Machine learning

Unit 1

syllabus

- Introduction - Well-posed learning problems, designing a learning system, Perspectives and issues in machine learning
- Concept learning and the general to specific ordering – introduction, a concept learning task, concept learning as search, find-S: finding a maximally specific hypothesis, version spaces and the candidate elimination algorithm, remarks on version spaces and candidate elimination, inductive bias.
- **Decision Tree Learning –** Introduction, decision tree representation, appropriate problems for decision tree learning, the basic decision tree learning algorithm, hypothesis space search in decision tree learning, inductive bias in decision tree learning, issues in decision tree learning

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.
- For any problem we must identity 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

- **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
- **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
  - **Training experience E:** a sequence of images and steering commands recorded while observing a human driver

Designing a Learning System

1. **Choosing the Training Experience**
2. **Choosing the Target Function**
3. **Choosing a Representation for the Target Function**
4. **Choosing a Function Approximation Algorithm**
5. **The Final Design**

1.Choosing the Training Experience

- First is to choose the type of training experience from which our system will learn.
  - Has significant impact on failure or success of the learner
- One key attribute is Whether training experience provides direct or indirect feedback regarding the choices made by the performance system.
  - Direct feedback - correct move for every step
  - Indirect feedback- Consisting of move sequences and final outcomes.

  -No credit assignment for each move.

  -Credit assignment is difficult.
- learning is easy from direct feedback.

1.Choosing the Training Experience

- Second attribute of training experience is the degree to which the learner controls the sequence of training examples
- Learner self learns and ends up with confusion
  - Novel board states , increases skills
  - confusion
- Ask teacher to help by posing various queries
  - learner collects training examples by autonomously exploring its environment
- 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.

1.Choosing the Training Experience

- 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
- Self learning
  - No teacher required
  - is not sufficient
- Assumption: Distribution of training is identical to test data

1.Choosing the Training Experience

- 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

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
  - how to choose the best move from among these legal moves
  - Program, or function(***ChooseMove)***, that chooses the best move for any given board state
  - ***ChooseMove : B -> M***
  - B:set of legal board states
  - M: some move from a set of legal moves as output
- choice of the target function is key in design
  - Difficult to choose

2.Choosing the Target Function

- An Alternative target function that is easier to choose is
  - an evaluation function(V) that assigns a numerical score to any given board state
  - ***V : B ->R***
  - B: Set of board state, ***R :*** set of real numbers
  - function **V** assigns higher scores to better board states

2.Choosing the Target Function

- Let us therefore 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).
- Learning an ideal target function(V) is difficult so we do some approximations to target function.
- The process of learning the target function is often called ***function approximation**(*$\hat{V}$***)***
  - ***V : I**deal target function*

Representing the Target Function

- Target function can be represented in many ways: lookup table, symbolic rules, numerical function, neural network.
- There is a trade-off between the expressiveness of a representation and the ease of learning.
- The more expressive a representation, the better it will be at approximating an arbitrary function; however, the more examples will be needed to learn an accurate function.

3.Choosing a Representation for the Target Function

- we must choose a representation that the learning program will use to describe the function $\hat{V}$ that it will learn
  - A large table with a distinct entry for every state
  - Collection of rules that match against features of state
  - Quadratic polynomial function of predefined board features
  - An artificial neural network
- More expensive representation more close to ideal target function V , and more training data we require to choose among the hypotheses
- let us choose a simple representation for any given board state

3.Choosing a Representation for the Target Function

- The function $\hat{V}$ will be calculated as a linear combination of the following board features:
  - X1: the number of black pieces on the board
  - X2:the number of red pieces on the board
  - X3: the number of black kings on the board
  - X4: the number of red kings on the board
  - x5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
  - X6: the number of red pieces threatened by black

3.Choosing a Representation for the Target Function

- 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 W0 through W6 are numerical coefficients, or weights, to be chosen by the learning algorithm

3.Choosing a Representation for the Target Function

- 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 -> $\mathscr{R}$
  - Target function representation

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

4.Choosing a Function Approximation Algorithm

- Each training example is an ordered pair is of the form(b, V_{train}(b)).
- b : specific board state
- V_{train}(b) : Training value for b

This is board state b in which black has won the game (note *x2 = 0 indicates that red has no remaining pieces)*
V_{train}(b) = +100

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

4.1.ESTIMATING TRAINING VALUES

- Easy to assign scores to board states that correspond to the end of the game
- Difficult to assign scores to intermediate board states
- Despite the ambiguity inherent in estimating training values for intermediate board states, one simple approach has been found to be surprisingly successful

Rule for estimating training values.

