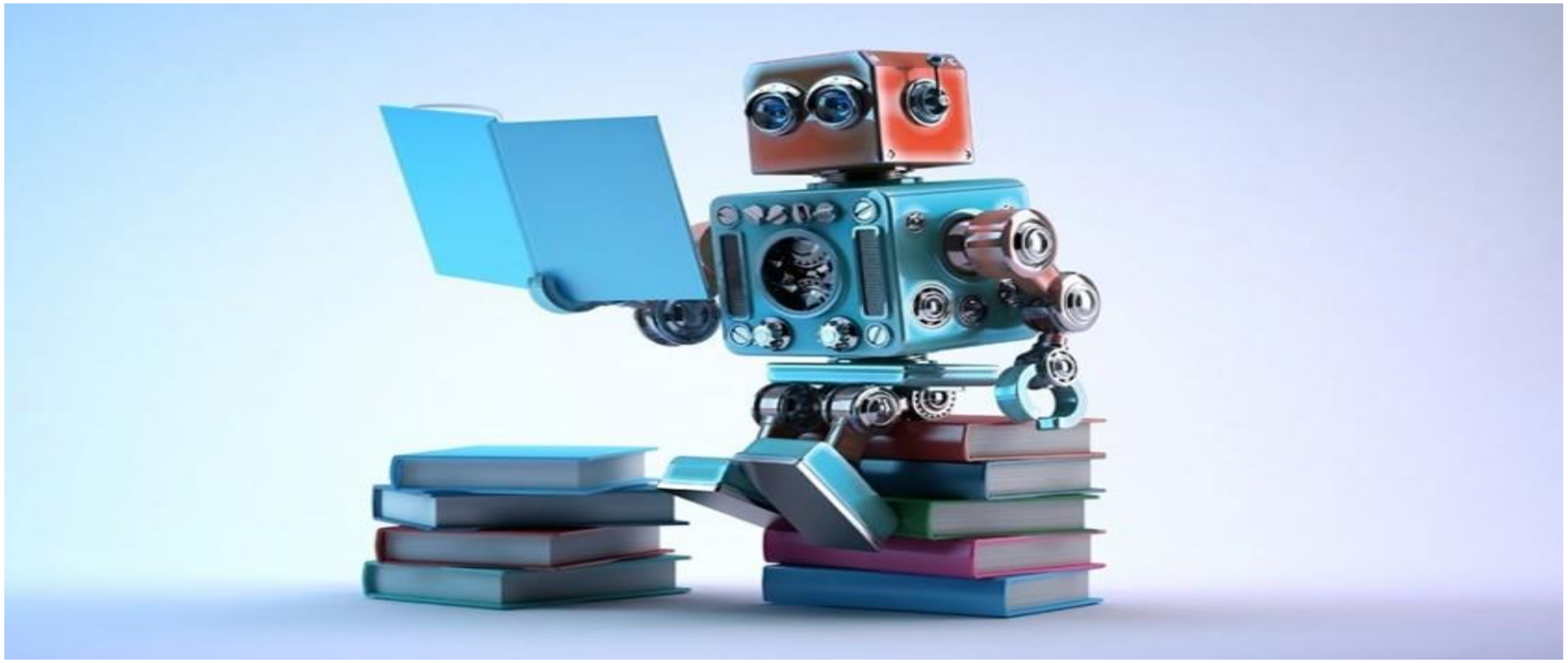


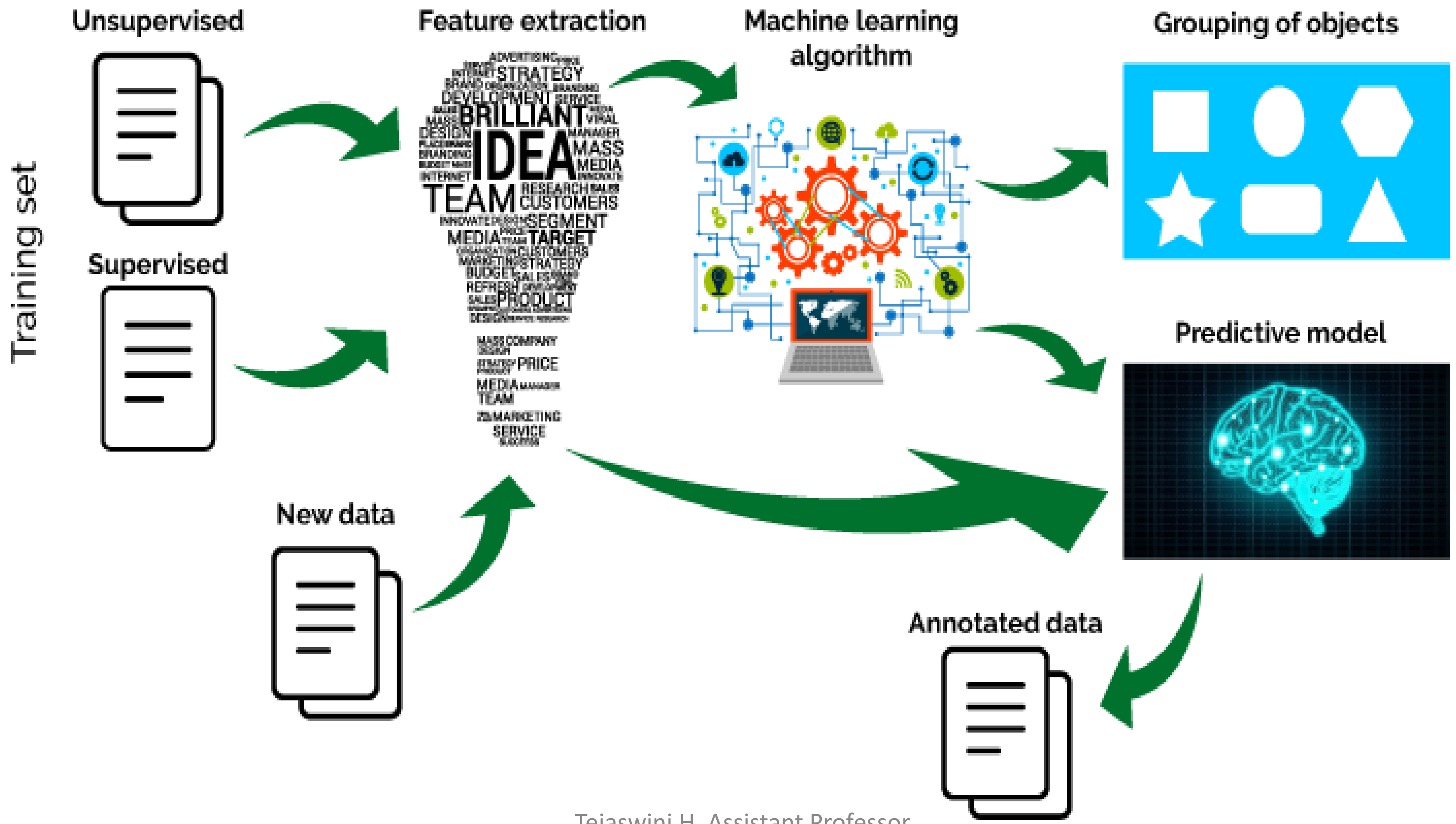
Module 1
Introduction to Machine Learning

Tejaswini H, Assistant Professor,
Department of CSE

What is Machine Learning?

- Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.
- **Machine learning focuses on the development of computer programs** that can access data and use it learn for themselves.

- The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.
- **The primary aim is to allow the computers learn automatically** without human intervention or assistance and adjust actions accordingly.


Training set
Unsupervised
Supervised
New data
Feature extraction
Machine learning algorithm
Grouping of objects
Predictive model
Annotated data

- https://www.artificial-intelligence.blog/news/how-companies-use-machine-learning

Some 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.**

* Learning to recognize spoken words

- For example, the SPHINX system (e.g., Lee 1989) learns speaker-specific strategies for recognizing the primitive sounds 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.

* 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.

* 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.

* 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.

Outline

1.1 WELL-POSED LEARNING PROBLEMS

1.2 DESIGNING A LEARNING SYSTEM

1.3 PERSPECTIVES AND ISSUES IN MACHINE LEARNING

1.1 WELL-POSED LEARNING PROBLEMS

- ***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**.

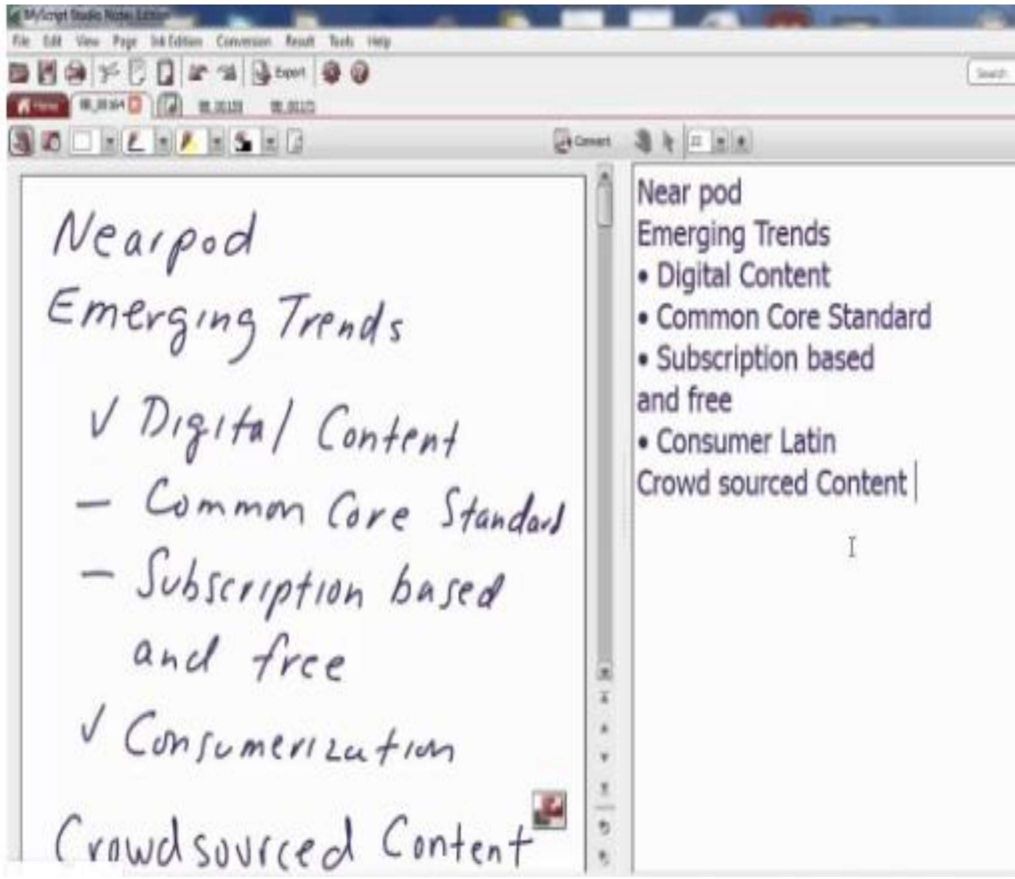
- In general, to have a well-defined learning problem, we must identity these three features:
  - the class of tasks (T)
  - the measure of performance to be improved (P)
  - and the source of experience (E)

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

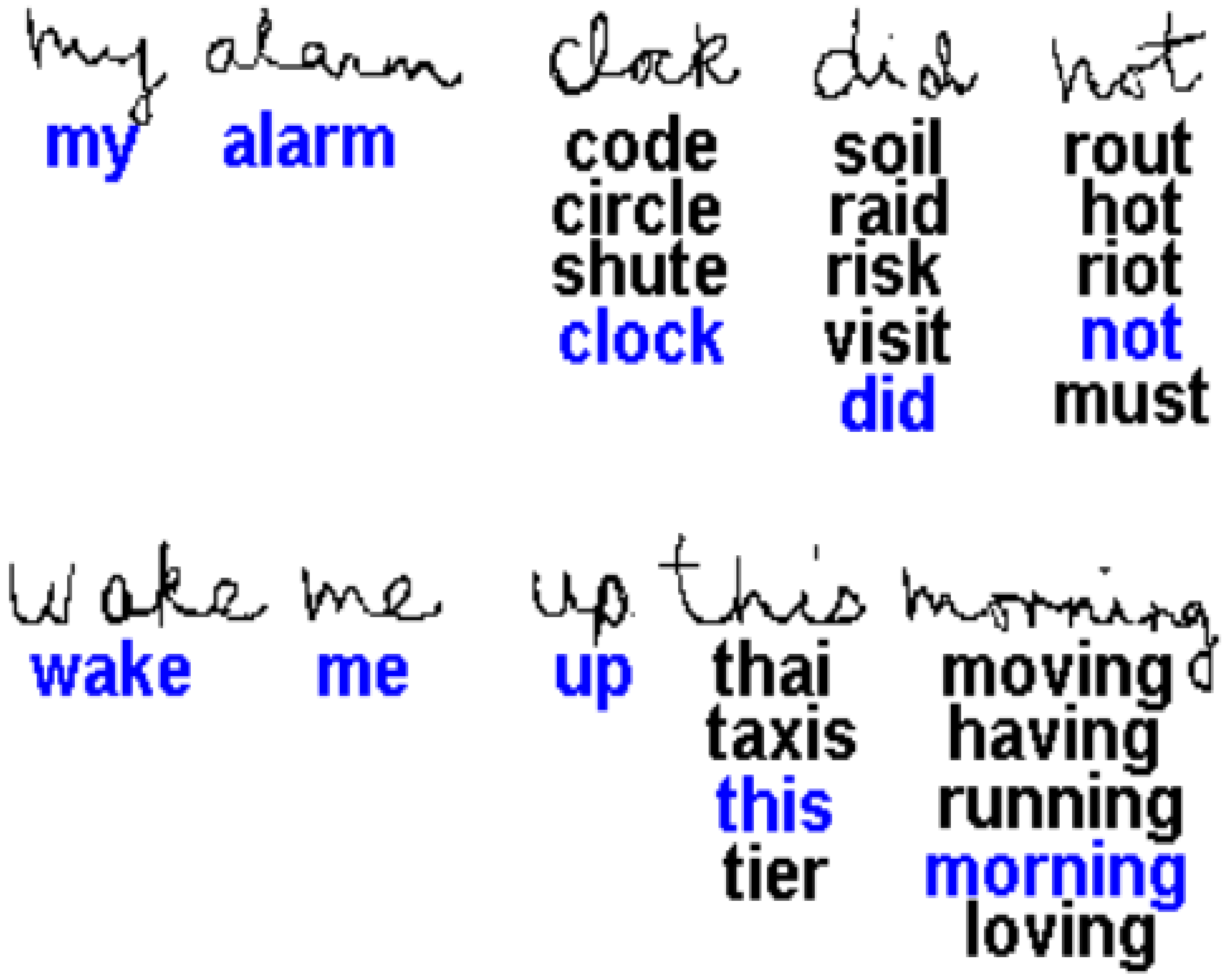
A handwriting recognition learning problem:

