Unit 3

- **Introduction to Machine Learning**: What is Machine Learning, Key Terminology, Key tasks of machine learning, Well posed learning problems, Designing a Learning system, Perspective and Issues in Machine Learning.
- **Concept Learning**: Introduction, A Concept Learning Task, Concept Learning as Search, Find-S, Version Spaces and the Candidate-Elimination Algorithm.
- **Decision Tree** - Decision Tree Representation, Appropriate Problems for Decision Tree Learning, Basic Decision Tree Learning Algorithm, Issues in Decision Tree Learning.
- Chapter 1, 2 and 3 of Text Book 2
- Tom M Mitchell, "Machine Learning", McGraw-Hill Education (Indian Edition), 2013.
- Peter Harrington. "Machine learning in action", Shelter Island, NY: ManningPublications Co, 2012. (Chapter 1.1,1.2,1.3)

OUTLINE

- **Introduction**
  - Well posed learning problems
  - Designing a Learning system
  - Perspective and Issues in Machine Learning
- **Concept Learning**
  - Concept learning task
  - Concept learning as search
  - Find-S algorithm, Version space
  - Candidate Elimination algorithm

Introduction

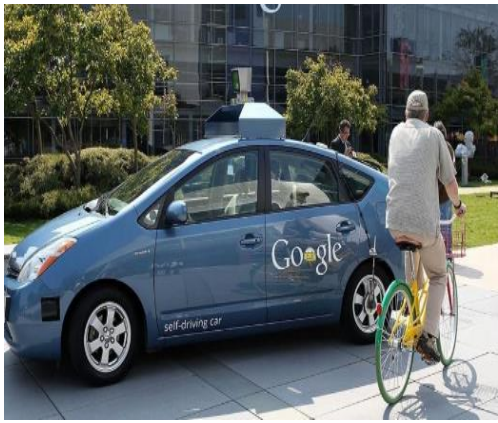
Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

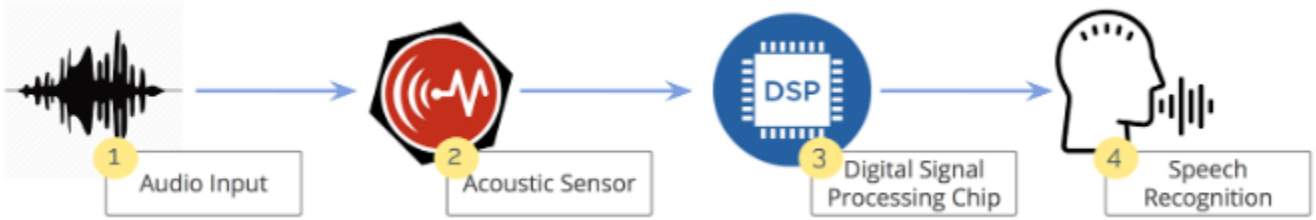
- Imagine computers learning from medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in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

Introduction

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





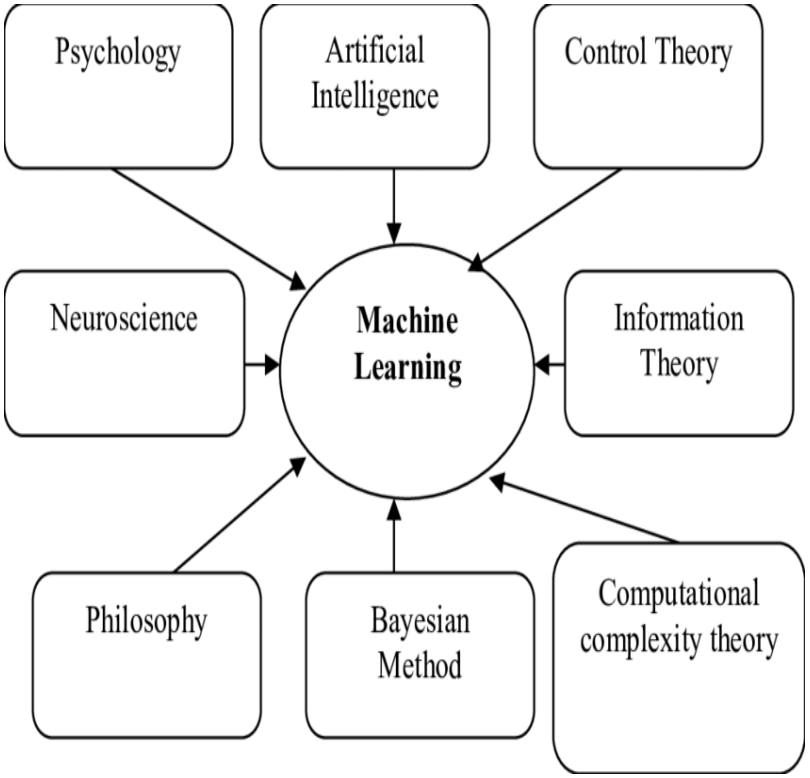
Why is Machine Learning Important?

- 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

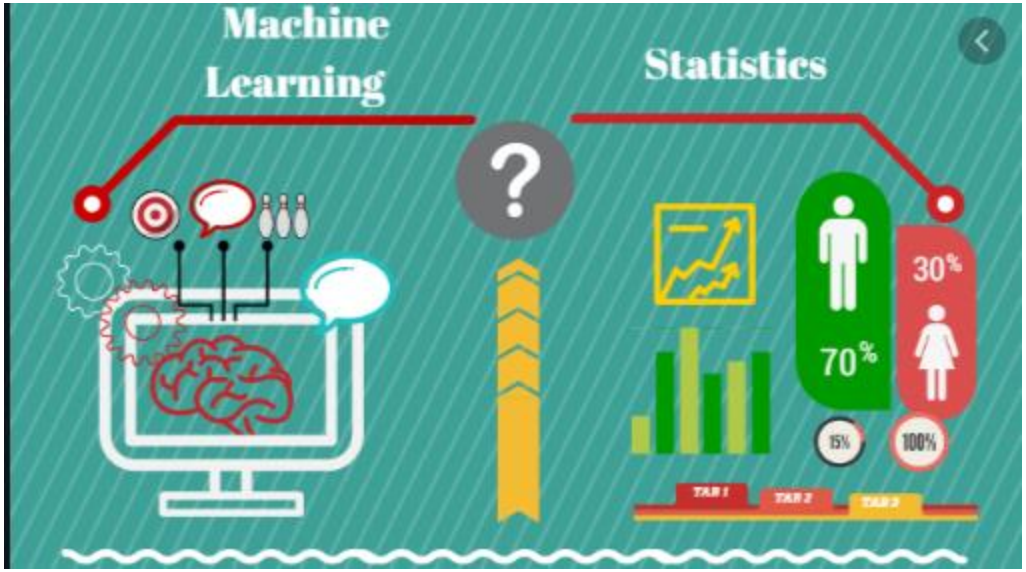
Machine learning research is often interdisciplinary. There are synergies in the following fields:

- Statistics
- Brain models
- Adaptive Control Theory
- Psychology
- Artificial Intelligence
- Evolutionary models
- Information theory
- Philosophy



Areas of Influence for Machine Learning

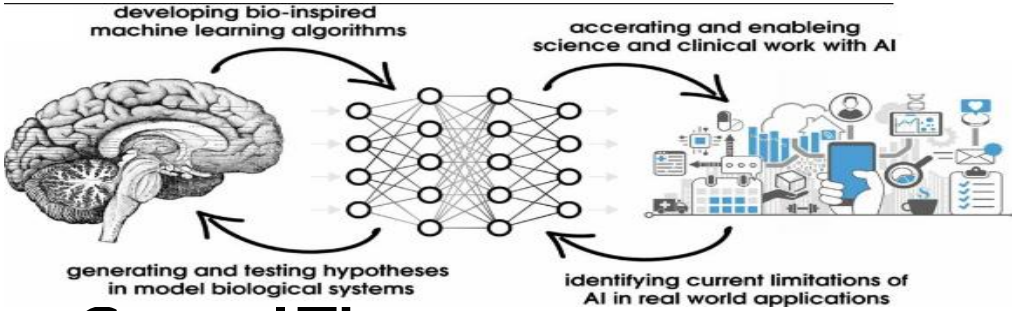
- **Statistics**
  How best to use samples drawn from unknown probability distributions to help decide from which distribution some new sample is drawn?



Areas of Influence for Machine Learning

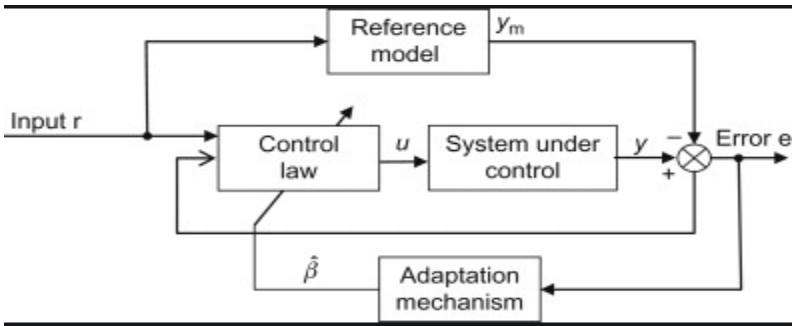
- **Brain Models**

  Non-linear elements with weighted inputs (ANN) 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?



Areas of Influence for Machine Learning

- **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?

- **EvolutionaryModels**

  How to model certain aspects of biological evolution to improve the performance of computer programs?

Machine Learning: 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.

P E T

Machine Learning - Examples

Example 1 **Learning to recognize faces**

- T: recognize faces
- P: % of correct recognitions
- E: opportunity to makes guesses and being told what the truth is

Example 2 **Learning to find clusters in data**

- T: finding clusters
- P: compactness of groups detected
- E: analyses of a growing set of data

Machine Learning – Other Examples

Learning to recognize spoken words.

SPHINX system (e.g., Lee 1989):the primitive sounds (phonemes) and words from the observed speech signal.

Neural network learning methods (e.g., Waibel et al. 1989) and methods for learning hidden Markov models

(e.g., Lee 1989) are effective for automatically customizing to,individual speakers, vocabularies, microphone characteristics, background noise, etc.

Machine Learning – Other Examples

Learning to drive an autonomous vehicle.

Machine learning methods have been used to train computer-controlled vehicles to steer correctly when driving on a variety of road types.

For example, the ALVINN system (Pomerleau 1989) has used its learned strategies to drive unassisted at 70 miles per hour for 90 miles on public highways among other cars. Similar techniques have possible applications in many sensor-based control problems.

Machine Learning – Other Examples

Learning to classify new astronomical structures. .

Machine learning methods have been applied to a variety of large databases to learn general regularities implicit in the data.

For example, decision tree learning algorithms have been used by NASA to learn how to classify celestial objects from the second Palomar Observatory Sky Survey (Fayyad et al. 1995).

This system is now used to automatically classify all objects in the Sky Survey, which consists of three terrabytes of image data.

Machine Learning – Other Examples

Learning to play world-class backgammon.

The most successful computer programs for playing games such as backgammon are based on machine learning algorithms.

For example, the world's top computer program for backgammon, TD-GAMMON (Tesauro 1992, 1995). learned its strategy by playing over one million practice games against itself. It now plays at a level competitive with the human world champion. Similar techniques have applications in many practical problems where very large search spaces must be examined efficiently.

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

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

P E T

What do you mean by a well –posed learning problem? Explain the important features that are required to well –define a learning problem with examples.

Well-Posed Learning Problem

Checkers 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.

Well-Posed Learning Problem

Checkers Game Basics

- 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.

Well-Posed Learning Problem

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.

Rules of the Game Cont.

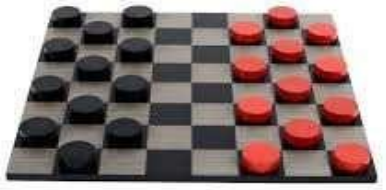
- When a piece is captured, it is removed from the board.
- If a player is able to make a capture, there is no option; the jump must be made.
- If more than one capture is available, the player is free to choose whichever he or she prefers.
- When a piece reaches the furthest row from the player who 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.

Rules of the Game Cont.

- 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-Posed Learning Problem

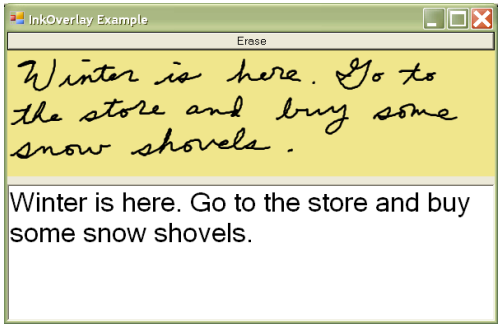


A checkers learning problem:

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

A 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



Well-Posed Learning Problem

A robot 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

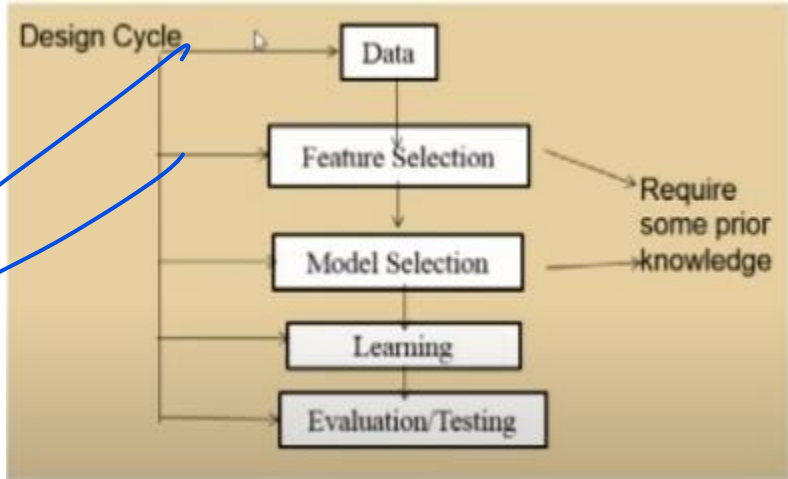
Designing a Learning System

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

Exemplify the various stages involved in designing a learning system.