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

Where $\hat{V}$ *is the learner's current approximation to V and where Successor(b) denotes the next board state following b for which it* is again the program's turn to move

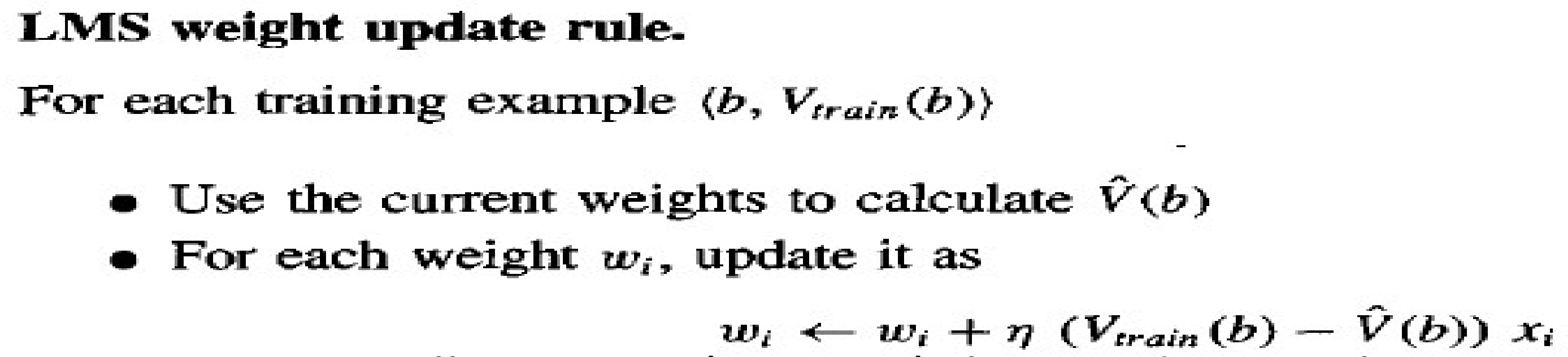
4.2.ADJUSTING THE WEIGHTS

- Choosing the weights w_i to best fit the set of training examples (b, V_{train}(b)).
- Best fit to the training data
- Define best hypothesis or set of weights to minimize squared error
- If E is small → most probable hypothesis

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

4.2.ADJUSTING THE WEIGHTS

- we seek the weights, or equivalently the $\hat{V}$ that minimize E for the observed training examples.
- 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
- **LMS training rule:** It adjusts the weights a small amount in the direction that reduces the error on this training example



LMS weight update rule.

