X = \sigma$$


Two-Sided and One-Sided Bounds

- **Two-sided bound:** It bounds the estimated quantity from above and below
- **One-sided bound:** If we are interested in questions like: What is the probability that $error_D(h)$ is at most U



Two-Sided and One-Sided Bounds 2

If the sample error is considered as normal distributed indicating that:

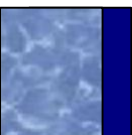
✓ the $error_D(h)$ couches with N% probability in the interval

$$error_D(h) \pm z_N \sqrt{\frac{error_S(h) + error_S(h)}{n}}$$

where z_N is a constant

Confidence level N%	50.00%	68.00%	80.00%	90.00%	95.00%	98.00%	99.00%
Constant	0.67	1	1.28	1.64	1.96	2.33	2.58

Table 1: Values of z_N for two sided N% confidence intervals



Two-Sided and One-Sided Bounds 3

Example:

✓ $n = 50$

✓ Hypothesis h makes $r = 16$ errors $\Rightarrow error_S(h) = \frac{16}{50}$

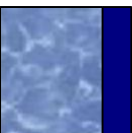
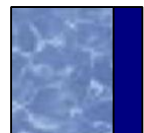
✓ Using the values from Table 1

✗ With 99% probability is $error_D(h)$ in the interval

$$0.32 \pm 2.58 \sqrt{\frac{0.32 \cdot 0.68}{50}}$$

✗ If the numbers of errors is 12 then $error_D(h)$ is in the interval with 50% probability

$$0.24 \pm 0.67 \sqrt{\frac{0.24 \cdot 0.76}{50}} \approx 0.24 \pm 0.04$$



Two-Sided and One-Sided Bounds 4

One-sided error bound

It can be computed with half of the probability of the error from normal distributed two-sided error bound

Example:

✓ h delivers 12 errors, $n = 40$

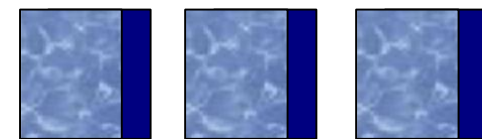
✓ It leads to a (two sided) 95% confidence interval of $0.30 \pm 0.14 \Rightarrow$

✓ In this case $100(1 - \alpha) = 95$ so $\alpha = 0.05$

✓ Thus, we can apply the rule with $100(1 - \alpha/2) = 97.5$ confidence that $error_D(h)$ is at most 0.30 ± 0.1

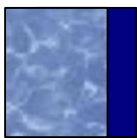
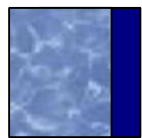
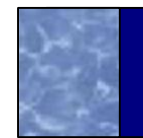
✓ Making no assumption about the lower bound on $error_D(h)$

✓ Thus we have a one-sided error bound on error with double the confidence that we had in the corresponding two-sided bound



Context

- ' Motivation
- ' Estimating Hypothesis Accuracy
- ' Basics of Sampling Theory
- Difference in Error of Two Hypotheses
 - ✓ Hypothesis Testing
- ' Comparing Learning Algorithms
- ' Summary



Difference in Errors of Two Hypotheses

Consider:

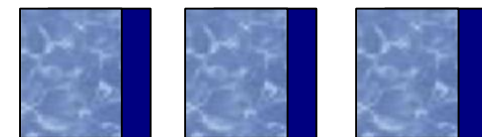
- ✓ two hypotheses h_1 and h_2 for some discrete-valued target function
- ✓ h_1 has been tested on a sample S_1 containing n_1 randomly drawn examples
- ✓ h_2 has been tested on a sample S_2 containing n_2 randomly drawn examples

Suppose we wish to estimate the difference d between the true errors of these two hypotheses

$$d = \text{error}_D(h_1) - \text{error}_D(h_2)$$

4-step procedure to derive confidence interval estimates for d

- ✓ Choose the estimator $\hat{d} = \text{error}_{S_1}(h_1) - \text{error}_{S_2}(h_2)$
- ✓ We do not prove but it can be shown that \hat{d} gives an unbiased estimate of d ; that is $E[\hat{d}] = d$

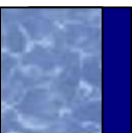


Hypothesis Testing

Question: What is the probability distribution governing the random variable \hat{d} ?

Answer:

- ✓ n_1, n_2 - both errors $error_{s_1}(h_1)$ and $error_{s_2}(h_2)$ follow a distribution that is approximately normal
- ✓ Difference of two normal distributions is also normal \Rightarrow
 \hat{d} is also approximately normal
- ✓ The variance of this distribution is the sum of the variances of $error_{s_1}(h_1)$ and $error_{s_2}(h_2)$
- ✓ We have
$$\sigma_{\hat{d}}^2 \approx \frac{error_{s_1}(h_1)(1 - error_{s_1}(h_1))}{n_1} + \frac{error_{s_2}(h_2)(1 - error_{s_2}(h_2))}{n_2}$$
- ✓ For random variable obeying a normal distribution with mean d and variance $\sigma_{\hat{d}}^2$
the $N\%$ confidence interval estimate for d is $\hat{d} \pm z_N \sigma_{\hat{d}}$



Hypothesis Testing 2

✓ So $\hat{d} \pm z_N \sqrt{\frac{\text{error}_{S_1}(h_1)(1 - \text{error}_{S_1}(h_1))}{n_1} + \frac{\text{error}_{S_2}(h_2)(1 - \text{error}_{S_2}(h_2))}{n_2}}$

z_N is the same constant as described in Table 1

Test over same data

✓ h_1 And h_2 are tested on a single sample S (where S is still independent of h_1 and h_2)

✓ Redefine $\hat{d} = (\text{error}_S(h_1) - \text{error}_S(h_2))$

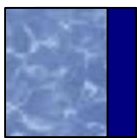
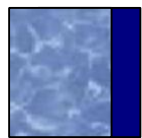
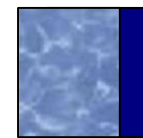
✓ The variance in this new \hat{d} will usually be smaller than the variance of the original

✓ Using a single sample S eliminates the variance due to random differences in the compositions of S_1 and S_2



Context

- ' Motivation
- ' Estimating Hypothesis Accuracy
- ' Basics of Sampling Theory
- ' Difference in Error of Two Hypotheses
- Comparing Learning Algorithms
- ' Summary



Comparing Learning Algorithms

Goal: Comparing the performance of two learning algorithms

L_A and L_B

Question:

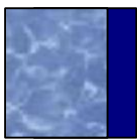
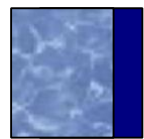
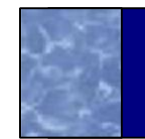
- ✓ What is an appropriate test for comparing learning algorithms?
- ✓ How can we determine whether an observed difference between the algorithms is statistically significant?

Active debate within the machine-learning research community regarding the best method for comparison

Task: Determine which of L_A and L_B is the better learning method on average for learning some particular target function f

- ✓ „On average“ is to consider the relative performance of these two algorithms averaged over all the training set of size n that might be drawn from the underlying instance distribution D

$$E_{S \sim D} [error_D(L_A(S)) - error_D(L_B(S))]$$



Comparing Learning Algorithms 2

In practice:

- ✓ We have only a limited sample D_0
- ✓ Divide D_0 into a training set S_0 and a disjoint test set T_0
- ✓ The training data can be used to train both L_A and L_B
- ✓ Test set can be used to compare the accuracy of the two learned hypothesis $[error_{T_0}(L_A(S_0)) \quad error_{T_0}(L_B(S_0))]$

Improvement:

- ✓ Partition the available data D_0 into k disjoint subsets T_1, T_2, \dots, T_k of equal size, where this size is at least 30
- ✓ For i from 1 to k , do
 use T_i for the test and the remaining data for training set S_i
 $S_i \leftarrow D_0 - T_i$
 $h_A \leftarrow L_A(S_i)$
 $h_B \leftarrow L_B(S_i)$
 $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$

$$\bar{\delta}$$

$$\bar{\delta} = \frac{1}{k} \sum_{i=1}^k \delta_i$$



Comparing Learning Algorithms 3

The approximate $N\%$ confidence interval for estimating the quantity in $[error_{T_0}(L_A(S_0)) \quad error_{T_0}(L_B(S_0))]$ using $\bar{\delta}$ is given by

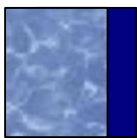
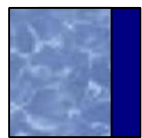
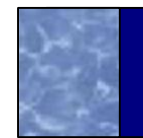
$$\bar{\delta} \pm t_{N, k-1} S_{\hat{\delta}}$$

where $t_{N, k-1}$ is a constant that plays a role analogous to that of z_N
 $S_{\hat{\delta}}$ defined as following $S_{\hat{\delta}} = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^k (\delta_i - \hat{\delta})^2}$

			Confidence lev	
			90%	95%
v	=	2	2,92	4,3
v	=	5	2,02	2,57
v	=	10	1,81	2,23
v	=	20	1,72	2,09
v	=	30	1,7	2,04
v	=	##	1,66	1,98
v	=	∞	1,64	1,96

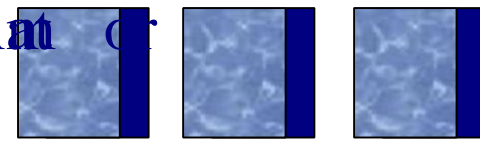
Context

- ' Motivation
- ' Estimating Hypothesis Accuracy
- ' Basics of Sampling Theory
- ' Difference in Error of Two Hypotheses
- ' Comparing Learning Algorithms
- Summary



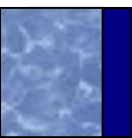
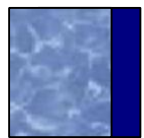
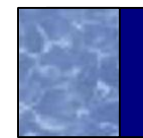
Summary

- Statistical theory provides a basis for estimating the true error ($error_D(h)$) of hypothesis h , based on its observed error ($error_S(h)$) over a sample S of data.
- In general, the problem of estimating confidence intervals is approached by identifying the parameter to be estimated ($error_D(h)$) and an estimator ($error_S(h)$) for this quantity.
- Because the estimator is a random variable it can be characterised by the probability distribution that governs its value.
- Confidence intervals can then be calculated by determining the interval that contains the desired probability mass under this distribution.
- A cause of estimation error is the variance in the estimate. Even with an unbiased estimator, the observed value of the estimator is likely to vary from one experiment to another.
The variance of the distribution governing the estimate characterises how widely this estimate is likely to



Summary 2

- ' Comparing the relative effectiveness of two learning algorithms is an estimation problem that is relatively easy when data and time are unlimited, but more difficult when these resources are limited.
- ' One approach to run the learning algorithms on different subsets of available data, testing the learned hypotheses on the remaining data, then averaging the result of these experiments.
- ' In most cases considered here, deriving confidence intervals involves making a number of assumptions and approximations.



MODULE -4

BAYEIAN LEARNING

CONTENT

- Introduction
- Bayes theorem
- Bayes theorem and concept learning
- Maximum likelihood and Least Squared Error Hypothesis
- Maximum likelihood Hypotheses for predicting probabilities
- Minimum Description Length Principle
- Naive Bayes classifier
- Bayesian belief networks
- EM algorithm

INTRODUCTION

Bayesian learning methods are relevant to study of machine learning for two different reasons.

- First, Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems
- The second reason is that they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities.

Features of Bayesian Learning Methods

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting (1) a prior probability for each candidate hypothesis, and (2) a probability distribution over observed data for each possible hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

Practical difficulty in applying Bayesian methods

- One practical difficulty in applying Bayesian methods is that they typically require initial knowledge of many probabilities. When these probabilities are not known in advance they are often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions.
- A second practical difficulty is the significant computational cost required to determine the Bayes optimal hypothesis in the general case. In certain specialized situations, this computational cost can be significantly reduced.

BAYES THEOREM

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Notations

- $P(h)$ *prior probability of h*, reflects any background knowledge about the chance that h is correct
- $P(D)$ *prior probability of D*, probability that D will be observed
- $P(D|h)$ probability of observing D given a world in which h holds
- $P(h|D)$ *posterior probability of h*, reflects confidence that h holds after D has been observed

Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability $P(\mathbf{h}/\mathbf{D})$, from the prior probability $P(\mathbf{h})$, together with $P(\mathbf{D})$ and $P(\mathbf{D}/\mathbf{h})$.

Bayes Theorem:

$$P(\mathbf{h}|\mathbf{D}) = \frac{P(\mathbf{D}|\mathbf{h})P(\mathbf{h})}{P(\mathbf{D})}$$

$P(\mathbf{h}/\mathbf{D})$ increases with $P(\mathbf{h})$ and with $P(\mathbf{D}/\mathbf{h})$ according to Bayes theorem.

$P(\mathbf{h}/\mathbf{D})$ decreases as $P(\mathbf{D})$ increases, because the more probable it is that \mathbf{D} will be observed independent of \mathbf{h} , the less evidence \mathbf{D} provides in support of \mathbf{h} .

Maximum a Posteriori (MAP) Hypothesis

- In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data D . Any such maximally probable hypothesis is called a *maximum a posteriori (MAP) hypothesis*.
- Bayes theorem to calculate the posterior probability of each candidate hypothesis is h_{MAP} is a MAP hypothesis provided

$$\begin{aligned} h_{MAP} &= \underset{h \in H}{\operatorname{argmax}} P(h|D) \\ &= \underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h) \end{aligned}$$

- $P(D)$ can be dropped, because it is a constant independent of h

Maximum Likelihood (ML) Hypothesis

In some cases, it is assumed that every hypothesis in H is equally probable a priori ($P(h_i) = P(h_j)$ for all h_i and h_j in H).

In this case the below equation can be simplified and need only consider the term $P(D/h)$ to find the most probable hypothesis.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

the equation can be simplified

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} P(D|h)$$

$P(D/h)$ is often called the *likelihood* of the data D given h , and any hypothesis that maximizes $P(D/h)$ is called a *maximum likelihood* (ML) hypothesis

Example

Consider a medical diagnosis problem in which there are two alternative hypotheses

- The patient has a particular form of cancer (denoted by *cancer*)
- The patient does not (denoted by \neg *cancer*)

The available data is from a particular laboratory with two possible outcomes: + (positive) and - (negative)

$$P(cancer) = .008 \quad P(\neg cancer) = 0.992$$

$$P(\oplus|cancer) = .98 \quad P(\ominus|cancer) = .02$$

$$P(\oplus|\neg cancer) = .03 \quad P(\ominus|\neg cancer) = .97$$

- Suppose a new patient is observed for whom the lab test returns a positive (+) result.
- Should we diagnose the patient as having cancer or not?

$$\begin{aligned}P(\oplus|cancer)P(cancer) &= (.98).008 = .0078 \\P(\oplus|\neg cancer)P(\neg cancer) &= (.03).992 = .0298 \\ \Rightarrow h_{MAP} &= \neg cancer\end{aligned}$$

BAYES THEOREM AND CONCEPT LEARNING

What is the relationship between Bayes theorem and the problem of concept learning?

Since Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data, and can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable.

Brute-Force Bayes Concept Learning

We can design a straightforward concept learning algorithm to output the maximum a posteriori hypothesis, based on Bayes theorem, as follows:

Brute-Force MAP LEARNING algorithm

1. For each hypothesis h in H calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D)$$

In order specify a learning problem for the **BRUTE-FORCE MAP LEARNING** algorithm we must specify what values are to be used for $P(h)$ and for $P(D/h)$?

Lets choose $P(h)$ and for $P(D/h)$ to be consistent with the following assumptions:

- The training data D is noise free (i.e., $d_i = c(x_i)$)
- The target concept c is contained in the hypothesis space H
- We have no a priori reason to believe that any hypothesis is more probable than any other.

What values should we specify for $P(h)$?

- Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis h in H .
- Assume the target concept is contained in H and require that these prior probabilities sum to 1.

$$P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

What choice shall we make for $P(D/h)$?

- $P(D/h)$ is the probability of observing the target values $D = (d_1 \dots d_m)$ for the fixed set of instances $(x_1 \dots x_m)$, given a world in which hypothesis h holds
- Since we assume noise-free training data, the probability of observing classification d_i given h is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$. Therefore,

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

Given these choices for $P(h)$ and for $P(D|h)$ we now have a fully-defined problem for the above **BRUTE-FORCE MAP LEARNING** algorithm.

