



**VIVEKANANDA**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
Nehrunagar post, Puttur, D.K. 574203



**Lecture Notes**  
**on**

 **MACHINE  
LEARNING**  
**BY HARIVINOD N**

**Subject Code: 15CS73**  
**(CBCS Scheme)**

**Prepared by**

**Mr. Harivinod N**

Dept. of Computer Science and Engineering,  
VCET Puttur

**Module-1**  
**Introduction to Machine Learning**

**Course website:**  
**[www.techjourney.in](http://www.techjourney.in)**



## Module-1

### Introduction to Machine Learning

#### 1.1 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 or personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper. This course presents the field of machine learning, describing a variety of learning paradigms, algorithms, theoretical results, and applications.

Some successful applications of machine learning are,

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

Examples of supervised machine learning tasks include:

- Identifying the zip code from handwritten digits on an envelope
- Determining whether a tumor is benign based on a medical image
- Detecting fraudulent activity in credit card transactions
- Identifying topics in a set of blog posts
- Segmenting customers into groups with similar preferences
- Detecting abnormal access patterns to a website

#### 1.2 Well posed learning problems

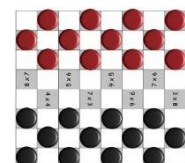
Learning is broadly defined as any computer program that improves its performance at some task through experience.

**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  $E$ .

For example, a computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself. In general, to have a well-defined learning problem, we must identify these three features: the class of tasks, the measure of performance to be improved, and the source of experience.

##### 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





We can specify many learning problems in this fashion, such as learning to recognize handwritten words, or learning to drive a robotic automobile autonomously.

### 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

Winter is here. Go to the store and buy some snow shovels.

### A robot driving learning problem

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

## 1.3 Designing a Learning system

In order to illustrate some of the basic design issues and approaches to machine learning, let us 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.

### 1.3.1 Choosing the Training Experience

The type of training experience available can have a significant impact on success or failure of the learner.

- One key attribute is 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. 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. Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

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

For example, the learner might rely on the teacher to select informative board states and to provide the correct move for each.

Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move. Or 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.



- A third important attribute of the training experience is **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 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 evaluated

To proceed with our design, let us decide that our system will train by playing games against itself. This has the advantage that no external trainer need be present, and it therefore allows the system to generate as much training data as time permits. We now have a fully specified learning task.

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

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

### 1.3.2. Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program. Consider checkers-playing program. The program needs only to learn how to choose the best move from among some large search space are known a priori. Here we discuss two such methods.

- **Method-1:** Let us use the function *ChooseMove*:  $B \rightarrow M$  to indicate that accepts any board from the set of legal board states  $B$  as input and produces as output some move from the set of legal moves  $M$ .

The choice of the target function *ChooseMove* is a key design choice.

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.

- **Method-2:** An alternative target function and one that will turn out to be easier to learn in this setting is an evaluation function that assigns a numerical score to any given board state.

Let us call this target function  $V$  and again use the notation  $V: B \rightarrow \mathcal{R}$  to denote that  $V$  maps any legal board state from the set  $B$  to some real value in  $\mathcal{R}$ . 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.

For example, 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 (assuming the opponent plays optimally, as well).

While this recursive definition specifies a value of  $V(b)$  for every board state  $b$ , this definition is not usable by our checkers player because it is not efficiently computable.

The goal of learning in this case is to discover an operational description of  $V$ ; i.e. 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  $V$ . In practice, implementation of learning the target function is often called function approximation. 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  $V$ .

### 1.3.3 Choosing a Representation for the Target Function

We have several ways to represent  $\hat{V}$  like; using a large table with a distinct entry specifying the value for each distinct board state or using a collection of rules that match against features of the board state, or a quadratic polynomial function of predefined board features, or an artificial neural network.

On the other hand, the more expressive the representation, the more training data the program will require in order to choose among the alternative hypotheses it can represent. To keep the discussion brief, let us choose a simple representation: for any given board state, the function  $c$  will be calculated as a linear combination of the following board features:

- $x_1$ : 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  $w_0$  through  $w_6$  are numerical coefficients, or weights, to be chosen by the learning algorithm.

### Partial design of a checkers learning program:

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

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

### 1.3.4 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  $\langle b, V_{train}(b) \rangle$ . For instance, the following training example describes a board state  $b$  in which black has won the game (note  $x_2 = 0$  indicates that red has no remaining pieces) and for which the target function value  $V_{train}(b)$  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  $w_i$  to best fit these training examples

**Estimating Training Values:** Recall that according to our formulation of the learning problem, the only training information available to our learner is whether the game was eventually **won or lost**.

Even if the program loses the game, it may still be the case that board states occurring early in the game should be rated very highly and that the cause of the loss was a subsequent poor move.