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

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

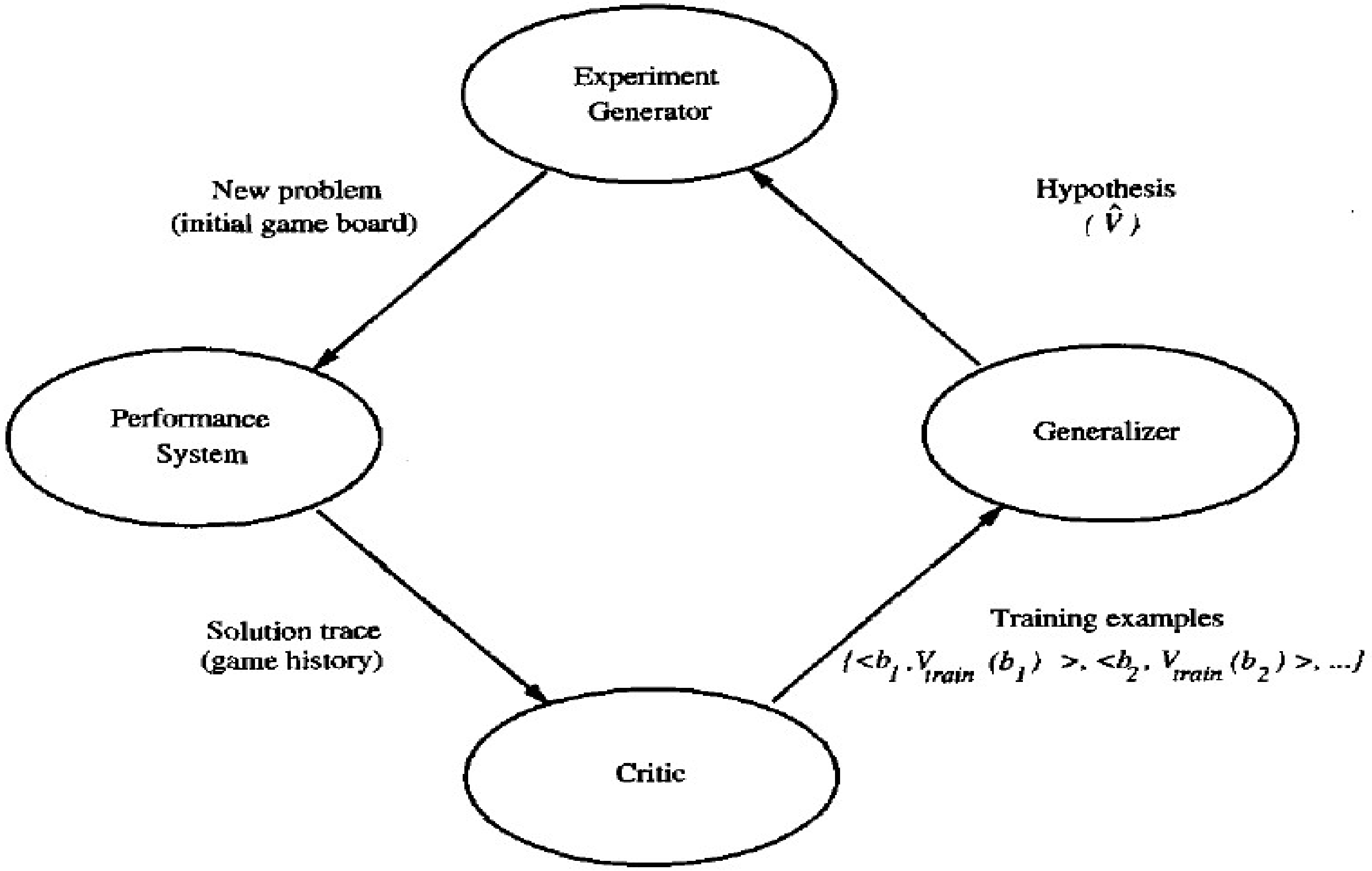
$$w_i \leftarrow w_i + \eta\ (V_{train}(b) - \hat{V}(b))\ x_i$$

- Here n is a small constant (e.g., 0.1) that moderates the size of the weight update
- For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