In a first step, we have to determine the probabilities for $P(h|D)$

h is **inconsistent** with training data D

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$$

h is **consistent** with training data D

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} = \frac{1}{|VS_{H,D}|}$$

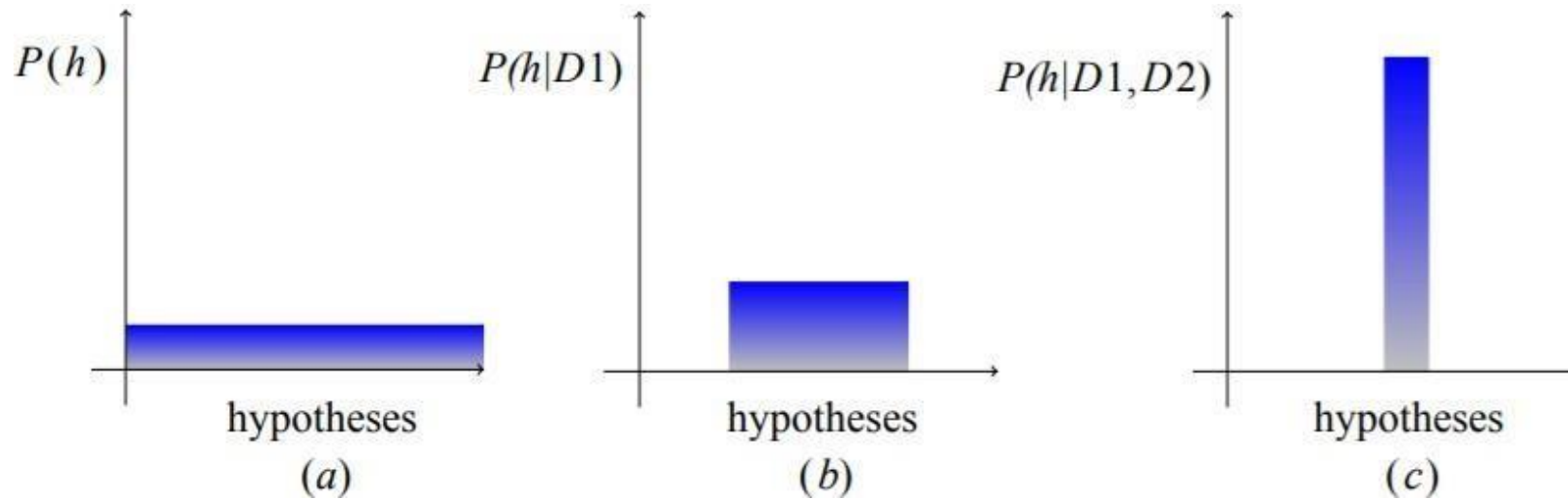
To summarize, Bayes theorem implies that the posterior probability $P(h|D)$ under our assumed $P(h)$ and $P(D|h)$ is

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$$

where $|VS_{H,D}|$ is the number of hypotheses from H consistent with D

The Evolution of Probabilities Associated with Hypotheses

- Figure (a) all hypotheses have the same probability.
- Figures (b) and (c), As training data accumulates, the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to 1 is shared equally among the remaining consistent hypotheses.



MAP Hypotheses and Consistent Learners

A learning algorithm is a consistent learner if it outputs a hypothesis that commits zero errors over the training examples.

Every consistent learner outputs a MAP hypothesis, if we assume a uniform prior probability distribution over H ($P(h_i) = P(h_j)$ for all i, j), and deterministic, noise free training data ($P(D|h) = 1$ if D and h are consistent, and 0 otherwise).

Example:

- FIND-S outputs a consistent hypothesis, it will output a MAP hypothesis under the probability distributions $P(h)$ and $P(D|h)$ defined above.
- Are there other probability distributions for $P(h)$ and $P(D|h)$ under which FIND-S outputs MAP hypotheses? Yes.
- Because FIND-S outputs a maximally specific hypothesis from the version space, its output hypothesis will be a MAP hypothesis relative to any prior probability distribution that favours more specific hypotheses.

- Bayesian framework is a way to characterize the behaviour of learning algorithms
- By identifying probability distributions $P(h)$ and $P(D|h)$ under which the output is a optimal hypothesis, implicit assumptions of the algorithm can be characterized (Inductive Bias)
- Inductive inference is modelled by an equivalent probabilistic reasoning system based on Bayes theorem

MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

Consider the problem of learning a continuous-valued target function such as neural network learning, linear regression, and polynomial curve fitting

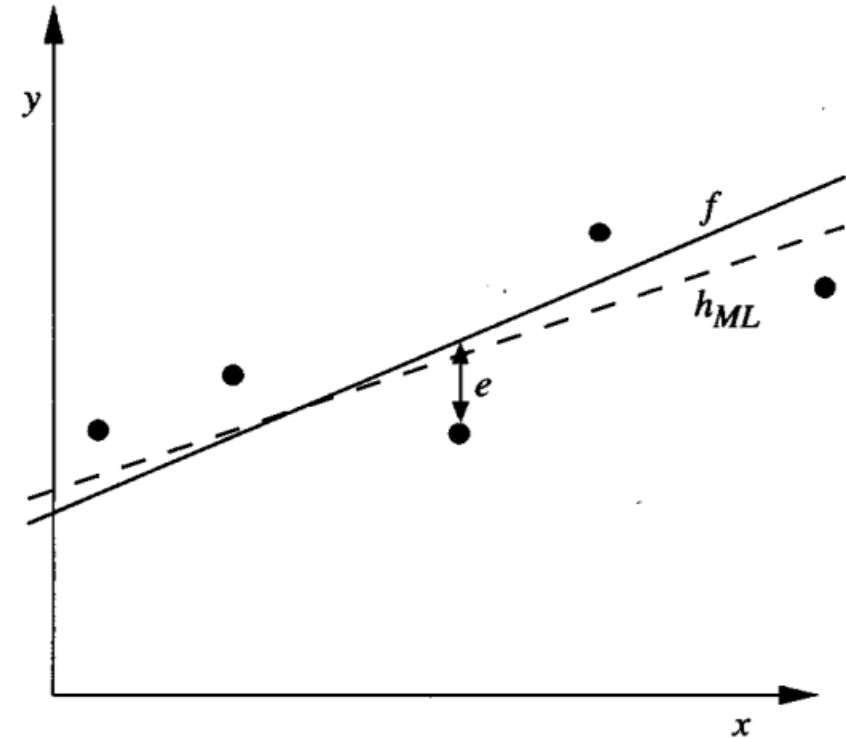
A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood (ML) hypothesis

Learning A Continuous-Valued Target Function

- Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X , i.e., $(\forall h \in H)[h : X \rightarrow \mathbb{R}]$ and training examples of the form $\langle x_i, d_i \rangle$
- The problem faced by L is to learn an unknown target function $f : X \rightarrow \mathbb{R}$
- A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution with zero mean ($d_i = f(x_i) + e_i$)
- Each training example is a pair of the form (x_i, d_i) where $d_i = f(x_i) + e_i$.
 - Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise.
 - It is assumed that the values of the e_i are *drawn independently* and that they are distributed according to a *Normal distribution* with zero mean.
- The task of the learner is to output a *maximum likelihood hypothesis*, or, equivalently, a MAP hypothesis assuming all hypotheses are equally probable a priori.

Learning A Linear Function

- The target function f corresponds to the solid line.
- The training examples (x_i, d_i) are assumed to have Normally distributed noise e_i with zero mean added to the true target value $f(x_i)$.
- The dashed line corresponds to the hypothesis h_{ML} with least-squared training error, hence the maximum likelihood hypothesis.
- Notice that the maximum likelihood hypothesis is not necessarily identical to the correct hypothesis, f , because it is inferred from only a limited sample of noisy training data



Before showing why a hypothesis that minimizes the sum of squared errors in this setting is also a maximum likelihood hypothesis, let us quickly review *probability densities and Normal distributions*

Probability Density for continuous variables

$$p(x_0) \equiv \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} P(x_0 \leq x < x_0 + \epsilon)$$

e : a random noise variable generated by a Normal probability distribution

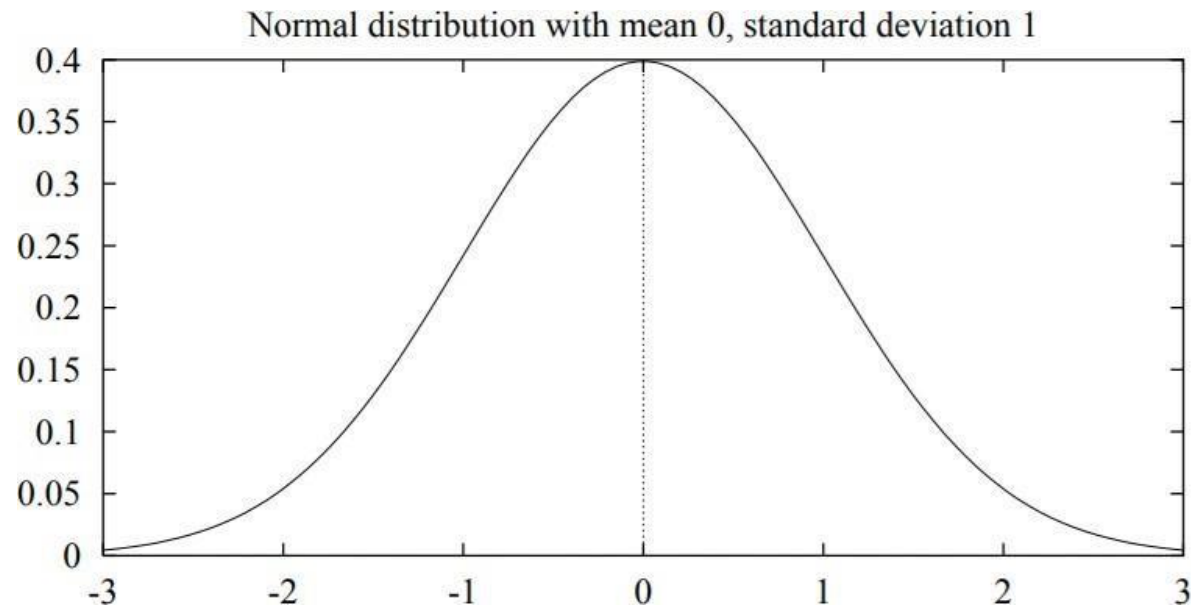
$\langle x_1 \dots x_m \rangle$: the sequence of instances (as before)

$\langle d_1 \dots d_m \rangle$: the sequence of target values with $d_i = f(x_i) + e_i$

Normal Probability Distribution (Gaussian Distribution)

A Normal distribution is a smooth, bell-shaped distribution that can be completely characterized by its mean μ and its standard deviation σ

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



- A Normal distribution is fully determined by two parameters in the formula: μ and σ .
- If the random variable X follows a normal distribution:
 - The probability that X will fall into the interval (a, b) is $\int_a^b p(x)dx$
 - The expected, or *mean value of X* , $E[X] = \mu$
 - The *variance of X* , $\text{Var}(X) = \sigma^2$
 - The *standard deviation of X* , $\sigma_x = \sigma$
- The *Central Limit Theorem* states that the sum of a large number of independent, identically distributed random variables follows a distribution that is approximately *Normal*.

Using the previous definition of h_{ML} we have

$$h_{ML} = \underset{h \in H}{argmax} p(D|h)$$

Assuming training examples are mutually independent given h , we can write $P(D|h)$ as the product of the various $(d_i|h)$

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^m p(d_i|h)$$

Given the noise e_i obeys a Normal distribution with zero mean and unknown variance σ^2 , each d_i must also obey a Normal distribution around the true targetvalue $f(x_i)$. Because we are writing the expression for $P(D|h)$, we assume h is the correct description of f . Hence, $\mu = f(x_i) = h(x_i)$

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2} (d_i - h(x_i))^2}$$

$$\begin{aligned}
h_{ML} &= \operatorname{argmax}_{h \in H} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2} \\
&= \operatorname{argmax}_{h \in H} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2}
\end{aligned}$$

It is common to maximize the less complicated logarithm, which is justified because of the monotonicity of function p .

$$= \operatorname{argmax}_{h \in H} \sum_{i=1}^m \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of \mathbf{h} and can therefore be discarded

$$= \operatorname{argmax}_{h \in H} \sum_{i=1}^m -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative term is equivalent to minimizing the corresponding positive term.

$$= \operatorname{argmin}_{h \in H} \sum_{i=1}^m \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally Discard constants that are independent of h

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^m (d_i - h(x_i))^2$$

- the h_{ML} is one that minimizes the sum of the squared errors

Why is it reasonable to choose the Normal distribution to characterize noise?

- good approximation of many types of noise in physical systems
- Central Limit Theorem shows that the sum of a sufficiently large number of independent, identically distributed random variables itself obeys a Normal distribution

Only noise in the target value is considered, not in the attributes describing the instances themselves

MAXIMUM LIKELIHOOD HYPOTHESES FOR PREDICTING PROBABILITIES

Consider the setting in which we wish to learn a nondeterministic (probabilistic) function $f : X \rightarrow \{0, 1\}$, which has two discrete output values.

We want a function approximator whose output is the probability that $f(x) = 1$

In other words , learn the target function

$$f' : X \rightarrow [0, 1] \text{ such that } f'(x) = P(f(x) = 1)$$

How can we learn f' using a neural network?

Use of brute force way would be to first collect the observed frequencies of 1's and 0's for each possible value of x and to then train the neural network to output the target frequency for each x .

What criterion should we optimize in order to find a maximum likelihood hypothesis for f in this setting?

- First obtain an expression for $P(D|h)$
- Assume the training data D is of the form $D = \{(x_1, d_1) \dots (x_m, d_m)\}$, where d_i is the observed 0 or 1 value for $f(x_i)$.
- Both x_i and d_i as random variables, and assuming that each training example is drawn independently, we can write $P(D|h)$ as

$$P(D | h) = \prod_{i=1}^m P(x_i, d_i | h)$$

Applying the product rule

$$P(D | h) = \prod_{i=1}^m P(d_i | h, x_i)P(x_i)$$

The probability $P(d_i|h, x_i)$

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases} \quad \text{equ (3)}$$

Re-express it in a more mathematically manipulable form, as

$$P(d_i|h, x_i) = h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \quad \text{equ (4)}$$

Equation (4) to substitute for $P(d_i |h, x_i)$ in Equation (5) to obtain

$$P(D|h) = \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i) \quad \text{equ (5)}$$

We write an expression for the maximum likelihood hypothesis

$$h_{ML} = \operatorname{argmax}_{h \in H} \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i)$$

The last term is a constant independent of h , so it can be dropped

$$h_{ML} = \operatorname{argmax}_{h \in H} \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \quad \text{equ (6)}$$

It easier to work with the log of the likelihood, yielding

$$h_{ML} = \operatorname{argmax}_{h \in H} \sum_{i=1}^m d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \quad \text{equ (7)}$$

Equation (7) describes the quantity that must be maximized in order to obtain the maximum likelihood hypothesis in our current problem setting

Gradient Search to Maximize Likelihood in a Neural Net

Derive a weight-training rule for neural network learning that seeks to maximize $G(h, D)$ using gradient ascent

- The gradient of $G(h, D)$ is given by the vector of partial derivatives of $G(h, D)$ with respect to the various network weights that define the hypothesis h represented by the learned network
- In this case, the partial derivative of $G(h, D)$ with respect to weight w_{jk} from input k to unit j is

$$\begin{aligned}\frac{\partial G(h, D)}{\partial w_{jk}} &= \sum_{i=1}^m \frac{\partial G(h, D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\ &= \sum_{i=1}^m \frac{\partial (d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\ &= \sum_{i=1}^m \frac{d_i - h(x_i)}{h(x_i)(1 - h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}}\end{aligned}\quad \text{equ (1)}$$

Suppose our neural network is constructed from a single layer of sigmoid units. Then,

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x_i) x_{ijk} = h(x_i)(1 - h(x_i)) x_{ijk}$$

where x_{ijk} is the k^{th} input to unit j for the i^{th} training example, and $d(x)$ is the derivative of the sigmoid squashing function.

Finally, substituting this expression into Equation (1), we obtain a simple expression for the derivatives that constitute the gradient

$$\frac{\partial G(h, D)}{\partial w_{jk}} = \sum_{i=1}^m (d_i - h(x_i)) x_{ijk}$$

Because we seek to maximize rather than minimize $P(D|h)$, we perform ***gradient ascent*** rather than ***gradient descent search***. On each iteration of the search the weight vector is adjusted in the direction of the gradient, using the weight update rule

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where,

$$\Delta w_{jk} = \eta \sum_{i=1}^m (d_i - h(x_i)) x_{ijk} \quad \text{equ (2)}$$

where η is a small positive constant that determines the step size of the i gradient ascent search

It is interesting to compare this weight-update rule to the weight-update rule used by the BACKPROPAGATION algorithm to minimize the sum of squared errors between predicted and observed network outputs.

The BACKPROPAGATION update rule for output unit weights, re-expressed using our current notation, is

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where,

$$\Delta w_{jk} = \eta \sum_{i=1}^m h(x_i)(1 - h(x_i))(d_i - h(x_i)) x_{ijk}$$

MINIMUM DESCRIPTION LENGTH PRINCIPLE

- A Bayesian perspective on Occam's razor
- Motivated by interpreting the definition of h_{MAP} in the light of basic concepts from information theory.

$$h_{MAP} = \underset{h \in H}{argmax} P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the \log_2

$$h_{MAP} = \underset{h \in H}{argmax} \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{argmin} -\log_2 P(D|h) - \log_2 P(h) \quad \text{equ (1)}$$

- This equation can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data

Introduction to a basic result of information theory

- Consider the problem of designing a code to transmit messages drawn at random
- i is the message
- The probability of encountering message i is p_i
- Interested in the most compact code; that is, interested in the code that minimizes the expected number of bits we must transmit in order to encode a message drawn at random
- To minimize the expected code length we should assign shorter codes to messages that are more probable
- Shannon and Weaver (1949) showed that the optimal code (i.e., the code that minimizes the expected message length) assigns $-\log_2 p_i$ bits to encode message i .
- The number of bits required to encode message i using code C as the *description length of message i with respect to C* , which we denote by $L_c(i)$.

Interpreting the equation

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} -\log_2 P(D|h) - \log_2 P(h) \quad \text{equ (1)}$$

- $-\log_2 P(h)$: the description length of h under the optimal encoding for the hypothesis space H : $L_{CH}(h) = -\log_2 P(h)$, where C_H is the optimal code for hypothesis space H .
- $-\log_2 P(D|h)$: the description length of the training data D given hypothesis h , under the optimal encoding from the hypothesis space H : $L_{CH}(D|h) = -\log_2 P(D|h)$, where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know the hypothesis h .

Rewrite Equation (1) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

where C_H and $C_{D|h}$ are the optimal encodings for H and for D given h

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths of equ.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Minimum Description Length principle:

$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D | h)$$

Where, codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis

The above analysis shows that if we choose C_1 to be the optimal encoding of hypotheses C_H , and if we choose C_2 to be the optimal encoding $C_{D|h}$, then $h_{MDL} = h_{MAP}$

Application to Decision Tree Learning

Apply the MDL principle to the problem of learning decision trees from some training data.