Despite the ambiguity inherent in estimating training values for intermediate board states, one **simple approach** has been found to be surprisingly successful. This approach is to assign the training value of  $V_{train}(b)$  for any intermediate board state  $b$  to be  $\hat{V}(\text{Successor}(b))$  where  $\hat{V}$  is the learner's current approximation to  $V$  and where  $\text{Successor}(b)$  denotes the next board state following  $b$  for which it is again the program's turn to move (i.e., the board state following the program's move and the opponent's response). This rule for estimating training values can be summarized as

$$\text{Rule for estimating training values: } V_{train}(b) \leftarrow \hat{V}(\text{Successor}(b))$$

While it may seem strange to use the current version of  $\hat{V}$  to estimate training values that will be used to refine this very same function, notice that we are using estimates of the value of the  $\text{Successor}(b)$  to estimate the value of board state  $b$ . Intuitively, we can see this will make sense if  $\hat{V}$  tends to be more accurate for board states closer to game's end.





**Adjusting the weights:** All that remains is to specify the learning algorithm for choosing the weights  $w_i$  to best fit the set of training examples  $\{(b, V_{train}(b))\}$ . As a first step we must define what we mean by the best-fit to the training data. One common approach is to 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  $V$ .

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

Thus, we seek the weights, or equivalently the  $\hat{V}$ , that minimize  $E$  for the observed training examples.

In our case, we require an algorithm that will incrementally refine the weights as new training examples become available and that will be robust to errors in these estimated training values. One such algorithm is called the **least mean squares (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. The LMS algorithm is defined as follows:

#### **LMS Weight update rule**

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

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

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

Here  $\eta$  is a small constant (e.g., 0.1) that moderates the size of the weight update. To get an intuitive understanding for why this weight update rule works, notice that when the error  $(V_{train}(b) - \hat{V}(b))$  is zero, no weights are changed. When  $(V_{train}(b) - \hat{V}(b))$  is positive (i.e., when  $f(b)$  is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of  $\hat{V}(b)$ , reducing the error. Notice that if the value of some feature  $x_i$  is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board. Surprisingly, in certain settings this simple weight-tuning method can be proven to converge to the least squared error approximation to the  $V_{train}$  values.

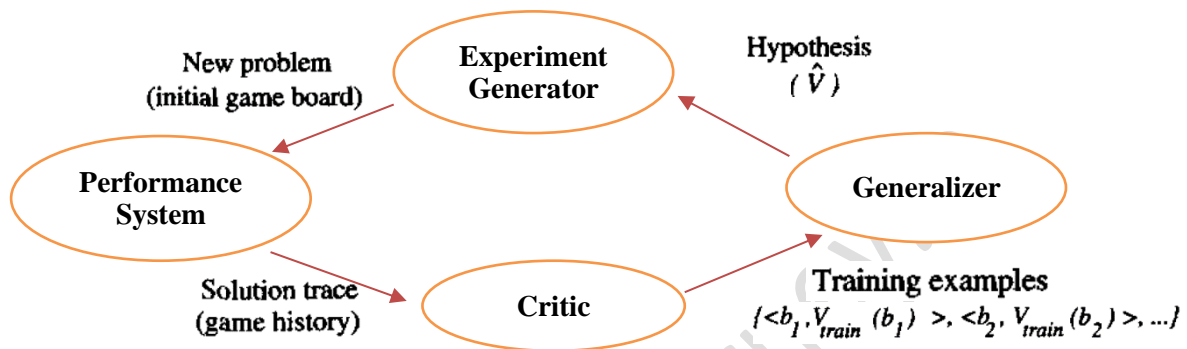
### **1.3.5 The final design**

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.

- a) The **Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.
- b) The **Critic** takes as input - history or trace of the game and produces as output - a set of training examples of the target function.

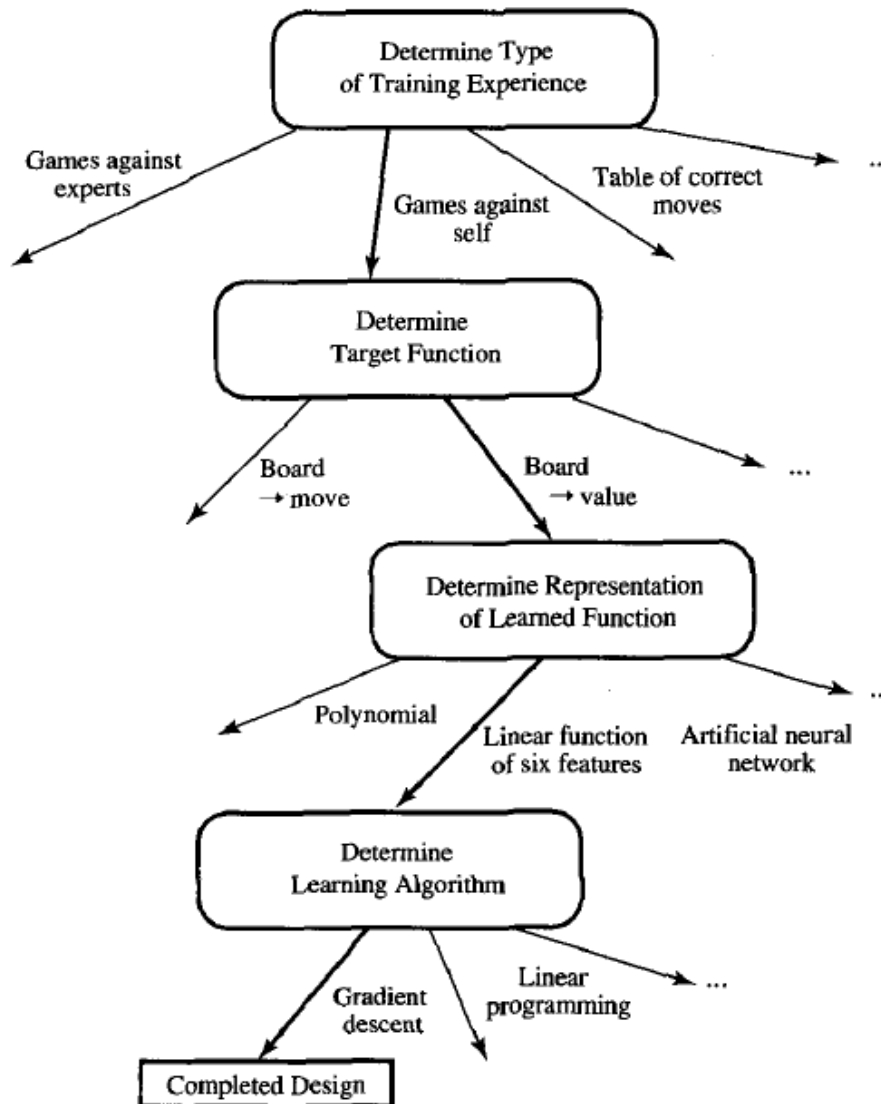
- c) The **Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.
- d) The **Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