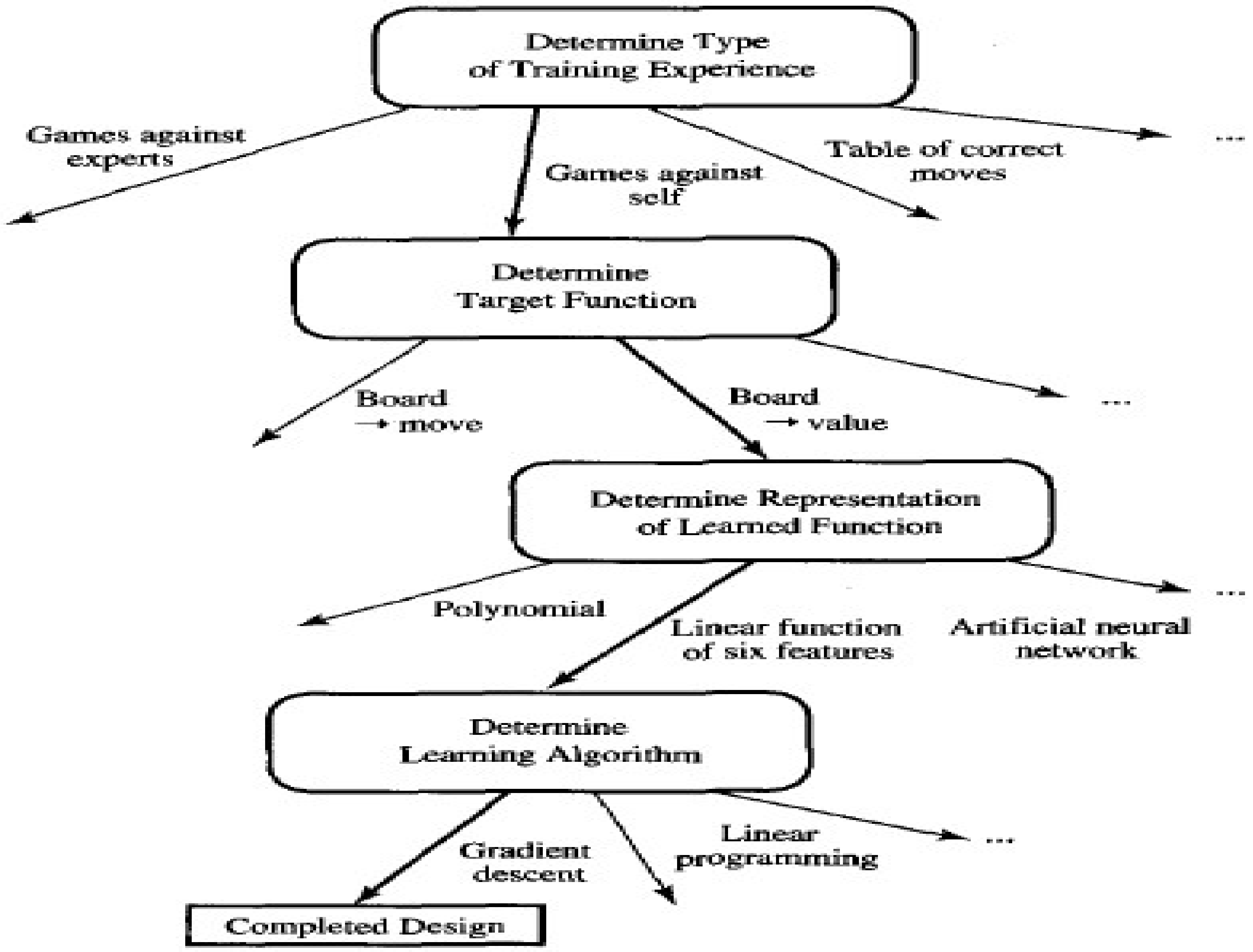
5.The Final Design

- The final design of our checkers learning system can be done by **four distinct program modules** that represent the central components in many learning systems
  - **Performance System**
    - **Solve the task, outputs the game history**
  - **Critic**
    - **Outputs the set of training examples of the target function.**
  - **Generalizer**
    - **Outputs the target function hypothesis**
  - **Experiment Generator**
    - **Outputs a new problem(Board state)**

5.The Final Design


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

5.The Final Design


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

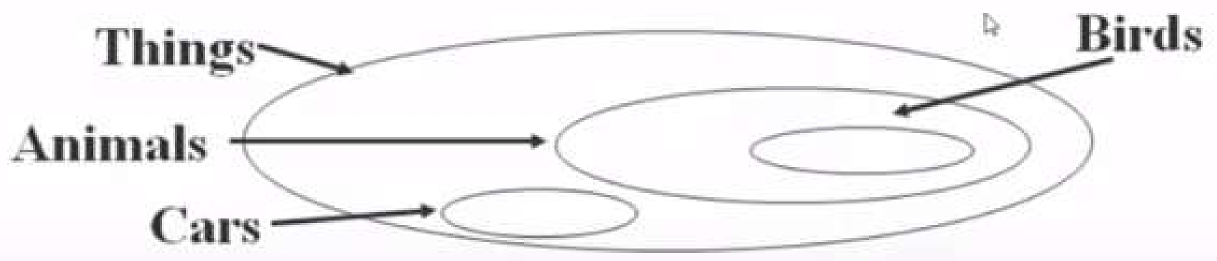
- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training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

Issues in Machine Learning

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

WHAT IS A CONCEPT?