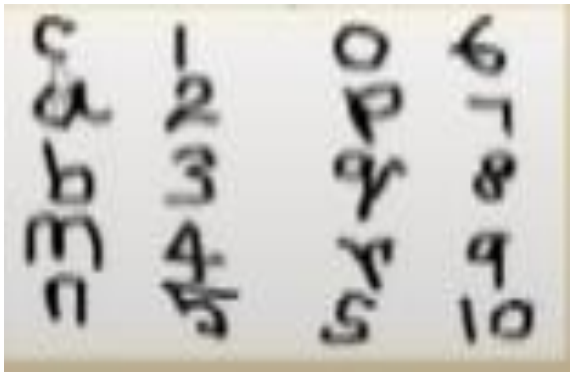
List and explain the steps involved in designing a learning system.

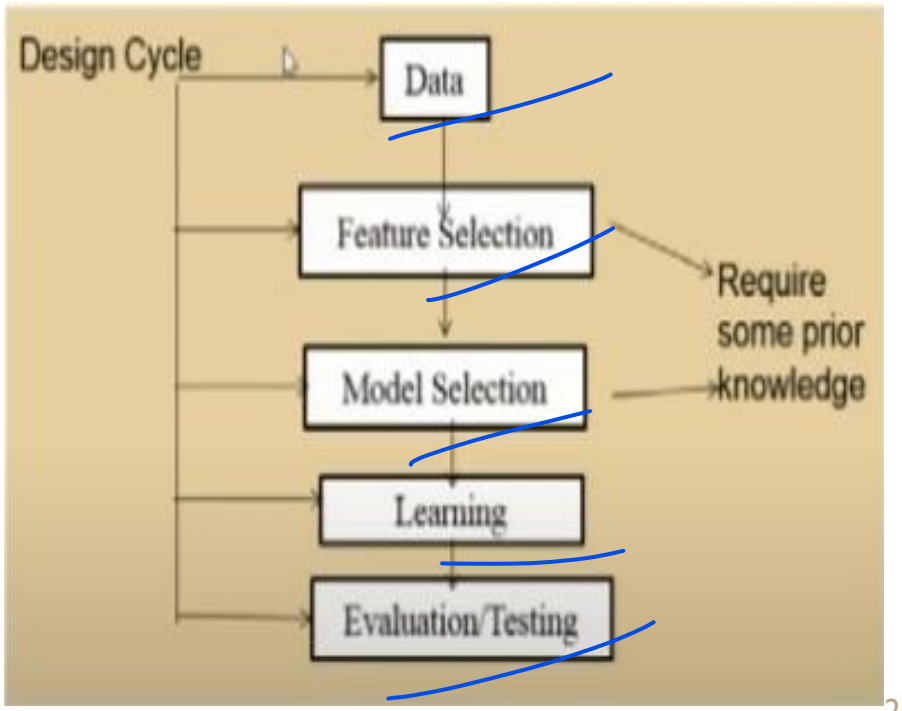
1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
   1. Estimating training values
   2. Adjusting the weights
5. The Final Design



Designing a Learning System

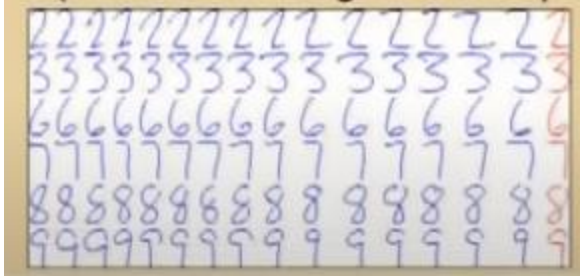
- Task T: recognize handwritten characters and numbers
- Performance measure P: percent of characters correctly cclassified
- Training experience E: a database of handwritten characters with given classifications





Step 1: Collect Training Examples (Experience).

- Without examples, our system will not learn (so-called learning from examples)



Step 2: Representing Experience

- Choose a representation scheme for the experience/examples

(1,1,0,1,1,1,1,1,1,1,0,0,0,0,1,1,1, 1,1,0, ..., 1) 64-d Vector

(1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1, 1,1,0, ..., 1) 64-d Vector

The sensor input represented by an n-d vector, called the **feature vector**, $\mathbf{X} = (x_1, x_2, x_3, \ldots, x_n)$

Step 2: Representing Experience

- So, what would D be like? There are many possibilities.
- Assuming our system is to recognise 10 digits only, then D can be a 10-d binary vector; each correspond to one of the digits

D = (d0, d1, d2, d3, d4, d5, d6, d7, d8, d9)

X = (1,1,0,1,1,1,1,1,1,1,0,0,0,0,1,1,1, 1,1,0, ..., 1);
64-d Vector D= (0,0,0,0,0,1,0,0,0,0)

X= (1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1, 1,1,0, ..., 1);
64-d Vector -D= (0,0,0,0,0,0,0,0,1,0)

Step 3: Choose a Representation for the Black Box

- We need to choose a function F to approximate the black box and for a given X, the value of F will give the classification of X.

Step 4: Learning/Adjusting the Weights

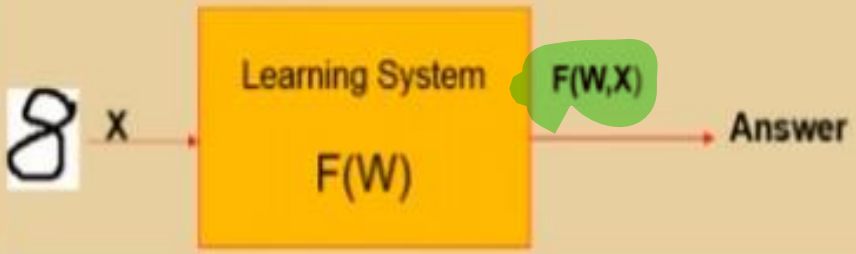
- We need a learning algorithm to adjust the weights such that the experience/prior knowledge from the training data can be learned into the system:

E=(X,D)

F(W,X) = L

Step 5: Use/Test the System

- After learning is completed, all parameters are fixed and an unknown input X can be presented to the system for which the system computes its answer according to F(W,X)



Designing a Learning System

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.

Designing a Learning System

1. Choosing the Training Experience

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.

Designing a Learning System

1. Choosing the Training Experience

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

For example, in checkers game:

Direct Feedback: In the case of direct training experience, an individual board states and correct move for each board state are given.

Indirect Feedback: In case of indirect training experience, the move sequences for a game and the final result (win, loss or draw) are given for a number of games.

Designing a Learning System

1. Choosing the Training Experience

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:

Teacher or Not — Supervised — The training experience will be labeled, which means, all the board states will be labeled with the correct move. So the learning takes place in the presence of a supervisor or a teacher.

Designing a Learning System

Unsupervised — The training experience will be unlabeled, which means, all the board states will not have the moves. So the learner generates random games and plays against itself with no supervision or teacher involvement.

Designing a Learning System

1. Choosing the Training Experience

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.

For example, in checkers game:

Is the training experience good — Do the training examples represent the distribution of examples over which the final system performance will be measured?

Performance is best when training examples and test examples are from the same/a similar distribution.

Designing a Learning System

2. Choosing the Target Function

In this design step, we need to determine exactly what type of knowledge has to be learned and it's used by the performance program.

- Here there are 2 considerations
  - direct and indirect experience.

Designing a Learning System

2. Choosing the Target Function

- Direct experience

- Let us call the function ChooseMove and use the notation

ChooseMove : B → M

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.

Designing a Learning System

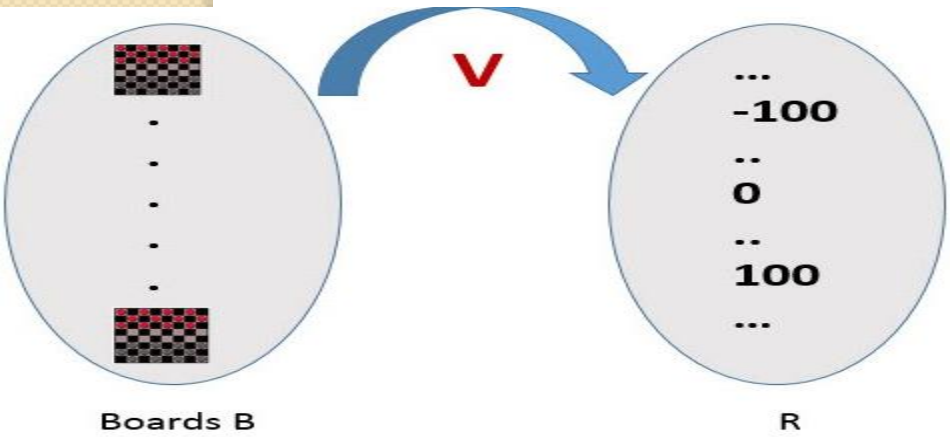
2. Choosing the Target Function

- Indirect experience
- **When there is an indirect experience**, it becomes difficult to learn such function.
- How about assigning a real score to the board state.
- So the function be **V : B →R** indicating that this accepts as input any board from the set of legal board states B and produces an output a real score.
- This function assigns the higher scores to better board states.

Designing a Learning System

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 a 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



Designing a Learning System

- It may be very difficult in general to learn such an operational form of V perfectly.

- We expect learning algorithms to acquire only some approximation to the target function ^V.

Designing a Learning System

3. Choosing a Representation for the Target Function

To choose a representation that the learning program will use to describe the function ^V that it will learn.

The representation of ^V can be as follows.

- A table specifying values for each possible board state?
- collection of rules?
- neural network?
- a polynomial function of board features?

Designing a Learning System

3. Choosing a Representation for the Target Function

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

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

Designing a Learning System

3. Choosing a Representation for the Target Function

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

$$\hat{V} = w0 + w1 \cdot x1(b) + w2 \cdot x2(b) + w3 \cdot x3(b) + w4 \cdot x4(b) + w5 \cdot x5(b) + w6 \cdot x6(b)$$

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

Designing a Learning System

The checkers learning task can be summarized as below.

- Task T : Play Checkers
- Performance Measure : % of games won in world tournament
- Training Experience E : opportunity to play against itself
- Target Function : V : Board → R
- Target Function Representation : $\hat{V} = w0 + w1 \cdot x1(b) + w2 \cdot x2(b) + w3 \cdot x3(b) + w4 \cdot x4(b) + w5 \cdot x5(b) + w6 \cdot x6(b)$

Designing a Learning System

4. Choosing a Function Approximation Algorithm

- To train our learning program, we need a set of training data, each describing a specific board state b and the training value V_train (b) for b.

- Each training example is an ordered pair <b,V_train(b)>

- For example, a training example may be <(x1 = 3, x2 = 0, x3 = 1, x4 = 0, x5 = 0, x6 = 0), +100">.

  Black has won the game since x2 = 0 or red has no remaining pieces.

  Here assigning a training value V_train(b) for the specific boards b that are clean win, loss or draw is direct as they are direct training experience.

Designing a Learning System

4. Choosing a Function Approximation Algorithm

- But in the case of indirect training experience, assigning a training value V_train(b) for the intermediate boards is difficult.
- In such case, the training values are updated using temporal difference learning.

Designing a Learning System

4. Choosing a Function Approximation Algorithm

- **Temporal difference (TD) learning is a concept central to reinforcement learning, in which learning happens through the iterative correction of your estimated returns towards a more accurate target return.**

Designing a Learning System

4. Choosing a Function Approximation Algorithm

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

Designing a Learning System

4. Choosing a Function Approximation Algorithm

1. Estimating training values

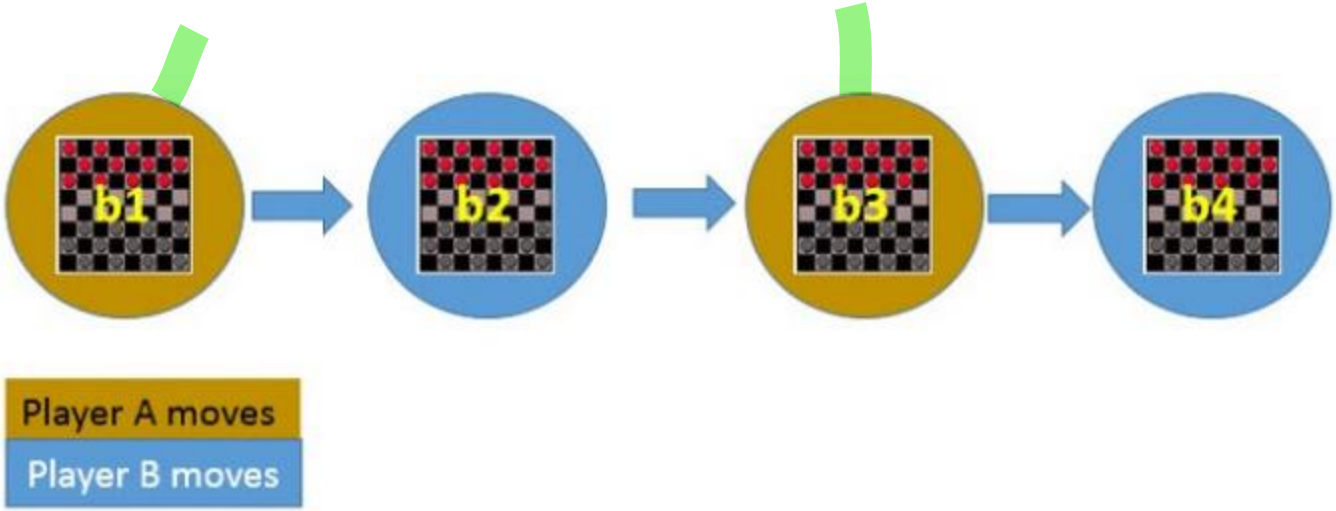
Let Successor(b) denotes the next board state following b for which it is again the program's turn to move.