These four modules are summarized as follows:



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





**FIGURE 1.2**  
Summary of choices in designing the checkers learning program.

#### 1.4 Perspective and Issues in Machine Learning.

**Perspective in ML** - One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner.

For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights  $w_0$  through  $w_6$ . The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples. The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This algorithm works



well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

**Issues in ML** - Our checker's example raises a number of generic questions about machine learning. The field of machine learning, is concerned with answering questions such as the following:

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?



## 2 Concept Learning

Much of learning involves acquiring general concepts from specific training examples. People, for example, continually learn general concepts or categories such as "bird," "car," etc. Each concept can be viewed as describing some subset of objects/events defined over a larger set.

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.

### 2.1 A Concept learning task

To ground our discussion of concept learning, consider the example task of learning the target concept "**Days on which my friend Sachin enjoys his favorite water sport**". Table given below describes a set of example days, each represented by a set of attributes.

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*.

What hypothesis representation shall we provide to the learner in this case?

For each attribute, the hypothesis will either

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

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

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

(?, Cold, High, ?, ?, ?)

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

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

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

(Φ, Φ, Φ, Φ, Φ, Φ)

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, and the set of available training examples.

### Notation

- The set of items over which the concept is defined is called the **set of instances**, which we denote by  $X$ . In the current example,  $X$  is the set of all possible days, each represented by the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*.
- The concept or function to be learned is called the **target concept**, which we denote by  $c$ . In general,  $c$  can be any boolean valued function defined over the instances  $X$ ; that is,  $c: X \rightarrow \{0, 1\}$ . In the current example, the target concept corresponds to the value of the attribute *EnjoySport* (i.e,  $c(x)=1$  if *EnjoySport*=*Yes*, and  $c(x)=0$  if *EnjoySport*=*No*).
- When learning the target concept, the learner is presented by a set of training examples, each consisting of an instance  $x$  from  $X$ , along with 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. 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  $c(x)$ .
- We use the symbol  $D$  to denote **the set of available training examples**.
- 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  $X$ ; that is,  $h: X \rightarrow \{0, 1\}$ . The goal of the learner is to find a hypothesis  $h$  such that  $h(x) = c(x)$  for all  $x$  in  $X$ .

#### • Given:

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

#### • Determine:

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

The *EnjoySport* concept learning task.



## Inductive learning hypothesis

- 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.

## 2.2 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.**

Consider, for example, the instances  $X$  and hypotheses  $H$  in the *EnjoySport* learning task. Given that the attribute *Sky* has three possible values, and that *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast* each have two possible values, the instance space  $X$  contains exactly  $3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96$  distinct instances. A similar calculation shows that there are  $5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120$  syntactically distinct hypotheses within  $H$  (including  $?$  and  $\Phi$  for each). Most practical learning tasks involve much larger, sometimes infinite, hypothesis spaces.