What should we choose for the representations C_1 and C_2 of hypotheses and data?

- **For C_1 :** C_1 might be some obvious encoding, in which the description length grows with the number of nodes and with the number of edges
- **For C_2 :** Suppose that the sequence of instances $(x_1 \dots x_m)$ is already known to both the transmitter and receiver, so that we need only transmit the classifications $(f(x_1) \dots f(x_m))$.

Now if the training classifications $(f(x_1) \dots f(x_m))$ are identical to the predictions of the hypothesis, then there is no need to transmit any information about these examples. The description length of the classifications given the hypothesis ZERO

If examples are misclassified by h , then for each misclassification we need to transmit a message that identifies which example is misclassified as well as its correct classification

The hypothesis h_{MDL} under the encoding C_1 and C_2 is just the one that minimizes the sum of these description lengths.

- MDL principle provides a way for trading off hypothesis complexity for the number of errors committed by the hypothesis
- MDL provides a way to deal with the issue of overfitting the data.
- Short imperfect hypothesis may be selected over a long perfect hypothesis.

Machine Learning: Lecture 8

Computational Learning Theory

(Based on Chapter 7 of Mitchell T.,
Machine Learning, 1997)

Overview

- Are there general laws that govern learning?
 - ***Sample Complexity***: How many training examples are needed for a learner to converge (with high probability) to a successful hypothesis?
 - ***Computational Complexity***: How much computational effort is needed for a learner to converge (with high probability) to a successful hypothesis?
 - ***Mistake Bound***: How many training examples will the learner misclassify before converging to a successful hypothesis?
- These questions will be answered within two analytical frameworks:
 - The ***Probably Approximately Correct (PAC)*** framework
 - The ***Mistake Bound*** framework

Overview (Cont'd)

- Rather than answering these questions for individual learners, we will answer them for broad classes of learners. In particular we will consider:
 - The size or complexity of the hypothesis space considered by the learner.
 - The accuracy to which the target concept must be approximated.
 - The probability that the learner will output a successful hypothesis.
 - The manner in which training examples are presented to the learner.

The PAC Learning Model

- **Definition:** Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H . C is *PAC-learnable* by L using H if for all $c \in C$, distributions D over X , ε such that $0 < \varepsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will, with probability at least $(1 - \delta)$, output a hypothesis $h \in H$ such that $\text{error}_D(h) \leq \varepsilon$, in time that is *polynomial* in $1/\varepsilon$, $1/\delta$, n , and $\text{size}(c)$.

Sample Complexity for *Finite* Hypothesis Spaces

- Given any *consistent* learner, the number of examples sufficient to assure that any hypothesis will be probably (with probability $(1 - \delta)$) approximately (within error ϵ) correct is $m = \frac{1}{\epsilon} (\ln|H| + \ln(1/\delta))$
- If the learner is *not consistent*, $m = \frac{1}{2\epsilon^2} (\ln|H| + \ln(1/\delta))$
- Conjunctions of Boolean Literals are also PAC-Learnable and $m = \frac{1}{\epsilon} (n \ln 3 + \ln(1/\delta))$
- k-term DNF expressions are not PAC learnable because even though they have polynomial sample complexity, their computational complexity is not polynomial.
- Surprisingly, however, k-term CNF is PAC learnable.

Sample Complexity for Infinite Hypothesis Spaces I: VC-Dimension

- The PAC Learning framework has 2 disadvantages:
 - It can lead to weak bounds
 - Sample Complexity bound cannot be established for infinite hypothesis spaces
- We introduce new ideas for dealing with these problems:
 - **Definition:** A set of instances S is shattered by hypothesis space H iff for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.
 - **Definition:** The Vapnik-Chervonenkis dimension, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H . If arbitrarily large finite sets of X can be shattered by H , then $VC(H)=\infty$

Sample Complexity for Infinite Hypothesis Spaces II

- *Upper-Bound* on sample complexity, using the *VC-Dimension*: $m \geq 1/\varepsilon (4\log_2(2/\delta) + 8VC(H)\log_2(13/\varepsilon))$
- *Lower Bound* on sample complexity, using the *VC-Dimension*:

Consider any concept class C such that $VC(C) \geq 2$, any learner L , and any $0 < \varepsilon < 1/8$, and $0 < \delta < 1/100$. Then there exists a distribution D and target concept in C such that if L observes fewer examples than

$$\max[1/\varepsilon \log(1/\delta), (VC(C)-1)/(32\varepsilon)]$$

then with probability at least δ , L outputs a hypothesis h having $error_D(h) > \varepsilon$.

VC-Dimension for Neural Networks

- Let G be a layered directed acyclic graph with n input nodes and $s \geq 2$ internal nodes, each having at most r inputs. Let C be a concept class over \mathbf{R}^r of VC dimension d , corresponding to the set of functions that can be described by each of the s internal nodes. Let C_G be the G -composition of C , corresponding to the set of functions that can be represented by G . Then $VC(C_G) \leq 2ds \log(es)$, where e is the base of the natural logarithm.
- This theorem can help us bound the VC-Dimension of a neural network and thus, its sample complexity (See, [Mitchell, p.219])!

The *Mistake Bound* Model of Learning

- The *Mistake Bound* framework is different from the PAC framework as it considers learners that receive a sequence of training examples and that predict, upon receiving each example, what its target value is.
- The question asked in this setting is: “*How many mistakes will the learner make in its predictions before it learns the target concept?*”
- This question is significant in practical settings where learning must be done while the system is in actual use.

Optimal Mistake Bounds

- **Definition:** Let C be an arbitrary nonempty concept class. The optimal mistake bound for C , denoted $Opt(C)$, is the minimum over all possible learning algorithms A of $M_A(C)$.

$$Opt(C) = \min_{A \in \text{Learning_Algorithm}} M_A(C)$$

- For any concept class C , the optimal mistake bound is bound as follows:

$$VC(C) \leq Opt(C) \leq \log_2(|C|)$$

A Case Study: The Weighted-Majority Algorithm

a_i denotes the i^{th} prediction algorithm in the pool A of algorithm. w_i denotes the weight associated with a_i .

- For all i initialize $w_i \leftarrow 1$
- For each training example $\langle x, c(x) \rangle$
 - Initialize q_0 and q_1 to 0
 - For each prediction algorithm a_i
 - If $a_i(x)=0$ then $q_0 \leftarrow q_0 + w_i$
 - If $a_i(x)=1$ then $q_1 \leftarrow q_1 + w_i$
 - If $q_1 > q_0$ then predict $c(x)=1$
 - If $q_0 > q_1$ then predict $c(x)=0$
 - If $q_0=q_1$ then predict 0 or 1 at random for $c(x)$
 - For each prediction algorithm a_i in A do
 - If $a_i(x) \neq c(x)$ then $w_i \leftarrow \beta w_i$

Relative Mistake Bound for the Weighted-Majority Algorithm

- Let D be any sequence of training examples, let A be any set of n prediction algorithms, and let k be the minimum number of mistakes made by any algorithm in A for the training sequence D . Then the number of mistakes over D made by the *Weighted-Majority* algorithm using $\beta=1/2$ is at most $2.4(k + \log_2 n)$.
- This theorem can be generalized for any $0 \leq \beta \leq 1$ where the bound becomes

$$(k \log_2 1/\beta + \log_2 n) / \log_2(2/(1+\beta))$$

INSTANCE-BASE LEARNING

- Instance-based learning methods simply store the training examples instead of learning explicit description of the target function.
 - Generalizing the examples is postponed until a new instance must be classified.
 - When a new instance is encountered, its relationship to the stored examples is examined in order to assign a target function value for the new instance.
- Instance-based learning includes *nearest neighbor*, *locally weighted regression* and *case-based reasoning* methods.
- Instance-based methods are sometimes referred to as **lazy** learning methods because they delay processing until a new instance must be classified.
- A key advantage of lazy learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.

k-Nearest Neighbor Learning

- k-Nearest Neighbor Learning algorithm assumes all instances correspond to points in the n-dimensional space \mathbb{R}^n
- The nearest neighbors of an instance are defined in terms of Euclidean distance.
- Euclidean distance between the instances $x_i = \langle x_{i1}, \dots, x_{in} \rangle$ and $x_j = \langle x_{j1}, \dots, x_{jn} \rangle$ are:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^n (x_{ir} - x_{jr})^2}$$

- For a given query instance x_q , $f(x_q)$ is calculated the function values of k-nearest neighbor of x_q

k-Nearest Neighbor Learning

- Store all training examples $\langle x_i, f(x_i) \rangle$
- Calculate $f(x_q)$ for a given query instance x_q using k-nearest neighbor
- **Nearest neighbor: (k=1)**
 - Locate the nearest training example x_n , and estimate $f(x_q)$ as
 - $f(x_q) \leftarrow f(x_n)$
- **k-Nearest neighbor:**
 - Locate k nearest training examples, and estimate $f(x_q)$ as
 - If the target function is real-valued, take mean of f-values of k nearest neighbors.

$$f(x_q) = \frac{\sum_{i=1}^k f(x_i)}{k}$$

- If the target function is discrete-valued, take a vote among f-values of k nearest neighbors.

When To Consider Nearest Neighbor

- Instances map to points in \mathbb{R}^n
- Less than 20 attributes per instance
- Lots of training data
- **Advantages**
 - Training is very fast
 - Learn complex target functions
 - Can handle noisy data
 - Does not lose any information
- **Disadvantages**
 - Slow at query time
 - Easily fooled by irrelevant attributes

Distance-Weighted kNN

Might want weight nearer neighbors more heavily...

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

and $d(x_q, x_i)$ is distance between x_q and x_i

Note now it makes sense to use *all* training examples instead of just k

Curse of Dimensionality

Imagine instances described by 20 attributes, but only 2 are relevant to target function

Curse of dimensionality: nearest nbr is easily mislead when high-dimensional X

One approach:

- Stretch j th axis by weight z_j , where z_1, \dots, z_n chosen to minimize prediction error
- Use cross-validation to automatically choose weights z_1, \dots, z_n
- Note setting z_j to zero eliminates this dimension altogether

Locally Weighted Regression

- KNN forms local approximation to f for each query point x_q
- Why not form an explicit approximation $f(x)$ for region surrounding x_q
 - ➔ Locally Weighted Regression
- **Locally weighted regression** uses nearby or distance-weighted training examples to form this local approximation to f .
- We might approximate the target function in the neighborhood surrounding x , using a linear function, a quadratic function, a multilayer neural network.
- The phrase "locally weighted regression" is called
 - *local* because the function is approximated based only on data near the query point,
 - *weighted* because the contribution of each training example is weighted by its distance from the query point, and
 - *regression* because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Locally Weighted Regression

- Given a new query instance x_q , the general approach in locally weighted regression is to construct an approximation f that fits the training examples in the neighborhood surrounding x_q .
- This approximation is then used to calculate the value $f(x_q)$, which is output as the estimated target value for the query instance.

Locally Weighted Linear Regression

f is approximated near x_q using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

$a_i(x)$ denotes the value of the i th attribute of the instance x .

Minimize the squared error

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

Kernel function K is the function of distance that is used to determine the weight of each training example.

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) (f(x) - \hat{f}(x)) a_j(x)$$

Radial Basis Functions

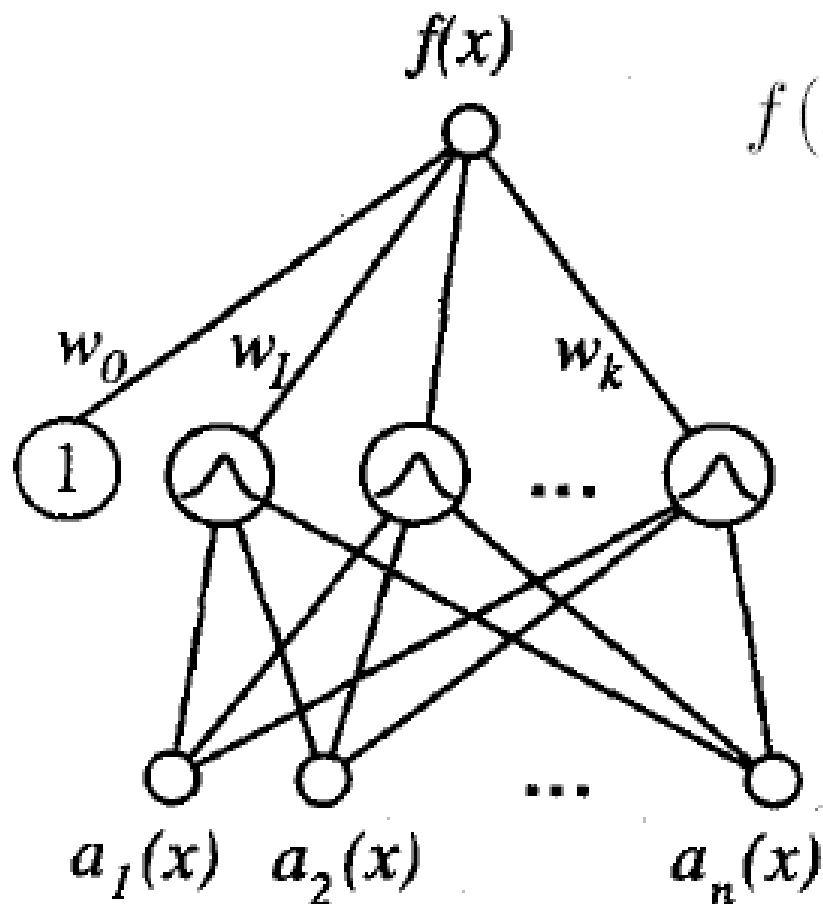
- One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions.
- The learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^k w_u K_u(d(x_u, x))$$

where each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases. Here k is a user-provided constant that specifies the number of kernel functions to be included. Even though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u . It is common to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at the point x_u with some variance σ_u^2 .

$$K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Radial Basis Function Networks



$$f(x) = w_0 + \sum_{u=1}^k w_u K_u(d(x_u, x))$$

$$K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Each hidden unit produces an activation determined by a Gaussian function centered at some instance x_u .

Therefore, its activation will be close to zero unless the input x is near x_u .

The output unit produces a linear combination of the hidden unit activations.

Case-based reasoning

- Instance-based methods
 - lazy
 - classification based on classifications of near (similar) instances
 - data: points in n -dim. space
- Case-based reasoning
 - as above, but data represented in symbolic form
- New distance metrics required

Lazy & eager learning

- Lazy: generalize at query time
 - kNN, CBR
- Eager: generalize before seeing query
 - Radial basis, ID3, ...
- Difference
 - eager *must* create global approximation
 - lazy *can* create many local approximation
 - lazy can represent more complex functions using same H (H = linear functions)

Machine Learning: Lecture 12

Genetic Algorithms

(Based on Chapter 9 of Mitchell, T.,
Machine Learning, 1997)

Overview of Genetic Algorithms (GAs)

- GA is a learning method motivated by analogy to biological evolution.
- GAs search the hypothesis space by generating successor hypotheses which repeatedly mutate and recombine parts of the best currently known hypotheses.
- In Genetic Programming (GP), entire computer programs are evolved to certain fitness criteria.

General Operation of GAs

- **Initialize Population**: generate p hypotheses at random.
- **Evaluate**: for each p , compute $fitness(p)$
- While $Max_h Fitness(h) < Threshold$ do
 - **Select**: probabilistically select a fraction of the best p 's in P . Call this new generation P_{New}
 - **Crossover**: probabilistically form pairs of the selected p 's and produce two offsprings by applying the crossover operator. Add all offsprings to P_{new} .
 - **Mutate**: Choose $m\%$ of P_{New} with uniform probability. For each, invert one randomly selected bit in its representation.
 - **Update**: $P \leftarrow P_{new}$
 - **Evaluate**: for each p in P , compute $fitness(p)$
- Return the hypothesis from P that has the highest fitness.

Representing Hypotheses

- In GAs, hypotheses are often represented by bit strings so that they can be easily manipulated by genetic operators such as mutation and crossover.
- Examples:

(Outlook = Overcast \vee Rain) \wedge (Wind = Strong)
 \Leftrightarrow **011 10**

IF Wind = Strong THEN PlayTennis = yes
 \Leftrightarrow **111 10 10**

where group 1 = 3-valued outlook,
group 2 = 2-valued Wind
group 3 = 2-valued PlayTennis

Genetic Operators

- **Crossover Techniques:**

- Single-point Crossover.

Mask example: 111110000000

- Two-point Crossover.

Mask example: 00111110000

- Uniform Crossover.

Mask example: 10011010011

- **Mutation Techniques:**

- Point Mutation

- **Other Operators:**

- Specialization Operator
- Generalization Operator

Fitness Function and Selection

- A simple measure for modeling the probability that a hypothesis will be selected is given by the *fitness proportionate selection* (or *roulette wheel selection*):

$$Pr(h_i) = \text{Fitness}(h_i) / \sum_{j=1}^P \text{Fitness}(h_j)$$

- Other methods: *Tournament Selection* and *Rank Selection*.
- In classification tasks, the Fitness function typically has a component that scores the classification accuracy over a set of provided training examples. Other criteria can be added (e.g., complexity or generality of the rule)

Hypothesis Space Search (I)

- GA search can move very abruptly (as compared to Backpropagation, for example), replacing a parent hypothesis by an offspring that may be radically different from the parent.
- The problem of **Crowding**: when one individual is more fit than others, this individual and closely related ones will take up a large fraction of the population.
- **Solutions:**
 - Use tournament or rank selection instead of roulette selection.
 - Fitness sharing
 - restriction on the kinds of individuals allowed to recombine to form offsprings.