- ***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



Nearpod
Emerging Trends
✓ Digital Content
– Common Core Standard
– Subscription based
and free
✓ Consumerization
Crowdsourced Content

Near pod
Emerging Trends
• Digital Content
• Common Core Standard
• Subscription based
and free
• Consumer Latin
Crowd sourced Content


my
alarm
code
circle
shute
clock
soil
raid
risk
visit
did
rout
hot
riot
not
must
wake
me
up
thai
taxis
this
tier
moving
having
running
morning
loving

A robot driving learning problem:

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

1.2 DESIGNING A LEARNING SYSTEM

Consider:

- **designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament.**
- We adopt the obvious **performance measure:** ***the percent of games it wins in this world tournament***.

- **1.2.1 Choosing the Training Experience**
- **1.2.2 Choosing the Target Function**
- **1.2.3 Choosing a Representation for the Target Function**
- **1.2.4 Choosing a Function Approximation Algorithm**
- **1.2.5 The Final Design**

1.2.1 Choosing the Training Experience

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

Training experience
Key Attribute-1

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

- For example, 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.
  - Alternatively, it might have available only ***indirect*** information consisting of the move sequences and final outcomes of various games played.

- In this later case, 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.

Training experience
Key Attribute-2

- The degree to which the learner controls the sequence of training examples.

- The learner might rely on the teacher to select informative board states and to provide the correct move for each.
- 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.

Training experience
Key Attribute-3

- How well it represents the distribution of examples over which the final system performance P must be measured.

- In general, learning is most reliable when the training examples follow a distribution similar to that of future test examples.

- In our 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 obvious 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.

- In practice, it is often necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be.
- Such situations are problematic because mastery of one distribution of examples will not necessary lead to strong performance over some other distribution.

- **A checkers learning problem:**
  - **Task T**: playing checkers
  - **Performance measure *P:*** percent of games won in the world tournament
  - **Training experience E:** games played against itself

Checkers game

- Step 1: Position board Sit across from the other player, with the board between you. Turn the board so that you each have a red corner square on your right. Tip If your board isn't black and red, just imagine that the lighter color on your board is red, and the darker color is black to follow these instructions.
- Step 2: Decide first player Decide who's going first. You can flip a coin, base it on who won the last game, or just agree. Whatever you choose, the starting player gets the black checkers and the other player gets the red.

- Step 3: Set up board Set up the board. Each player arranges all twelve of his checkers on the black squares (and only black squares!) of the first three rows of his side of the board.
- Step 4: Make first move The first player (black) begins by moving any one of his checkers in the row closest to the middle of the board diagonally one space. Checkers can only move diagonally, which means the game is played entirely on the black squares.

- Step 5: Next player moves The next player moves one of his checkers diagonally one space. At this stage of the game, you can only move your pieces forward. Step 6: Take turns Keep taking turns this way, moving another checker or the same checker further forward.

- Step 7: Jump & capture On your turn, if your opponent's checker is in front of you—that is, on a black square diagonal to yours—and there's an empty square on the other side, you can jump your checker over his, landing in the empty square. Take the other player's checker that you just jumped over off the board and put it to the side. Tip Remember: if your checker is jumpable on your opponent's turn, he can jump and capture it, so before that happens, try to move out of the way or fill in the empty square behind you so there's no place for him to jump.
- Step 8: Multiple capture You can jump over and capture more than one checker in a turn. Just remember that there must be one empty square between each one, so that you're leapfrogging one checker at a time—not long-jumping over two.

- Step 9: "Crown me!" Continue to take turns moving and jumping. If you manage to get a checker all the way to the other side of the board, tell your opponent, 'Crown me!' He'll have to take one of the checkers of yours that he captured and place it on top of your checker. Now you've got a tall 'king' that can move both forward and backward.
- Step 10: Take turns until win Continue to take turns moving, jumping, and being crowned, until one player captures all of the other's checkers.

What’s next >>>

- In order to complete the design of the learning system, we must now choose

1. the exact type of knowledge to be learned

2. a representation for this target knowledge

3. a learning mechanism

1.2.2 Choosing the Target Function

- Let us 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.

- GOAL is >>>
  - 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.

- Let us call this function ***ChooseMove***
- Let us represent it as ***ChooseMove* : B --> M**
- i.e 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.**

- Problem???
  - Although ***ChooseMove*** is an obvious choice for the target function in our example, this function will turn out to be very difficult to learn given the kind of **indirect training experience** available to our system.

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

- Let us call this target function ***V***
- Let us use the notation ***V*** **: B -->** $\Re$
- *i.e.* ***V*** maps any legal board state from the set B to some real value
- (we use $\Re$ to denote the set of real numbers).

- 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.
- This can be accomplished by generating the successor board state produced by every legal move, then using ***V*** to choose the best successor state and therefore the best legal move.

- 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

- But this not efficiently computable by our checkers playing program, we say that it is a ***nonoperational*** definition.
- The goal >>
  - to discover an ***operational*** description of ***V ;*** that is, a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds.

- Thus, we have reduced the learning task in this case to the problem of discovering an ***operational description of the ideal target function*** V.
- It may be very difficult in general to learn such **an** operational form of V perfectly.

- In fact, we often expect learning algorithms to acquire only some ***approximation*** to the target function, and for this reason the process of learning the target function is often called ***function approximation.***
- In the current discussion we will use the symbol $\hat{V}$ to refer to the **function that is actually learned by our program**, to distinguish it from the **ideal target function V**.

1.2.3 Choosing a Representation for the Target Function

- Now that we have specified the **ideal target function V**, we must choose a representation that the learning program will use to describe the function $\hat{V}$ that it will learn.

Simple representation:

- For any given board state, the function c will be calculated as a linear combination of the following board features:
  - ***x_1:*** the number of black pieces on the board
  - ***x_2:*** the number of red pieces on the board
  - ***x_3:*** the number of black kings on the board
  - **x_4:** the number of red kings on the board
  - **x_5**: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
  - **X_6**: the number of red pieces threatened by black

Thus, our learning program will represent $\hat{V}(b)$ 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 $\boldsymbol{w_0}$ through $\boldsymbol{w_6}$ are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights $\boldsymbol{w_1}$ through $\boldsymbol{w_6}$ will determine the relative importance of the various board features in determining the value of the board, whereas the weight $\boldsymbol{w_o}$ will provide an additive constant to the board value.

Summary so far…….

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:* $V : Board \rightarrow \Re$
- *Target function representation*

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

1.2.4 Choosing a Function Approximation Algorithm

- In order to learn the target function $\hat{V}$ we require a set of training examples, each describing a specific board state **b** and the training value **$V_{train}(b)$** for b.
- In other words, each training example is an ordered pair of the form **$(b, V_{train}(b))$**.

- For instance, the following training example describes a board state b in which black has won the game (note $\boldsymbol{x_2} = 0$ indicates that **red** has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore **+100.**

$$\langle\langle x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0\rangle, +100\rangle$$

- Below we describe a procedure that first derives such training examples from the indirect training experience available to the learner, then adjusts the weights $\boldsymbol{w_i}$ to best fit these training examples

- **1.2.4.1 ESTIMATING TRAINING VALUES**
- **1.2.4.2 ADJUSTING THE WEIGHTS**

1.2.4.1 ESTIMATING TRAINING VALUES

- the only training information available to our learner is whether the game was eventually won or lost.
- But…
  - we require training examples that assign specific scores to specific board states.
  - the game was eventually won or lost does not necessarily indicate that ***every*** board state along the game path was necessarily good or bad.

- Need to assign specific scores to *intermediate* board states
- Approximate intermediate board state *b* using the learner's current approximation of the next board state following *b*

Rule for estimating training values.

$$V_{train}(b) \leftarrow \hat{V}(Successor(b)) \qquad (1.1)$$

- we are using estimates of the value of the ***Successor(b)*** to estimate the value of board state $\hat{V}$.
- More accurate for states closer to end states

1.2.4.2 ADJUSTING THE WEIGHTS

- What is remaining?!!!
  - to specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{<b, V_{train(b)}>\}$

- Best fit?!!!!
  - define the best hypothesis, or set of weights, as that which minimizes the square error **E** between the training values and the values predicted by the hypothesis V̂ .

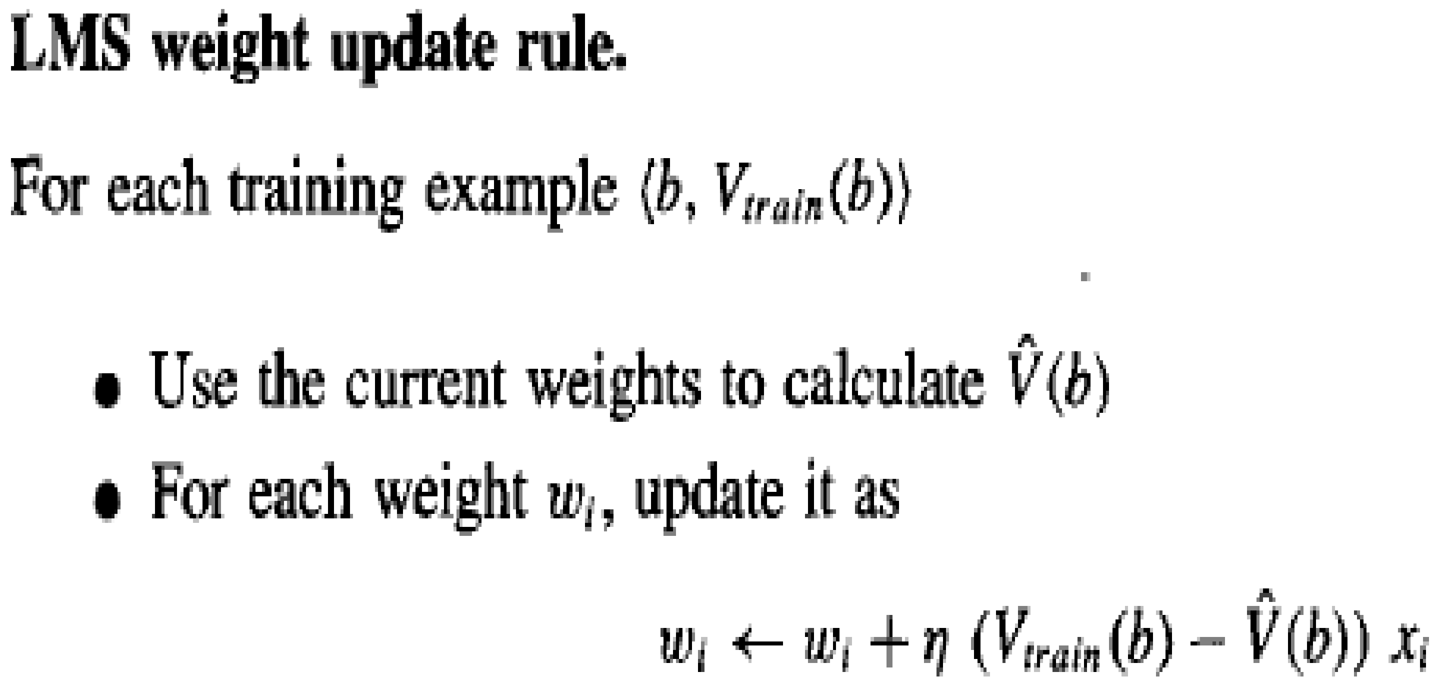
$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \ training\ examples} (V_{train}(b) - \hat{V}(b))^2$$

Best algorithm………

- Several algorithms are known for finding weights of a linear function that minimize E defined in this way.
- But
  - 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



LMS weight update rule.