### 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 illustrate the general-to-specific ordering, consider the two hypotheses

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

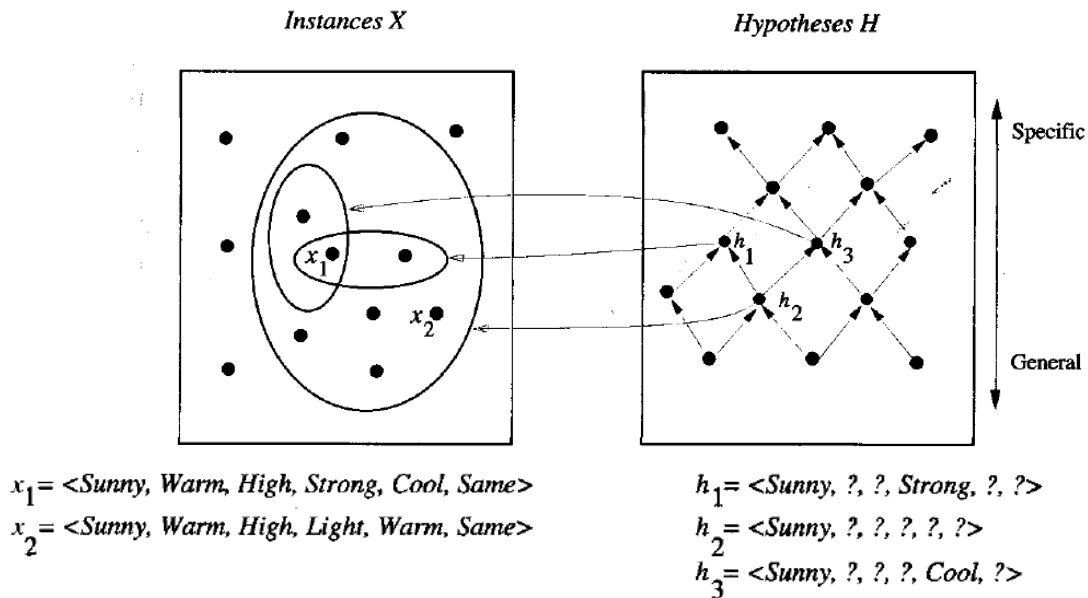
Now consider the sets of instances that are classified positive by  $h_1$  and by  $h_2$ . Because  $h_2$  imposes fewer constraints on the instance, it classifies more instances as positive. In fact, any instance classified positive by  $h_1$  will also be classified positive by  $h_2$ . Therefore, we say that  **$h_2$  is more general than  $h_1$ .**

This intuitive "more general than" relationship between hypotheses can be defined more precisely as follows.

**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

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

We will also find it useful to consider cases where one hypothesis is strictly more general than the other. Therefore, we will say that  $h_j$  is (strictly) **more general than**  $h_k$  (written  $h_j >_g h_k$ ) if and only if  $(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$ . Finally, we will sometimes find the inverse useful and will say that  $h_j$  is **more specific than**  $h_k$  when  $h_k$  is **more general than**  $h_j$ .



**FIGURE 2.1**

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

The  $\geq_g$  relation is important because it provides a useful structure over the hypothesis space  $H$  for *any* concept learning problem. The following sections present concept learning algorithms that take advantage of this partial order to efficiently organize the search for hypotheses that fit the training data.

### 2.3 Find-S: Finding A Maximally Specific Hypothesis

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. FIND-S algorithm is used for this purpose.

#### FIND-S Algorithm.

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

To illustrate this algorithm, assume the learner is given the sequence of training examples from Table 2.1 for the *EnjoySport* task.





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

**TABLE 2.1**

Positive and negative training examples for the target concept *EnjoySport*.

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

$$h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

Upon observing the **first training example** from Table 2.1, which happens to be a positive example, it becomes clear that our hypothesis is too specific. In particular, none of the “ $\Phi$ ” constraints in  $h$  are satisfied by this example, so each is replaced by the next more general constraint that fits the example; namely, the attribute values for this training example.

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

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

Next, the **second training example** (also positive in this case) forces the algorithm to further generalize  $h$ , this time substituting a “?” in place of any attribute value in  $h$  that is not satisfied by the new example. The refined hypothesis is

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

Upon encountering the **third training example**-in this case a negative example-the algorithm makes no change to  $h$ . In fact, the FIND-S algorithm simply ignores every negative example.

The **fourth** (positive) example leads to a further generalization of  $h$

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

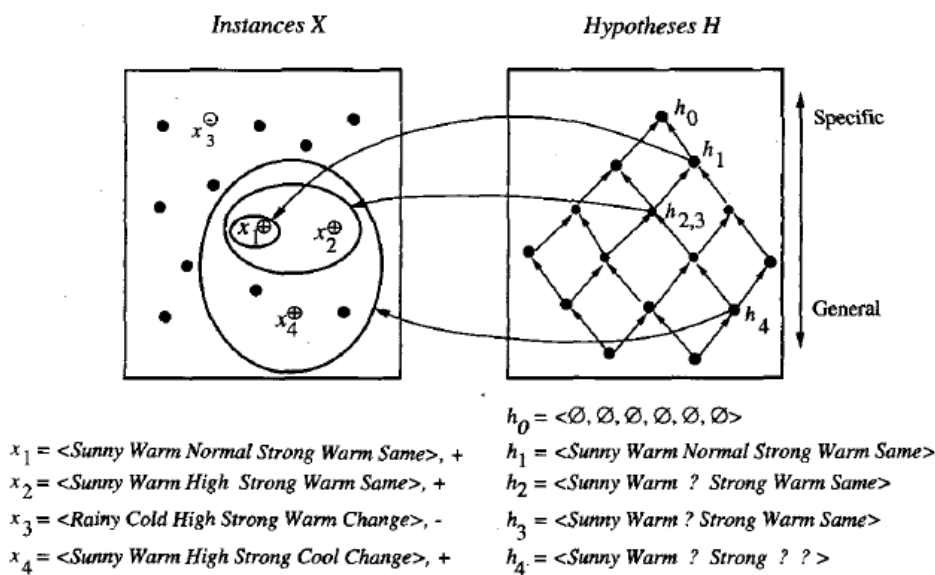
The FIND-S algorithm illustrates one way in which the more-general-than-partial ordering can be used to organize the search for an acceptable hypothesis. The search moves from hypothesis to hypothesis, searching from the most specific to progressively more general hypotheses along one chain of the partial ordering.