^V is the learner's current approximation to V.

Using these information, assign the training value of V_train(b) for any intermediate board state b as below :

V_train(b) ← ^V(Successor(b))

Designing a Learning System



In the above figure, V_train(b1) ← ^V(b3), where b3 is the successor of b1. Once the game is played, the training data is generated. For each training example, the V_train(b) is computed.

Designing a Learning System

4. Choosing a Function Approximation Algorithm

2. Adjusting the weights

One common approach is to define the best hypothesis as that which minimizes the squared error E between the training values and the values predicted by the hypothesis ^V.

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

Designing a Learning System

4. Choosing a Function Approximation Algorithm

Least Mean Square (LMS) training rule is the one training algorithm that will adjust weights a small amount in the direction that reduces the error.

The LMS algorithm is defined as follows:

For each training example b

1. Compute $error(b) = V_{train}(b) - \hat{V}(b)$
2. for each board feature x_i, update weight w_i

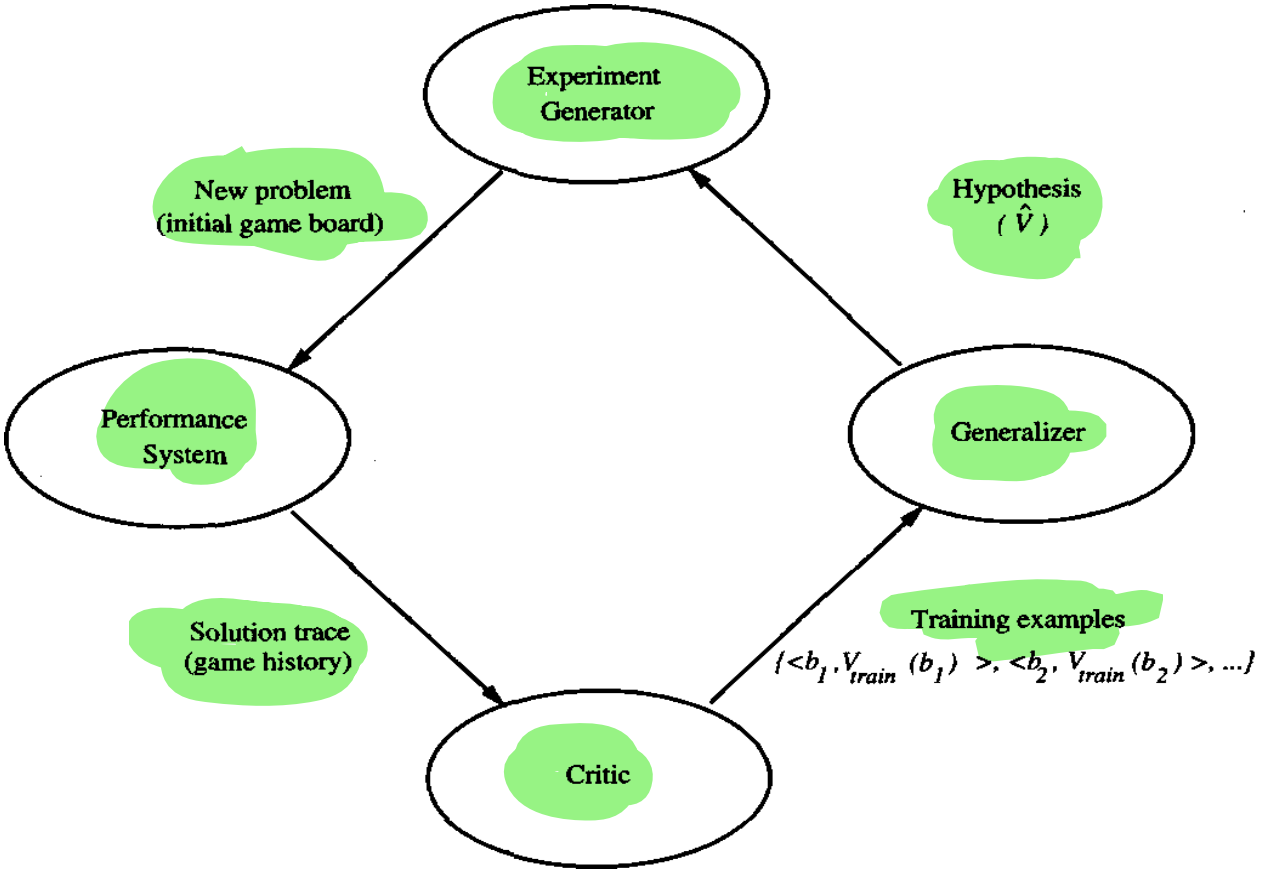
$$w_i \leftarrow w_i + \mu \, . error(b) \, . x_i$$

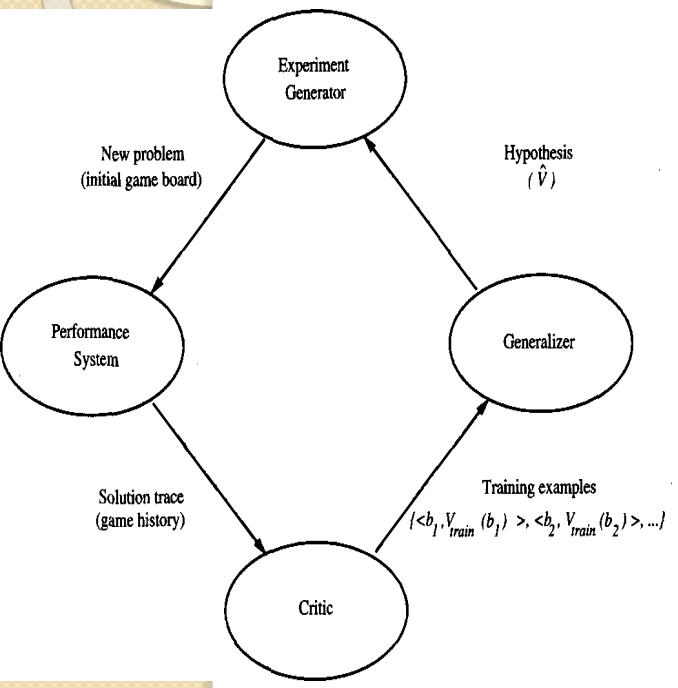
Here μ is a small constant (e.g., 0.1) that moderates the size of the weight update

Designing a Learning System

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



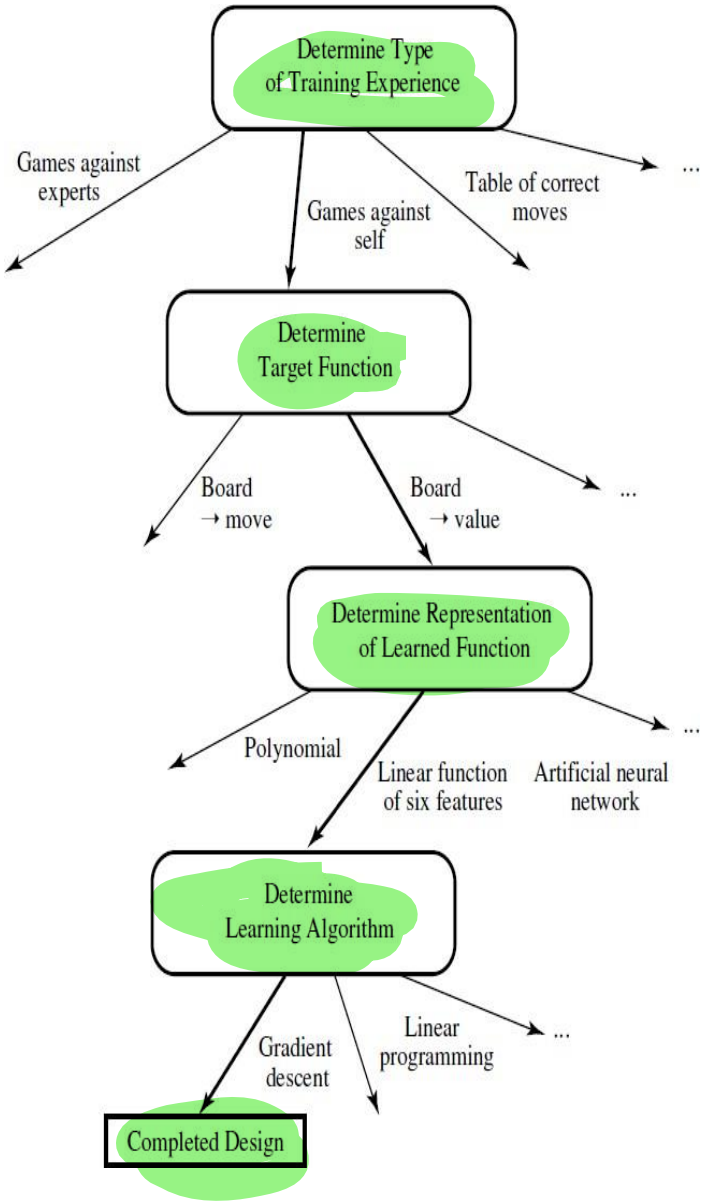


- The performance System — Takes a new board as input and outputs a trace of the game it played against itself.
- The Critic — Takes the trace of a game as an input and outputs a set of training examples of the target function.
- The Generalizer — Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.
- The Experiment Generator — Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.

Designing a Learning System

5. The Final Design

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



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 is the best way to reduce the learning task to one or more function approximation problems?

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?

Issues in Machine Learning

- 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?

OUTLINE

- **Introduction**
  - Well posed learning problems
  - Designing a Learning system
  - Perspective and Issues in Machine Learning
- **Concept Learning**
  - Concept learning task
  - Concept learning as search
  - Find-S algorithm,
  - Version space
  - Candidate Elimination algorithm
  - Inductive Bias

Text Book1- Sections: 1.1 – 1.3, 2.1-2.5, 2.7

Learning

What is Learning?

- The activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something.

Learning Methods

Learning Methods by Human

- Rote learning (memorization):
  - Memorizing things without knowing the concept/logic behind them.
- Passive learning (instructions):
  - Learning from a teacher/expert.
- Analogy (experience):
  - Learning new things from our past experience.
- Inductive learning (experience):
  - On the basis of past experience, formulating a generalized concept.
- Deductive learning:
  - Deriving new facts from past facts.

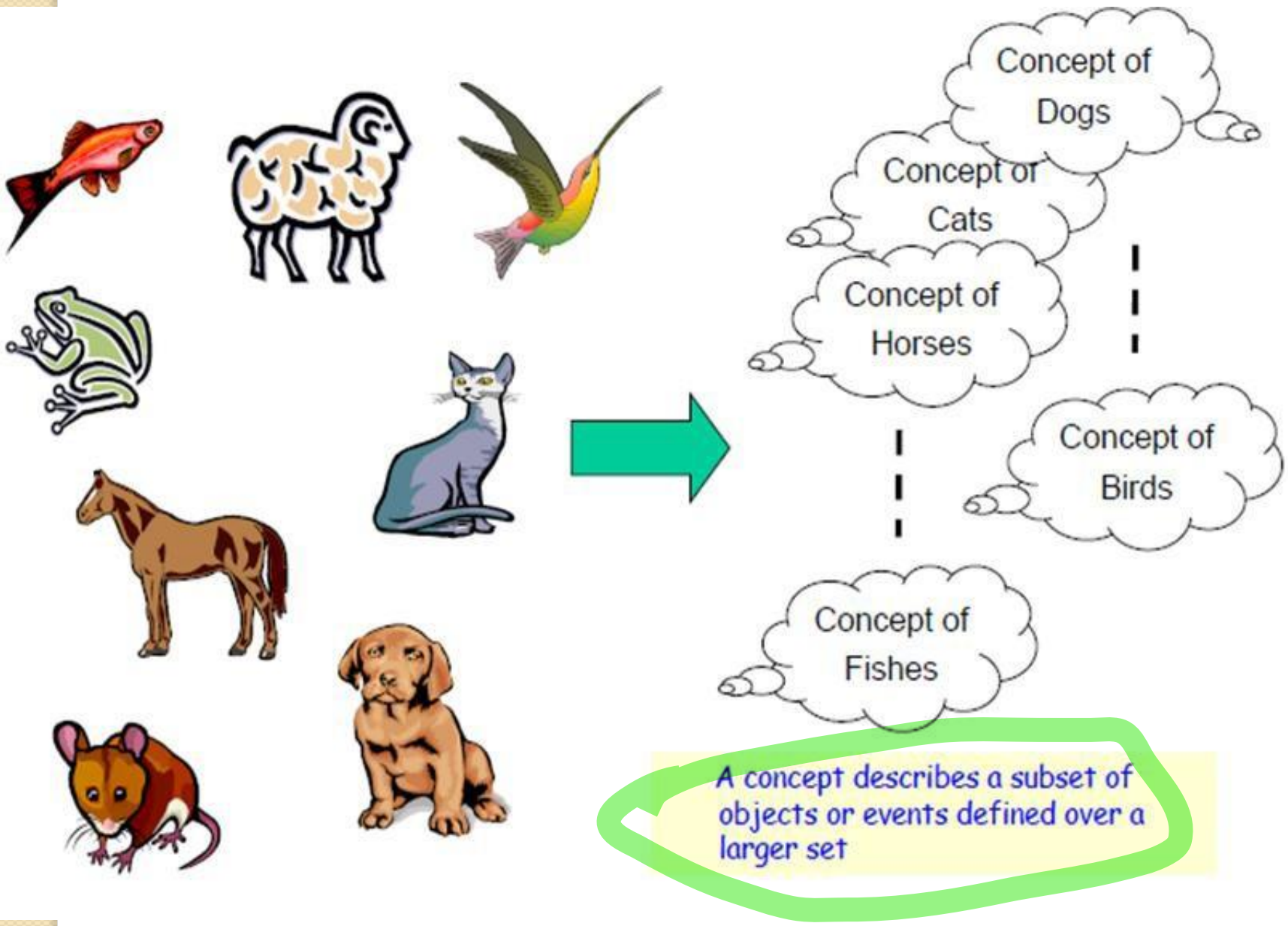
Concept Learning

Concept Learning

- Much of human learning involves acquiring general concepts from past experiences.
  - For example, humans identify different vehicles among all the vehicles based on specific sets of features defined over a large set of features.
  - This special set of features differentiates the subset of cars in a set of vehicles.
  - This set of features that differentiate cars can be called a concept.