Hypothesis Space Search (II): The Schema Theorem [Holland, 75]

- **Definition:** A schema is any string composed of 0s, 1s and *s where * means ‘don’t care’.
- **Example:** schema 0*10 represents strings 0010 and 0110.
- **The Schema Theorem:** More fit schemas will tend to grow in influence, especially schemas containing a small number of defined bits (i.e., containing a large number of *s), and especially when these defined bits are near one another within the bit string.

Genetic Programming: Representing Programs

- *Example:* $\sin(x) + \text{sqrt}(x^2 + y)$

Genetic Programming: Crossover Operation

- *Example:*

Models of Evolution and Learning I: Lamarckian Evolution [Late 19th C]

- **Proposition:** Experiences of a single organism directly affect the genetic makeup of their offsprings.
- **Assessment:** This proposition is wrong: the genetic makeup of an individual is unaffected by the lifetime experience of one's biological parents.
- **However:** Lamarckian processes can sometimes improve the effectiveness of computerized genetic algorithms.

Models of Evolution and Learning II: Baldwin Effect [1896]

- If a species is evolving in a changing environment, there will be evolutionary pressure to favor individuals with the capability to learn during their lifetime.
- Those individuals who are able to learn many traits will rely less strongly on their genetic code to “hard-wire” traits. As a result, these individuals can support a more diverse gene pool, relying on individual learning of the “missing” or “sub-optimized” traits in the genetic code. This more diverse gene pool can, in turn, support more rapid evolutionary adaptation. Thus the capability of learning can accelerate the rate of evolutionary adaptation of a population.

Parallelizing Genetic Algorithms

- GAs are naturally suited to parallel implementation.

Different approaches were tried:

- Coarse Grain: subdivides the population into distinct groups of individuals (*demes*) and conducts a GA search in each deme. Transfer between demes occurs (though infrequently) by a migration process in which individuals from one deme are copied or transferred to other demes
- Fine Grain: One processor is assigned per individual in the population and recombination takes place among neighboring individuals.



Machine Learning

Chapter 10. Learning Sets of Rules

Tom M. Mitchell



Learning Disjunctive Sets of Rules

- Method 1: Learn decision tree, convert to rules
- Method 2: Sequential covering algorithm:
 1. Learn one rule with high accuracy, any coverage
 2. Remove positive examples covered by this rule
 3. Repeat



Sequential Covering Algorithm

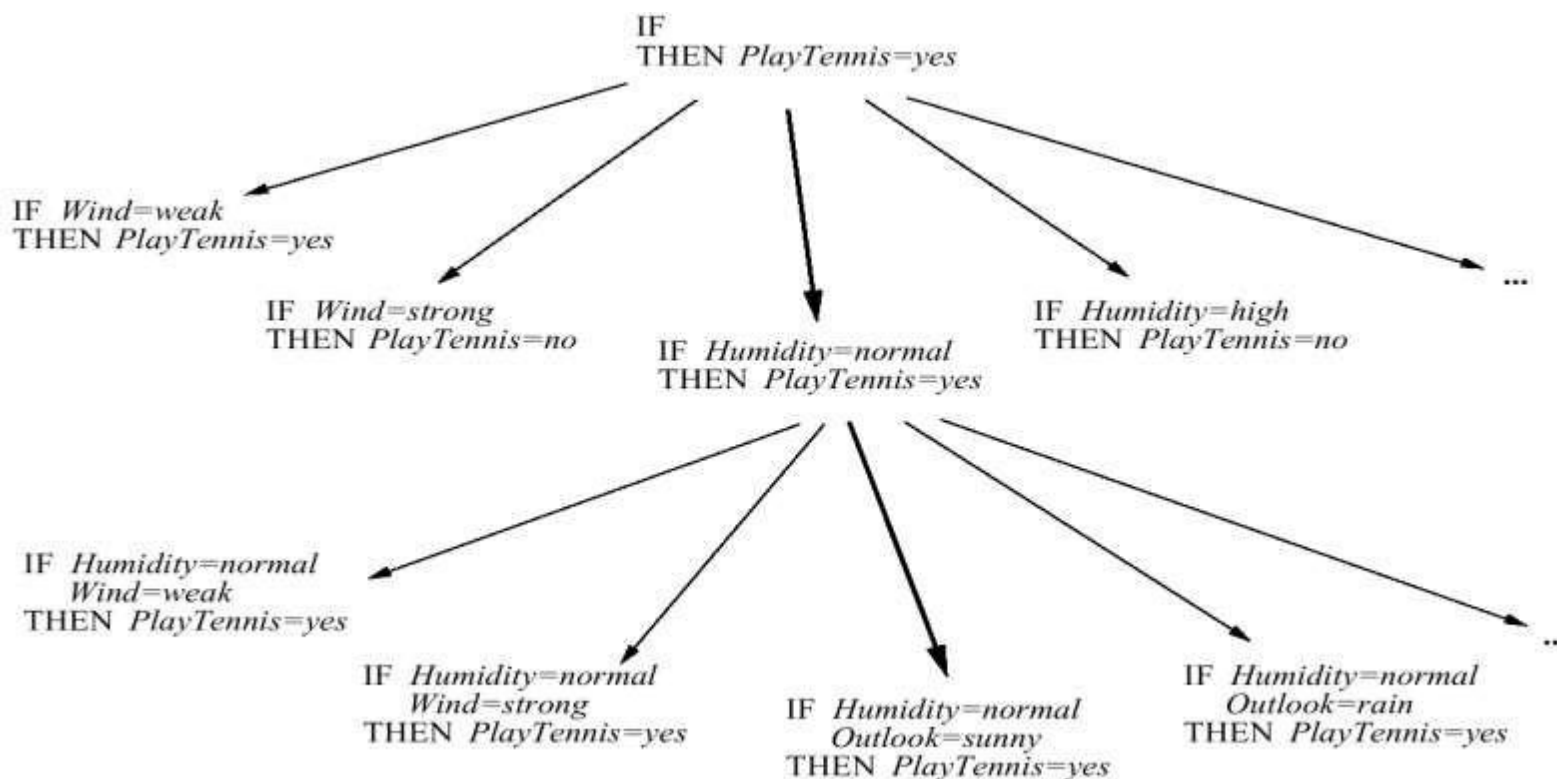
SEQUENTIAL-

COVERING (*Target attribute; Attributes; Examples; Threshold*)

- Learned rules $\leftarrow \{\}$
- Rule \leftarrow LEARN-ONE-RULE(*Target_attribute, Attributes, Examples*)
- while PERFORMANCE (*Rule, Examples*)
 > *Threshold*, do
 - Learned_rules \leftarrow Learned_rules + *Rule*
 - *Examples* \leftarrow *Examples* – {examples correctly classified by *Rule*}
 - *Rule* \leftarrow LEARN-ONE-RULE (*Target_attribute, Attributes, Examples*)
 - *Learned_rules* \leftarrow sort *Learned_rules* accord to PERFORMANCE over *Examples*
 - return *Learned_rules*



Learn-One-Rule





Learn-One-Rule(Cont.)

- $Pos \leftarrow$ positive *Examples*
- $Neg \leftarrow$ negative *Examples*
- while Pos , do
 - Learn a NewRule*
 - $NewRule \leftarrow$ most general rule possible
 - $NewRule \leftarrow Neg$
 - while $NewRuleNeg$, do
 - Add a new literal to specialize NewRule*
 - 1. $Candidate\ literals \leftarrow$ generate candidates
 - 2. $Best_literal \leftarrow \operatorname{argmax}_{L \in Candidate\ literals} Performance(SpecializeRule(NewRule; L))$
 - 3. add $Best_literal$ to $NewRule$ preconditions
 - 4. $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies $NewRule$ preconditions
 - $Learned_rules \leftarrow Learned_rules + NewRule$
 - $Pos \leftarrow Pos - \{members\ of\ Pos\ covered\ by\ NewRule\}$
- Return $Learned_rules$



Subtleties: Learn One Rule

1. May use *beam search*
2. Easily generalizes to multi-valued target functions
3. Choose evaluation function to guide search:

- Entropy (i.e., information gain)

- Sample accuracy: $\frac{n_c}{n}$

where n_c = correct rule predictions, n = all predictions

- m estimate: $\frac{n_c + mp}{n + m}$



Variants of Rule Learning Programs

- *Sequential* or *simultaneous* covering of data?
- General \rightarrow specific, or specific \rightarrow general?
- Generate-and-test, or example-driven?
- Whether and how to post-prune?
- What statistical evaluation function?



Learning First Order Rules

Why do that?

- Can learn sets of rules such as

$Ancestor(x, y) \leftarrow Parent(x; y)$

$Ancestor(x; y) \leftarrow Parent(x; z) \wedge Ancestor(z; y)$

- General purpose programming language
PROLOG : programs are sets of such rules



First Order Rule for Classifying Web Pages

[Slattery, 1997]

course(A) \leftarrow

has-word(A, instructor),

Not has-word(A, good),

link-from(A, B),

has-word(B, assign),

Not link-from(B, C)

Train: 31/31, Test: 31/34

FOIL(*Target_predicate*, *Predicates*, *Examples*)

- $Pos \leftarrow$ positive *Examples*
- $Neg \leftarrow$ negative *Examples*
- while Pos , do
 - Learn a NewRule*
 - $NewRule \leftarrow$ most general rule possible
 - $NewRuleNeg \leftarrow Neg$
 - while $NewRuleNeg$, do
 - Add a new literal to specialize NewRule*
 - 1. $Candidate_literals \leftarrow$ generate candidates
 - 2. $Best_literal \leftarrow$
 $\operatorname{argmax}_{L \in Candidate_literals} Foil_Gain(L, NewRule)$
 - 3. add $Best_literal$ to $NewRule$ preconditions
 - 4. $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$
that satisfies $NewRule$ preconditions
 - $Learned_rules \leftarrow Learned_rules + NewRule$
 - $Pos \leftarrow Pos - \{\text{members of } Pos \text{ covered by } NewRule\}$
- Return $Learned_rules$

Specializing Rules in FOIL

Learning rule: $P(x_1, x_2, \dots, x_k) \leftarrow L_1 \dots L_n$

Candidate specializations add new literal of form:

- $Q(v_1, \dots, v_r)$, where at least one of the v_i in the created literal must already exist as a variable in the rule.
- $Equal(x_j, x_k)$, where x_j and x_k are variables already present in the rule
- The negation of either of the above forms of literals

Information Gain in FOIL

$$Foil_Gain(L, R) \equiv t \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

Where

- L is the candidate literal to add to rule R
- p_0 = number of positive bindings of R
- n_0 = number of negative bindings of R
- p_1 = number of positive bindings of $R + L$
- n_1 = number of negative bindings of $R + L$
- t is the number of positive bindings of R also covered by $R + L$

Note

- $-\log_2 \frac{p_0}{p_0 + n_0}$ is optimal number of bits to indicate the class of a positive binding covered by R



Induction as Inverted Deduction

Induction is finding h such that

$$(\forall \langle x_i, f(x_i) \rangle \in D) \ B \wedge h \wedge x_i \vdash f(x_i)$$

where

- x_i is i th training instance
- $f(x_i)$ is the target function value for x_i
- B is other background knowledge

So let's design inductive algorithm by inverting operators for automated deduction!



Induction as Inverted Deduction(Cont')

“pairs of people, $\langle u, v \rangle$ such that child of u is v ,”

$f(x_i) :$ $Child(Bob, Sharon)$

$x_i :$ $Male(Bob), Female(Sharon), Father(Sharon, Bob)$

$B :$ $Parent(u, v) \leftarrow Father(u, v)$

What satisfies $(\forall \langle x_i, f(x_i) \rangle \in D) B \wedge h \wedge x_i \vdash f(x_i)$?

$h_1 : Child(u, v) \leftarrow Father(v, u)$

$h_2 : Child(u, v) \leftarrow Parent(v, u)$



Induction as Inverted Deduction(Cont')

Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise. Who thinks of asking whether addition or subtraction is the more important process in arithmetic? But at the same time much difference in difficulty may exist between a direct and inverse operation; : : : it must be allowed that inductive investigations are of a far higher degree of difficulty and complexity than any questions of deduction....

(Jevons 1874)



Induction as Inverted Deduction(Cont')

We have mechanical *deductive* operators

$F(A, B) = C$, where $A \wedge B \vdash C$

need *inductive* operators

$O(B, D) = h$ where $(\forall \langle x_i, f(x_i) \rangle \in D) (B \wedge h \wedge x_i) \vdash f(x_i)$

Induction as Inverted Deduction(Cont')

Positives:

- Subsumes earlier idea of finding h that “fits” training data
- Domain theory B helps define meaning of “fit” the data

$$B \wedge h \wedge x_i \vdash f(x_i)$$

- Suggests algorithms that search H guided by B



Induction as Inverted Deduction(Cont')

Negatives:

- Doesn't allow for noisy data. Consider

$$(\forall \langle x_i, f(x_i) \rangle \in D) (B \wedge h \wedge x_i) \vdash f(x_i)$$

- First order logic gives a *huge* hypothesis space H
 - overfitting...
 - intractability of calculating all acceptable h 's

Deduction: Resolution Rule

$$\frac{P \vee L \quad \neg L \vee R}{P \vee R}$$

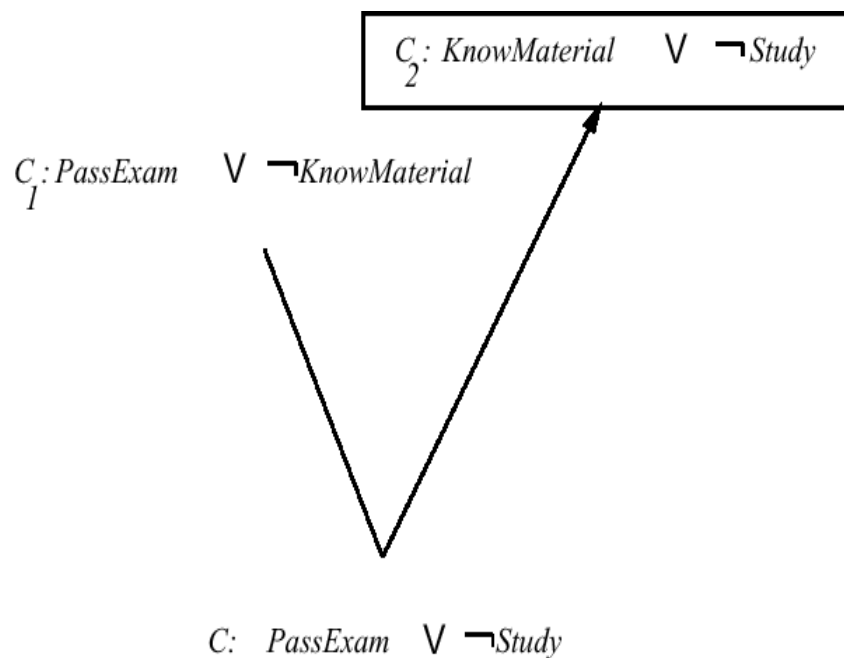
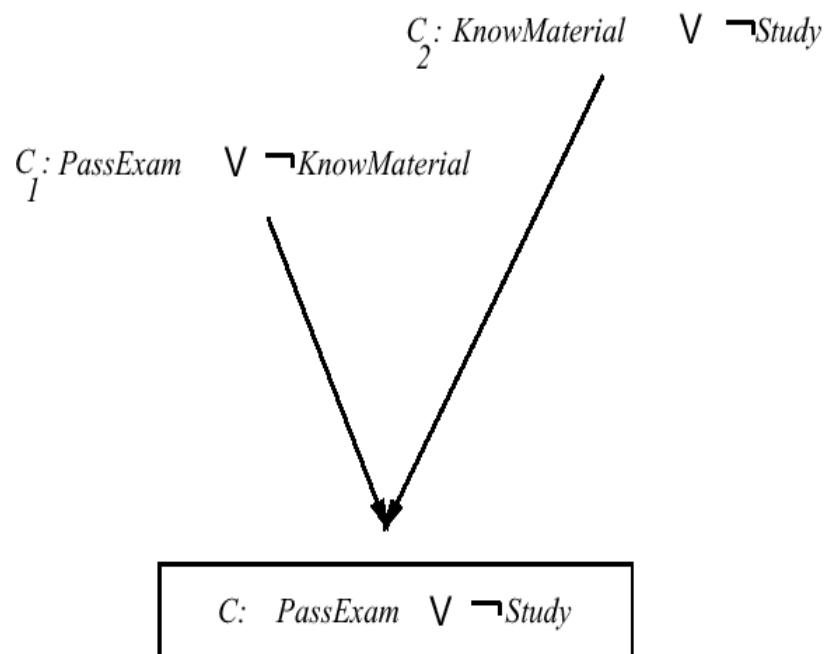
1. Given initial clauses C_1 and C_2 , find a literal L from clause C_1 such that $\neg L$ occurs in clause C_2
2. Form the resolvent C by including all literals from C_1 and C_2 , except for L and $\neg L$. More precisely, the set of literals occurring in the conclusion C is

$$C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$$

where \cup denotes set union, and “ $-$ ” denotes set difference.