Figure 2.2 illustrates this search in terms of the instance and hypothesis spaces.

### Key Property

The key property of the Find-S algorithm is that for hypothesis spaces described by conjunctions of attribute constraints (such as  $H$  for the EnjoySport task). Find-S is guaranteed to output the most specific hypothesis within  $H$  that is consistent with the positive training examples.




**FIGURE 2.2**

The hypothesis space search performed by FIND-S. The search begins ( $h_0$ ) with the most specific hypothesis in  $H$ , then considers increasingly general hypotheses ( $h_1$  through  $h_4$ ) as mandated by the training examples. In the instance space diagram, positive training examples are denoted by “+,” negative by “-,” and instances that have not been presented as training examples are denoted by a solid circle.

However, there are several **questions still left unanswered**, such as:

- **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.
- **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.
- **What if there are several maximally specific consistent hypotheses?** There can be several maximally specific hypotheses consistent with the data. Find S finds only one.

## 2.4 Version Space and Motivation to Candidate Elimination algorithm

Candidate Elimination algorithm (CEA), addresses limitations of FIND-S. It finds all describable hypotheses that are consistent with the observed training examples. In order to define this algorithm precisely, we begin with a few basic definitions.



**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  $D$ .

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

Notice the key difference between this definition of **consistent** and our earlier definition of **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  $h(x) = c(x)$ .

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.

**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  $D$ .

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

### The List-Then-Eliminate 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-Eliminate 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*

The List-Then-Eliminate algorithm first initializes the version space to contain all hypotheses in  $H$ , then eliminates any hypothesis found inconsistent with any training example. The version space of candidate hypotheses thus shrinks as more examples are observed, until ideally just one hypothesis remains that is consistent with all the observed examples.

It is intuitively plausible that we can represent the version space in terms of its most specific and most general members.

**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  $D$ .

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

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

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

As long as the sets  $G$  and  $S$  are well defined, they completely specify the version space. In particular, we can show that the version space is precisely the set of hypotheses contained in



G, plus those contained in S, plus those that lie between G and S in the partially ordered hypothesis space. (This is stated precisely in Theorem 2.1. Refer text book for more details)

## 2.5 Candidate Elimination algorithm

The computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples. It begins by initializing the version space to the set of all hypotheses in H; that is, by initializing the G boundary set to contain the most general hypothesis in H

$$G_0 \leftarrow \{ \langle ?, ?, ?, ?, ?, ? \rangle \}$$

and initializing the S boundary set to contain the most specific (least general) hypothesis

$$S_0 \leftarrow \{ \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle \}$$

These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than  $S_0$  and more specific than  $G_0$ . As each training example is considered, the S and G boundary sets are generalized and specialized, respectively, to eliminate from the version space any hypotheses found inconsistent with the new training example. After all examples have been processed, the computed version space contains all the hypotheses consistent with these examples and only these hypotheses. This algorithm is summarized in given below.

### Candidate Elimination Algorithm using Version Spaces

1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H
3. For each training example d, do
  - a. If d is a positive example
    - i. Remove from G any hypothesis inconsistent with d,
    - ii. 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, hypothesis that is more general than another in S
  - b. If d is a negative example
    - i. Remove from S any hypothesis inconsistent with d
    - ii. 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 in G

### An Illustrative Example

The Figure given below traces the algorithm. As described above, the boundary sets are first initialized to  $G_0$  and  $S_0$ , the most general and most specific hypotheses in H, respectively.