- Inferring a boolean-valued function from training examples of its input and output.

What is a Concept? – Examples .....



Concept- To identify given image is bird or not

- Birds have wings and most of them can fly.
- They have light and hollow bones.
- They have feet and claws.
- They do not have teeth ,instead have beak which is used to tear, bite ,chisel ,crush or chew their food.
- Birds have streamlined bodies which help them to fly.

A Concept Learning Task – EnjoySport Training

Example	Sky	Temp	Humidity	Wind	Water	Forecast	EnjoySports
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

ATTRIBUTES

CONCEPT

Concept – John can play Golf

Day	Outlook	Temperature	Humidity	Wind	Play Golf
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cold	Normal	Weak	Yes
6	Rain	Cold	Normal	Strong	No
7	Overcast	Cold	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cold	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Concept Learning

- A Concept defines an object

- In machine learning, given a concept, we want to acquire an ability to take an object, and say it belongs to a concept or not

Data Object	Concept
x	c

x- Belongs to a concept → 1 / +1 / True
- Does not belong to a concept → 0 / -1 / False

A concept label a data object into one of two types(Binary value)

Concept Learning

- The task of machine learning is to learn the concepts
- Given a data object, Apply the concept and be able to categorize the object(Belong or does not belong)
- How do we represent data

x – can be written with a bunch of features

Feature of object x = shape → square, rectangle
size → large, small
color → black, green
surface type→ smooth , irregular

Concept Learning

Features of object x = shape → square, rectangle
size → large, small
color → black, green
surface type → smooth , irregular

$2^4 = 16$ (4 features, so 16 object can exist)

X = { , , , , , „ , ….} 16 object in this set

So if d – is the binary feature , then Possible Concepts = 2^{2^d}

So if d =4 , then Possible Concepts = 2^{16}

Concept Learning

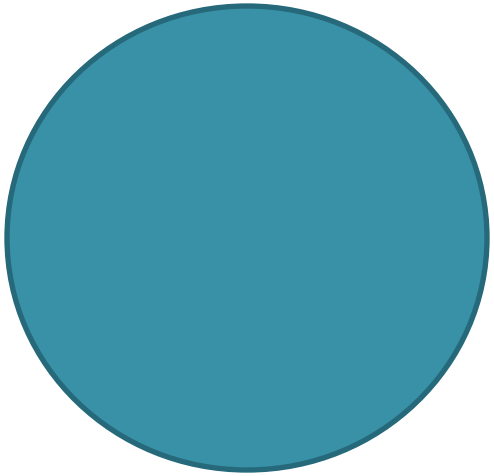
Features of object x = shape → square, rectangle
size → large, small
color → black, green
surface type → smooth , irregular

So if d =4 , then Possible Concepts = 2^{16}

Machine learning algorithm should look into the large Concept space and learn all data

And Assume Concept space is reduced

Inductive bias



Concept Learning

Concept Learning

- A Formal Definition for Concept Learning:
  - Inferring a boolean-valued function from training examples of its input and output.
- An example for concept-learning is the learning of bird-concept from the given examples of birds (positive examples) and non-birds (negative examples).

Concept Learning

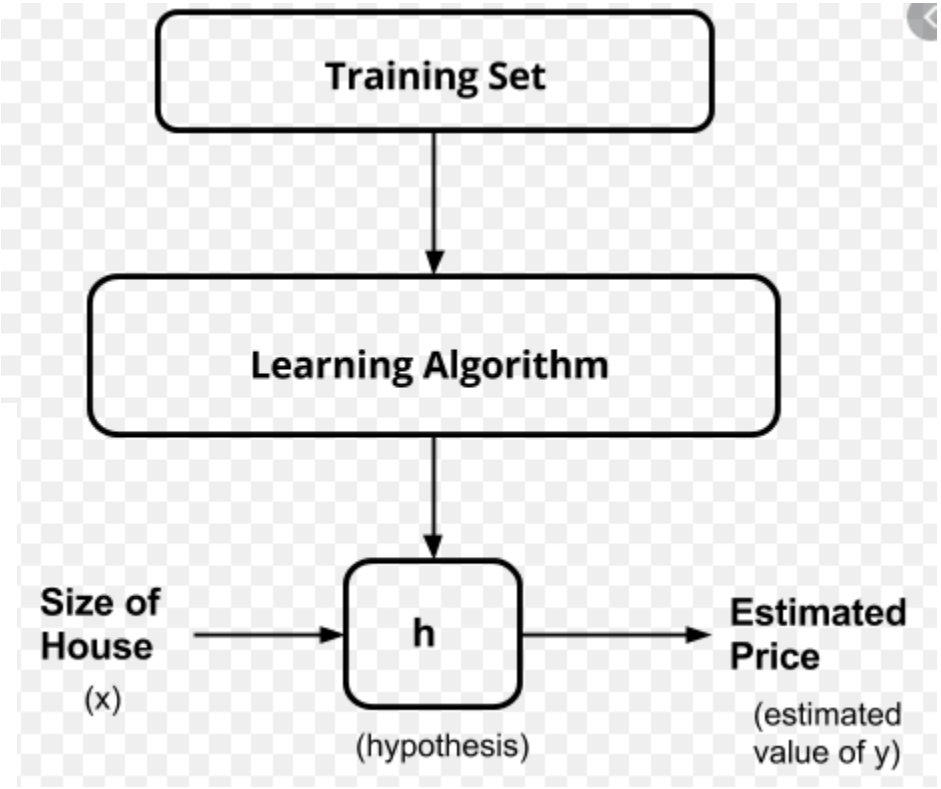
Concept Learning

- Concept Learning is acquiring the definition of a general category from given sample positive and negative training examples of the category.
- Concept Learning can seen as **a problem of searching through a predefined space of potential hypothesis for the hypothesis that best fits the training examples**.

What is hypothesis

- A **machine learning hypothesis** is a candidate model that approximates a target function for mapping inputs to outputs
- A common notation
  - **h (*hypothesis*)**: A specific hypothesis, e.g. an instance or specific candidate model that maps inputs to outputs and can be evaluated and used to make predictions.
  - **H (*hypothesis set/space*)**: A space of possible hypotheses for mapping inputs to outputs that can be searched.

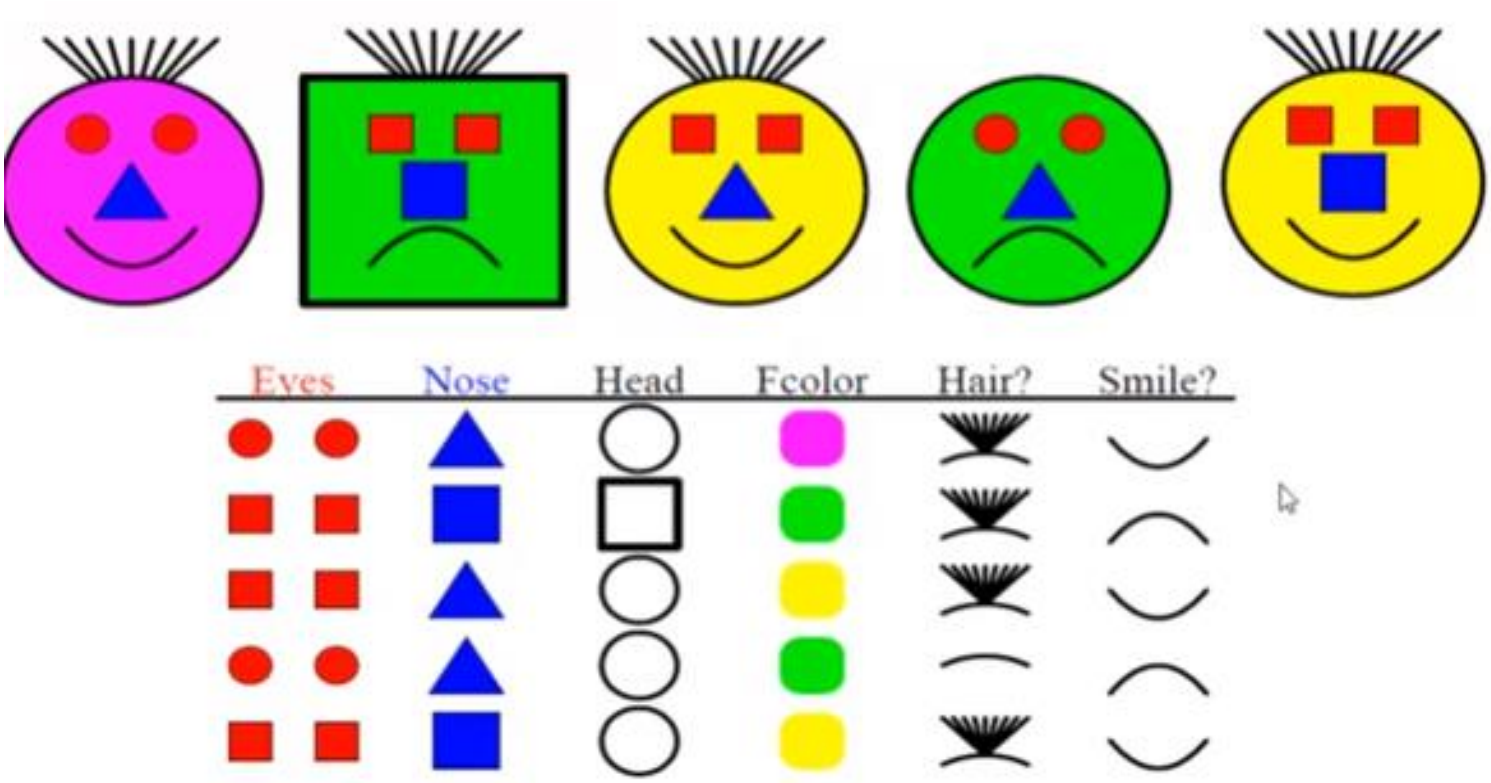
	home	nbhd	offers	sqft	brick	bedrooms	bathrooms	price
0	1	nbhd02	2	1790	No	2	2	114300
1	2	nbhd02	3	2030	No	4	2	114200
2	3	nbhd02	1	1740	No	3	2	114800
3	4	nbhd02	3	1980	No	3	2	94700
4	5	nbhd02	3	2130	No	3	3	119800
5	6	nbhd01	2	1780	No	3	2	114600
6	7	nbhd03	3	1830	Yes	3	3	151600
7	8	nbhd03	2	2160	No	4	2	150700
8	9	nbhd02	3	2110	No	4	2	119200
9	10	nbhd02	3	1730	No	3	3	104000



Terminology (Concept Learning)

- The set of items/objects over which the concept is defined is called the set of instances and denoted by X.
- The concept or function to be learned is called the target concept and denoted by c.
  - It can be seen as a boolean valued function defined over X and can be represented as **c: X -> {0, 1}**.
- H is used to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept.
- The goal of a learner is to find a hypothesis H that can identify all the objects in X so that **h(x) = c(x) for all x in X**.

Concept Example for Smiley Faces



Concept Features – Computer view

Eyes	Nose	Head	Fcolor	Hair?	Smile?
Round	Triangle	Round	Purple	Yes	Yes
Square	Square	Square	Green	Yes	No
Square	Triangle	Round	Yellow	Yes	Yes
Round	Triangle	Round	Green	No	No
Square	Square	Round	Yellow	Yes	Yes

Find S Algorithm

- Concept Learning - Is a learning task, in which we train our machine to learn some concept by giving some predefined example.
- Two categories
  - General Hypothesis
  - Specific Hypothesis

  - Example: Get something to eat (Friend)

Representation of Hypotheses

- General Hypothesis representation

  G = <?, ?, ?, ?, ?,..... ?>

Specific Hypothesis representation

S = <ø , ø , ø , ø , ø ,..... ø >

NUMBER OF ATTRIBUTES

1. specify a single required value for the attribute→ **(e.g, Eyes = round)**
2. any value is acceptable for the attribute → **(e.g, Eyes = ?)**
3. no value is acceptable→ **(e.g, Eyes = ø)**

Features – Computer view

Eyes	Nose	Head	Fcolor	Hair?	Smile?
Round	Triangle	Round	Purple	Yes	Yes
Square	Square	Square	Green	Yes	No
Square	Triangle	Round	Yellow	Yes	Yes
Round	Triangle	Round	Green	No	No
Square	Square	Round	Yellow	Yes	Yes

For example,

Eyes	Nose	Head	Fcolor	Hair?
<Round,	?,	Round,	?,	No>

A Concept Learning Task – Hypothesis Representation.....