For each training example $\langle b, V_{train}(b)\rangle$

- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i, update it as

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

1.2.5 The Final Design

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

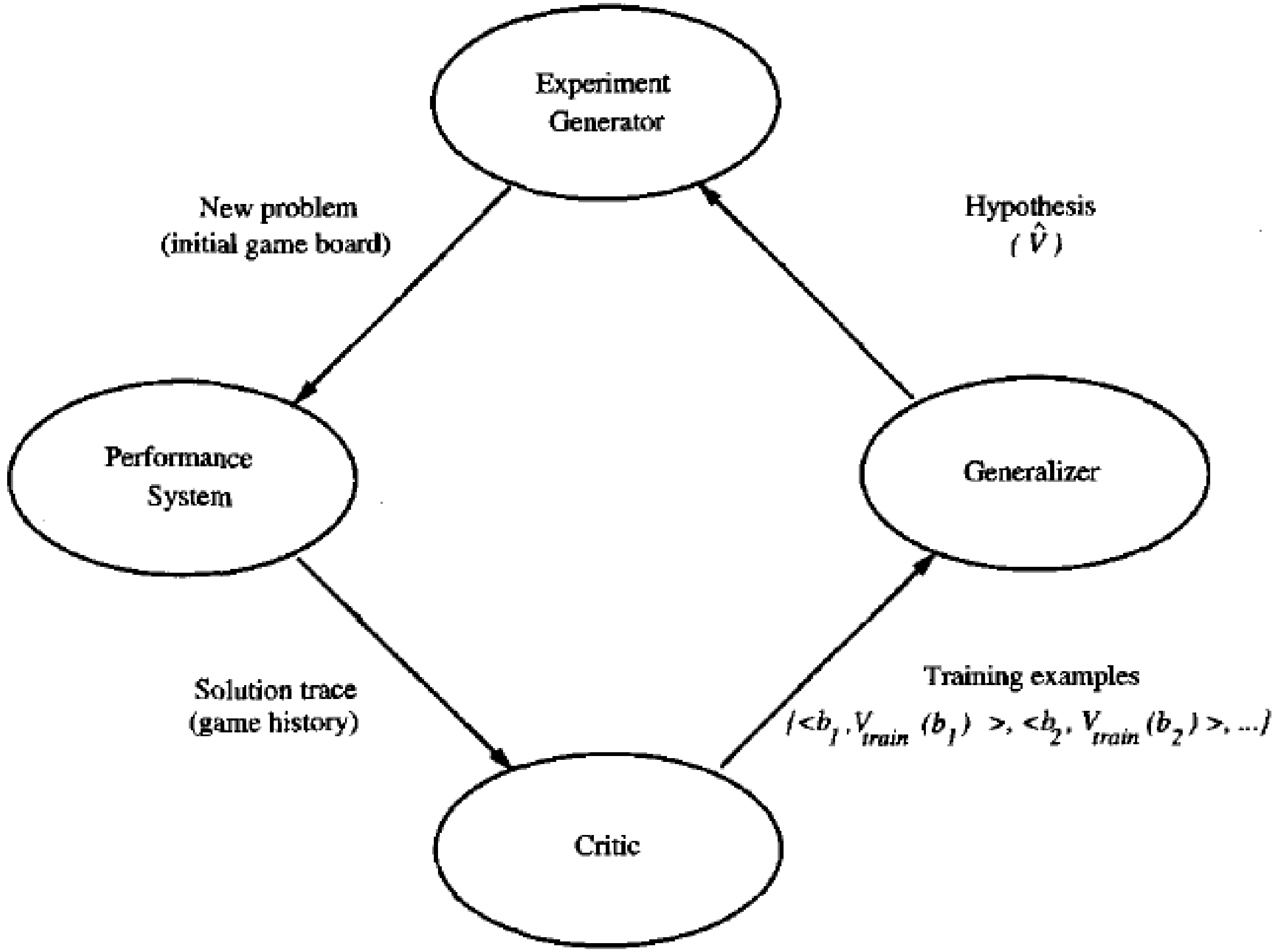

Experiment Generator
New problem (initial game board)
Hypothesis ($\hat{V}$)
Performance System
Generalizer
Solution trace (game history)
Training examples
$\{<b_1, V_{train}(b_1)>, <b_2, V_{train}(b_2)>, ...\}$
Critic

FIGURE 1.1
Final design of the checkers learning program.

Performance System

- The **Performance System** is the module that must solve the given performance task, by using the learned target function(s)
  - in this case: playing checkers
- It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.
- In our case, the strategy used by the Performance System to select its next move at each step is determined by the learned evaluation function $\hat{V}$

Critic

- 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 $\boldsymbol{V_{train}}$ of the target function value for this example.
- In our example, the Critic corresponds to the training rule given by Equation (1.1).

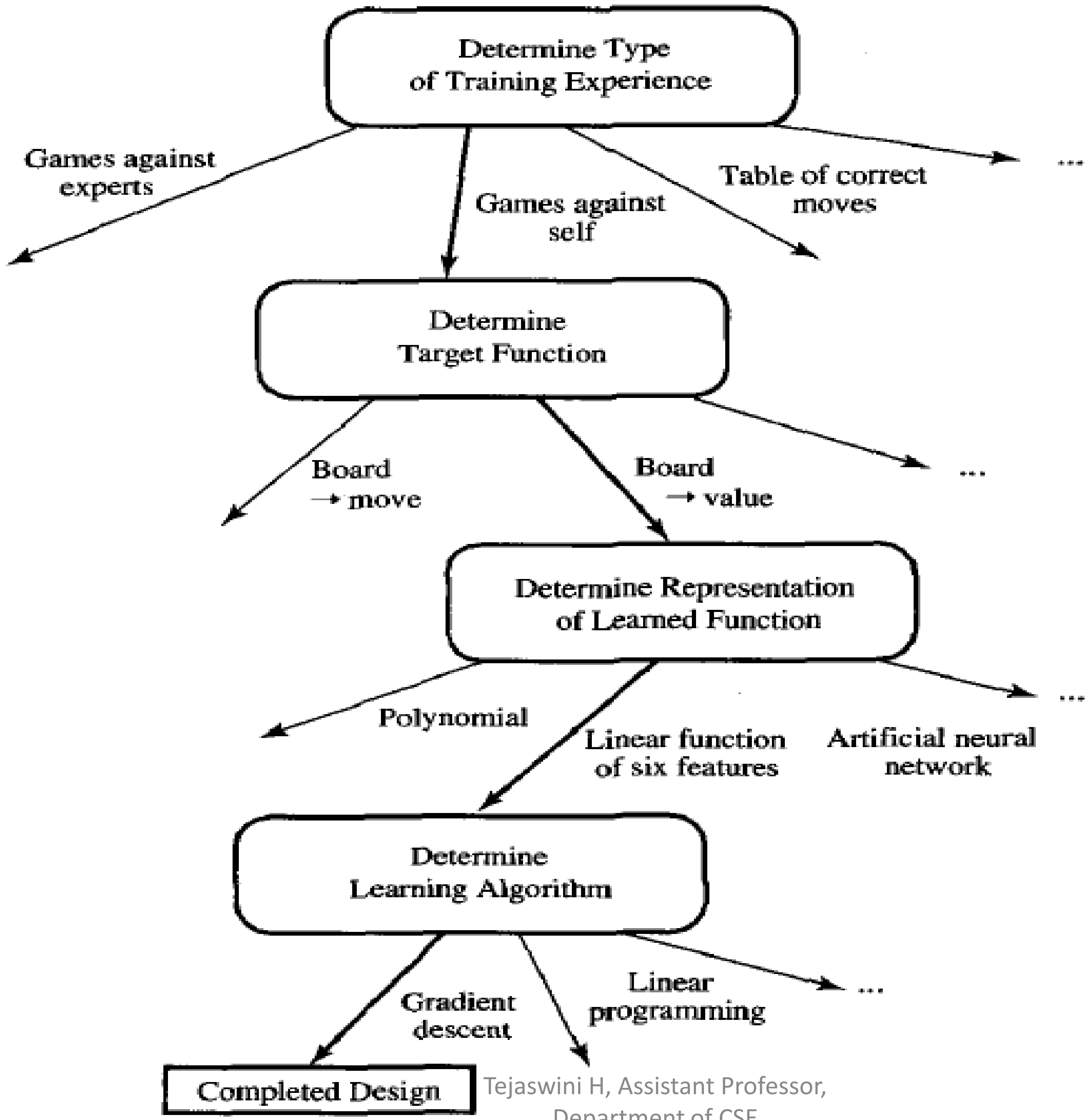
Generalizer

- 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 $\boldsymbol{w_0}$, . . . , $\boldsymbol{w_6}$.

Experiment Generator

- The **Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem 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 follows a very simple strategy: It always proposes the same initial game board to begin a new game.

Summary of choices in designing the checkers learning program.


Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
...
Determine Target Function
Board → move
Board → value
...
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
...
Determine Learning Algorithm
Gradient descent
Linear programming
...
Completed Design

Issues in Machine Learning

- What algorithms exist for learning general target functions from specific training examples? 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?

Module 1: Chapter 2
Concept Learning

- The problem of inducing general functions from specific training examples is central to learning.

- **concept learning**: acquiring the definition of a general category given a sample of positive and negative training examples of the category.

- Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.

- Much of learning involves acquiring general concepts from specific training examples.
- People, for example, continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.

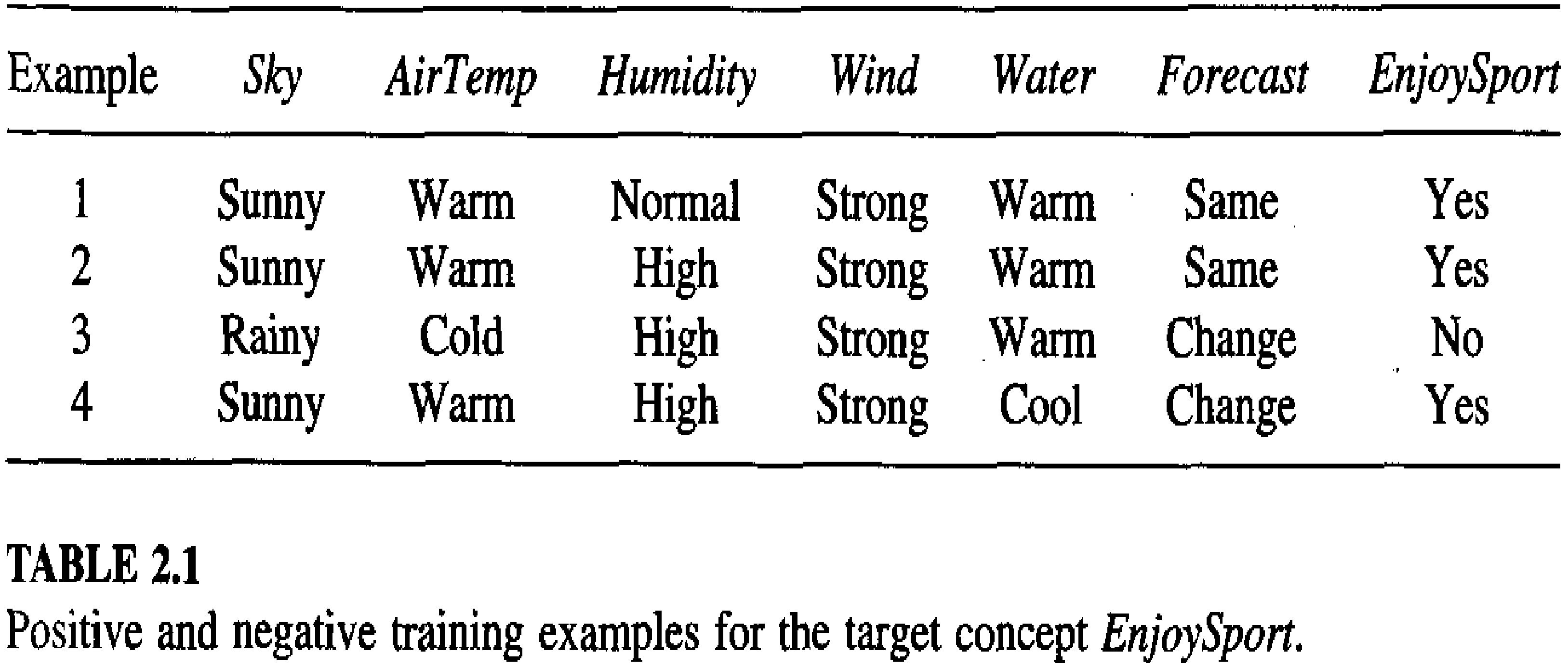
- Each such concept can be viewed as **describing some subset of objects or events defined over a larger set** (e.g., the subset of animals that constitute birds).
- Alternatively, each concept can be thought of as a **boolean-valued function defined over this larger set** (e.g., a function defined over all animals, whose value is true for birds and false for other animals).

- we consider the problem of automatically inferring the general definition of some concept, given examples labeled as **members or nonmembers of the concept**.
- This task is commonly referred to as ***concept learning***, or approximating a boolean-valued function from examples.

- **Concept learning:** Inferring a boolean-valued function from training examples of its input and output.

A CONCEPT LEARNING TASK

- concept learning:
  - 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 2.1
Positive and negative training examples for the target concept *EnjoySport*.