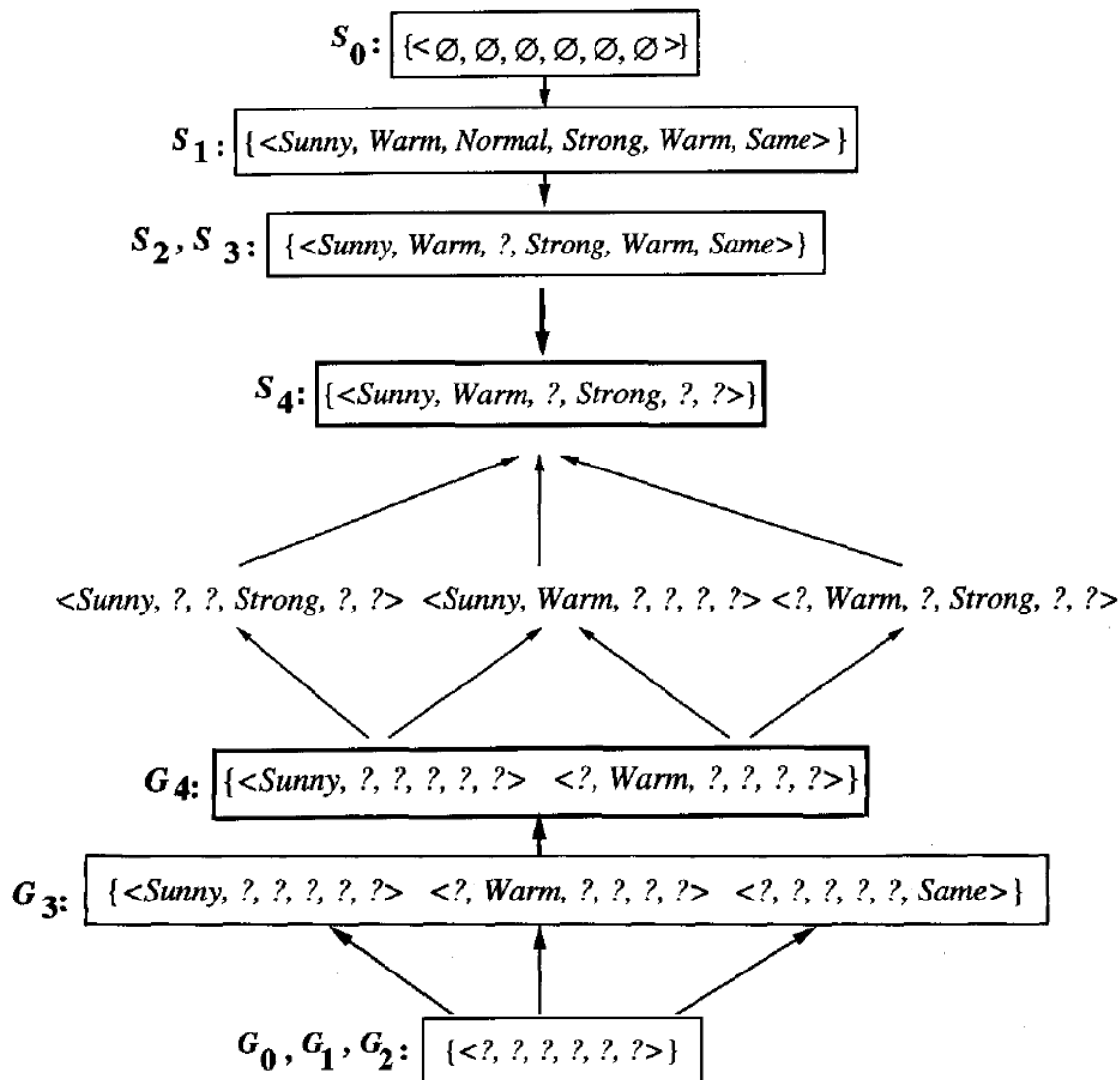
**First training Sample:** When the first training example is presented (a positive example in this case), the algorithm checks the  $S$  boundary and finds that it is overly specific-it fails to cover the positive example. The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example. This revised boundary is shown as  $S_1$  in Figure 2.4. No update of the  $G$  boundary is needed in response to this training example because  $G_0$  correctly covers this example.

**Second Training Sample:** When the second training example (also positive) is observed, it has a similar effect of generalizing  $S$  further to  $S_2$ , leaving  $G$  again unchanged (i.e.,  $G_2 = G_1 = G_0$ ). Notice the processing of these first two positive examples is very similar to the processing performed by the Find-S algorithm.

**Third Sample:** Negative training examples play the complimentary role of forcing the  $G$  boundary to become increasingly specific. Consider the third training example (negative sample), This negative example reveals that the  $G$  boundary of the version space is overly general; that is, the hypothesis in  $G$  incorrectly predicts that this new example is a positive example. The hypothesis in the  $G$  boundary must therefore be specialized until it correctly classifies this new negative example. There are several alternative minimally more specific hypotheses. All of these become members of the new  $G_3$  boundary set.

Given that there are six attributes that could be specified to specialize  $G_2$ , why are there only three new hypotheses in  $G_3$ ? For example, the hypothesis  $h = (?, ?, Normal, ?, ?, ?)$  is a minimal specialization of  $G_2$  that correctly labels the new example as a negative example, but it is not included in  $G_3$ . The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples. The algorithm determines this simply by noting that  $h$  is not more general than the current specific boundary,  $S_2$ . In fact, the  $S$  boundary of the version space forms a summary of the previously encountered positive examples that can be used to determine whether any given hypothesis is consistent with these examples. Any hypothesis more general than  $S$  will, by definition, cover any example that  $S$  covers and thus will cover any past positive example. In a dual fashion, the  $G$  boundary summarizes the information from previously encountered negative examples. Any hypothesis more specific than  $G$  is assured to be consistent with past negative examples. This is true because any such hypothesis, by definition, cannot cover examples that  $G$  does not cover.