- ✓ **Most General Hypothesis: Everyday** is a good day for water sports **<?, ?, ?, ?, ?, ?>** (Positive example)
- ✓ **Most Specific Hypothesis: No day** is a good day for water sports **<0, 0, 0, 0, 0, 0>** (No day is Positive example)
- ✓ **EnjoySport concept learning task requires learning the sets of days for which EnjoySport = yes, describing this set by a conjunction of constraints over the instance attributes.**

Find S Algorithm

- Find Most Specific Hypothesis
- Consider only Positive(+/ Yes) examples

Find S Algorithm

Concept : Day on which person Enjoy

Example	Sky	Temp	Humidity	Wind	Water	Forecast	Enjoy Sports
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

1. Initialize h to meet specific hypothesis

   h= { ‘ø’ , ‘ø’ , ‘ø’ , ‘ø’ , ‘ø’ , ‘ø’ }

2. For each positive example:

   For each attribute in the example:

   if attribute value = hypothesis value

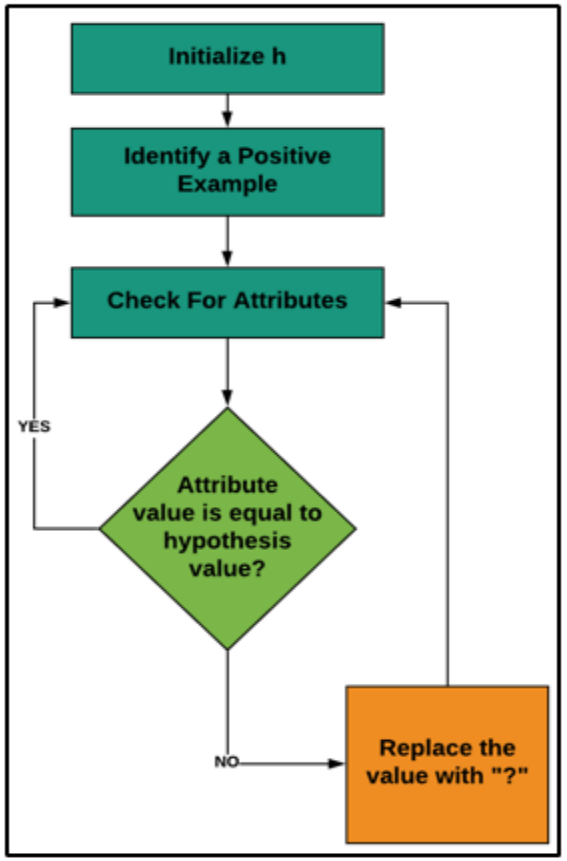
   do nothing

   else

   replace hypothesis value

   with more general constraint ‘?’

3. Output hypothesis h



Example	Sky	Temp	Humidity	Wind	Water	Forecast	EnjoySports
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

Concept Learning Task

Given:

- Instances X: Faces, each described by the attributes *Eyes*, *Nose*, *Head*, *Fcolor*, and *Hair?*
- Target function *c: Smile? : X* -> { no, yes }
- Hypotheses H: Conjunctions of literals such as
  <?,Square,Square,Yellow,?>
- Training examples D: Positive and negative examples of the target function

$$< x_1, c(x_1) >, < x_2, c(x_2) >, \ldots, < x_m, c(x_m) >$$

Determine: a hypothesis h in H such that $h(x)=c(x)$ for all x in D.

EnjoySport Concept Learning Task

Given:

- ✓ **Instances X**: Set of all **Possible days,** each described by the attributes
  - **Sky (Sunny, Cloudy, and Rainy)**
  - **Temp (Warm and Cold)**
  - **Humidity (Normal and High)**
  - **Wind (Strong and Weak)**
  - **Water (Warm and Cool)**
  - **Forecast (Same and Change)**

- ✓ **Target Concept (Function) c : EnjoySport : X → {0,1}**
- ✓ **Hypotheses H : Each hypothesis is described by a conjunction of constraints on the attributes.**
- ✓ **Training Examples D : positive and negative examples of the target function** along with their **target concept value c(x).**
- ✓ **Determine : A hypothesis h in H such that h(x) = c(x) for all x in D.**

Find –S Algorithm

1. Initialize h to meet specific hypothesis

   h= { ‘ø’ , ‘ø’ , ‘ø’ , ‘ø’ , ‘ø’ , ‘ø’ }

2. For each positive example:

   For each attribute in the example:

   if attribute value = hypothesis value

   do nothing

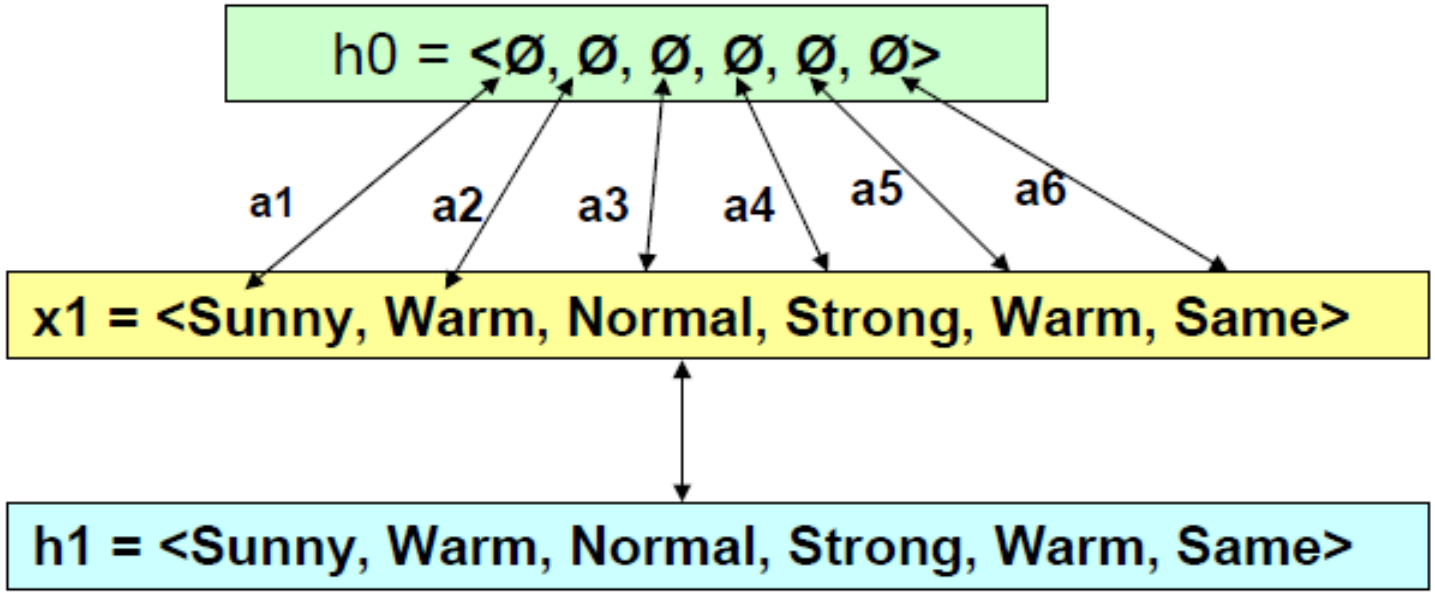
   else

   replace hypothesis value with more general constraint ‘?’

3. Output hypothesis h

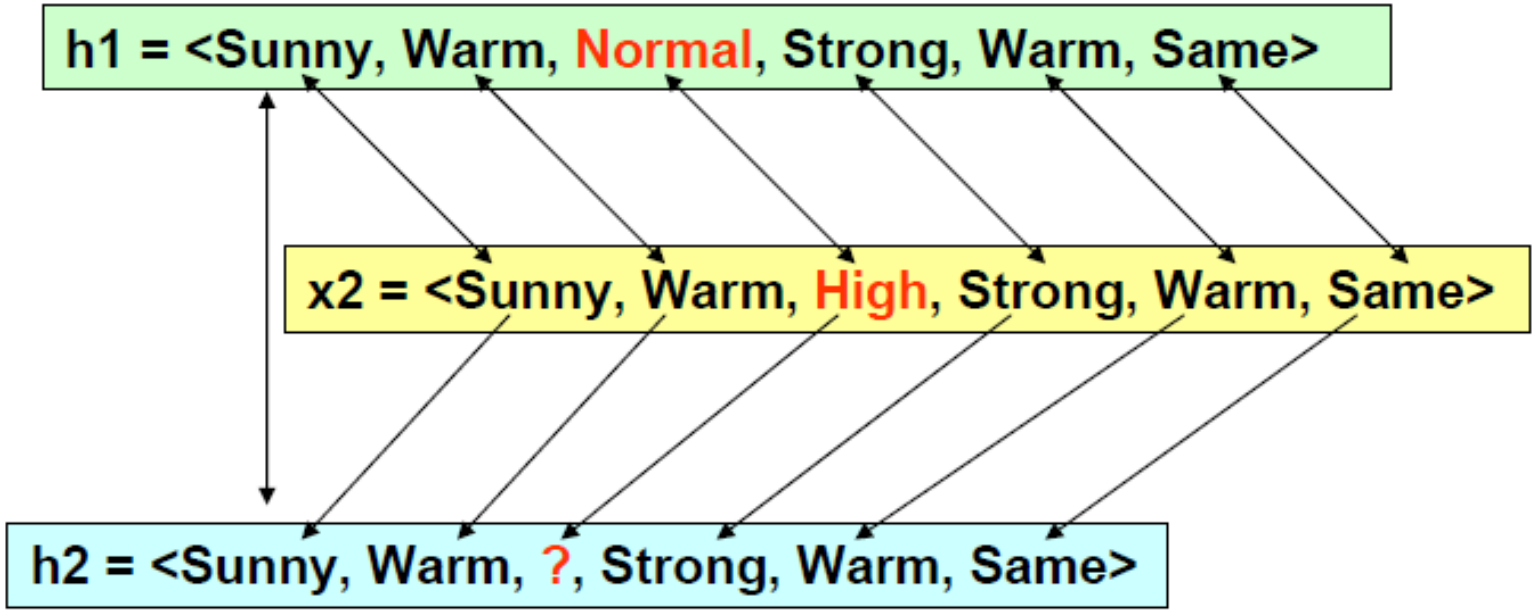
Example	Sky	Temp	Humidity	Wind	Water	Forecast	EnjoySports
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Iteration 1



Example	Sky	Temp	Humidity	Wind	Water	Forecast	EnjoySports
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

Iteration 2



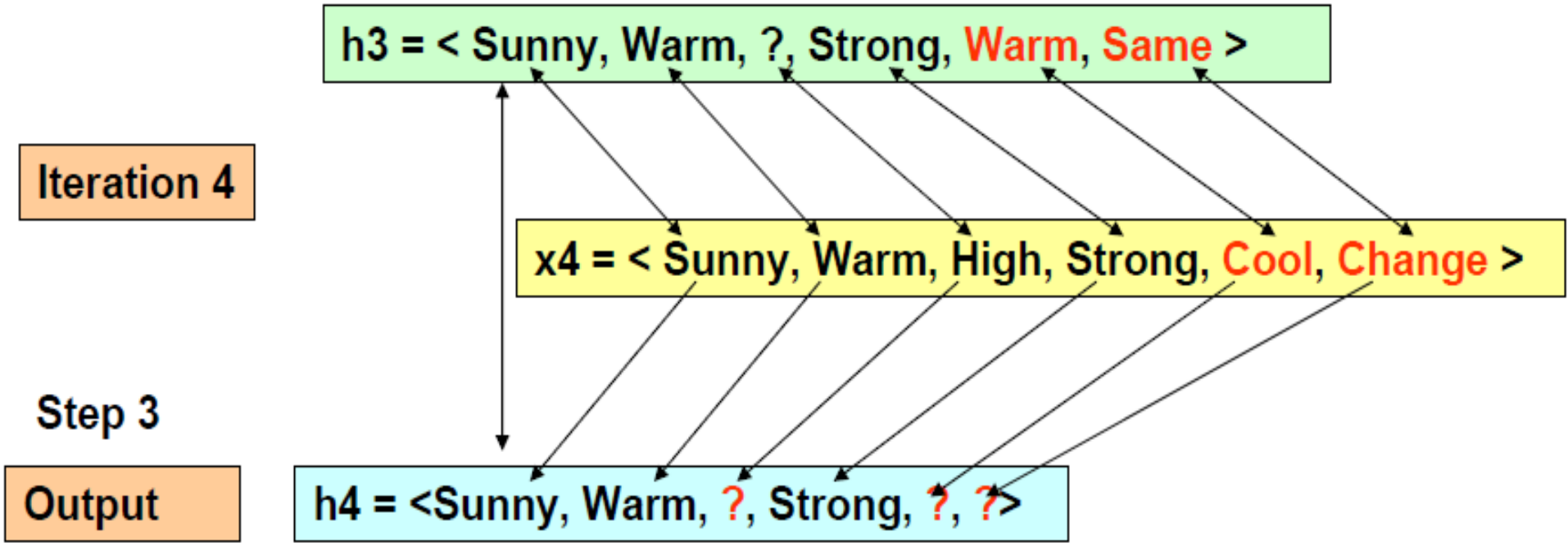
Example	Sky	Temp	Humidity	Wind	Water	Forecast	EnjoySports
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Iteration 3

Ignore

h3 = <Sunny, Warm, ?, Strong, Warm, Same>

Example	Sky	Temp	Humidity	Wind	Water	Forecast	EnjoySports
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



Example :

Consider the following data set having the data about which particular seeds are poisonous.