- Table 2.1 describes a set of example days, each represented by a set of ***attributes.***
- The attribute ***EnjoySport*** indicates whether or not **Aldo 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 shall we provide to the learner in this case?
  - Let us begin by considering 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***
  - ***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 *"0"* 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 **most general hypothesis**-that **every day** is a positive example-is represented by

  < ?, ?, ?, ?, ?, ? >

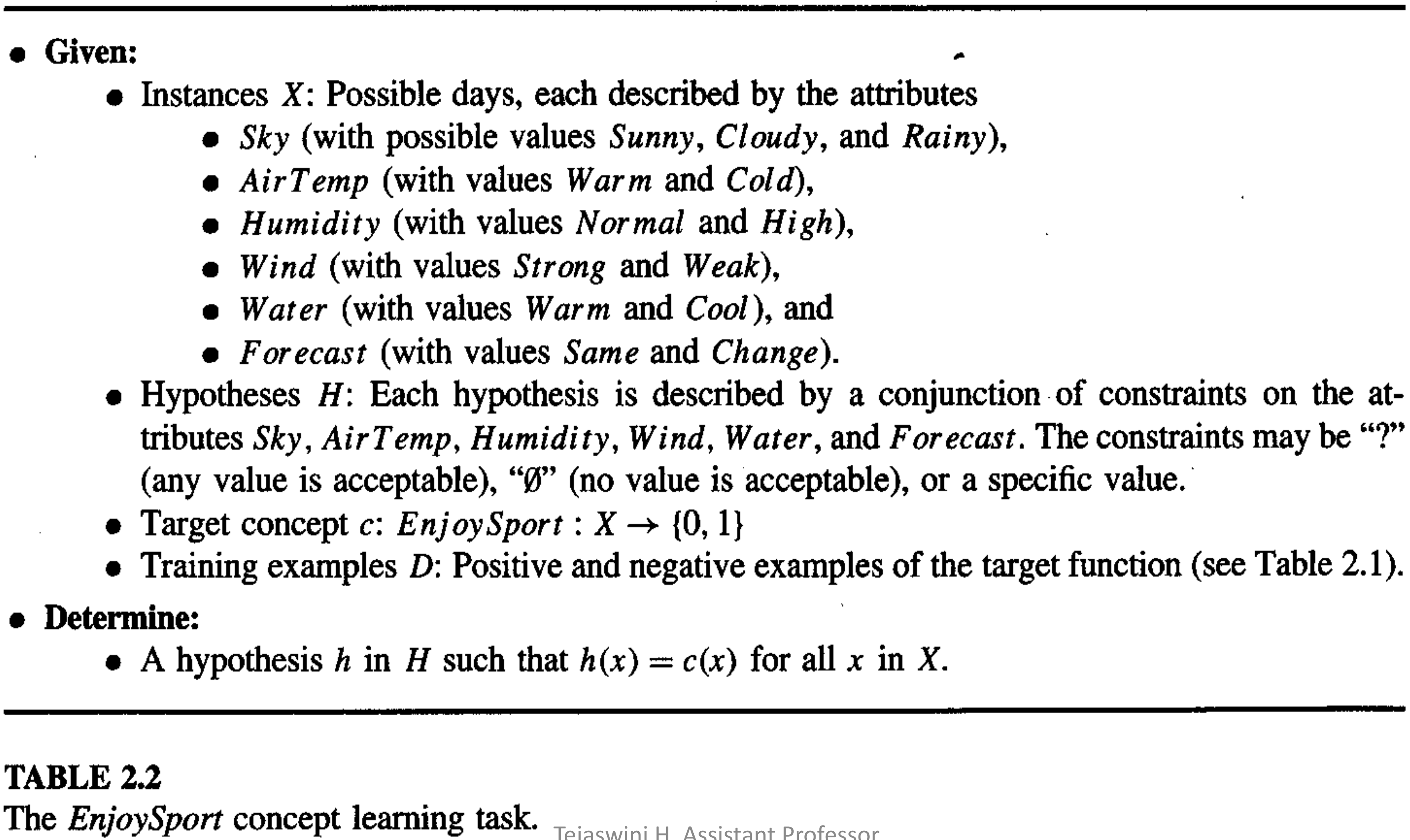
- the **most specific possible hypothesis**-that **no day** is a positive example-is represented by

  < 0, 0, 0, 0, 0, 0 >

- To summarize, the ***EnjoySport*** concept learning task requires learning the set of days for which ***EnjoySport = yes***, describing this set by a **conjunction of constraints** over the instance attributes.

- In general, any concept learning task can be described by
  - the set of instances over which the target function is defined
  - the target function
  - the set of candidate hypotheses considered by the learner
  - the set of available training examples

The definition of the *EnjoySport* concept learning task in this general form



- **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), "Ø" (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 2.2
The *EnjoySport* concept learning task.

Notation

- **Instances X**: The set of items over which the concept is defined is called the set of instances.
  - In the current example, X: is the set of all possible days, each represented by the attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast.

- **Target concept c**: The concept or function to be learned is called the target concept.
- In general, c can be any boolean valued function defined over the instances **X**; i.e. **c : X → {0, 1}**.
  - In the current example, the target concept corresponds to the value of the attribute ***EnjoySport***
  - c(x) = 1 if EnjoySport = Yes, and
  - c(x) = **0** if EnjoySport = No.

- ***training examples D:*** When learning the target concept, the learner is presented a set of ***training examples,*** each consisting of an instance ***x*** from X, along with its target concept value ***c(x)***
- We will often write the ordered pair ***<x, c(x)>*** to describe the training example consisting of the instance **x** and its target concept value ***c(x).***

- 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 nonmembers of the target concept.

- **Hypothesis h**: Given a set of training examples of the target concept ***c,*** the problem faced by the learner is to hypothesize, or estimate, ***c.***
- We use the symbol **H** to denote the set of ***all possible hypotheses*** that the learner may consider regarding the identity of the target concept.
- In general, each hypothesis ***h*** in H represents a boolean-valued function defined over X; i.e. $h : X \rightarrow \{0, 1\}$.
- The goal of the learner is to find a hypothesis ***h*** such that $h(x) = c(x)$ for all ***x*** in X.

The Inductive Learning Hypothesis

- Although the learning task is to determine a hypothesis **h** identical to the target concept ***c*** over the entire set of instances X, the only information available about ***c*** is its value over the training examples.

- Inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data.
- Lacking any further information, our assumption is that the best hypothesis regarding unseen instances is the **hypothesis that best fits the observed training data**.
- This is the fundamental assumption of inductive learning.

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**

- distinct instances
- syntactically distinct hypotheses within H
- semantically distinct hypotheses

- We are interested in **algorithms capable of efficiently searching very large or infinite hypothesis spaces**, to find the hypotheses that best fit the training data.

General-to-Specific Ordering of Hypotheses

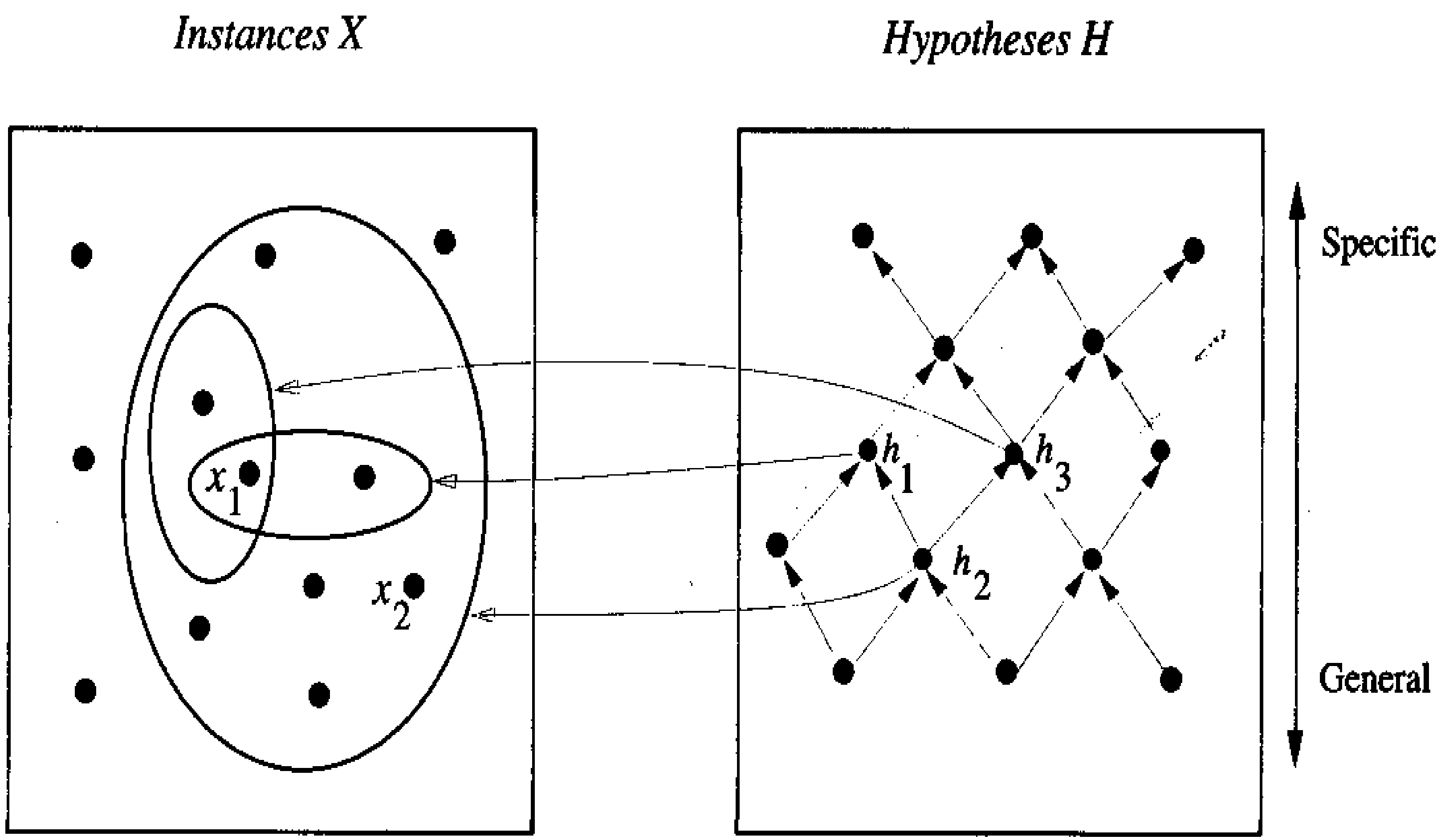
- Many algorithms for concept learning organize the search through the hypothesis space by relying on a very useful structure that exists for any concept learning problem: a general-to-specific ordering of hypotheses.
- To design learning algorithms that exhaustively search infinite hypothesis spaces without explicitly enumerating every hypothesis.

- h_1 = <Sunny, ?, ?, Strong, ?, ?>
- h_2 = <Sunny, ?, ?, ?, ?, ?>

- First, for any instance **x** in **X** and hypothesis **h in H**, we say that **x** satisfies **h**
  - if and only if **h(x) = 1**

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 $h_j \geq_g h_k$) if and only if

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



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, ?*>

- we will say that h_j is (strictly) more-general than $\boldsymbol{h_k}$ (written $\boldsymbol{h_j >_g h_k}$)
  - if and only if $\boldsymbol{(h_j \geq_g h_k) \wedge (h_k \,!\!\geq_g h_j)}$

- sometimes find the inverse useful and will say that $\boldsymbol{h_j}$ is ***more specific than*** $\boldsymbol{h_k}$ when $\boldsymbol{h_k}$ is ***more-general-than*** $\boldsymbol{h_j}$

Definition 4 (More General Relation)

Let X be a feature space and let h_1 and h_2 be two boolean-valued functions with domain X. Then function h_1 is called more general than function h_2, denoted as $h_1 \geq_g h_2$, iff:

$$\forall \mathbf{x} \in X : \; (h_2(\mathbf{x}) = 1 \; \text{implies} \; h_1(\mathbf{x}) = 1)$$

h_1 is called stricly more general than h_2, denoted as $h_1 >_g h_2$, iff:

$$(h_1 \geq_g h_2) \; \text{and} \; (h_2 \not\geq_g h_1)$$

About the maximally specific / general hypothesis:

- s_0 is minimum and g_0 is maximum with regard to $\geq_g$: no hypothesis is more specific wrt. s_0, and no hypothesis is more general wrt. g_0.
- We will consider only hypothesis spaces that contain s_0 and g_0.