Inverting Resolution





Inverted Resolution (Propositional)

1. Given initial clauses C_1 and C , find a literal L that occurs in clause C_1 , but not in clause C .
2. Form the second clause C_2 by including the following literals

$$C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$$

First order resolution

First order resolution:

1. Find a literal L_1 from clause C_1 , literal L_2 from clause C_2 , and substitution θ such that $L_1\theta = \neg L_2\theta$
2. Form the resolvent C by including all literals from $C_1\theta$ and $C_2\theta$, except for $L_1\theta$ and $\neg L_2\theta$. More precisely, the set of literals occurring in the conclusion C is

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$$

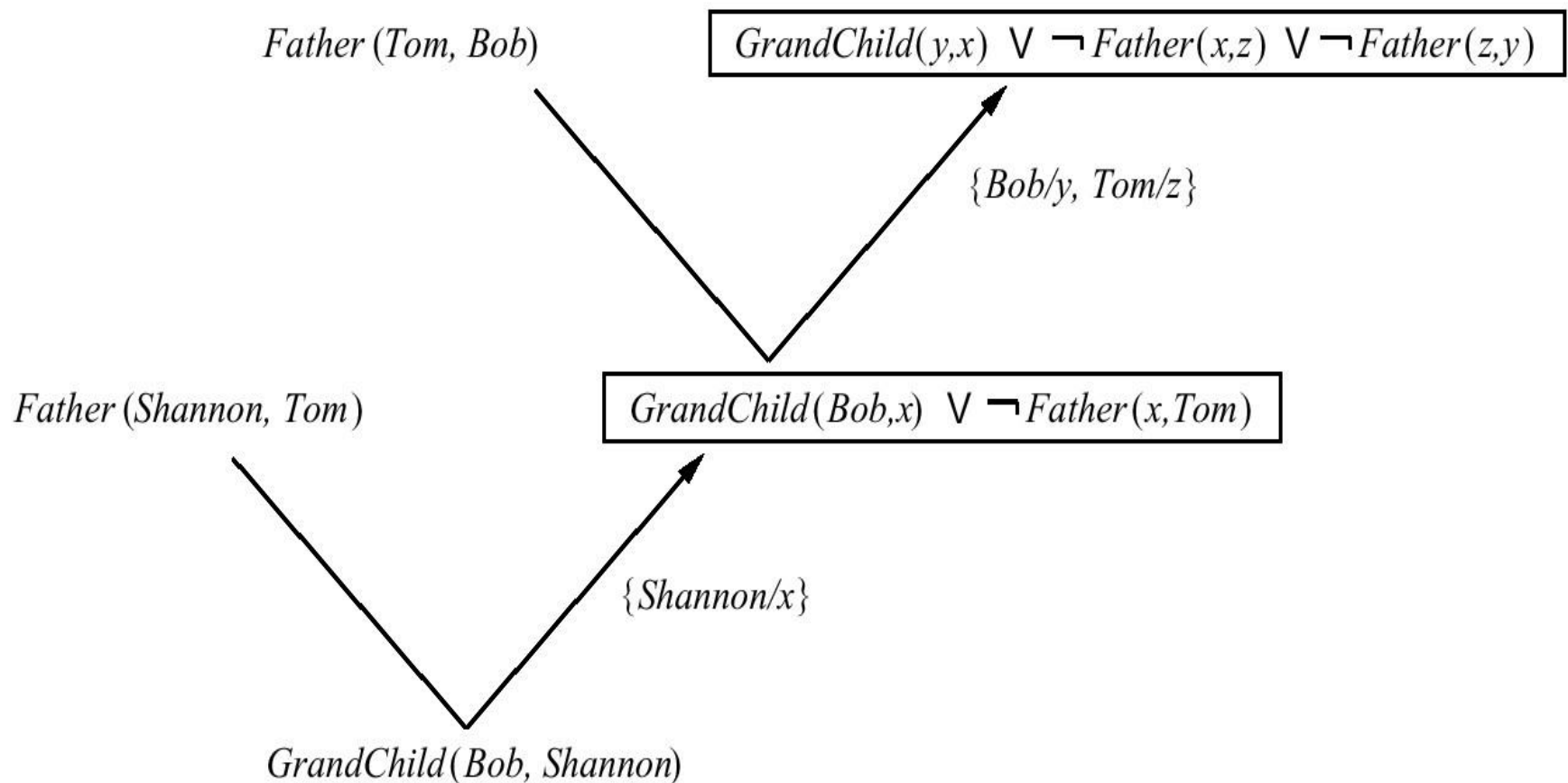


Inverting First order resolution

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\}$$



Cigol



Progol

PROGOL: Reduce comb explosion by generating the most specific acceptable h

1. User specifies H by stating predicates, functions, and forms of arguments allowed for each
2. PROGOL uses sequential covering algorithm.
For each $\langle x_i, f(x_i) \rangle$
 - Find most specific hypothesis h_i s.t.
 $B \wedge h_i \wedge x_i \vdash f(x_i)$
 - actually, considers only k -step entailment
3. Conduct general-to-specific search bounded by specific hypothesis h_i , choosing hypothesis with minimum description length



Machine Learning

Chapter 13. Reinforcement Learning

Tom M. Mitchell



Control Learning

Consider learning to choose actions, e.g.,

- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon

Note several problem characteristics:

- Delayed reward
- Opportunity for active exploration
- Possibility that state only partially observable
- Possible need to learn multiple tasks with same sensors/actuators



One Example: TD-Gammon

Learn to play Backgammon

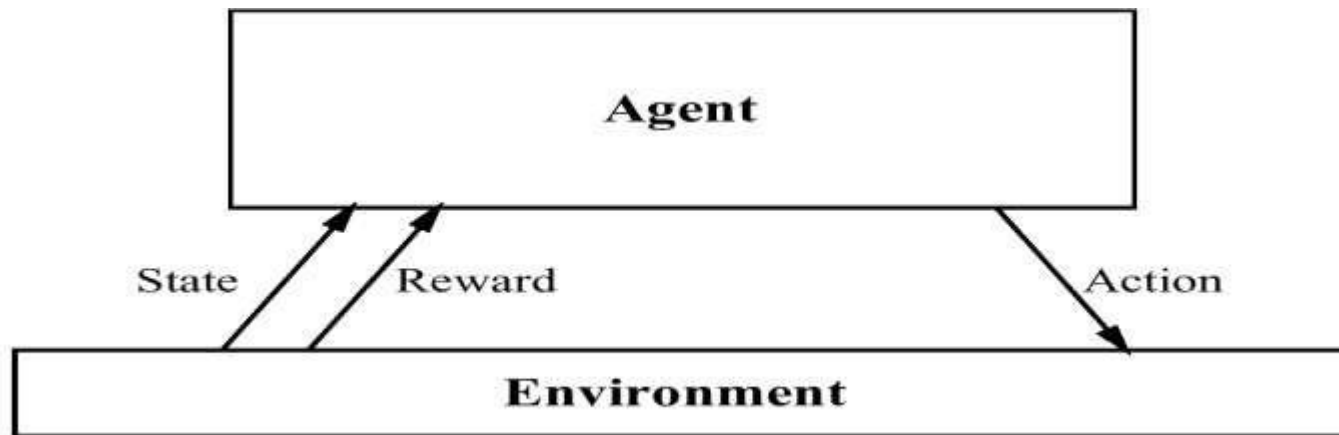
Immediate reward

- +100 if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games against itself

Now approximately equal to best human player

Reinforcement Learning Problem



Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ where } 0 \leq \gamma < 1$$



Markov Decision Processes

Assume

- finite set of states S
- set of actions A
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- then receives immediate reward r_t
- and state changes to s_{t+1}
- Markov assumption : $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
 - i.e., r_t and s_{t+1} depend only on *current* state and action
 - functions δ and r may be nondeterministic
 - functions δ and r not necessarily known to agent



Agent's Learning Task

Execute actions in environment, observe results, and

- learn action policy $\pi : S \rightarrow A$ that maximizes

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

from any starting state in S

- here $0 \leq \gamma < 1$ is the discount factor for future rewards

Note something new:

- Target function is $\pi : S \rightarrow A$
- but we have no training examples of form $\langle s, a \rangle$
- training examples are of form $\langle \langle s, a \rangle, r \rangle$



Value Function

To begin, consider deterministic worlds...

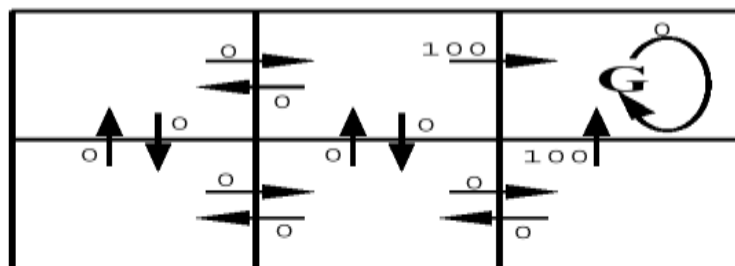
For each possible policy π the agent might adopt, we can define an evaluation function over states

$$\begin{aligned} V^\pi(s) &\equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

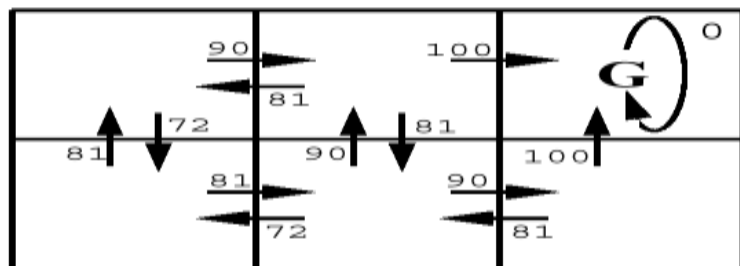
where r_t, r_{t+1}, \dots are generated by following policy π starting at state s

Restated, the task is to learn the optimal policy π^*

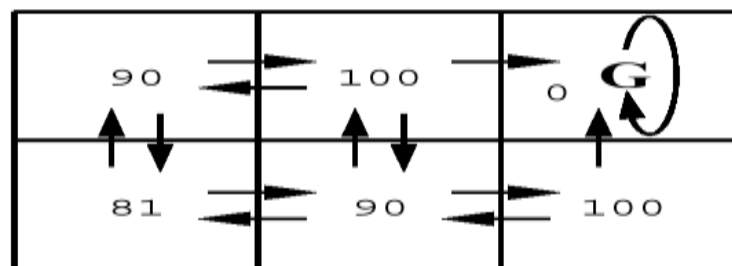
$$\pi^* \equiv \operatorname{argmax}_{\pi} V^\pi(s), (\forall s)$$



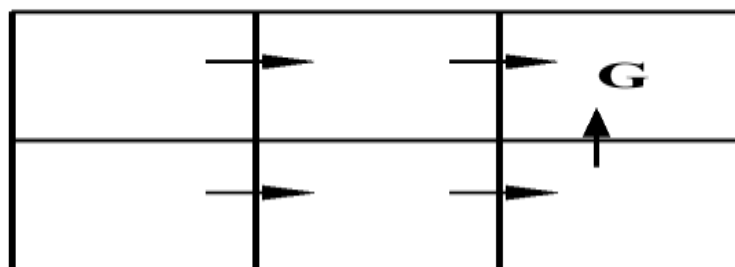
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy



What to Learn

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)

It could then do a lookahead search to choose best action from any state s because

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

A problem:

- This works well if agent knows $\delta : S \times A \rightarrow S$, and $r : S \times A \rightarrow \mathbb{R}$
- But when it doesn't, it can't choose actions this way



Q Function

Define new function very similar to V^*

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

If agent learns Q , it can choose optimal action even without knowing δ !

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

Q is the evaluation function the agent will learn

Training Rule to Learn Q

Note Q and V^* closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

Which allows us to write Q recursively as

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Nice! Let \hat{Q} denote learner's current approximation to Q . Consider training rule

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

where s' is the state resulting from applying action a in state s



Q Learning for Deterministic Worlds

For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state s

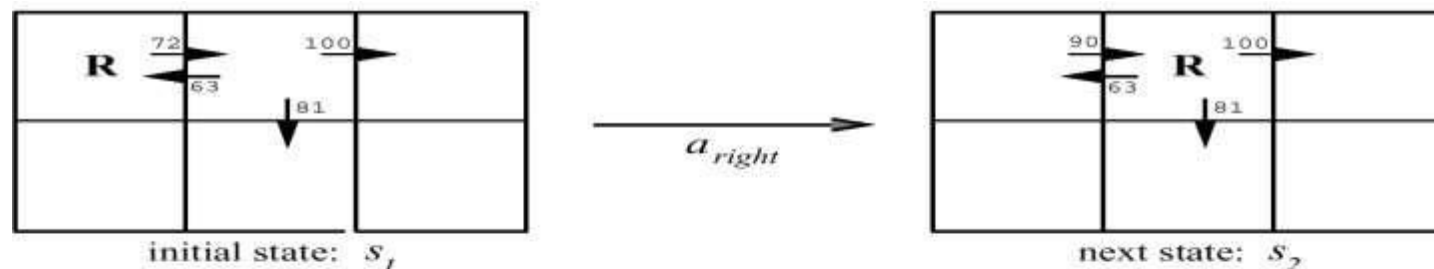
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

Updating \hat{Q}



$$\begin{aligned}
 \hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
 &\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\
 &\leftarrow 90
 \end{aligned}$$

notice if rewards non-negative, then

$$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

and

$$(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

\hat{Q} converges to Q . Consider case of deterministic world where see each $\langle s, a \rangle$ visited infinitely often.

Proof: Define a full interval to be an interval during which each $\langle s, a \rangle$ is visited. During each full interval the largest error in \hat{Q} table is reduced by factor of γ

Let \hat{Q}_n be table after n updates, and Δ_n be the maximum error in \hat{Q}_n ; that is

$$\Delta_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$$

For any table entry $\hat{Q}_n(s, a)$ updated on iteration $n + 1$, the error in the revised estimate $\hat{Q}_{n+1}(s, a)$ is

$$\begin{aligned} |\hat{Q}_{n+1}(s, a) - Q(s, a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\ &= \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')| \\ &\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\ &\leq \gamma \max_{s'', a'} |\hat{Q}_n(s'', a') - Q(s'', a')| \\ |\hat{Q}_{n+1}(s, a) - Q(s, a)| &\leq \gamma \Delta_n \end{aligned}$$



Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine V, Q by taking expected values

$$\begin{aligned} V^\pi(s) &\equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots] \\ &\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right] \end{aligned}$$

$$Q(s, a) \equiv E[r(s, a) + \gamma V^*(\delta(s, a))]$$



Nondeterministic Case(Cont')

Q learning generalizes to nondeterministic worlds

Alter training rule to

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n [r + \max_{a'} \hat{Q}_{n-1}(s', a')]$$

where

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

Can still prove convergence of \hat{Q} to Q [Watkins and Dayan, 1992]



Temporal Difference Learning

Q learning: reduce discrepancy between successive Q estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

Why not two steps?

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

Or n ?

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

Blend all of these:

$$Q^\lambda(s_t, a_t) \equiv (1-\lambda) [Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \cdots]$$



Temporal Difference Learning(Cont')

$$Q^\lambda(s_t, a_t) \equiv (1-\lambda) [Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots]$$

Equivalent expression:

$$Q^\lambda(s_t, a_t) = r_t + \gamma [(1 - \lambda) \max_a \hat{Q}(s_t, a) + \lambda Q^\lambda(s_{t+1}, a_{t+1})]$$



TD(λ) algorithm uses above training rule

- Sometimes converges faster than Q learning
- converges for learning V^* for any $0 \leq \lambda \leq 1$ (Dayan, 1992)
- Tesauro's TD-Gammon uses this algorithm




Subtleties and Ongoing Research

- Replace \hat{Q} table with neural net or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $\hat{\delta} : S \times A \rightarrow S$
- Relationship to dynamic programming



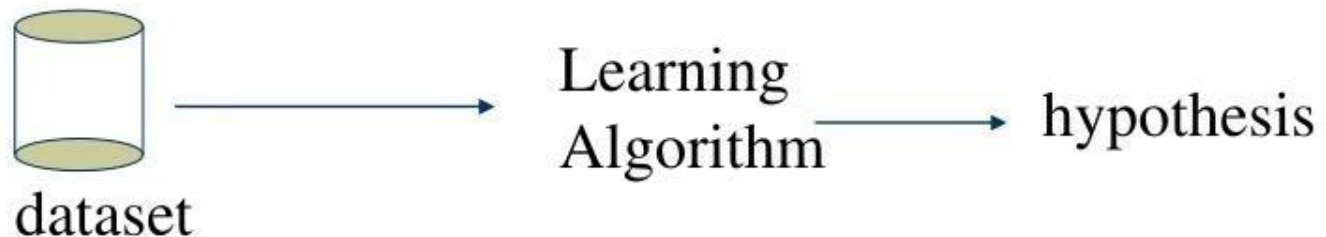
Analytical Learning



- *Introduction*
- *Learning with Perfect Domain Theories*
- *Explanation-Based Learning*
- Search Control Knowledge
- Summary

Introduction

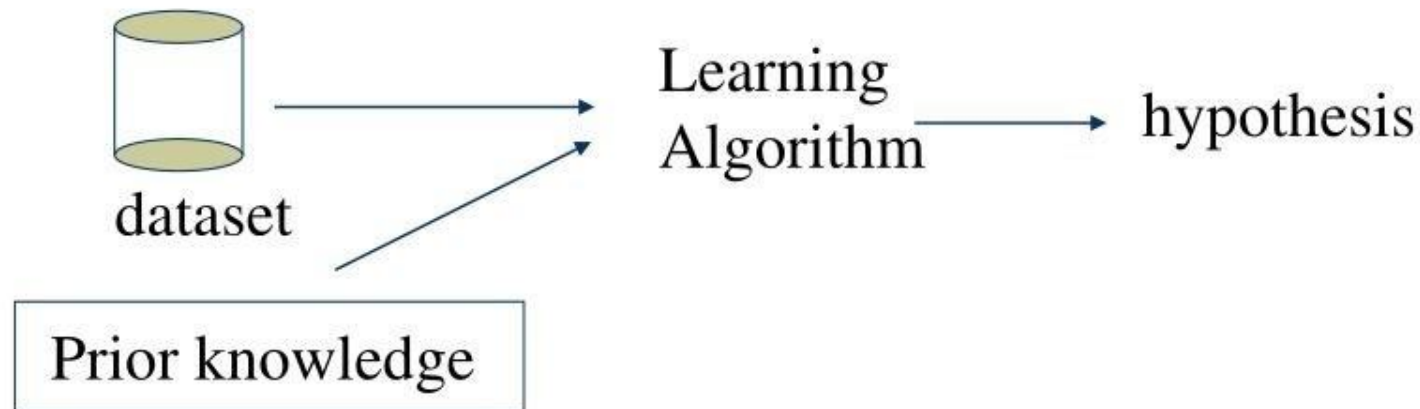
Learning algorithms like neural networks, decision trees, inductive logic programming, etc. all require a good number of examples to be able to do good predictions.