**Fourth training example:** This further generalizes the  $S$  boundary of the version space. It also results in removing one member of the  $G$  boundary, because this member fails to cover the new positive example. This last action results from the first step under the condition "If  $d$  is a positive example" in the algorithm. To understand the rationale for this step, it is useful to consider why the offending hypothesis must be removed from  $G$ . Notice it cannot be specialized, because specializing it would not make it cover the new example. It also cannot be generalized, because by the definition of  $G$ , any more general hypothesis will cover at least one negative training example. Therefore, the hypothesis must be dropped from the  $G$  boundary, thereby removing an entire branch of the partial ordering from the version space of hypotheses remaining under consideration.



**Training examples:**

1.  $\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle, \text{Enjoy Sport} = \text{Yes}$
2.  $\langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle, \text{Enjoy Sport} = \text{Yes}$
3.  $\langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle, \text{Enjoy Sport} = \text{No}$
4.  $\langle \text{Sunny, Warm, High, Strong, Cool, Change} \rangle, \text{Enjoy Sport} = \text{Yes}$

After processing these four examples, the boundary sets  $S_4$  and  $G_4$  delimit the version space of all hypotheses consistent with the set of incrementally observed training examples. The entire version space, including those hypotheses bounded by  $S_4$  and  $G_4$ . This learned version space is independent of the sequence in which the training examples are presented (because in the end it contains all hypotheses consistent with the set of examples). As further training data is encountered, the  $S$  and  $G$  boundaries will move monotonically closer to each other, delimiting a smaller and smaller version space of candidate hypotheses.



## 2.6 Inductive Bias.

The CEA 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. Here we examine them in the context of the CEA. The conclusions we draw from this analysis will apply to any concept learning system that outputs any hypothesis consistent with the training data.

### 2.6.1 A Biased Hypothesis Space

Suppose we wish to assure that the hypothesis space contains the unknown target concept. The obvious solution is to enrich the hypothesis space to include every possible hypothesis. To illustrate, consider *EnjoySport* example in which 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.

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

To see why 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  $H$   $S_2 : \{?, Warm, Normal, Strong, Cool, Change\}$  is

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. In this case we require a more expressive hypothesis space.

### 2.6.2 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 **power-set** of  $X$  ).





In the *EnjoySport* learning task, for example, the size of the instance space  $X$  of days described by the six available attributes is 96. In general, the number of distinct subsets that can be defined over a set  $X$  containing  $|x|$  elements is  $2^{|x|}$ . Thus, there are  $2^{96}$ , or approximately distinct target concepts that could be defined over this instance space and that our learner might be called upon to learn. Our conjunctive hypothesis space is able to represent only 973 of these—a very biased hypothesis space indeed!

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

$$\langle \text{Sunny}, ?, ?, ?, ?, ? \rangle \vee \langle \text{Cloudy}, ?, ?, ?, ?, ? \rangle$$

However, while this hypothesis space eliminates any problems of expressibility, it unfortunately raises a new, equally difficult problem: our concept learning algorithm is now completely unable to generalize beyond the observed examples! To see why, suppose we present three positive examples ( $x_1, x_2, x_3$ ) and two negative examples ( $x_4, x_5$ ) to the learner. At this point, the  $S$  boundary of the version space will be  $S : \{(x_1 \vee x_2 \vee x_3)\}$

That of  $G$  will be  $G : \{\neg(x_4 \vee x_5)\}$

Here in order to converge to a single, final target concept, we will have to present every single instance in  $X$  as a training example!

### 2.6.3 The Futility of Bias-Free Learning

The 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 CEA 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 CEA will mis-classify at least some instances from  $X$ .

Let us define this notion of inductive bias more precisely. Consider the general setting in which an arbitrary learning algorithm  $L$  is provided an arbitrary set of training data  $D_c = \{\langle x, c(x) \rangle\}$  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  $(D_c \wedge x_i) \succ L(x_i, D_c)$

where the notation  $y \succ z$  indicates that  $z$  is inductively inferred from  $y$ . For example, if we take  $L$  to be the CEA,  $D_c$ , to be the training data from Table 2.1, and  $x_i$  to be the first instance from Table 2.6, then the inductive inference performed in this case concludes that  $L(x_i, D_c) = (\text{EnjoySport} = \text{yes})$ .



Instance	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
A	Sunny	Warm	Normal	Strong	Cool	Change	?
B	Rainy	Cold	Normal	Light	Warm	Same	?
C	Sunny	Warm	Normal	Light	Warm	Same	?
D	Sunny	Cold	Normal	Strong	Warm	Same	?