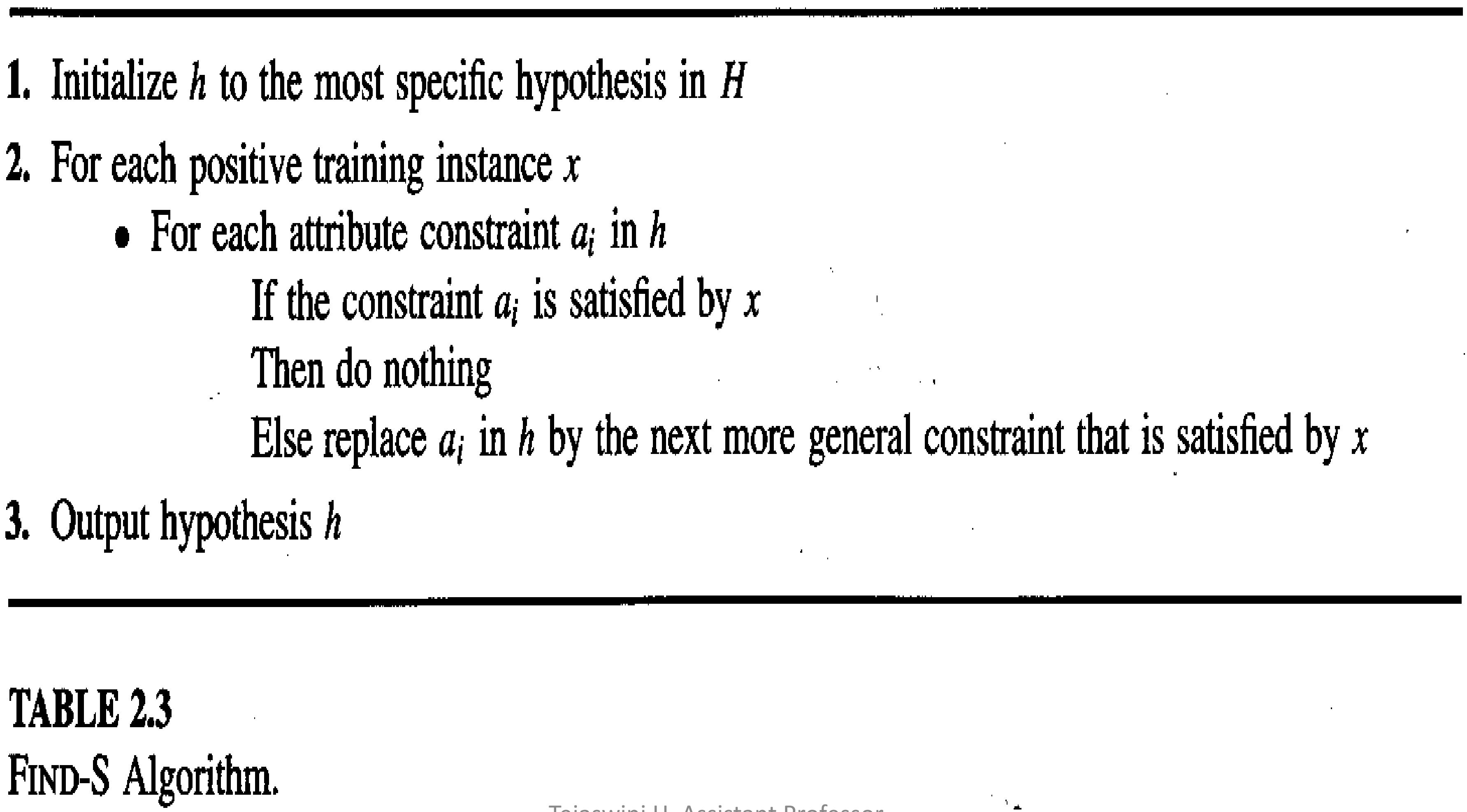
Point to ponder ……

- How can we use the *more-general-than* partial ordering to organize the search for a hypothesis consistent with the observed training examples?
  - One way is to **begin with the most specific possible hypothesis** in H, **then generalize** this hypothesis each time it fails to cover an observed positive training example.
- We say that a hypothesis "**covers**" a positive example if it correctly classifies the example as positive.

Approaches to concept learning task:

- 1. FIND-S algorithm
- 2. Candidate-elimination algorithm

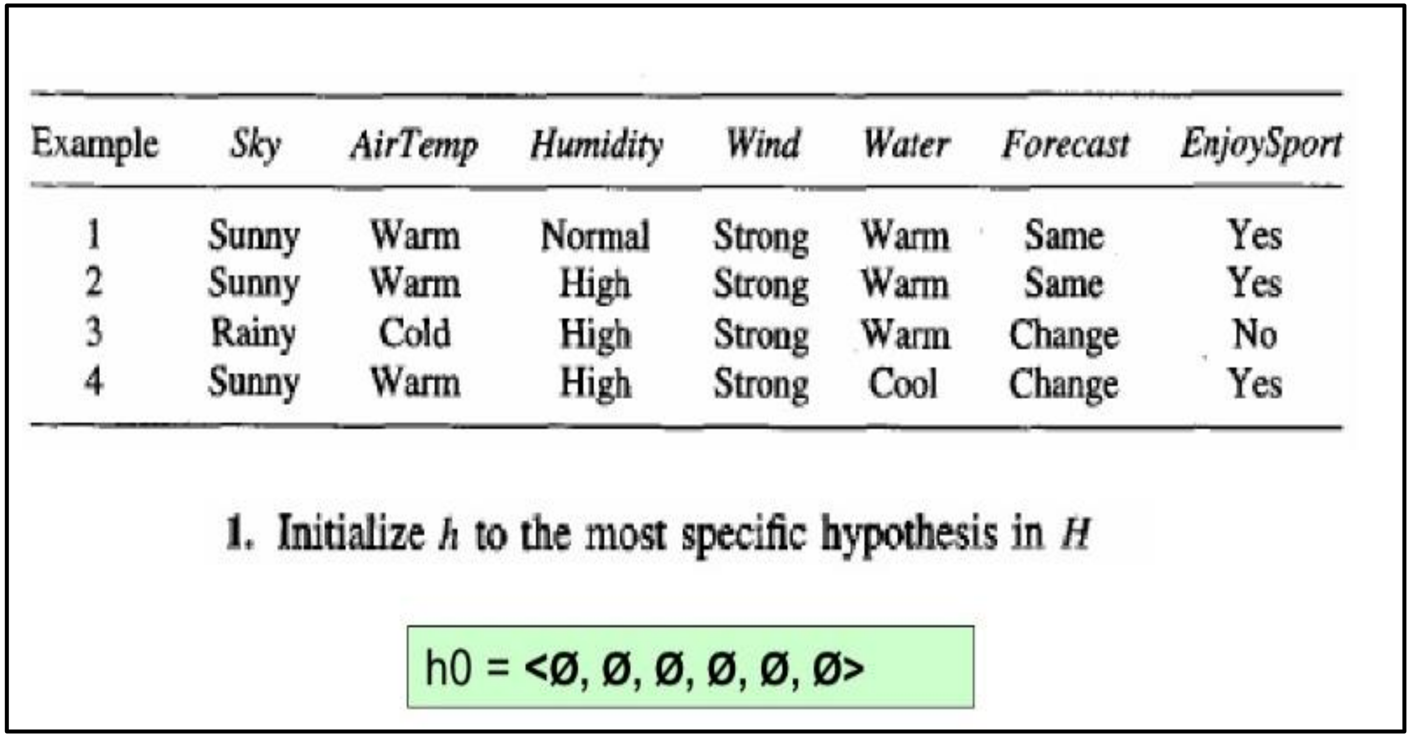
FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS



```
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 $h$
```

TABLE 2.3
FIND-S Algorithm.

Step1: Find S



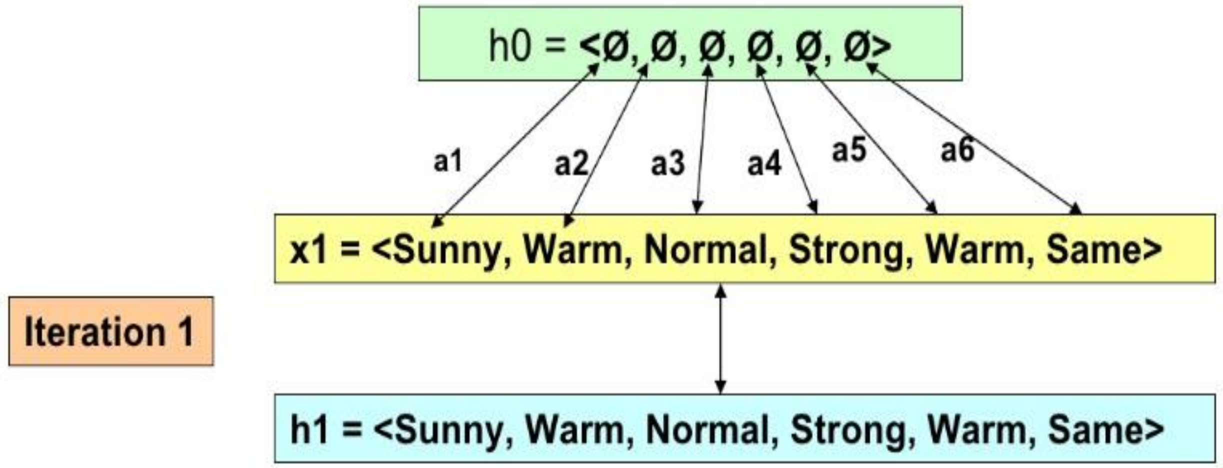
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

1. Initialize *h* **to the most specific hypothesis in** *H*

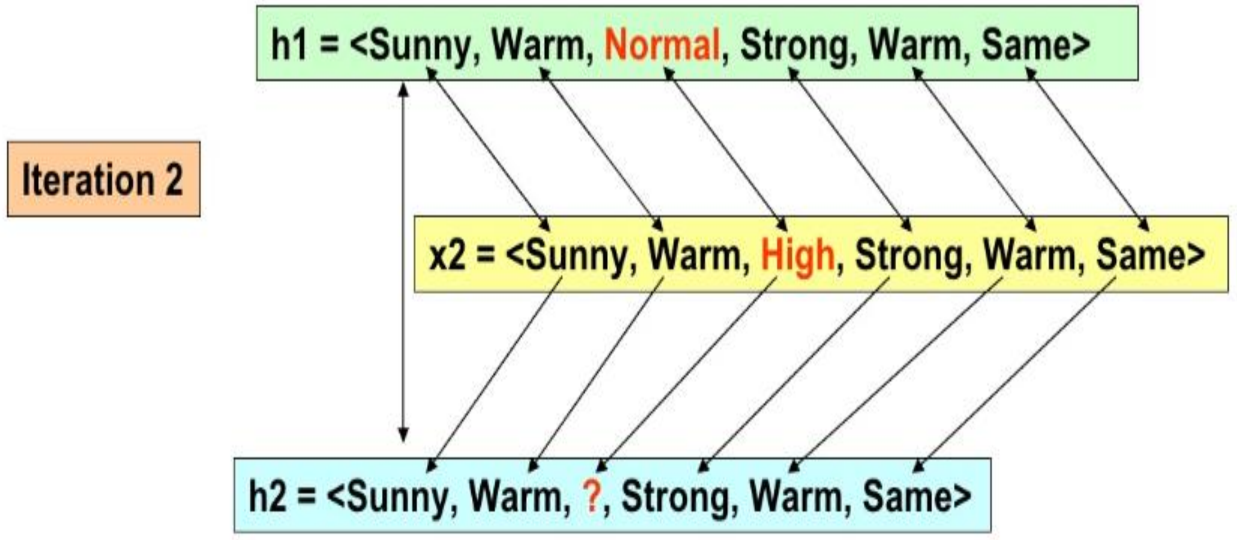
h0 = <Ø, Ø, Ø, Ø, Ø, Ø>

Step2 : Find S

```
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 $x$
```

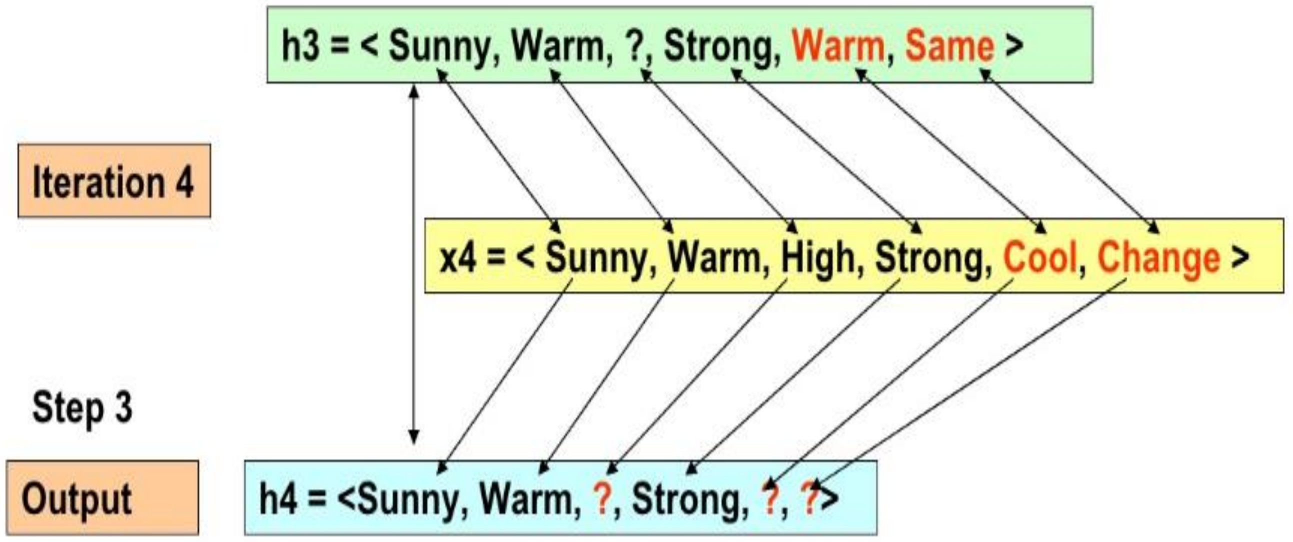

h0 = <Ø, Ø, Ø, Ø, Ø, Ø>
a1
a2
a3
a4
a5
a6
x1 = <Sunny, Warm, Normal, Strong, Warm, Same>
Iteration 1
h1 = <Sunny, Warm, Normal, Strong, Warm, Same>