Dataset must be sufficiently large

Introduction

We don't need a large dataset if besides taking examples as input, the learning algorithm can take prior knowledge.



Dataset does not need to be large

Introduction

Explanation-based learning uses prior knowledge to reduce the size of the hypothesis space.



It analyzes each example to infer which features are relevant and which ones are irrelevant.



Example



Learning to Play Chess

Suppose we want to learn a concept like “what is a board position in which black will lose the queen in x moves?”.

Chess is a complex game. Each piece can occupy many positions. We would need many examples to learn this concept.

But humans can learn these type of concepts with very few examples. Why?


Example

Humans can analyze an example and use prior knowledge related to legal moves.

From there it can generalize with only few examples.

Example: why is this a positive example?

“Because the white king is attacking both king and queen; black must avoid check, letting white capture the queen”

reasoning 

Example

What is the prior knowledge involved in playing chess?

It is knowledge about the rules of chess:

- Legal moves for the knight and other pieces.
- Players alternate moves in games.
- To win you must capture the opponent's king.

Inductive and Analytical Learning

Inductive Learning

Input: HS, D,
Output: hypothesis h
h is consistent with D

Analytical Learning

Input: HS, D, B
Output: h
h is consistent with D and B
($\sim h \not\models B$)

HS: Hypothesis Space

D: Training Set

B: Background knowledge

Example Analytical Learning

Input:

- Dataset where each instance is a pair of objects represented by the following predicates: Color, Volume, Owner, Material, Density, On. Example:

On(Obj1,Obj2)
Type(Obj1,Box)
Type(Obj2,Endtable)
Color(Obj1,Red)
Color(Obj2,Blue)
Volume(Obj1,2)

Owner(Obj1,Fred)
Owner(Obj2,Louise)
Density(Obj1,0.3)
Material(Obj1,Cardboard)
Material(Obj2,Wood)

Example Analytical Learning

Hypothesis space: set of Horn clause rules.

The head of each rule has the predicate SafeToStack.

Example:

$$\text{SafeToStack}(x,y) \leftarrow \text{Volume}(x,vx) \wedge \text{Volume}(y,vy) \\ \wedge \text{LessThan}(vx,vy)$$

Domain Theory:

$$\text{SafeToStack}(x,y) \leftarrow \sim \text{Fragile}(y)$$

$$\text{SafeToStack}(x,y) \leftarrow \text{Lighter}(x,y)$$

$$\text{Lighter}(x,y) \leftarrow \text{Weight}(x,wx) \wedge \text{Weight}(y,wy) \wedge \text{LessThan}(wx,wy)$$

Example Analytical Learning

Domain Theory:

$\text{SafeToStack}(x,y) \leftarrow \sim \text{Fragile}(y)$

$\text{SafeToStack}(x,y) \leftarrow \text{Lighter}(x,y)$

$\text{Lighter}(x,y) \leftarrow \text{Weight}(x,w_x) \wedge \text{Weight}(y,w_y) \wedge \text{LessThan}(w_x,w_y)$

...

$\text{Fragile}(x) \leftarrow \text{Material}(x,\text{Glass})$

Note:

- The domain theory refers to predicates not contained in the examples.
- The domain theory is sufficient to prove the example is true.



Perfect Domain Theories



A domain theory is **correct** if each statement is true.

A domain theory is **complete** if it covers every positive example of the instance space (w.r.t a target concept and instance space).

A **perfect** domain theory is correct and complete.



Perfect Domain Theories

Examples of where to find perfect domain theories:

Rules of chess

Examples of where not to find perfect domain theories:

SafetoStack problem

We will look into learning problems with perfect domain theories only.

Explanation Based Learning Algorithm

We consider an algorithm that has the following properties:

- ✓ It is a sequential covering algorithm considering the data incrementally
- ✓ For each positive example not covered by the current rules it forms a new rule by:
 - Explaining the new positive example.
 - Analyzing the explanation to find a generalization.
 - Refining the current hypothesis by adding a new Horn Clause rule to cover the example.

Explanation Based Learning Algorithm

Prolog-EBG (Kedar-Cabelli and McCarty 87).

1. $\text{LearnedRules} \leftarrow \{\}$
2. $\text{Pos} \leftarrow$ Positive examples from training examples
3. For each positive example X in Pos not covered by LearnedRules do
 - a. Explain
Use the domain theory to explain that X satisfies the target.

Explanation Based Learning Algorithm

Prolog-EBG (Kedar-Cabelli and McCarty 87).

b. Analyze

Find the most general set of features of X sufficient to satisfy the target according to the explanation.

c. Refine

LearnedRules += NewHornClause

NewHornClause: Target \leftarrow sufficient features

4. Return LearnedRules

Explaining the Example

1. For each positive example X in Pos not covered by LearnedRules do
 - a. *Explain*
Use the domain theory to explain that X satisfies the target concept.

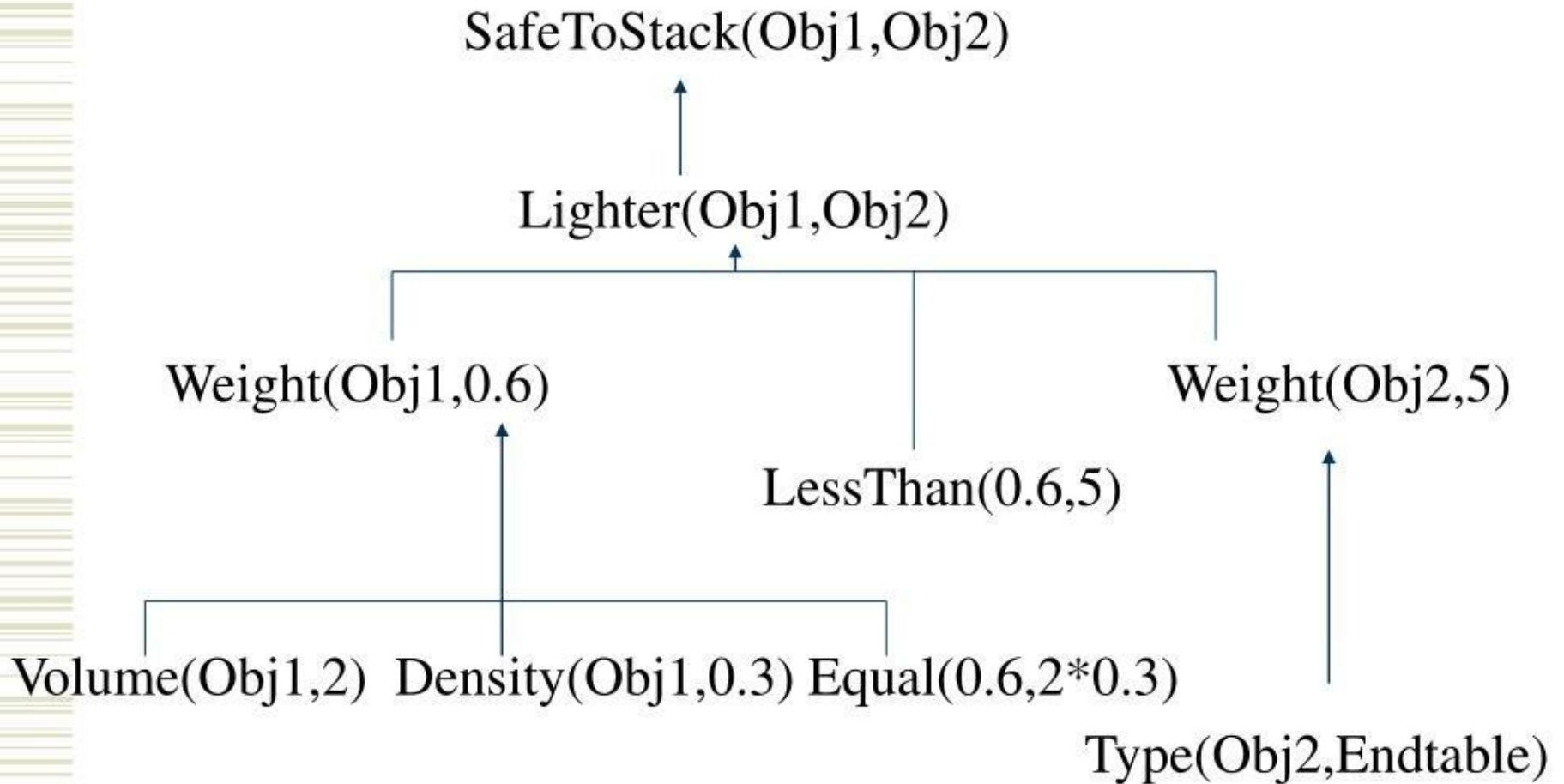
Explaining the Example

The explanation is a proof that the example belongs to the target (if the theory is perfect):

On(Obj1,Obj2)
Type(Obj1,Box)
Type(Obj2,Endtable)
Color(Obj1,Red)
Color(Obj2,Blue)
Volume(Obj1,2)

Owner(Obj1,Fred)
Owner(Obj2,Louise)
Density(Obj1,0.3)
Material(Obj1,Cardboard)
Material(Obj2,Wood)

Explanation



Explanation

Considerations:

- ❑ There might be more than one explanation to the example. In that case one or all explanations may be used.
- ❑ An explanation is obtained using a backward chaining search as is done by Prolog. Prolog-EBG stops when it finds the first proof.



Analyze



Many features appear in an example.
Of them, how many are truly relevant?

We consider as relevant those features that show in the explanation.

Example:

Relevant feature: Density

Irrelevant feature: Owner

Analyze

Taking the leaves of the explanation and substituting variables x and y for Obj1, and Obj2:

$$\text{SafeToStack}(x,y) \leftarrow \text{Volume}(x,2) \wedge \text{Density}(x,0.3) \\ \wedge \text{Type}(y,\text{Endtable})$$

Analyze

Considerations:

- ❖ We omit features independent of x and y such as `Equal(0.6,times(2,0.3))` and `LessThan(0.6,5)`.
- ❖ The rule is now more general and can serve to explain other instances matching the rule.
- ❖ A more general form of generalization called regression finds the most general rule explaining the example.

Refine

- The current hypothesis is the set of Horn clauses that we have constructed up to this point.
- Using sequential covering we keep adding more rules, thus refining our hypothesis.
- A new instance is negative if it is not covered by any rule.



Remarks

- Explanation Based Learning (EBL) justifies the hypothesis by using prior knowledge.
- The explanation or proof shows which features are relevant.
- Each Horn clause is a sufficient condition for satisfying the target concept.



Remarks



- The success of the method depends on how the domain theory is formulated.
- We assumed the theory is correct and complete.
If this is not the case the learned concept may be incorrect.



Analytical Learning



- Introduction
- Learning with Perfect Domain Theories
- *Explanation-Based Learning*
- *Search Control Knowledge*
- *Summary*

Discovering New Features

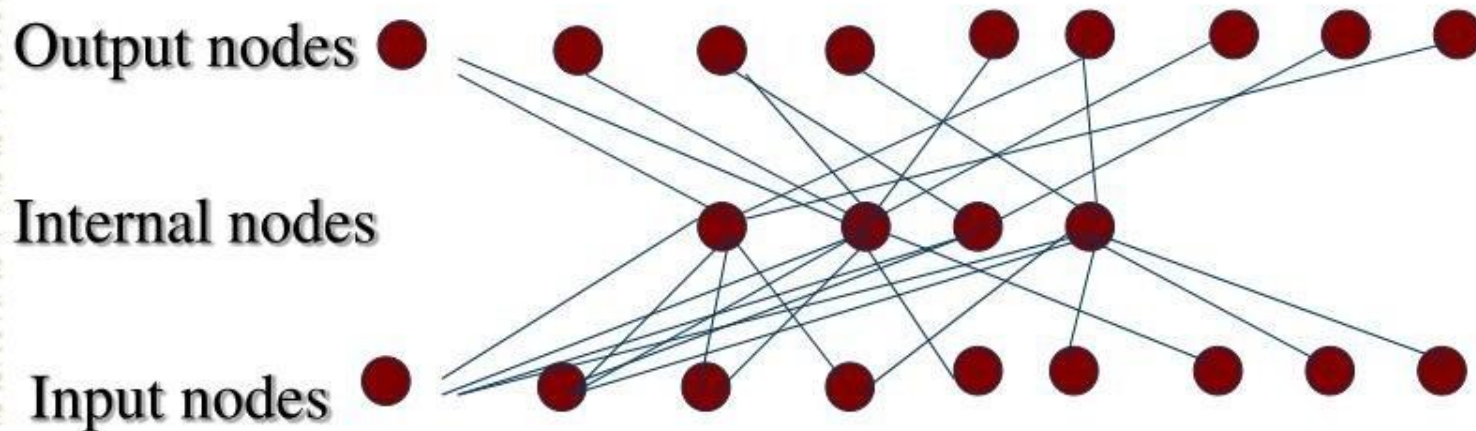
The Prolog-EBG system we described before can formulate new features that do not show up in the examples.

Example: $\text{Volume} * \text{Density} > 5$
(derived from the domain theory)

This is different from neural networks using hidden nodes. why?

Discovering New Features

This is different from neural networks using hidden nodes. why?



Inductive Bias in Explanation-Based Learning

What is the inductive bias of explanation based learning?

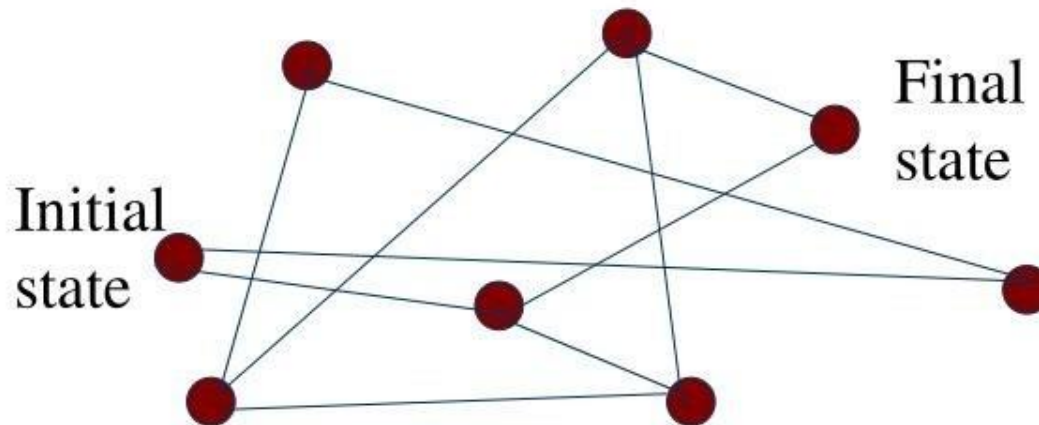
The hypothesis h follows deductively from D and B

D : database B : Background knowledge

Bias: Prefer small sets of maximally general Horn Clauses

Search Control Knowledge

Problem: learning to speed up search programs.
This is also called “speedup learning”



How do we improve our search control strategy to find a solution quickly?

Search Control Knowledge

Examples include: playing chess scheduling and optimization problems.

Problem formulation:

S: set of possible search states

O: set of legal operators (transform one state into another state)

G: predicate over S indicating the goal states

Search Control Knowledge

Example:

N
E
S
R

V

U

L

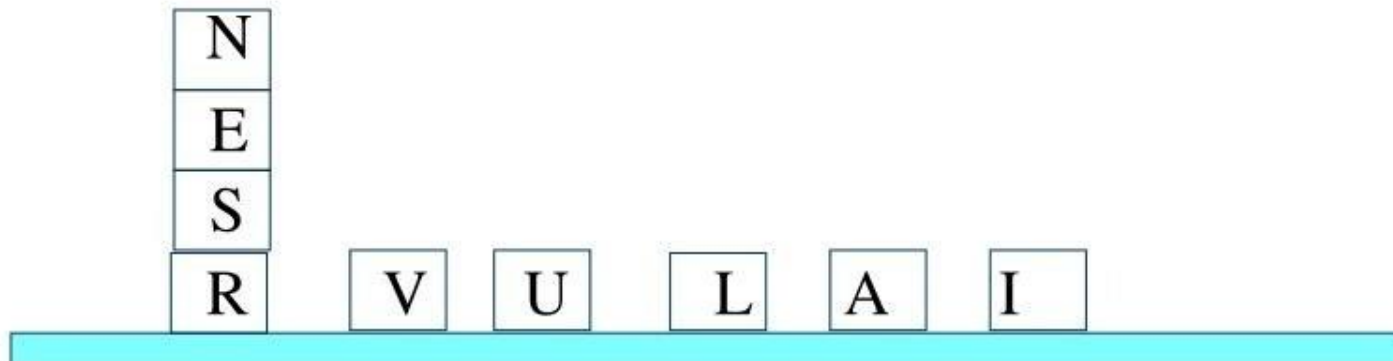
A

I

Learn an algorithm to stack the blocks
so that it reads “UNIVERSAL”

Search Control Knowledge

- S: all possible configurations of blocks on the table
- O: {(MS x) move block x to stack if x is on table,
(MT x) move block x to table if x is on the stack}
- G: $G(s_i) = \text{true}$ if s_i is the configuration where the blocks read UNIVERSAL



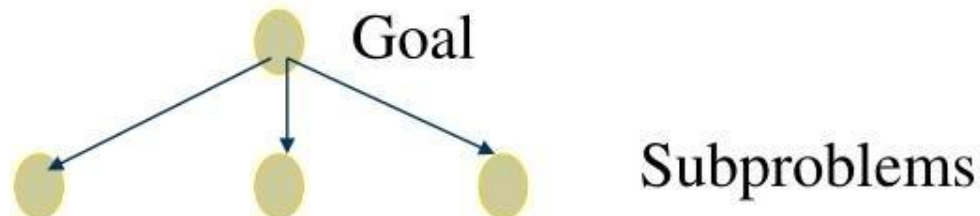
Prodigy

Prodigy is a planning system.