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

ATTRIBUTES ON WHICH THE CONCEPT DEPENDS ON

CONCEPT

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

STEP I

$h_0 = <$ ‘ø’ , ‘ø’ , ‘ø’ , ‘ø’ $>$

$h_1 = <$ ‘green’ , ‘hard’ , ‘no’ , ‘wrinkled’ $>$

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

STEP 2

h_1= < ‘green’ , ‘hard’ , ‘no’ , ‘wrinkled’ >

IGNORE

h_2= < ‘green’ , ‘hard’ , ‘no’ , ‘wrinkled’ >

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

STEP 3

h_2= < ‘green’ , ‘hard’ , ‘no’ , ‘wrinkled’ >

IGNORE

h_3= <‘green’ , ‘hard’ , ‘no’ , ‘wrinkled’ >

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

STEP 4

h_3= < ‘green’ , ‘hard’ , ‘no’ , ‘wrinkled’ >

x4= { ‘orange’ , ‘hard’ , ‘no’ , ‘wrinkled’ }

h_4= < ? , ‘hard’ , ‘no’ , ‘wrinkled’ >

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

STEP 5

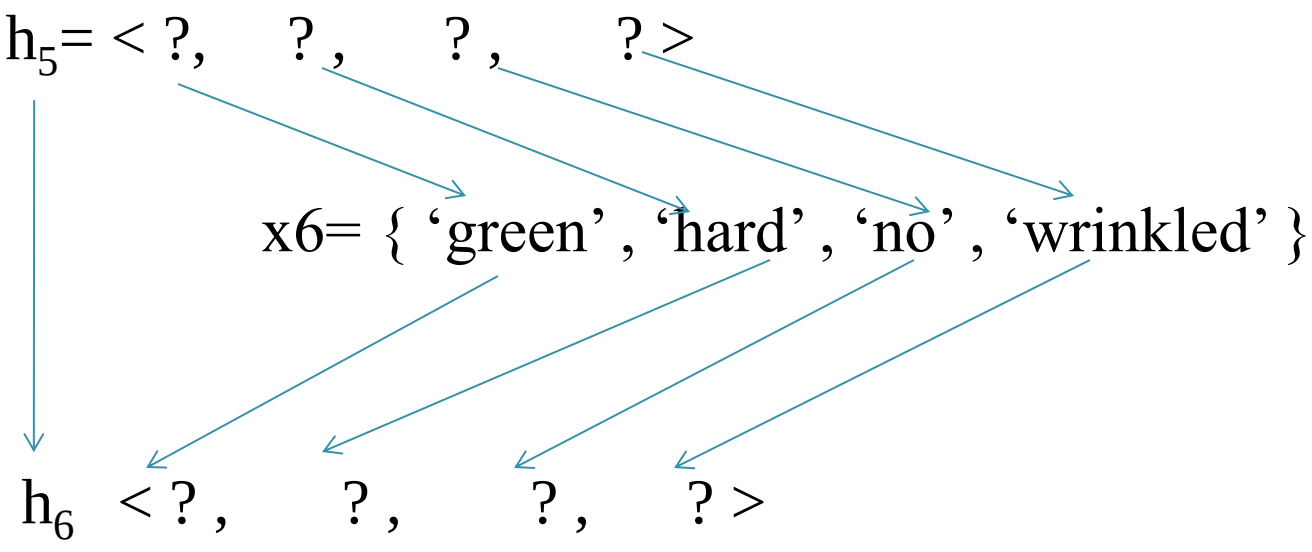
h_4= < ?, ‘hard’ , ‘no’ , ‘wrinkled’ >

x5= { ‘green’ , ‘soft’ , ‘yes’ , ‘smooth’ }

h_5 < ? , ? , ? , ? >

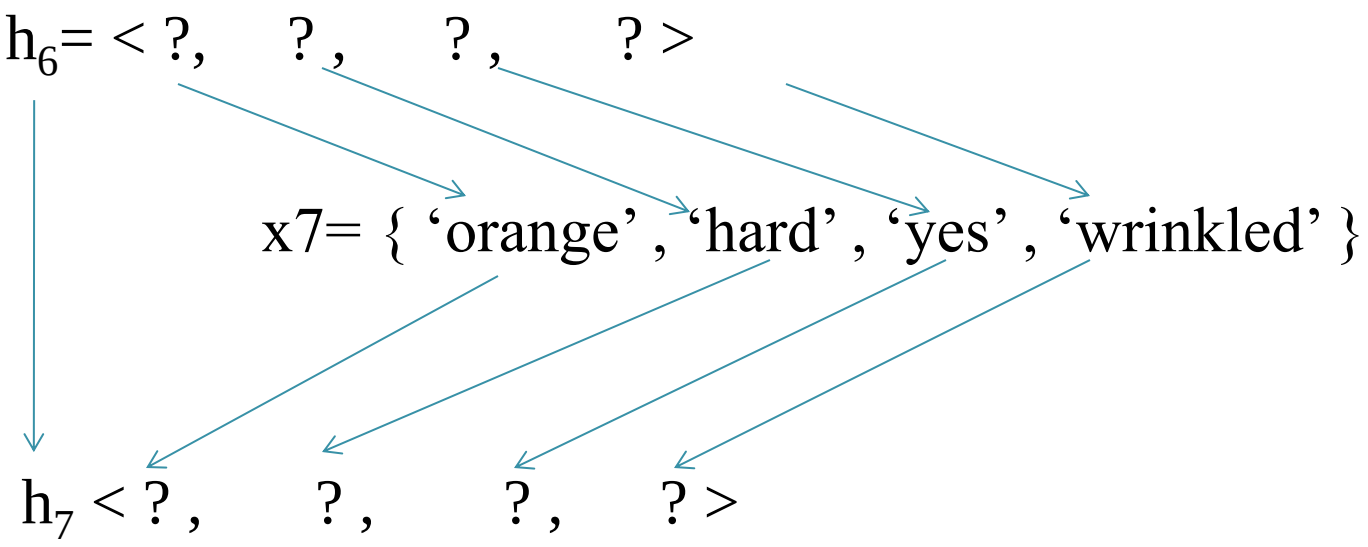
EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

STEP 6



EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

STEP 7



Hence, for the given data the final hypothesis would be :

Final Hyposthesis: h = { ?, ?, ?, ? }

Data set with a bunch of examples to decide if a person wants to go for a walk.

The concept of this particular problem will be on what days does a person likes to go on walk.

Time	Weather	Temperature	Company	Humidity	Wind	Goes
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

Data set with a bunch of examples to decide if a person wants to go for a walk.

The concept of this particular problem will be on what days does a person likes to go on walk.