Step2 : Find S

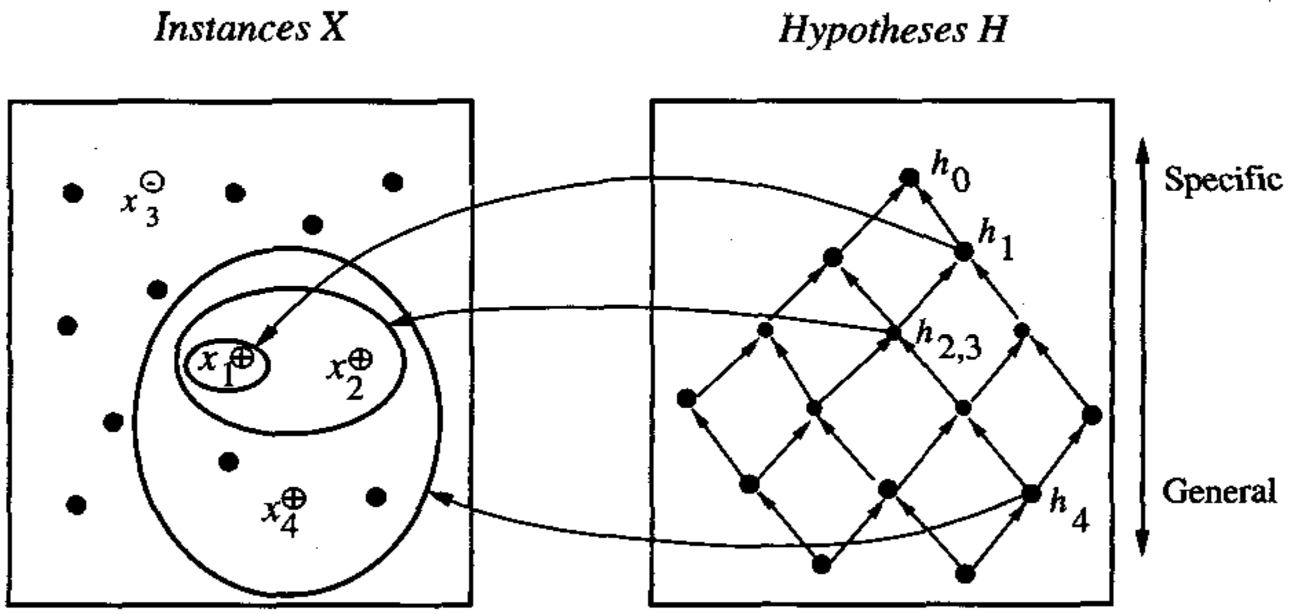

Iteration 2
h1 = <Sunny, Warm, Normal, Strong, Warm, Same>
x2 = <Sunny, Warm, High, Strong, Warm, Same>
h2 = <Sunny, Warm, ?, Strong, Warm, Same>

Iteration 3 | Ignore | h3 = <Sunny, Warm, ?, Strong, Warm, Same>

Iteration 4 and Step 3 : Find S


h3 = < Sunny, Warm, ?, Strong, Warm, Same >
Iteration 4
x4 = < Sunny, Warm, High, Strong, Cool, Change >
Step 3
Output
h4 = <Sunny, Warm, ?, Strong, ?, ?>

Hypothesis Space Search by Find-S



	h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
x_1 = <*Sunny Warm Normal Strong Warm Same*>, +	h_1 = <*Sunny Warm Normal Strong Warm Same*>
x_2 = <*Sunny Warm High Strong Warm Same*>, +	h_2 = <*Sunny Warm ? Strong Warm Same*>
x_3 = <*Rainy Cold High Strong Warm Change*>, -	h_3 = <*Sunny Warm ? Strong Warm Same*>
x_4 = <*Sunny Warm High Strong Cool Change*>, +	h_4 = <*Sunny Warm ? Strong ? ?* >

Issues/Questions!!?!!

- Has the learner converged to the correct target concept?
  - Although FIND-S will find a hypothesis consistent with the training data, it has no way to determine whether it has found the ***only*** hypothesis in ***H*** consistent with the data (i.e., the correct target concept), or whether there are many other consistent hypotheses as well.

- We would prefer a learning algorithm that could determine whether it had converged and, if not, at least characterize its uncertainty regarding the true identity of the target concept.

- Why prefer the most specific hypothesis? In case there are multiple hypotheses consistent with the training examples, FIND-S will find the most specific.
  - It is unclear whether we should prefer this hypothesis over, say, the most general, or some other hypothesis of intermediate generality.

- Are the training examples consistent? In most practical learning problems there is some chance that the training examples will contain at least some errors or noise.
  - Such inconsistent sets of training examples can severely mislead FIND-S, given the fact that it ignores negative examples.
  - We would prefer an algorithm that could at least detect when the training data is inconsistent and, preferably, accommodate such errors.

- What if there are several maximally specific consistent hypotheses?
  - In the hypothesis language H for the ***EnjoySport*** task, there is always a unique, most specific hypothesis consistent with any set of positive examples.
  - However, for other hypothesis spaces there can be several maximally specific hypotheses consistent with the data.

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

- Although FIND-S outputs a hypothesis from H, that is consistent with the training examples, this is just one of many hypotheses from H that might fit the training data equally well.
- The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of ***all hypotheses consistent with the training examples.***

- The CANDIDATE-ELIMINATION algorithm finds all describable hypotheses that are consistent with the observed training examples.

Basic definitions 1

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.

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

consistent v/s ***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.
- However, whether such an example is ***consistent*** with ***h*** depends on the target concept, and in particular, whether ***h(x)*** = ***c(x).***

- The CANDIDATE-ELIMINATION algorithm represents the set of *all* hypotheses consistent with the observed training examples.
- This subset of all hypotheses is called the ***version space*** with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.

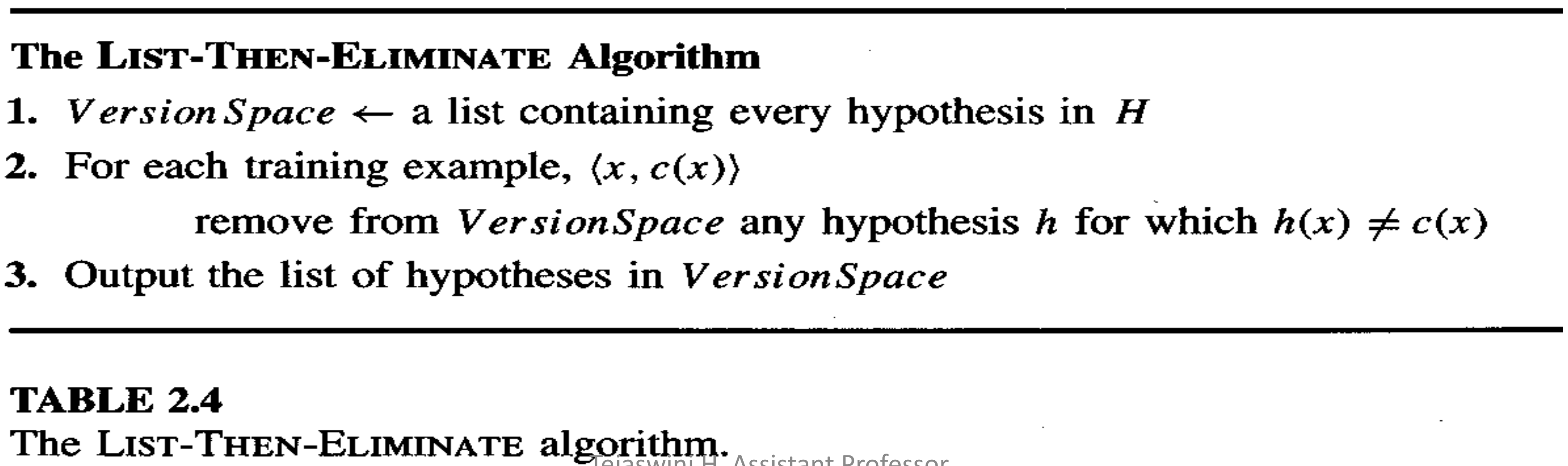
Basic Definition 2

***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.

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

Version Space:
LIST-THEN-ELIMINATION algorithm

- One obvious way to represent the version space is simply to list all of its members.
- This leads to a simple learning algorithm, which we might call the LIST-THEN-ELIMINATION algorithm



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 $VersionSpace$
```

TABLE 2.4
The LIST-THEN-ELIMINATE algorithm.

Basic definition 3, 4

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.

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

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.

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

Theorem 2.1. Version space representation 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 $\{\langle x, c(x)\rangle\}$. For all X, H, c, and D such that S and G are well defined,

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

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
```

TABLE 2.5
CANDIDATE-ELIMINATION algorithm using version spaces. Notice the duality in how positive and negative examples influence S and G.

An Illustrative Example

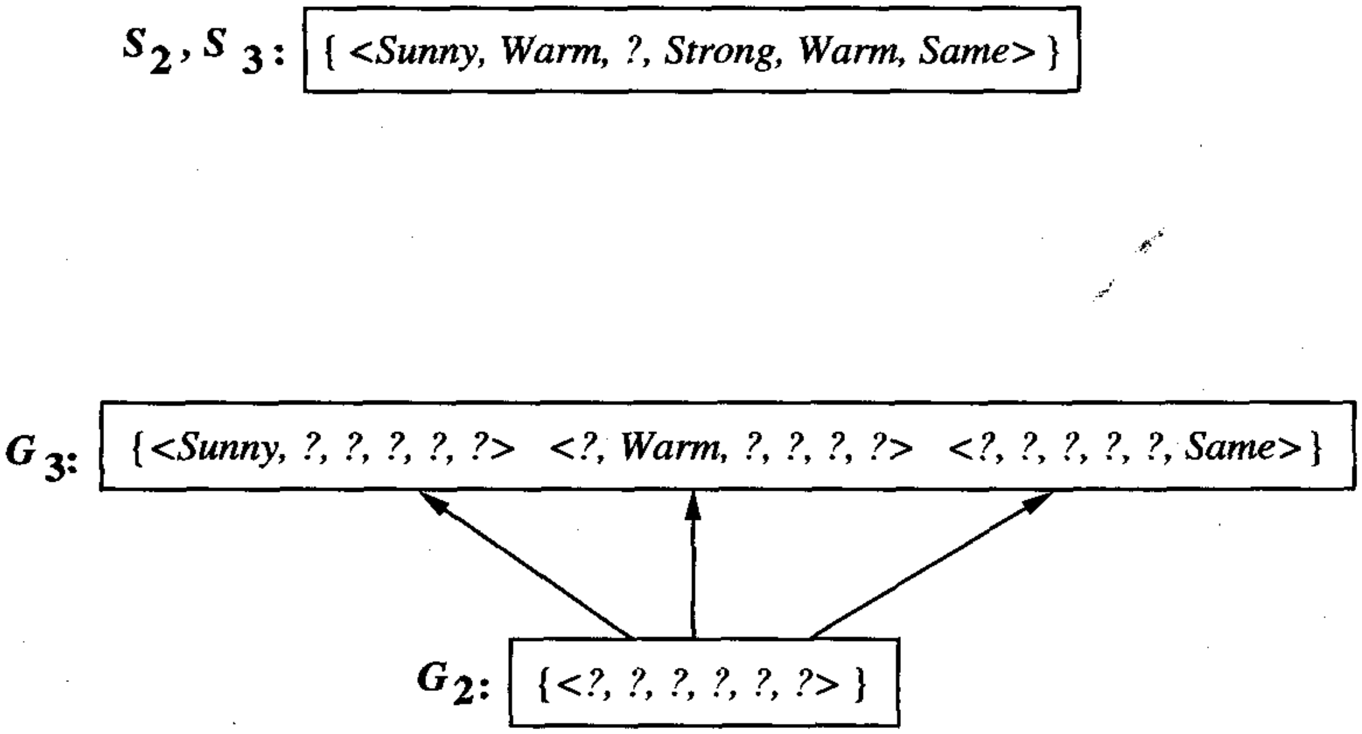
$$\begin{tikzcd}
S_0: \boxed{\{<\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing>\}} \arrow[d] \\
S_1: \boxed{\{<\mathit{Sunny}, \mathit{Warm}, \mathit{Normal}, \mathit{Strong}, \mathit{Warm}, \mathit{Same}>\}} \arrow[d] \\
S_2: \boxed{\{<\mathit{Sunny}, \mathit{Warm}, ?, \mathit{Strong}, \mathit{Warm}, \mathit{Same}>\}}
\end{tikzcd}$$

$$G_0, G_1, G_2: \boxed{\{<?, ?, ?, ?, ?, ?>\}}$$

Training examples:

1 . *<Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Yes*

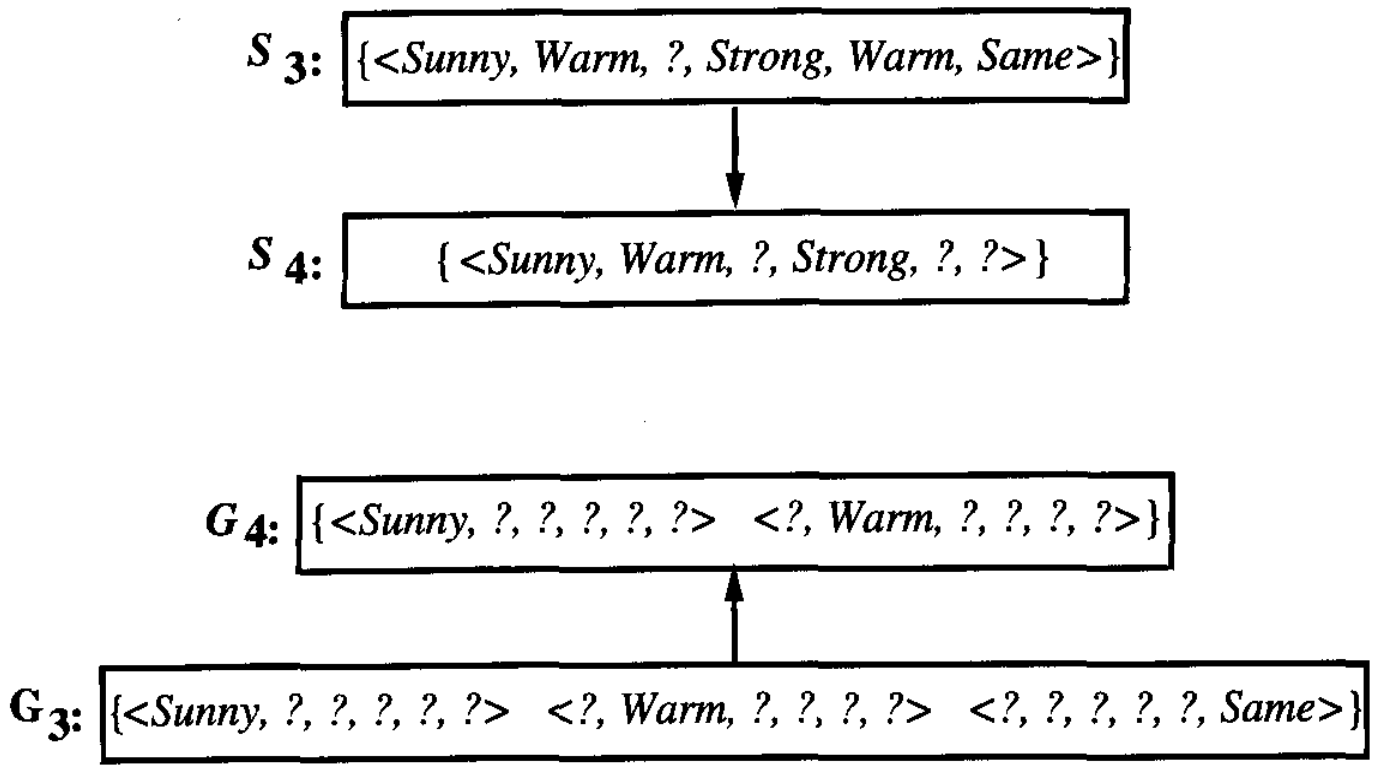
2 . *<Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes*



Training Example:

3. <*Rainy, Cold, High, Strong, Warm, Change*>, *EnjoySport=No*

FIGURE 2.5
CANDIDATE-ELIMINATION Trace 2. Training example 3 is a negative example that forces the G_2 boundary to be specialized to G_3. Note several alternative maximally general hypotheses are included in G_3.



Training Example:

4. <*Sunny, Warm, High, Strong, Cool, Change*>, *EnjoySport* = *Yes*

FIGURE 2.6
CANDIDATE-ELIMINATION Trace 3. The positive training example generalizes the S boundary, from S_3 to S_4. One member of G_3 must also be deleted, because it is no longer more general than the S_4 boundary.

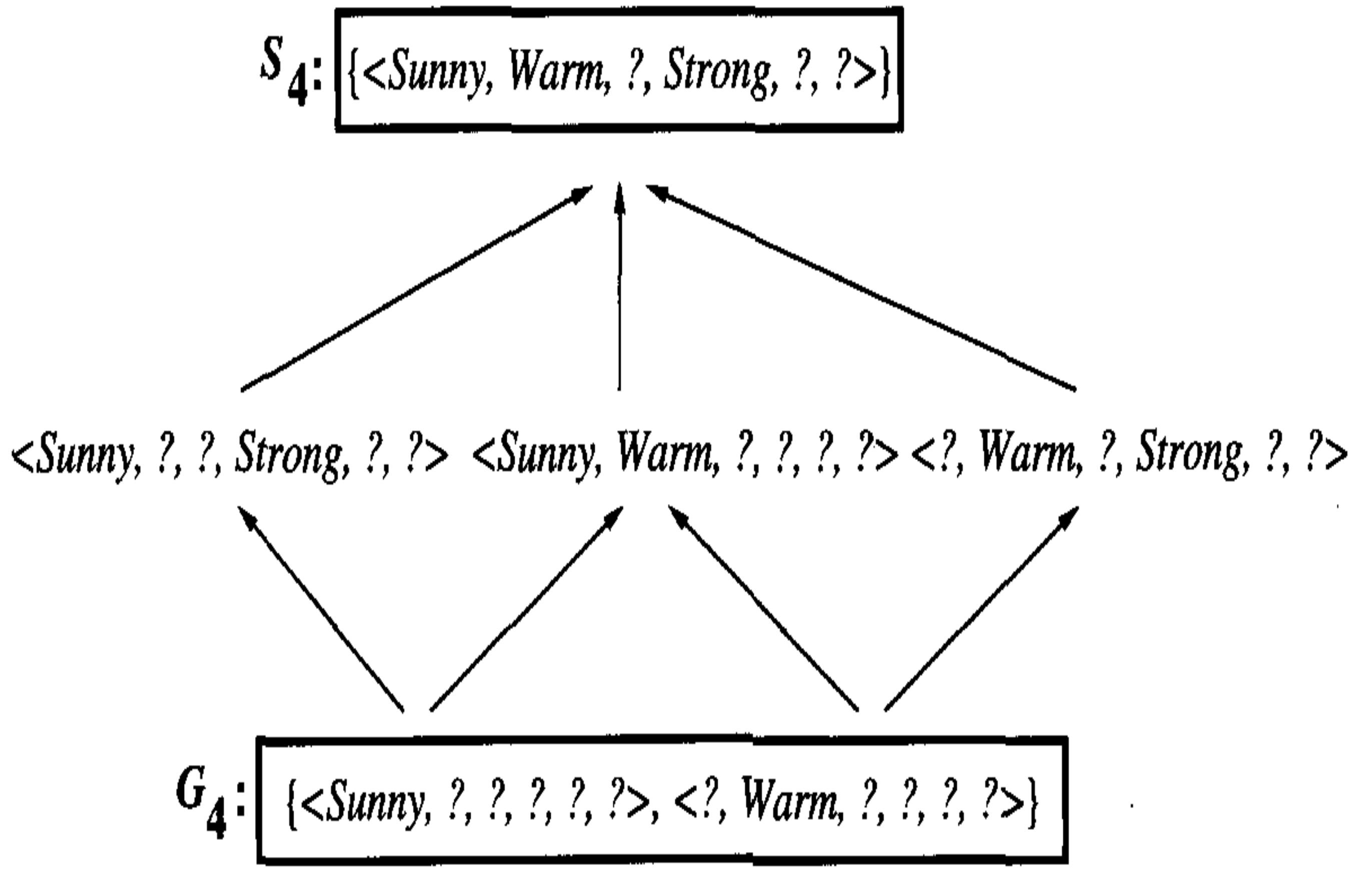
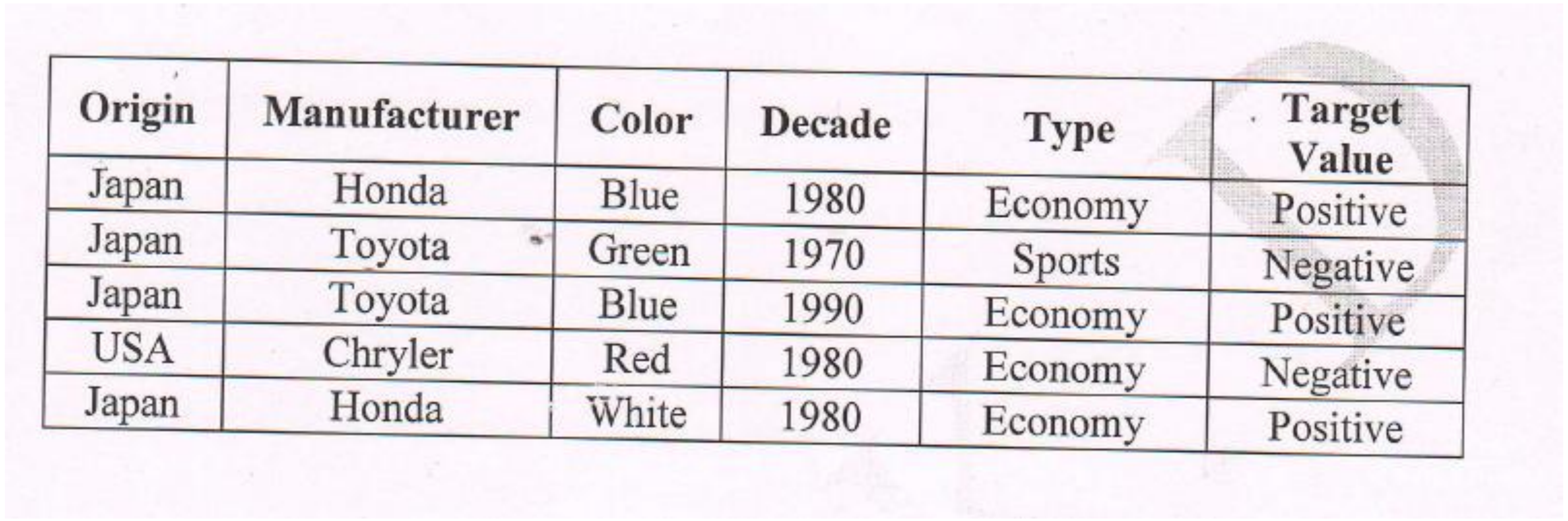


FIGURE 2.7
The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

Example 2



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

$G_0 = \{ <?, \quad ?, \quad ?, \quad ?, \quad ? > \}$

$S_0 = \{ <\phi, \quad \phi, \quad \phi, \quad \phi, \quad \phi > \}$

x_1 : <Japan, Honda, Blue, 1980, Economy> , Positive

Positive Example

G_1 = { <?, ?, ?, ?, ? > }
S_1 = { <Japan, Honda, Blue, 1980, Economy > }

x_2 : <Japan, Toyota, Green, 1970, Sports>, Negative

Negative Example

G_2 = {	<?,	Honda,	?,	?,	? >,
	<?,	?,	Blue,	?,	? >,
	<?,	?,	?,	1980,	? >,
	<?,	?,	?,	?,	Economy >}
S_2= {	<Japan,	Honda ,	Blue,	1980,	Economy >}

x_3 : <Japan, Toyota, Blue, 1990, Economy>, Positive

Positive Example

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

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

x_4 : <USA, Chryler, Red, 1980, Economy>, Negative

G_4 = { <?, ?, Blue, ?, ? >,

<Japan, ?, ?, ?, Economy >}

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

x_5 : <Japan, Honda, White, 1980, Economy> , Positive

G_5={ <Japan, ?, ?, ?, Economy >}

S_5={ <Japan, ?, ?, ?, Economy >}

Example 3:

SIZE	COLOR	SHAPE	CLASS
SMALL	BLUE	CIRCLE	POSITIVE
BIG	RED	CIRCLE	NEGATIVE
SMALL	RED	TRIANGLE	NEGATIVE
SMALL	RED	CIRCLE	POSITIVE
BIG	BLUE	CIRCLE	NEGATIVE

$G_0 = \{ <?, \quad ?, \quad ?, \quad ?, \quad ? > \}$

$S_0 = \{ <\phi, \ \phi, \quad \phi, \quad \phi, \quad \phi > \}$

x_1 : <small, blue, circle> , Positive

Positive Example

G_1 = { <?, ?, ? > }

S_1 = {<small, blue, circle> }

x_2 : <big, red, circle>, Negative

Negative Example

G_2 = { <small, ?, ?>,
<?, blue, ?>}

S_2= { <small, blue, circle>}

x_3 : <small, red, triangle>, Negative

Negative Example

G_3 = { <small, ?, circle>,
<?, blue, ?>}

S_3= { <small, blue, circle>}

x_4 : <small, red, circle>, Positive

Positive Example

G_4 = { <small, ?, circle> }

S_4 = { <small, ?, circle> }

x_5 : <big, blue, circle>, Negative

Negative Example

G_5= { <small, ?, circle>}

S_5= { <small, ?, circle>}

2.7 INDUCTIVE BIAS

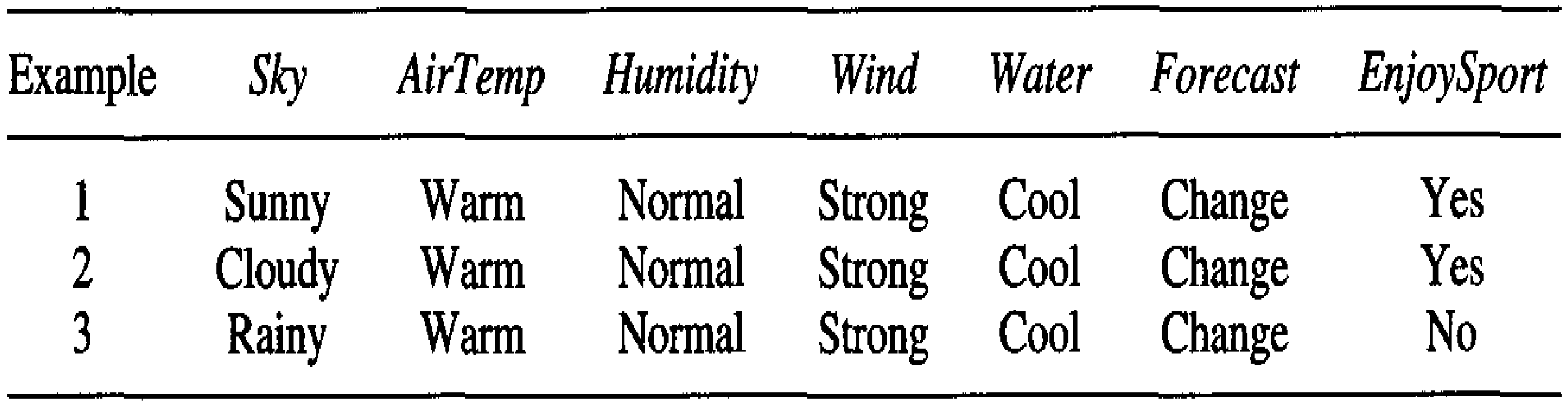
- **CANDIDATE-ELIMINATION** algorithm will converge toward the true target concept provided it is given **accurate training examples** and provided its **initial hypothesis space contains the target concept**.