Input: state space S and operators O .

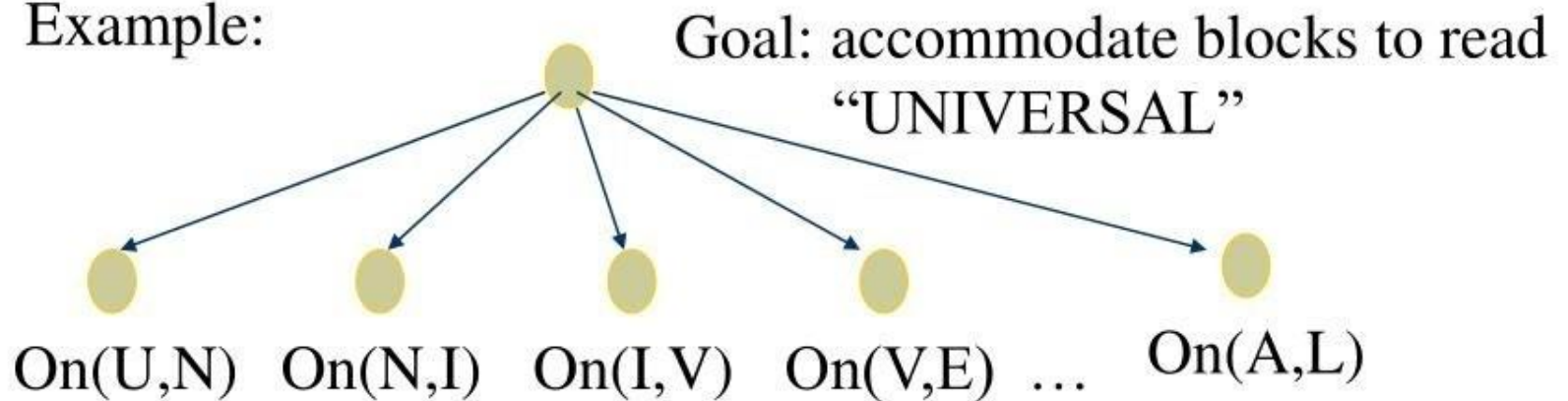
Output: A sequence of operators that lead from the initial state to the final state.

Prodigy uses a **means-end planner**: we decompose goals into subgoals:



Prodigy

Example:



Question: what subgoal should be attacked first?

Answer is given by *search control knowledge*

Prodigy and Explanation Based Learning

Prodigy defines a set of target concepts to learn, e.g., which operator given the current state takes you to the goal state?

An example of a rule learned by Prodigy in the block-stacking problem is:

IF	One subgoal to be solved is $\text{On}(x,y)$ AND
	One subgoal to be solved is $\text{On}(y,z)$
THEN	Solve the subgoal $\text{On}(y,z)$ before $\text{On}(x,y)$



Prodigy and Explanation Based Learning



The rationale behind the rule is that it would avoid a conflict when stacking blocks.

Prodigy learns by first encountering a conflict, then explaining the reason for the conflict and creating a rule like the one above.

Experiments show an improvement in efficiency by a factor of two to four.

Problems with EBL

- ✓ The number of control rules that must be learned is very large.
- ✓ If the control rules are many, much time will be spent looking for the best rule.

Utility analysis is used to determine what rules to keep and what rules to forget.

Prodigy:

328 possible rules \longrightarrow 69 pass test \longrightarrow 19 were retained

Problems with EBL

- ✓ Another problem with EBL is that it is sometimes intractable to create an explanation for the target concept.

For example, in chess, learning a concept like:
“states for which operator A leads to a solution”
The search here grows exponentially.

Summary

- ❖ Different from inductive learning, analytical learning looks for a hypothesis that fit the background knowledge and covers the training examples.
- ❖ Explanation based learning is one kind of analytical learning that divides into three steps:
 - a. Explain the target value for the current example
 - b. Analyze the explanation (generalize)
 - c. Refine the hypothesis



Summary



- ❖ Prolog-EBG constructs intermediate features after analyzing examples.
- ❖ Explanation based learning can be used to find search control rules.
- ❖ In all cases we depend on a perfect domain theory.



Machine Learning

Chapter 12. Combining Inductive and Analytical Learning

Tom M. Mitchell



Inductive and Analytical Learning

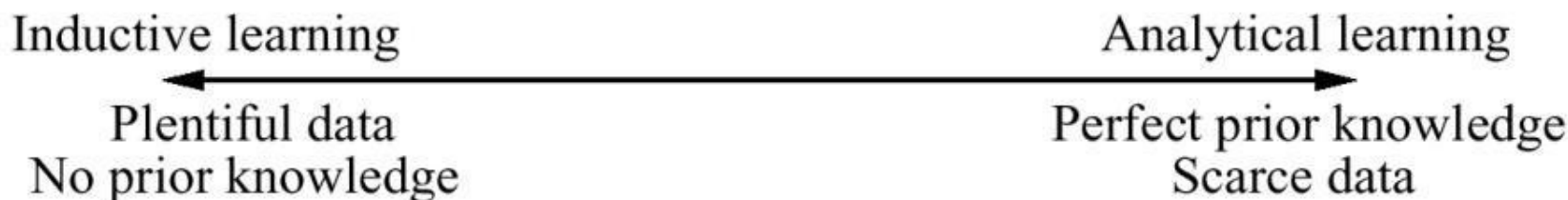
Inductive learning

- Hypothesis fits data
- Statistical inference
- Requires little prior knowledge
- Syntactic inductive bias

Analytical learning

- Hypothesis fits domain the
- Deductive inference
- Learns from scarce data
- Bias is domain theory

What We Would Like



General purpose learning method:

- No domain theory \rightarrow learn as well as inductive methods
- Perfect domain theory \rightarrow learn as well as Prolog-EBG
- Accomodate arbitrary and unknown errors in domain theory
- Accomodate arbitrary and unknown errors in training data



Domain theory:

Cup \leftarrow Stable, Lifiable, Open Vessel

Stable \leftarrow BottomIsFlat

Lifiable \leftarrow Graspable, Light

Graspable \leftarrow HasHandle

Open Vessel \leftarrow HasConcavity, ConcavityPointsUp

Training examples:

	Cups				Non-Cups				
BottomIsFlat	✓	✓	✓	✓	✓	✓	✓		✓
ConcavityPoints Up	✓	✓	✓	✓	✓		✓	✓	
Expensive	✓		✓				✓	✓	
Fragile	✓	✓			✓	✓	✓		✓
HandleOnTop					✓		✓		
HandleOnSide	✓			✓				✓	
HasConcavity	✓	✓	✓	✓	✓		✓	✓	✓
HasHandle	✓			✓	✓		✓	✓	
Light	✓	✓	✓	✓	✓	✓	✓	✓	
MadeOfCeramic	✓				✓		✓	✓	
MadeOfPaper				✓				✓	
MadeOfStyrofoam		✓	✓			✓			✓

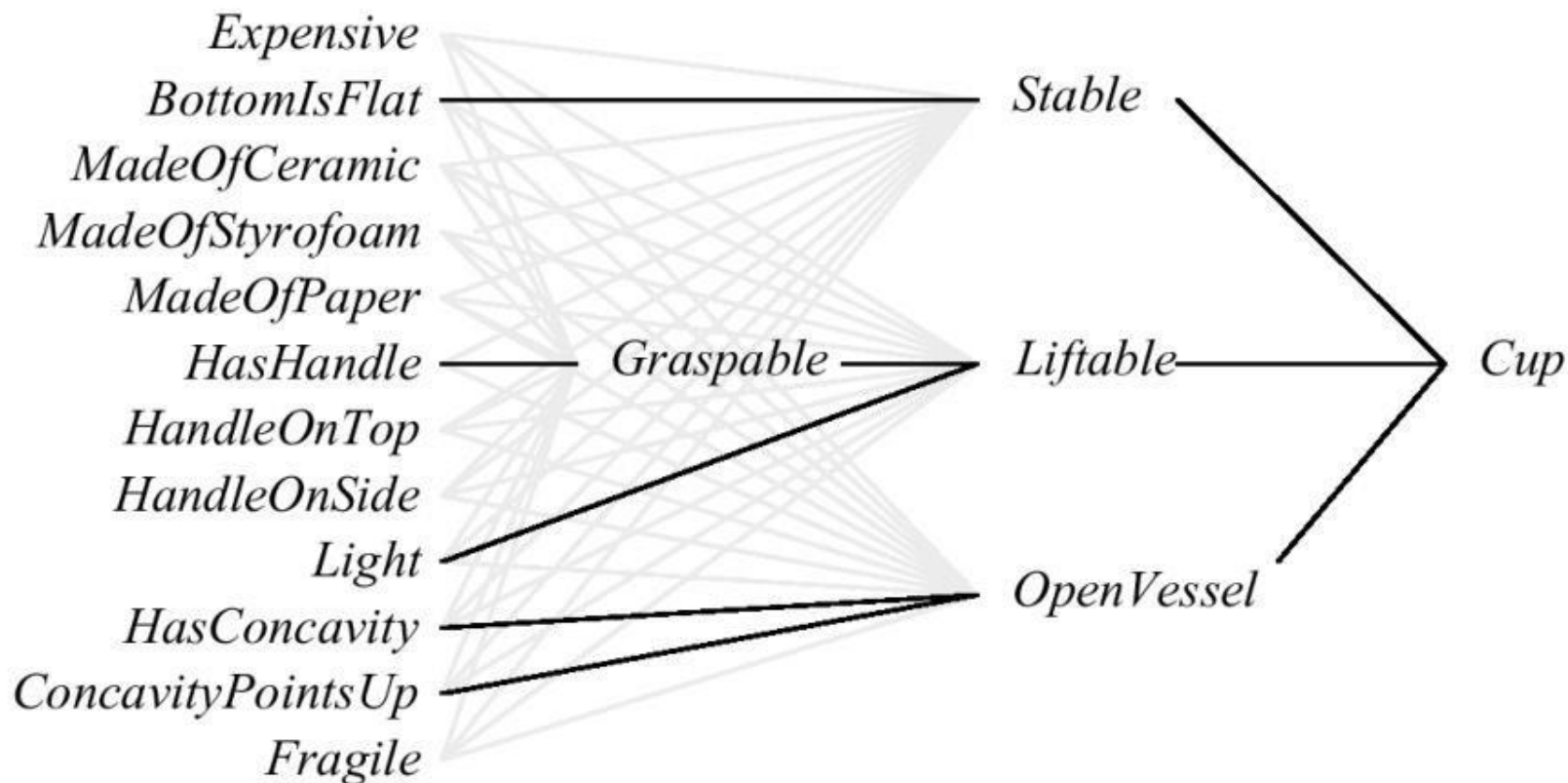


KBANN

- KBANN (data D , domain theory B)
 1. Create a feedforward network h equivalent to B
 2. Use BACKPROP to tune h to $t D$



Neural Net Equivalent to Domain Theory





Creating Network Equivalent to Domain Theory

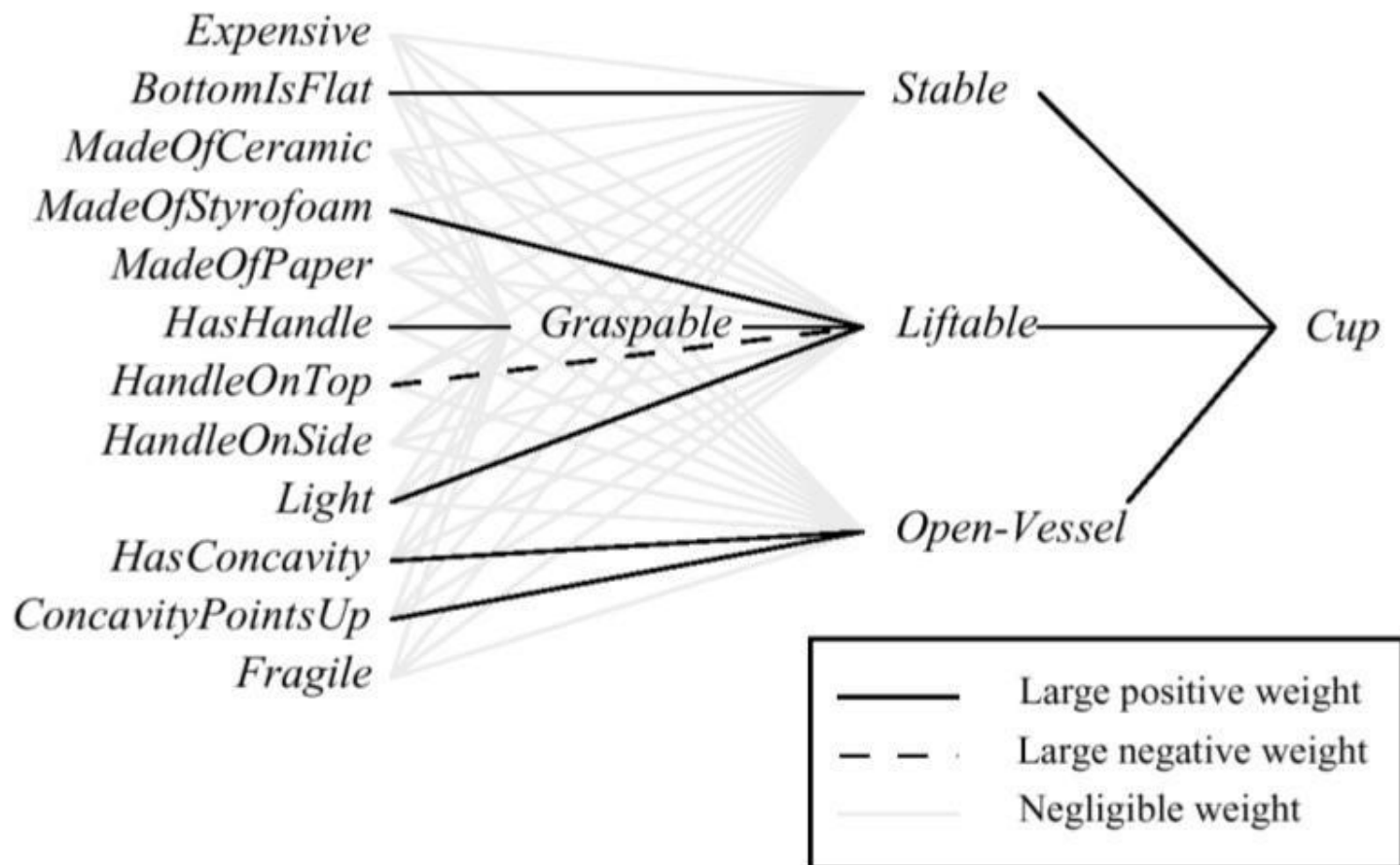
Create one unit per horn clause rule (i.e., an AND unit)

- Connect unit inputs to corresponding clause antecedents
- For each non-negated antecedent, corresponding input weight $w \leftarrow W$, where W is some constant
- For each negated antecedent, input weight $w \leftarrow -W$
- Threshold weight $w_0 \leftarrow -(n-.5)W$, where n is number of non-negated antecedents

Finally, add many additional connections with near-zero weights

$$\textit{Liftable} \leftarrow \textit{Graspable}, \neg \textit{Heavy}$$

Result of refining the network





KBANN Results

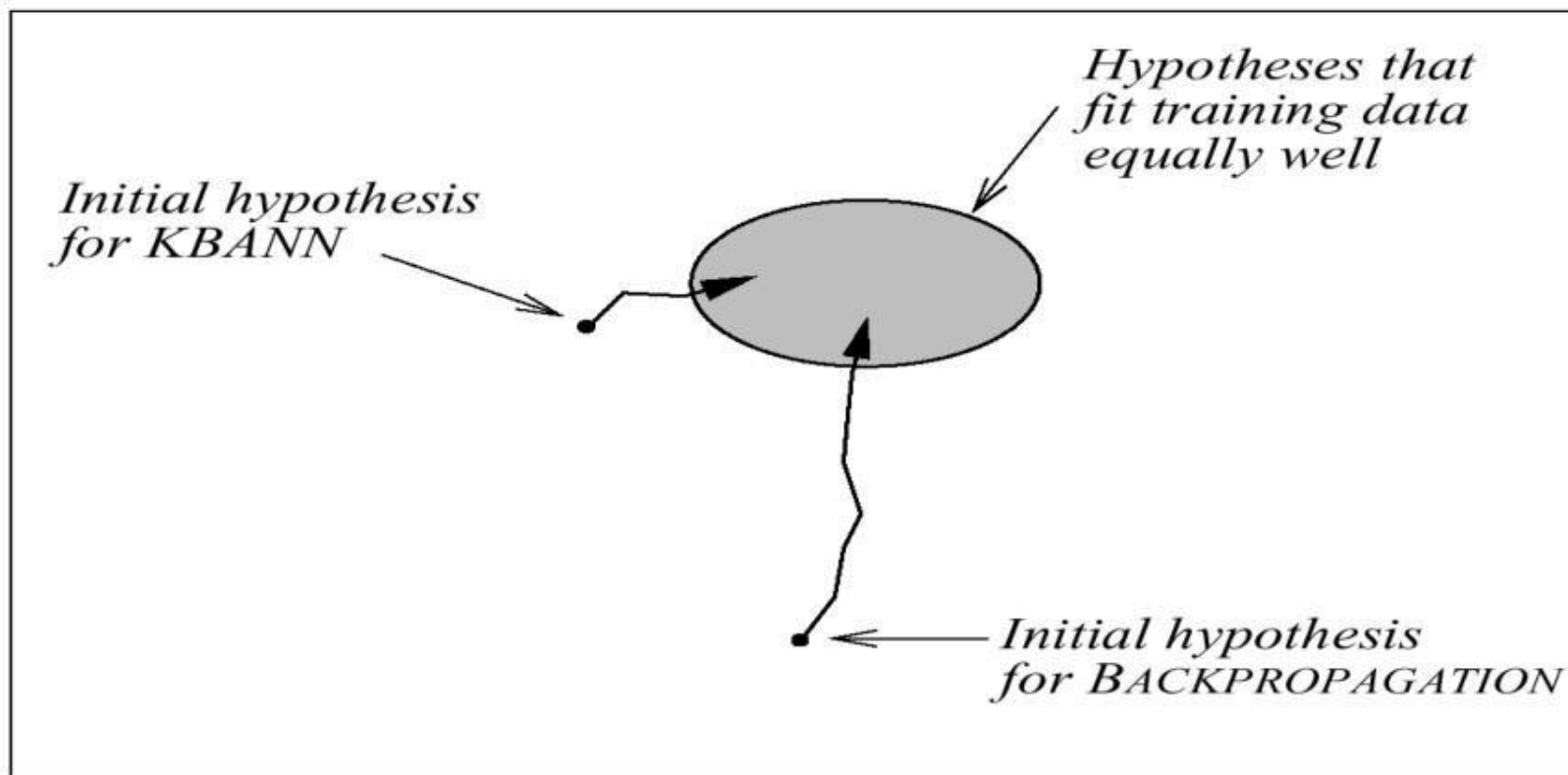
Classifying promoter regions in DNA leave one out testing:

- Backpropagation : error rate 8/106
- KBANN: 4/106

Similar improvements on other classification, control tasks.