A Concept is a subset of objects or events defined over a larger set. For example, We refer to the set of everything (i.e. all objects) as the set of things. Animals are a subset of things, and birds are a subset of animals.



In more technical terms, a concept is a boolean-valued function defined over this larger set.

For example, a function defined over all *animals* whose value is *true* for birds and *false* for every other animal.

WHAT IS A CONCEPT LEARNING?

Given a set of examples labeled as members or non-members of a concept, *concept-learning* consists of automatically inferring the general definition of this concept.

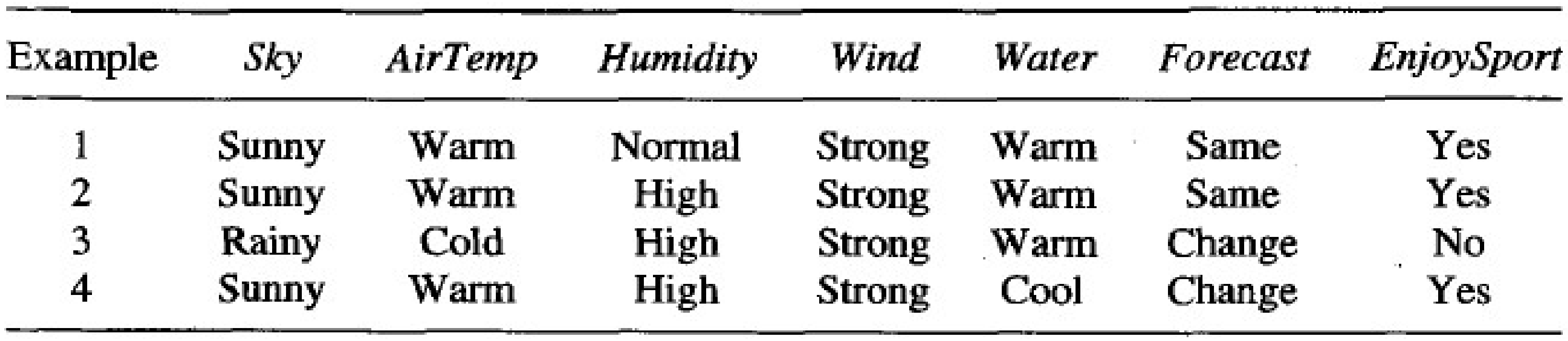
In other words, *concept-learning* consists of approximating a boolean-valued function from training examples of its input and output.

Concept Learning Task

- **Concept**: Good days for WaterSports
- **Task**: to learn to predict the value of ***EnjoySport*** *for an arbitrary day, based on the values of its other attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast*
- each **hypothesis** be a vector of six attributes

  Sky, AirTemp, Humidity, Wind, Water, and Forecast
- For each attribute, the hypothesis will either
  - indicate by a "?' that any value is acceptable for this attribute,
  - *specific value (e.g., Warm) for the attribute*
  - *indicate by a "Φ" that no value is acceptable.*