Time	Weather	Temperature	Company	Humidity	Wind	Goes
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

```
The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

Example – Car Dataset

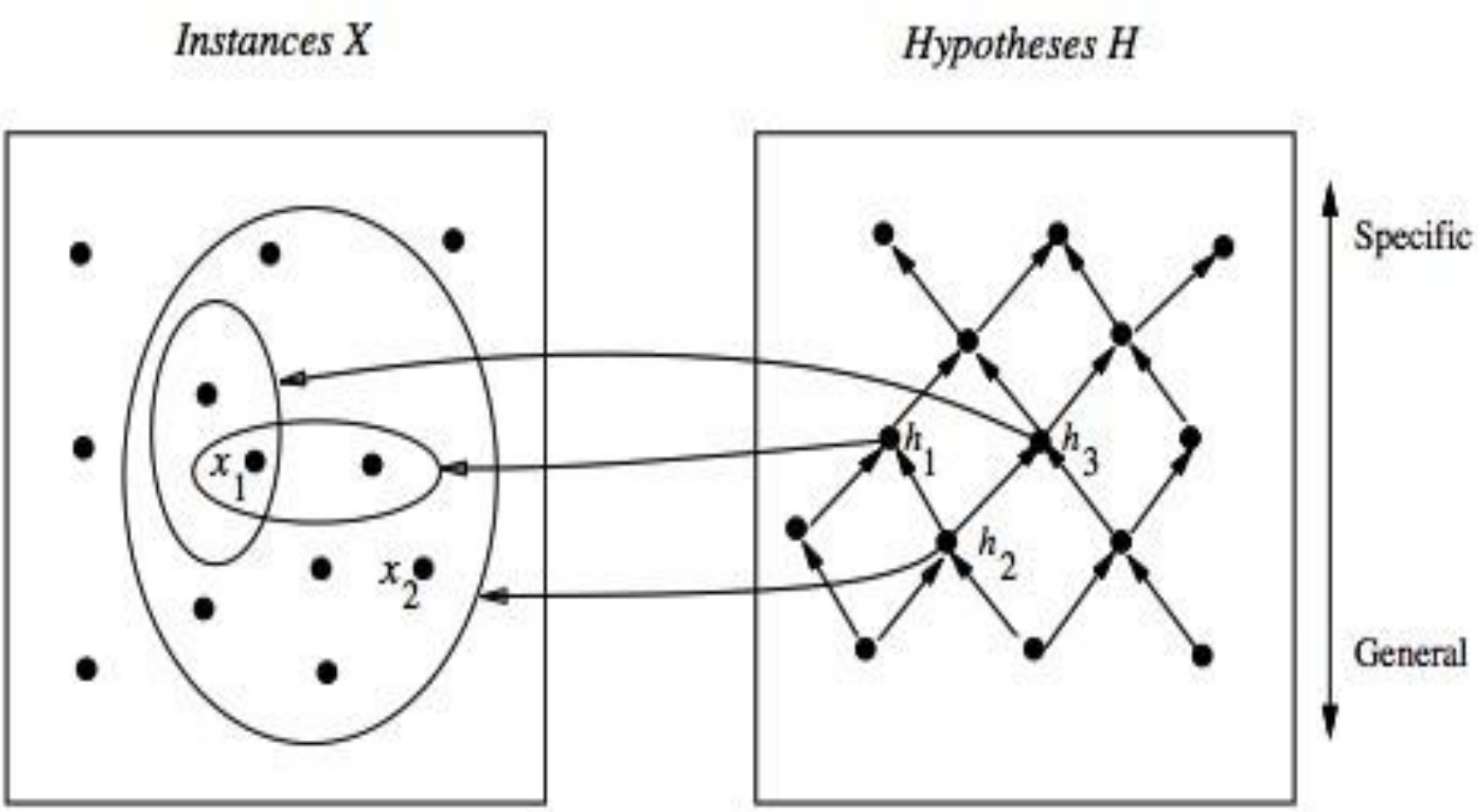
Org	Manf.	Color	Year	Type	Class
JP	HO	Blue	1980	ECO	+
JP	TO	Green	1970	SPO	−
JP	TO	Blue	1990	ECO	+
JP	TO	Red	1980	ECO	−
USA	CH	White	1980	ECO	+
JP	HO	Green	1980	ECO	+
JP	TO	Red	1990	ECO	−
JP	HO				

Example – Car Dataset

Org	Manf.	Color	Year	Type	Class
JP	HO	Blue	1980	ECO	+
JP	TO	Green	1970	SPO	−
JP	TO	Blue	1990	ECO	+
JP	TO	Red	1980	ECO	−
USA	CH	White	1980	ECO	+
JP	HO	Green	1980	ECO	+
JP	TO	Red	1990	ECO	−
JP	HO				

h_6= < ‘JP’ , ? , ?, ?, ‘ECO’>

More General than Relation



x_1 = <Sunny, Warm, High, Strong, Cool, Same>

x_2 = <Sunny, Warm, High, Light, Warm, Same>

h_1 = <Sunny, ?, ?, Strong, ?, ?>

h_2 = <Sunny, ?, ?, ?, ?, ?>

h_3 = <Sunny, ?, ?, ?, Cool, ?>

Properties and Shortcomings of Find-S

- Find-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples

- Is it a good strategy to prefer the most specific hypothesis?

- What if the training set is inconsistent (noisy)?
- What if there are several maximally specific consistent hypotheses? Find-S cannot backtrack!

- Negative example are not considered

Concept Learning as Search

Concept learning can be viewed as the task of searching through a large space of hypothesis implicitly defined by the hypothesis representation.

The goal of the concept learning search is to find the hypothesis that best fits the training examples.

Version Space

Explain version space representation theorem.

What is version space and mention the need for it?

The set of all valid hypotheses provided by an algorithm is called **version space (VS)** with respect to the hypothesis space **H** and the given example set **D**.

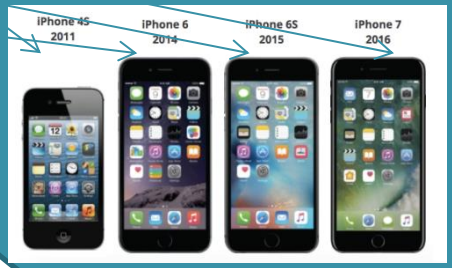
Definition: The **version space**, denoted $VS_{H,D}$, with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with the training examples in D.

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

Version Space

Hypotheses space H

Version space VS_{HD}



equal/ consistent

Training example D

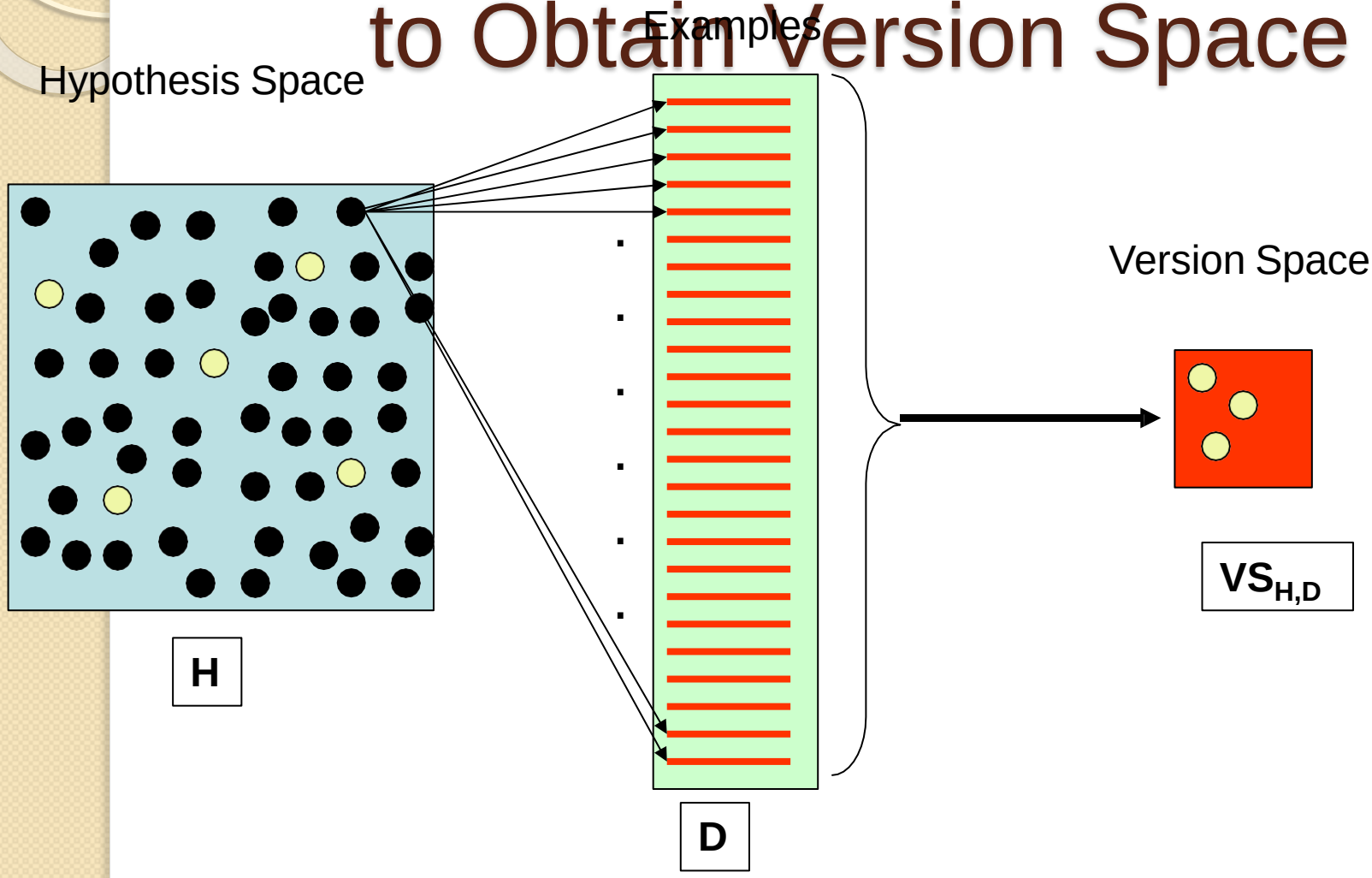
LIST-THEN-ELIMINATE Algorithm to Obtain Version Space

The LIST-THEN-ELIMINATE Algorithm

1. $VersionSpace \leftarrow$ a list containing every hypothesis in H
2. For each training example, $\langle x, c(x) \rangle$

   remove from $VersionSpace$ any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in $VersionSpace$

LIST-THEN-ELIMINATE Algorithm to Obtain Version Space



LIST-THEN-ELIMINATE Algorithm to Obtain Version Space

- In principle, the **LIST-THEN-ELIMINATE** algorithm can be applied whenever the hypothesis space H is finite.

- It is guaranteed to output all hypotheses consistent with the training data.

- Unfortunately, it requires exhaustively enumerating all hypotheses in H-an unrealistic requirement for all but the most trivial hypothesis spaces.

Candidate-Elimination Algorithm

The **Candidate-Elimination** algorithm finds all describable hypotheses that are consistent with the observed training examples

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

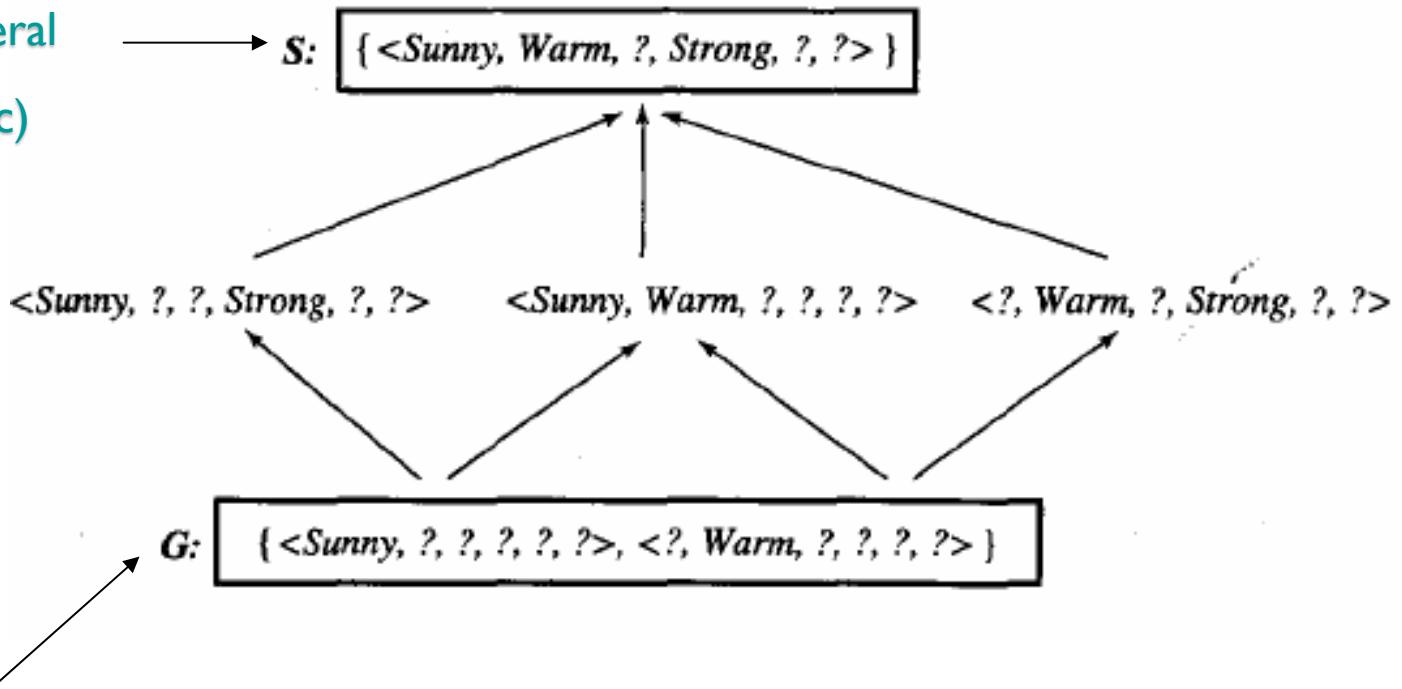
$$Consistent(h, D) \equiv (\forall \langle x, c(x)\rangle \in D)\ h(x) = c(x)$$

Hypothesis is derived from examples regardless of whether x is positive or negative example

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

Least General

(Specific)



Most General

Candidate-Elimination Algorithm

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

Example

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

Initialization

G0 ← {<?, ?, ?, ?, ?, ?>}

S0 ← {<Ø, Ø, Ø, Ø, Ø, Ø >}

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

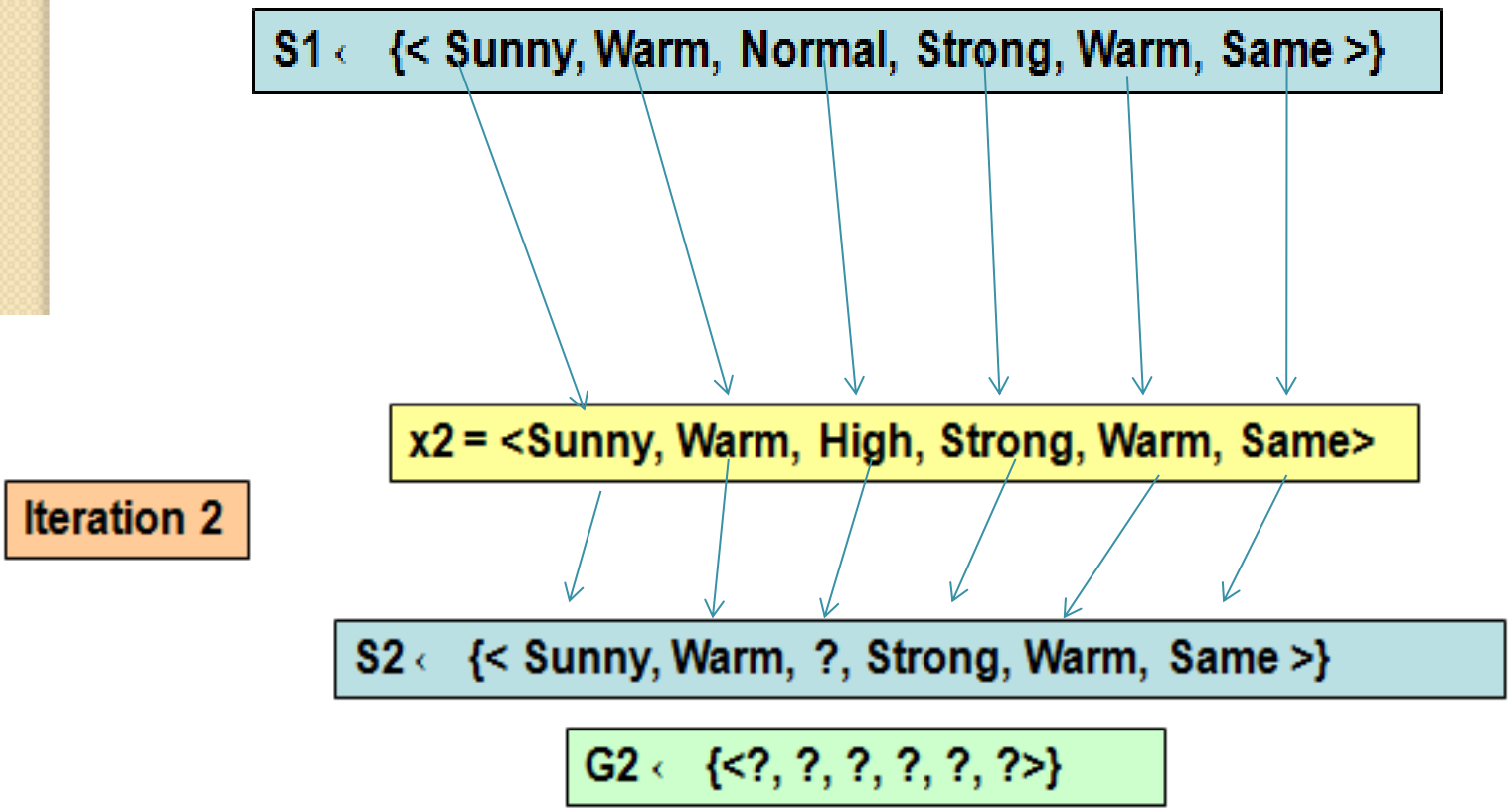
Iteration 1

x1 = <Sunny, Warm, Normal, Strong, Warm, Same>

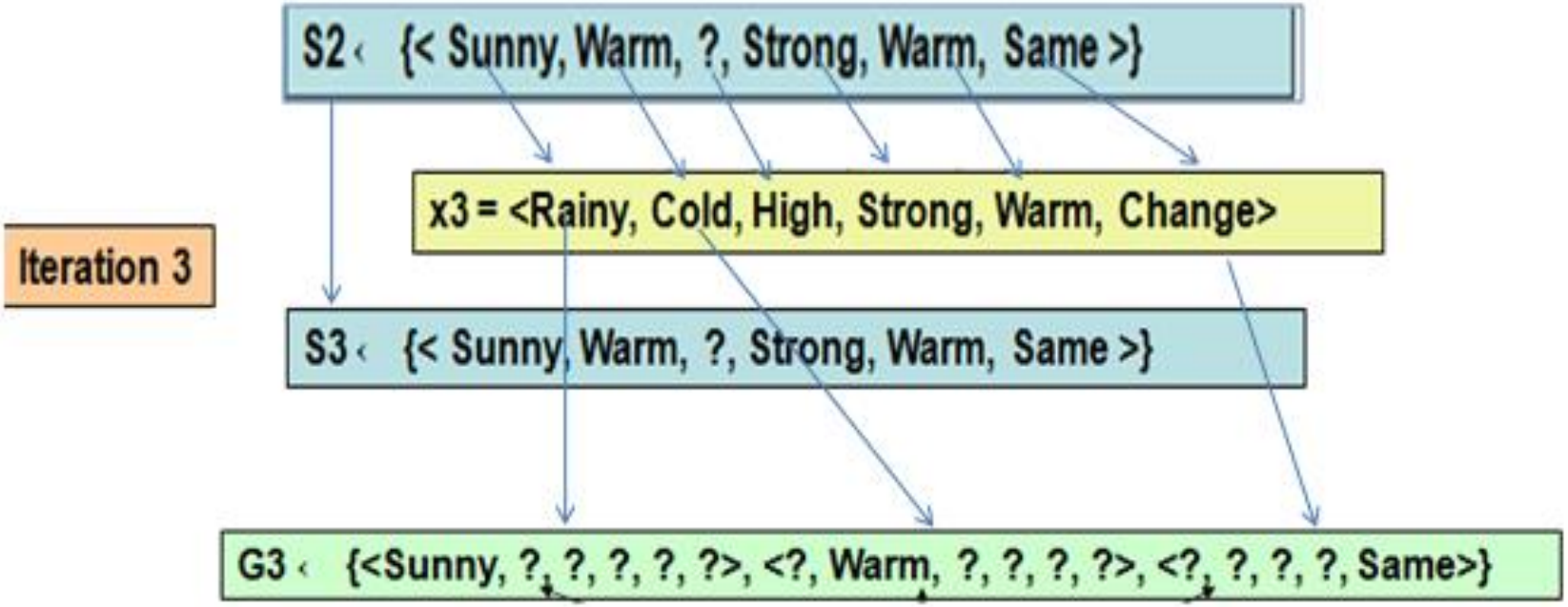
G1 ‹ {<?, ?, ?, ?, ?, ?>}

S1 ‹ {< Sunny, Warm, Normal, Strong, Warm, Same >}

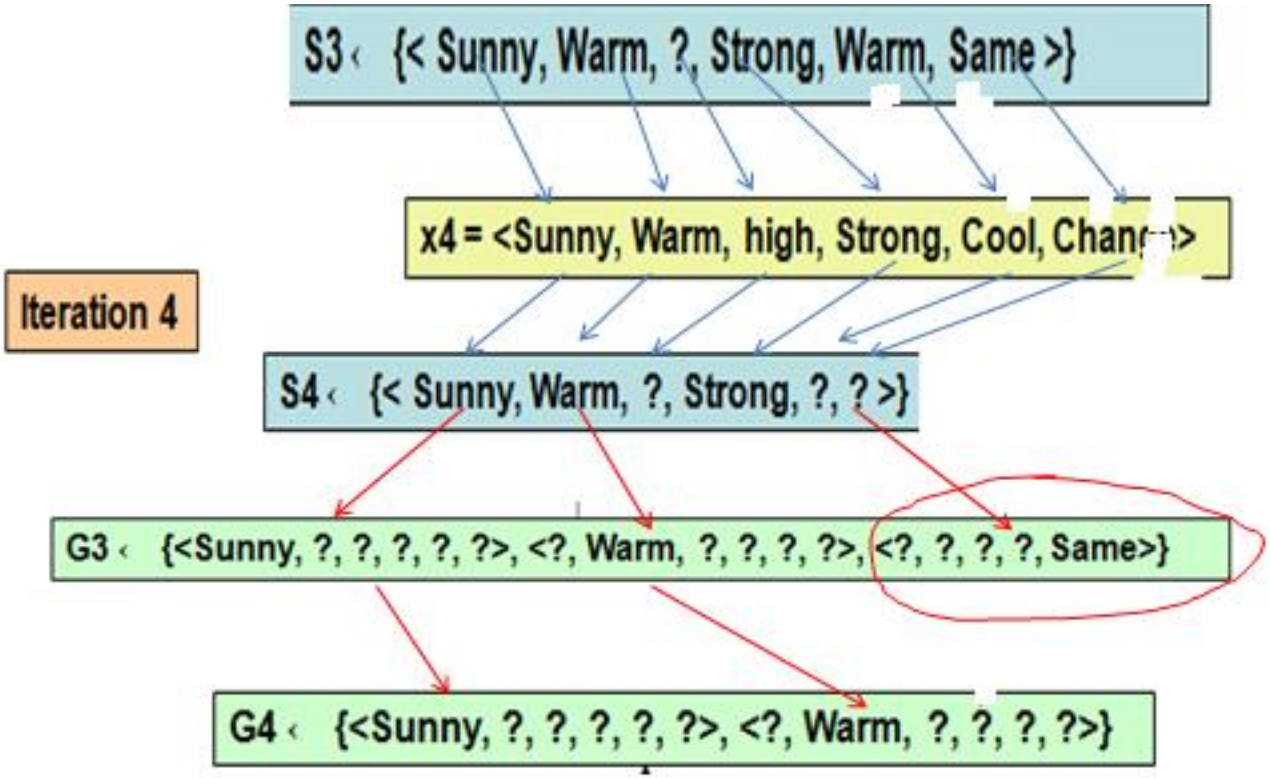
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

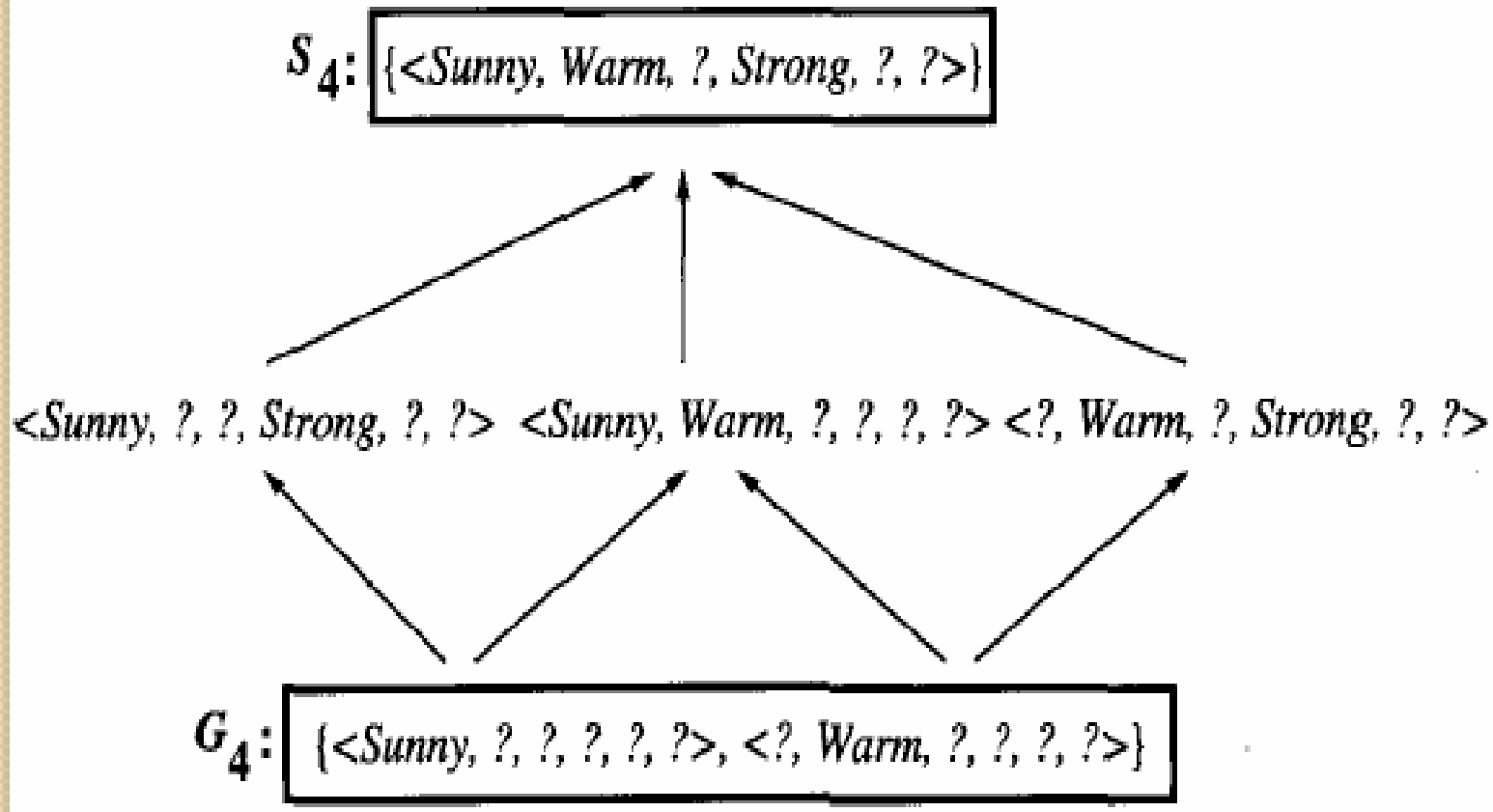


Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



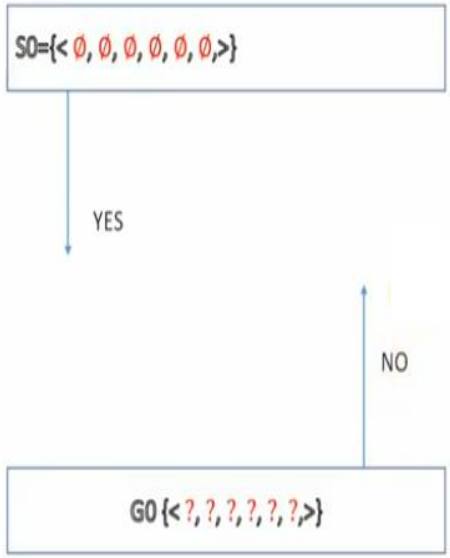
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes




S_4: {<Sunny, Warm, ?, Strong, ?, ?>}
<Sunny, ?, ?, Strong, ?, ?>
<Sunny, Warm, ?, ?, ?, ?>
<?, Warm, ?, Strong, ?, ?>
G_4: {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>}

- Candidate elimination algorithm computes the version space containing all hypotheses H.
- Candidate Elimination Algorithm finds all describable hypotheses that are consistent with the observed training examples.

- There are two terms used consistent and satisfy .
- Target concept is enjoy sport yes or no.
- An example x is said to satisfy hypothesis h when h(x)=1.Regardless of weather x is positive or negative example of target concept.