- 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?
- **These are fundamental questions for inductive inference in general.**

- **A Biased Hypothesis Space**
- **An Unbiased Learner**
- **The Futility of Bias-Free Learning**

A Biased Hypothesis Space



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

- There are no hypotheses consistent with these three examples,
  - note that the most specific hypothesis consistent with the first two examples ***and representable in the given hypothesis space H*** is

    S2 : ***<?, Warm, Normal, Strong, Cool, Change>***

- This hypothesis, although it is the maximally specific hypothesis from H that is consistent with the first two examples, is **already overly general**:
  - it incorrectly covers the third (negative) training example.

- The problem is that we have biased the learner to consider only conjunctive hypotheses.
  - we restricted the hypothesis space to include only conjunctions of attribute values.
- Because of this restriction, the hypothesis space is unable to represent even simple disjunctive target concepts such as ***"Sky*** = ***Sunny*** or ***Sky*** = ***Cloudy."***

- In fact, given the following three training examples of this disjunctive hypothesis, our algorithm would find that there are zero hypotheses in the version space.
- In this case we require a more expressive hypothesis space.

An Unbiased Learner

- The obvious solution to the problem of assuring that the target concept is in the hypothesis space H is:
  - to provide a hypothesis space capable of representing ***every teachable concept;***
  - that is, **it is capable of representing every possible subset of the instances X**.

- In general, the set of all subsets of a set X is called *the* ***powerset*** of X.

- In the ***EnjoySport*** learning task,
  - Number of attributes : 6
  - the size of the instance space X of days described by the six available attributes is |X|: 3*2*2*2*2*2 = 96.

- In general, 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 $2^{|X|}$

- Thus, there are $2^{|96|}$ distinct target concepts that could be defined over this instance space and that our learner might be called upon to learn.

Let us reformulate….

- 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

  (Sunny, ?, ?, ?, ?, ?) v (Cloudy, ?, ?, ?, ?, ?)

- Given this hypothesis space, we can safely use the CANDIDATE-ELIMINATION algorithm without worrying that the target concept might not be expressible.
- However, it unfortunately raises a new, equally difficult problem:
  - our concept learning algorithm is now completely unable to generalize beyond the observed examples

- Suppose we present three positive examples **(x1, x2, x3)** and two negative examples **(x4, x5)** to the learner.
- At this point, the ***S*** boundary of the version space will contain the hypothesis which is just the disjunction of the positive examples

$$S : \{(x_1 \vee x_2 \vee x_3)\}$$

- Similarly, the G boundary will consist of the hypothesis that rules out only the observed negative examples

$$G : \{\neg(x_4 \vee x_5)\}$$

- Therefore, the only examples that will be unambiguously classified by ***S*** and G are the observed training examples themselves.
- In order to converge to a single, final target concept, we will have to present every single instance in X as a training example

- Unfortunately, the only instances that will produce a unanimous vote are the previously observed training examples.
- For, all the other instances, taking a vote will be futile: each unobserved instance will be classified positive by ***precisely*** half the hypotheses in the version space and will be classified negative by the other half.

The Futility of Bias-Free Learning

- fundamental property of inductive inference:
  - ***a learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.***

- In fact, the only reason that the CANDIDATE-ELIMINATION was able to generalize beyond the observed training examples in our original formulation of the *EnjoySport* task is that it was biased by the implicit assumption that the target concept could be represented by a conjunction of attribute values.

- In cases where this assumption is correct (and the training examples are error-free), its classification of new instances will also be correct.
- If this assumption is incorrect, however, it is certain that the CANDIDATE-ELIMINATION will misclassify at least some instances from X.

- Because inductive learning requires some form of prior assumptions, or inductive bias, we will find it useful to characterize different learning approaches by the inductive bias they employ

Need of generalizing beyond observed data…

- consider the general setting in which an arbitrary learning algorithm **L** is provided an arbitrary set of training data $\mathbf{D_c = \{(x, c(x))\}}$ of some arbitrary target concept **c**.

- After training, L is asked to classify a new instance x_i.
- Let $L(x_i, D_c)$ denote the classification (e.g., positive or negative) that L assigns to x_i after learning from the training data D_c.
- We can describe this inductive inference step performed by L as follows:

$$(D_c \wedge x_i) \succ L(x_i, D_c)$$

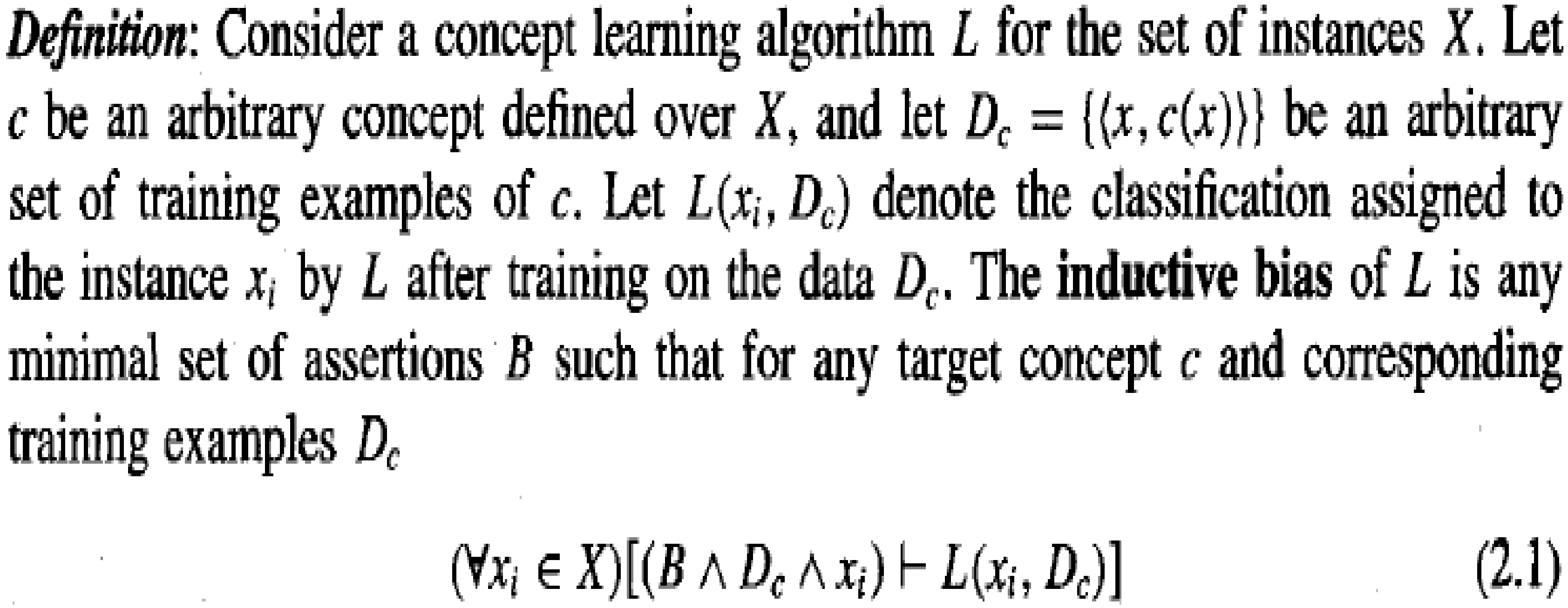
- For example, if we take L to be the CANDIDATE-ELIMINATION, D_c, to be the training data from Table 2.1, and x_i to be the fist instance from Table 2.6, then the inductive inference performed in this case concludes that $L(x_i, D_c)$ = *(EnjoySport = yes).*

- it is interesting to ask what additional assumptions could be added to $\mathbf{D_c \wedge x_i}$ so that $L(x_i, D_c)$ would follow deductively.
- We define the inductive bias of ***L*** as this set of additional assumptions.

- More precisely, we define the inductive bias of ***L*** to be the set of assumptions ***B*** such that for all new instances $\boldsymbol{x_i}$

$$(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)$$

- Thus, we define the inductive bias of a learner as the set of additional assumptions B sufficient to justify its inductive inferences as deductive inferences.



Definition: Consider a concept learning algorithm L for the set of instances X. Let c be an arbitrary concept defined over X, and let $D_c = \{\langle x, c(x)\rangle\}$ 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

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)] \qquad (2.1)$$

- **Inductive bias of CANDIDATE-ELIMINATION Algorithm:** The target concept c is contained in the given hypothesis space H.

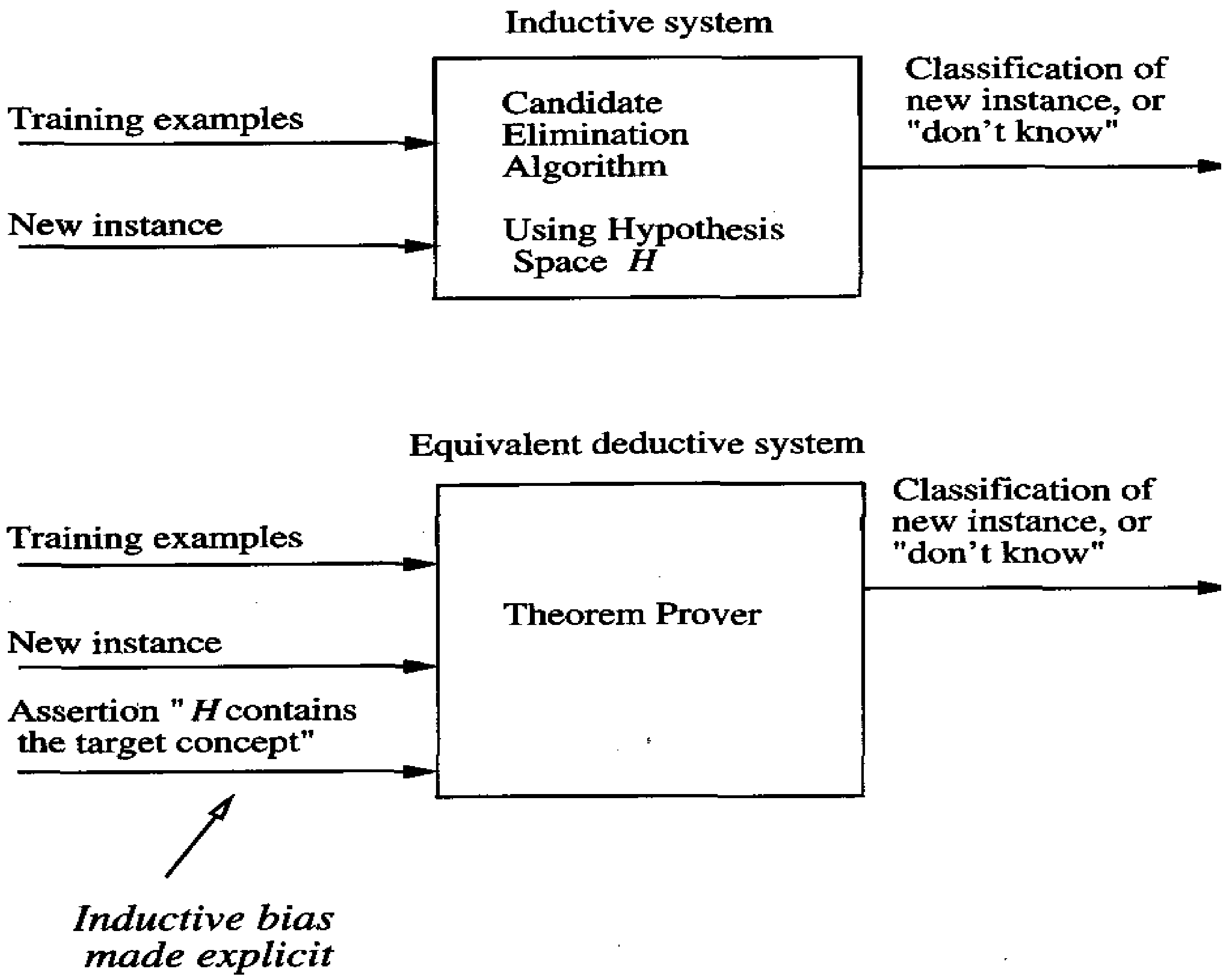

Inductive system
Training examples
New instance
Candidate Elimination Algorithm
Using Hypothesis Space H
Classification of new instance, or "don't know"
Equivalent deductive system
Training examples
New instance
Assertion " H contains the target concept"
Theorem Prover
Classification of new instance, or "don't know"
Inductive bias made explicit

FIGURE 2.8
Modeling inductive systems by equivalent deductive systems.

- One advantage of viewing inductive inference systems in terms of their inductive bias is that it provides a nonprocedural means of characterizing their policy for generalizing beyond the observed data.
- A second advantage is that it allows comparison of different learners according to the strength of the inductive bias they employ.

1. **ROTE-LEARNER:** Learning corresponds simply to storing each observed training example in memory. Subsequent instances are classified by looking them up in memory. If the instance is found in memory, the stored classification is returned. Otherwise, the system refuses to classify the new instance.
2. **CANDIDATE-ELIMINATION algorithm:** New instances are classified only in the case where all members of the current version space agree on the classification. Otherwise, the system refuses to classify the new instance.
3. **FIND-S:** This algorithm, described earlier, finds the most specific hypothesis consistent with the training examples. It then uses this hypothesis to classify all subsequent instances.