Example of a Concept Learning Task



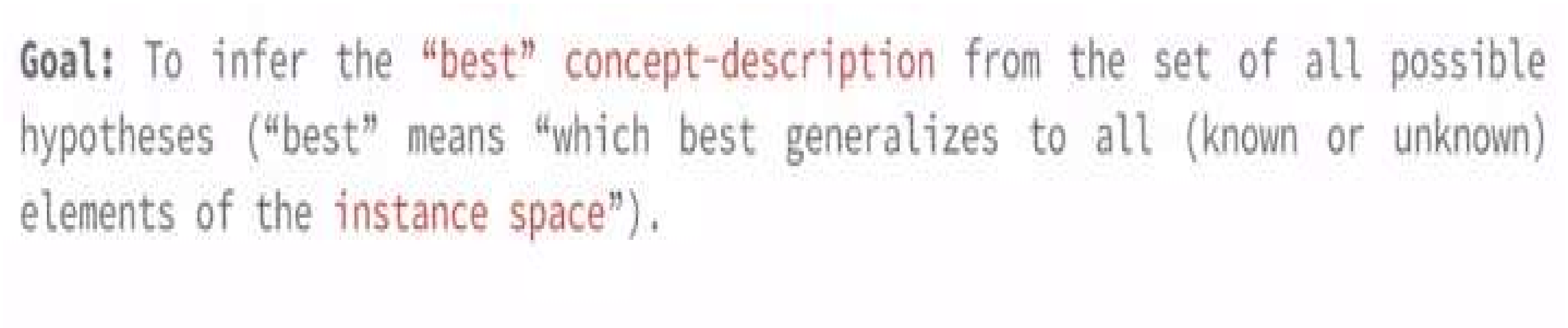
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- **Example of a hypothesis:** If the air temperature is cold and high humidity high then it is a good day for water sports.
- It can be represented by the expression

$$\langle ?, Cold, High, ?, ?, ? \rangle$$

- According to above training data , given hypothesis is false.
- If some instance x satisfies all the constraints of hypothesis h, then h classifies x as a positive example h(x) = 1

Concept Learning Task


Goal: To infer the "best" concept-description from the set of all possible hypotheses ("best" means "which best generalizes to all (known or unknown) elements of the instance space").

- The most general hypothesis-that every day is a good day for water sports is represented by

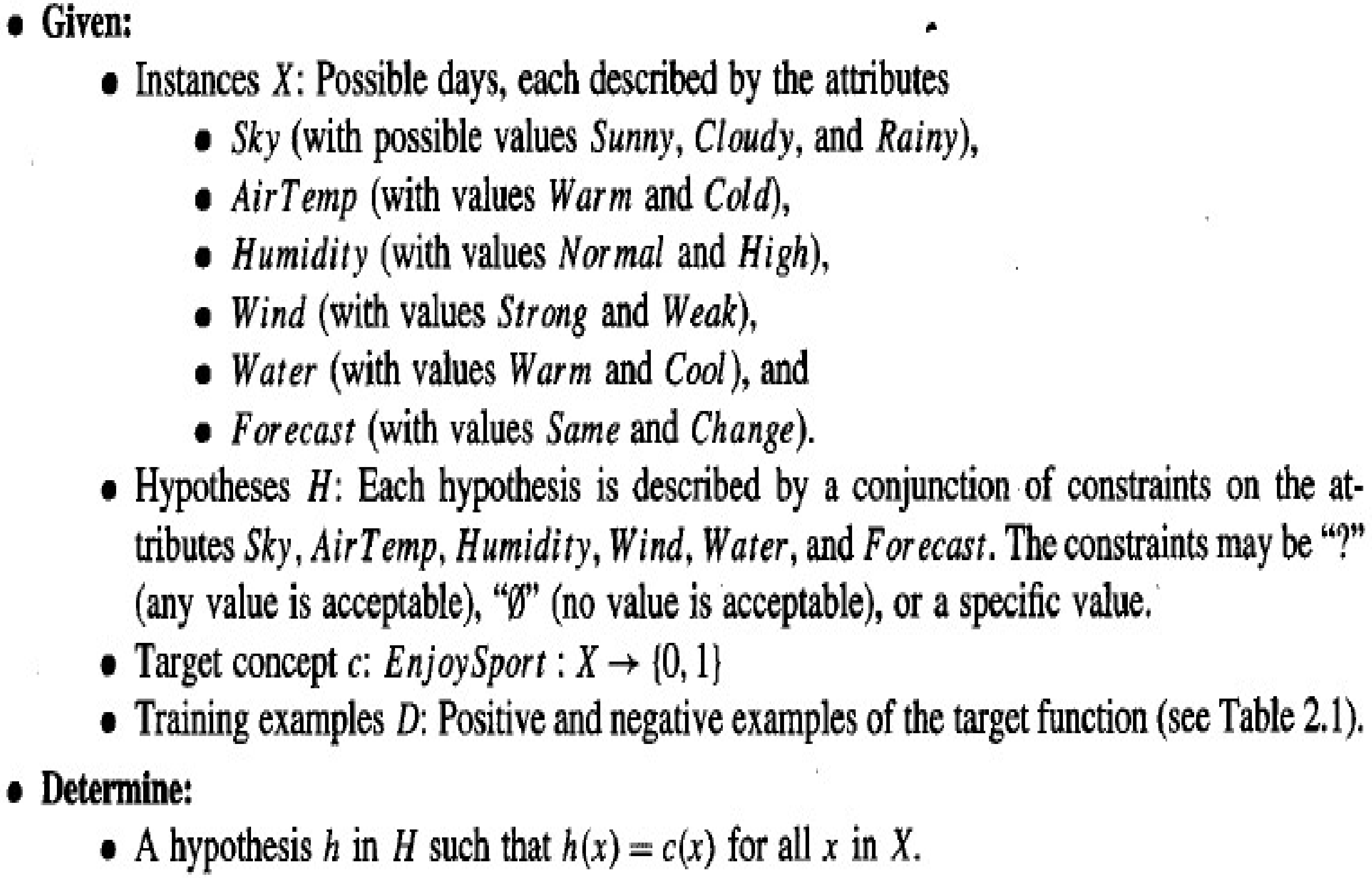
  (?,?,?,?,?,?)