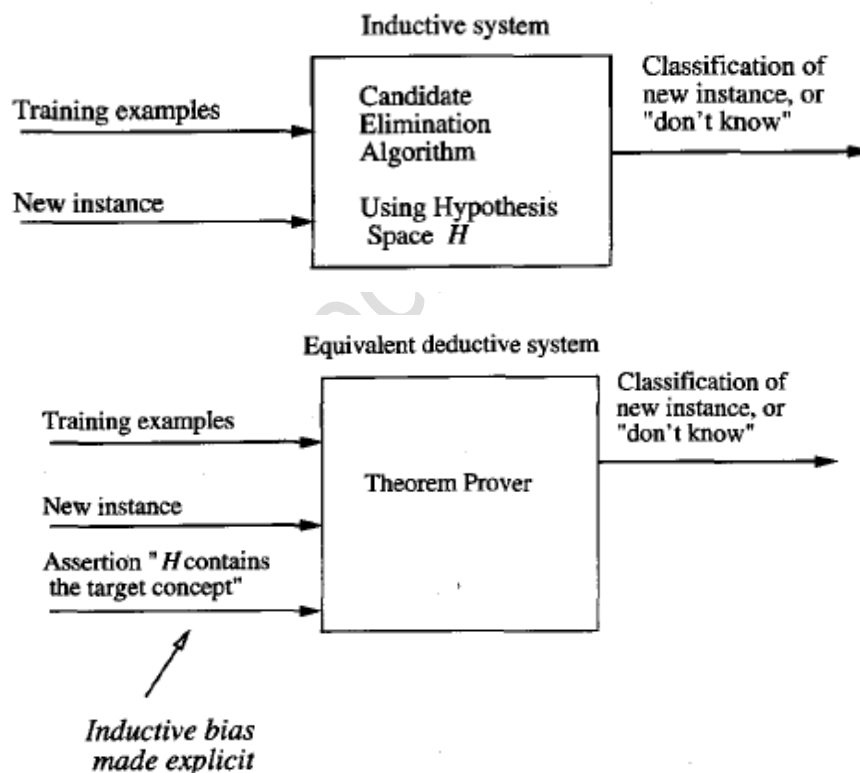
**TABLE 2.6**  
New instances to be classified.

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

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

**Inductive bias of CEA:** The target concept  $c$  is contained in the given hypothesis space  $H$ .

The figure given below summarizes the situation schematically.



**FIGURE 2.8**

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



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. Consider, for example, the following three learning algorithms, which are listed from weakest to strongest bias.

- **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.
- **CEA:** 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.
- **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.

The Rote-Learner has no inductive bias. The classifications it provides for new instances follow deductively from the observed training examples, with no additional assumptions required. The CEA has a stronger inductive bias: that the target concept can be represented in its hypothesis space. Because it has a stronger bias, it will classify some instances that the Rote-Learner will not. Of course, the correctness of such classifications will depend completely on the correctness of this inductive bias. The FIND-S algorithm has an even stronger inductive bias. In addition to the assumption that the target concept can be described in its hypothesis space, it has an additional inductive bias assumption: that all instances are negative instances unless the opposite is entailed by its other knowledge.

### 3. Summary

Machine learning addresses the question of how to build computer programs that improve their performance at some task through experience. Major points of this topic include:

- Machine learning algorithms have proven to be of great practical value in a variety of application domains. They are especially useful in (a) data mining problems where large databases may contain valuable implicit regularities that can be discovered automatically (b) poorly understood domains where humans might not have the knowledge needed to develop effective and (c) domains where the program must dynamically adapt to changing conditions
- Machine learning draws on ideas from a diverse set of disciplines, including artificial intelligence, probability and statistics, computational complexity, information theory, psychology and neurobiology, control theory, and philosophy.
- A well-defined learning problem requires a well-specified task, performance metric, and source of training experience.
- Designing a machine learning approach involves a number of design choices, including choosing the type of training experience, the target function to be learned, a



representation for this target function, and an algorithm for learning the target function from training examples.

- Learning involves search: searching through a space of possible hypotheses to find the hypothesis that best fits the available training examples and other prior constraints or knowledge.

The main points in the Concept Learning include:

- Concept learning can be cast as a problem of searching through a large predefined space of potential hypotheses.
- The general-to-specific partial ordering of hypotheses, which can be defined for any concept learning problem, provides a useful structure for organizing the search through the hypothesis space.
- The Find-S algorithm utilizes this general-to-specific ordering, performing a specific-to-general search through the hypothesis space along one branch of the partial ordering, to find the most specific hypothesis consistent with the training examples.
- The CEA utilizes this general-to-specific ordering to compute the version space (the set of all hypotheses consistent with the training data) by incrementally computing the sets of maximally specific (S) and maximally general (G) hypotheses.
- The version space of alternative hypotheses can be examined to determine whether the learner has converged to the target concept, to determine when the training data are inconsistent, to generate informative queries to further refine the version space, and to determine which unseen instances can be unambiguously classified based on the partially learned concept.
- Version spaces and the CEA provide a useful conceptual framework for studying concept learning. However, this learning algorithm is not robust to noisy data or to situations in which the unknown target concept is not expressible in the provided hypothesis space.
- Inductive learning algorithms are able to classify unseen examples only because of their implicit inductive bias for selecting one consistent hypothesis over another. The bias associated with the CEA is that the target concept can be found in the provided hypothesis space ( $c \in H$ ). The output hypotheses and classifications of subsequent instances follow deductively from this assumption together with the observed training data.
- If the hypothesis space is enriched to the point where there is a hypothesis corresponding to every possible subset of instances (the power set of the instances), this will remove any inductive bias from the CEA. Unfortunately, this also removes the ability to classify any instance beyond the observed training examples. An unbiased learner cannot make inductive leaps to classify unseen examples.

\*\*\*\*\*