Hypothesis space search in KBANN

Hypothesis Space





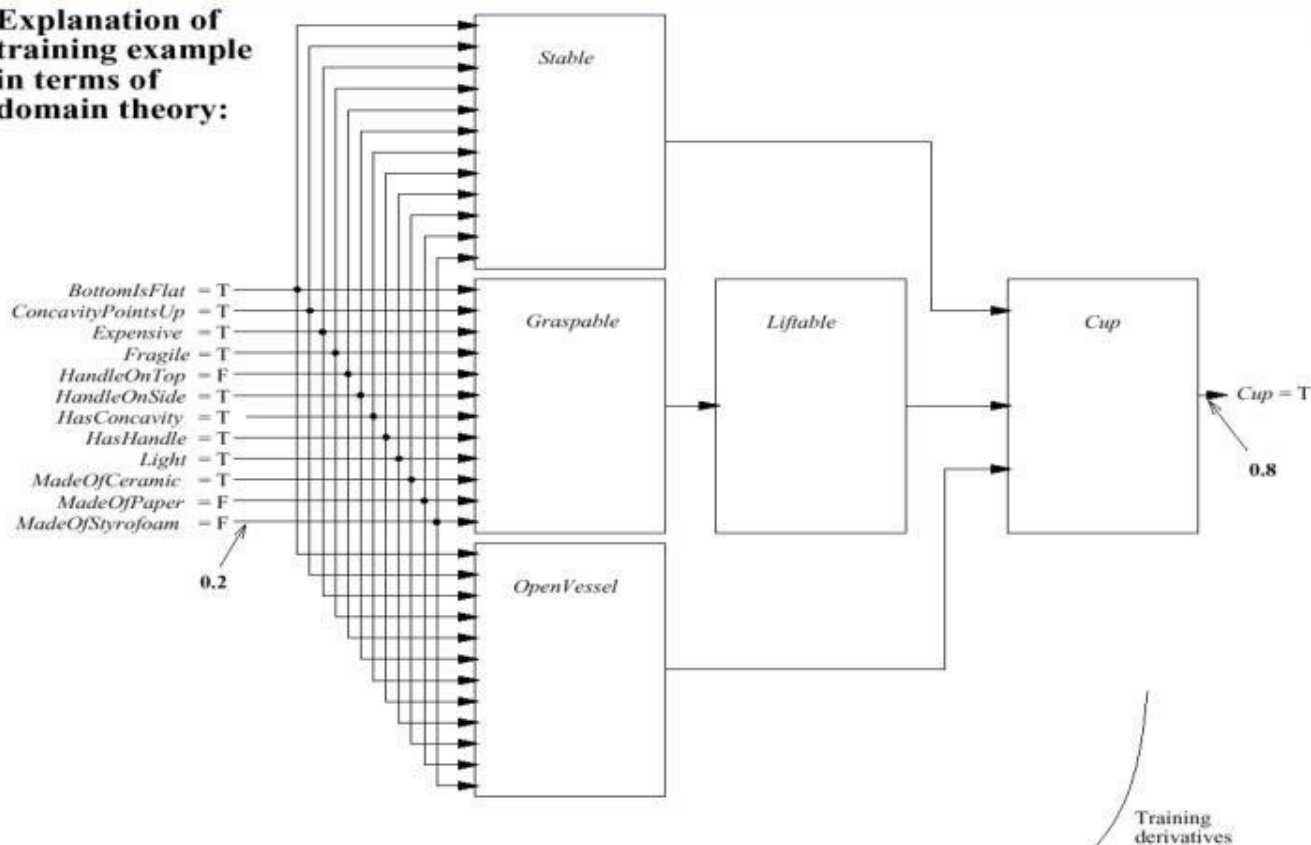
EBNN

Key idea:

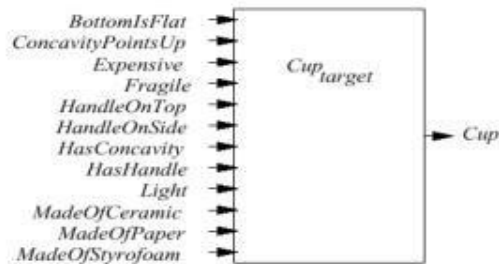
- Previously learned approximate domain theory
- Domain theory represented by collection of neural networks
- Learn target function as another neural network



**Explanation of
training example
in terms of
domain theory:**



Target network:



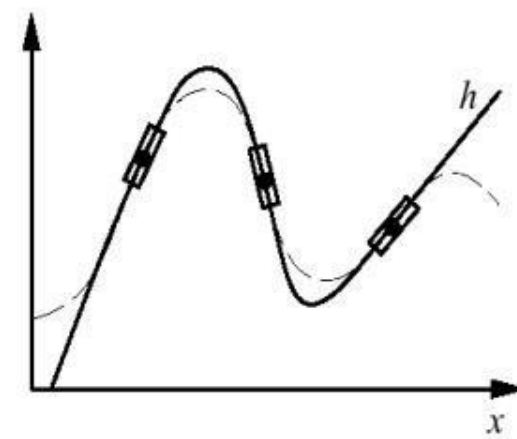
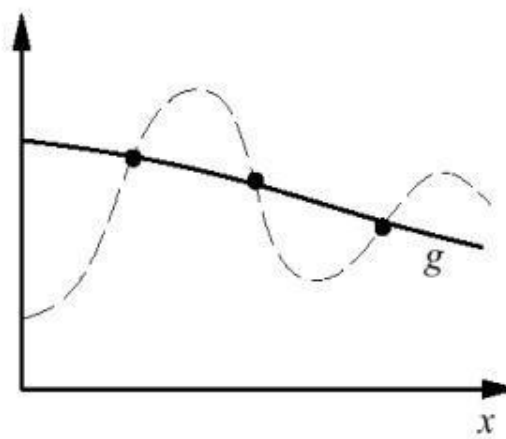
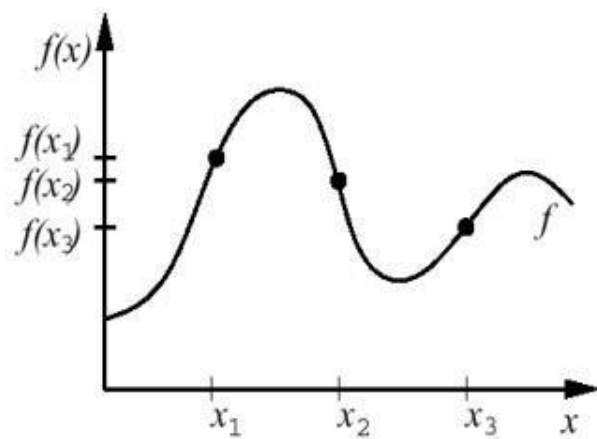
Modified Objective for Gradient Descent

$$E = \sum_i \left[(f(x_i) - \hat{f}(x_i))^2 + \mu_i \sum_j \left(\frac{\partial A(x)}{\partial x^j} - \frac{\partial \hat{f}(x)}{\partial x^j} \right)^2_{(x=x_i)} \right]$$

where

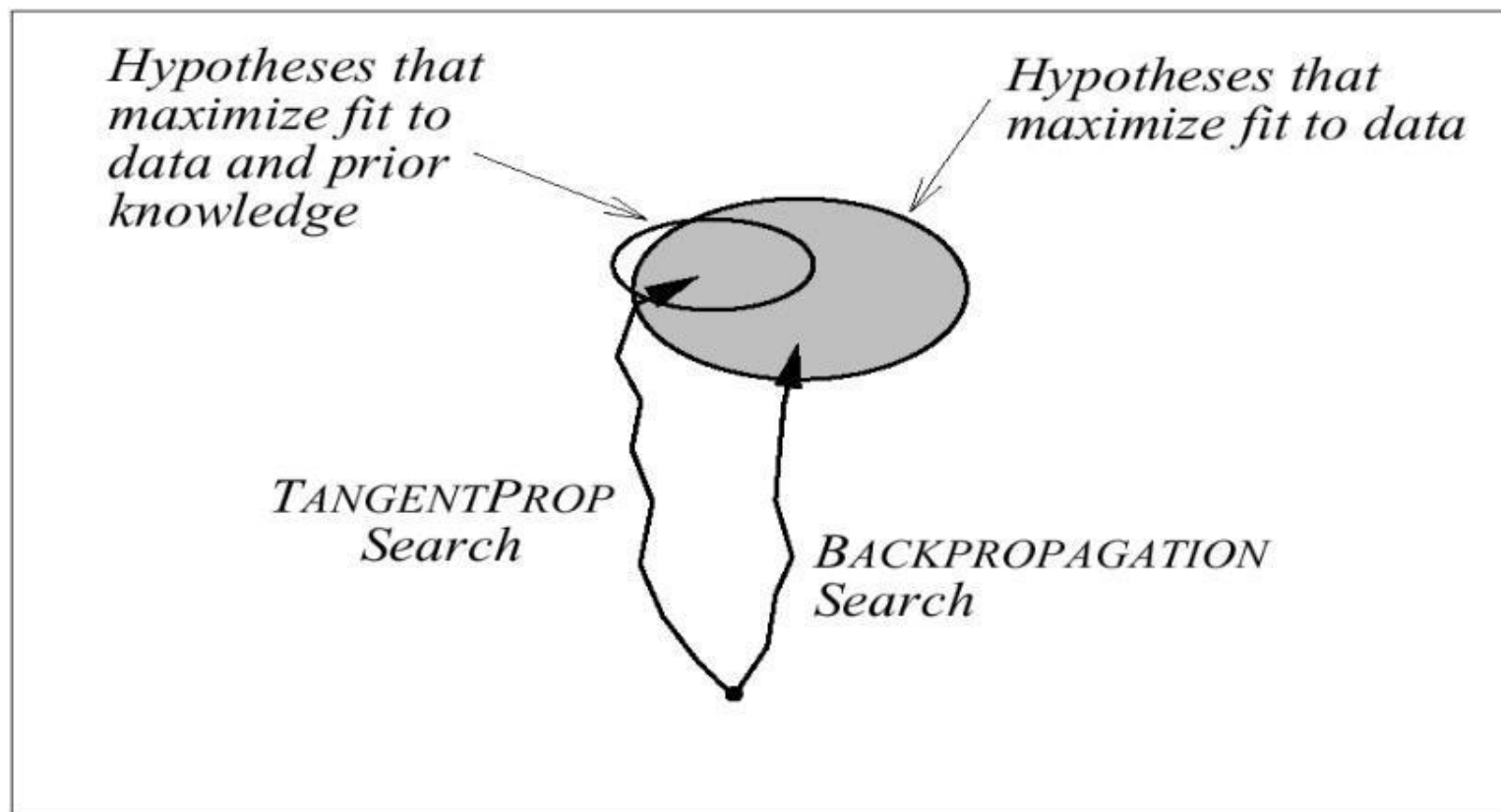
$$\mu_i \equiv 1 - \frac{|A(x_i) - f(x_i)|}{c}$$

- $f(x)$ is target function
- $\hat{f}(x)$ is neural net approximation to $f(x)$
- $A(x)$ is domain theory approximation to $f(x)$



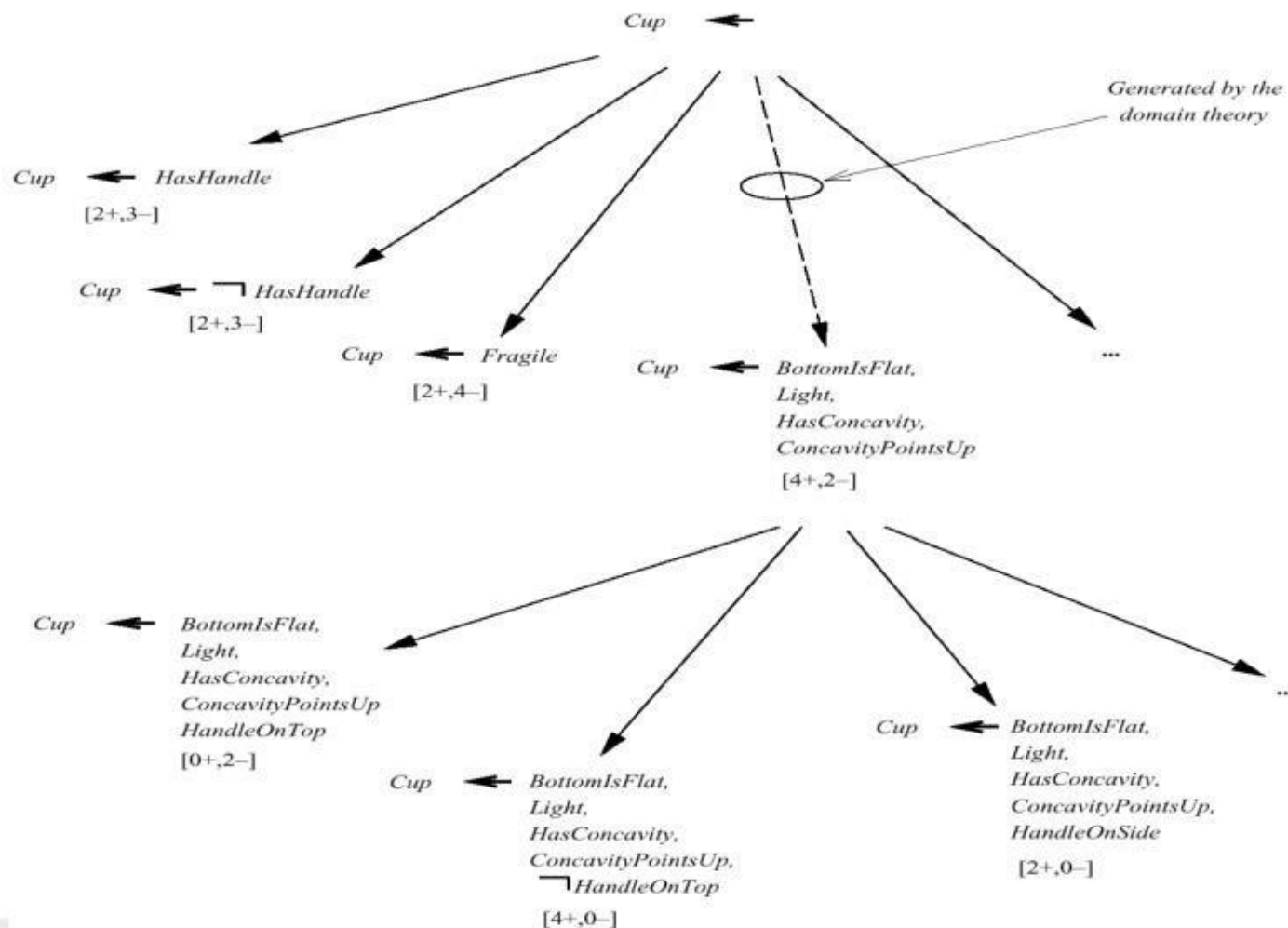
Hypothesis Space Search in EBNN

Hypothesis Space





Search in FOCL





FOCL Results

Recognizing legal chess endgame positions:

- 30 positive, 30 negative examples
- FOIL : 86%
- FOCL : 94% (using domain theory with 76% accuracy)

NYNEX telephone network diagnosis

- 500 training examples
- FOIL : 90%
- FOCL : 98% (using domain theory with 95% accuracy)