- the most specific possible hypothesis-that no day is a good day for water sports is represented by

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

Concept Learning Task

- The definition of the EnjoySport concept learning task in this general form is given below



- **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.

Terminology and Notations

- The set of items over which the concept is defined is called a **set of instances** (denoted b X)
- The concept to be learned is called **target concept** (denoted by c : X → {0,1})
- A set of **training examples** is an ordered pair ***(x, c(x)).***
- ***members of the target concept (***Instances for which ***c(x) = 1)*** *are called positive examples*
- nonmembers of the target concept (Instances for which c***(x) = 0****) are called negative examples*
- *H represents* ***set of all possible hypotheses****. H is designed by human designer's choice of a hypothesis representation.*
- *The* ***goal of Concept Learning*** *is to find a hypothesis h : X → {0,1} such that h(x)=c(x) for all x in X.*

The Inductive Learning Hypothesis

- The learning task is to determine a hypothesis *h* identical to the target concept *c over the entire set of instances X*
- **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
- Selecting a **hypothesis representation** is an is important step since it restricts
- (or biases) the space that can be searched.
- **For Example** "If the air temperature is cold <u>or</u> humidity is high then it is a good day for water sports" **cannot be represented by our representation**
- The **goal of this search** is to find the hypothesis that best fits the training examples
- Sky has 3 possible values, and other 5 attributes have 2 possible values .
- The instance space X contains exactly 3.2.2.2.2.2 = 96 distinct instances
- 5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H

  (adding ? and Φ)
- Therefore, the number of semantically distinct hypotheses is only

  1 + (4.3.3.3.3.3) = 973
- Although EnjoySport has small, finite hypothesis space, most learning tasks have much larger (even infinite) hypothesis spaces. – We need efficient search algorithms on the hypothesis spaces

General-to-Specific Ordering of Hypotheses

- Many algorithms for concept learning organize the search through the hypothesis space by relying on a general-to-specific ordering of hypotheses.
- By taking advantage of this naturally occurring structure over the hypothesis space, we can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis
- Consider two hypothesis

$$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$$
$$h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$$

- Now consider the sets of instances that are classified positive by hl and by h2.

– Because h2 imposes fewer constraints on the instance, it classifies more instances as positive.

– In fact, any instance classified positive by h1 will also be classified positive by h2.

– Therefore, we say that h2 is more general than h1.