- Consistent with h depends on the target concept and in particular h(x)=c(x) h(x) = yes.



Remarks on Version Spaces and Candidate-Elimination

The version space learned by the **CANDIDATE-ELIMINATION** algorithm will converge toward the hypothesis that correctly describes the target concept, provided

(1) there are no errors in the training examples, and

(2)there is some hypothesis in H that correctly describes the target concept.

Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H For
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

Concept: Japanese Economy Car

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Initialize

$$S_0 = \langle \Phi, \Phi, \Phi, \Phi, \Phi \rangle$$

$$G_0 = \langle ?, ?, ?, ?, ? \rangle$$

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Iteration 1

x_1 = <Japan, Honda, Blue, 1980, Economy>

S_1 = <Japan, Honda, Blue, 1980, Economy>

G_1 = < ? , ?, ? , ? , ?>

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Iteration 2

S_1 = <Japan, Honda, Blue, 1980, Economy>

x_2 = <Japan, Toyota, Green, 1970, Sports>

S_2 = <Japan, Honda, Blue, 1980, Economy>

G_2 = < ?, Honda, ?, ? , ?> < ?, ? , Blue, ? , ?> < ?, ? ,? , 1980, ?> < ?, ? ,? ,?, Economy>

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Iteration 3

S_2 = <Japan, Honda, Blue, 1980, Economy>

x_3 = <Japan, Toyota, Blue, 1990, Economy>

S_3 = <Japan, ?, Blue, ?, Economy>

G_2 = < ?, Honda, ?, ?, ?> < ?, ?, Blue, ?, ?> < ?, ?, ?, 1980, ?> < ?, ?, ?, ?, Economy>

G_3 = < ?, ?, Blue, ?, ?> < ?, ?, ?, ?, Economy>

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Iteration 4

S_3 = = <Japan, ?,Blue, ?,Economy>

x_4 = <USA, Chrysler, Red,1980, Economy>

S_4 = <Japan, ?,Blue, ?,Economy>

G_4 = < ?, ? ,Blue, ? , ?> < Japan, ? ,? ,?, Economy>

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Iteration 5

S_4 = <Japan, ?,Blue, ?,Economy>

x_5 = <Japan, Honda, White, 1980, Economy>

S_5 = <Japan, ?, Blue, ?, Economy>

G_5 = < ?, ? ,Blue, ? , ?> < Japan, ? ,? ,?, Economy>

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Iteration 6

S_5 = <Japan, ?, Blue, ?, Economy>

x_6 = <Japan, Honda, Green, 1980, Economy>

S_6 = <Japan, ?, ?, ?, Economy>

G_6 = < Japan, ? ,? ,?, Economy>

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

Iteration 7

S_6 = <Japan, ?, ?, ?, Economy>

x_7 = <Japan, Honda, Red, 1990, Economy>

S_7 = <Japan, ?, ?, ?, Economy>

G_7 = < Japan, ? ,? ,?, Economy>

- Example is inconsistent with the version-space.

- G cannot be specialized.
- S cannot be generalized.

- • The version space collapses.
- • Conclusion: No conjunctive hypothesis is consistent with the data set.

Consider the following training data sets

Size	Color	Shape	Class/Label
big	red	circle	No
small	red	triangle	No
small	red	circle	Yes
big	blue	circle	No
small	blue	circle	Yes

Size	Color	Shape	Class/Label
big	red	circle	No
small	red	triangle	No
small	red	circle	Yes
big	blue	circle	No
small	blue	circle	Yes

$G0 = \{\langle ?, ?, ? \rangle\}$

$S0 = \{\langle 0, 0, 0 \rangle\}$

Size	Color	Shape	Class/Label
big	red	circle	No
small	red	triangle	No
small	red	circle	Yes
big	blue	circle	No
small	blue	circle	Yes

- First (negative) example, (big, red, circle),
- So, the minimal specializations would make the new hypothesis space

$G1 = \{\langle small, ?, ? \rangle, \langle ?, blue, ? \rangle, \langle ?, ?, triangle \rangle\}$

$S1 = S0 = \{\langle 0, 0, 0 \rangle\}$

Size	Color	Shape	Class/Label
big	red	circle	No
small	red	triangle	No
small	red	circle	Yes
big	blue	circle	No
small	blue	circle	Yes

Second example, (small, red, triangle), which is also negative, you will need to further specialize G.

G2 = {<small, blue, ?>, <small, ?, circle>, <?, blue, ?>, <big, ?, triangle>, <?, blue, triangle>}

G1 = {<small, ? , ?>, <?, blue, ?>, <?, ?, triangle>}

S2 = S1 = S0 = {<0, 0, 0>}

Size	Color	Shape	Class/Label
big	red	circle	No
small	red	triangle	No
small	red	circle	Yes
big	blue	circle	No
small	blue	circle	Yes

G2 = {<small, blue, ?>, <small, ?, circle>, <?, blue, ?>, <big, ?, triangle>, <?, blue, triangle>}

G3 = {<small, ?, circle>}

S3 = {<small, red, circle>}

Size	Color	Shape	Class/Label
big	red	circle	No
small	red	triangle	No
small	red	circle	Yes
big	blue	circle	No
small	blue	circle	Yes

G4 = G3 = {<small, ?, circle>}

S4 = S3 = {<small, red, circle>}

Size	Color	Shape	Class/Label
big	red	circle	No
small	red	triangle	No
small	red	circle	Yes
big	blue	circle	No
small	blue	circle	Yes

$G5 = \{\langle small, ?, circle \rangle\}$

$S5 = \{\langle small, ?, circle \rangle\}$

- Since G and S are equal, you have learned the concept of "small circles".

Comparison

Algorithm	Order	Strategy	N/P
FIND-S	Specific-to-general	Top-down	Positive
LIST-THEN-ELIMINATE	General-to-Specific	Bottom-up	Negative
CANDIDATE-ELIMINATION	Bi-directional	Bi-directional	Both

Thank You