General-to-Specific Ordering of Hypotheses

- **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*

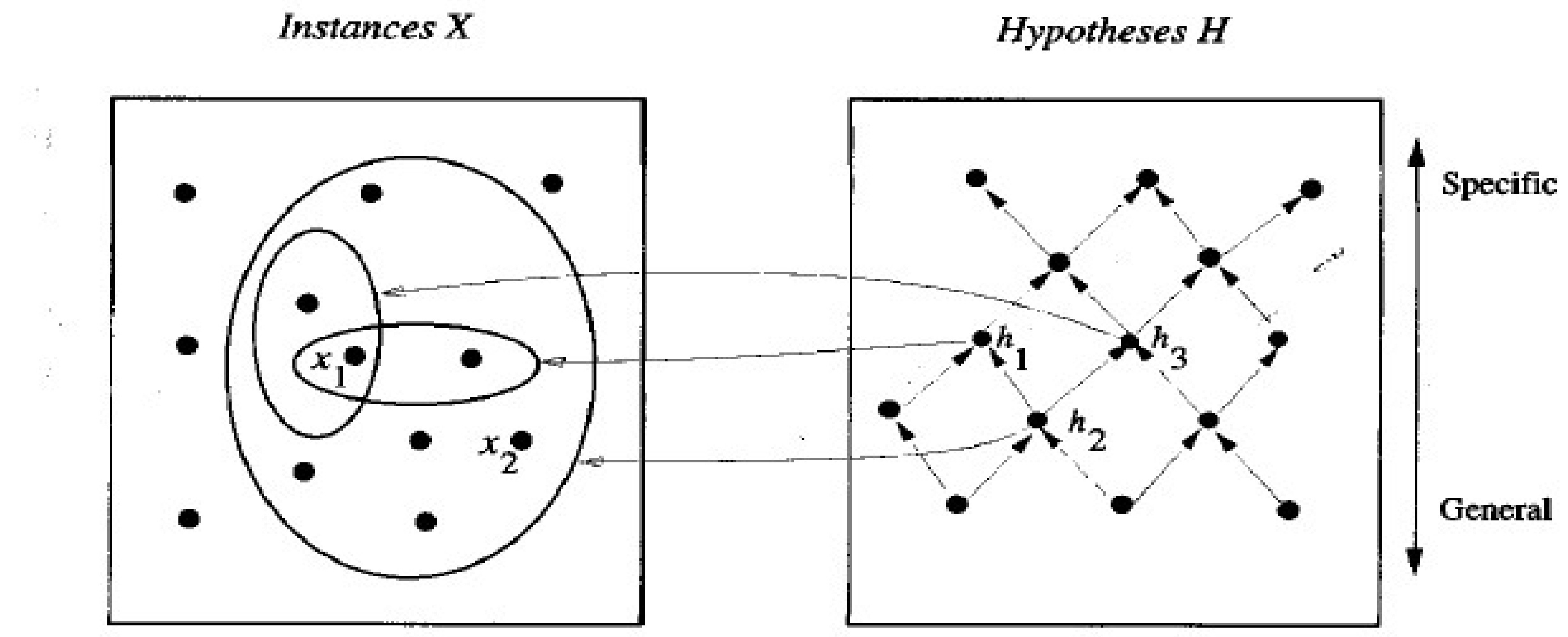
$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

- For any instance x in X and hypothesis h in H, we say that x satisfies h if and only if h(x) = 1.
- **More-General-Than-Or-Equal Relation**:

  Let h1 and h2 be two boolean-valued functions defined over X.

  Then h1 is **more-general-than-or-equal-to** h2 (written h1 ≥ h2) if and only if any instance that satisfies h2 also satisfies h1.
- h1 is (strictly) **more-general-than** h2 (h1 > h2) if and only if h1≥h2 is true and h2≥h1 is false. We also say h2 is **more-specific-than** h1.
- The $\geq_g$ ***, relation defines a partial order over*** the hypothesis space H (the relation is reflexive, antisymmetric, and transitive

General-to-Specific Ordering of Hypotheses Example



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, ?*>

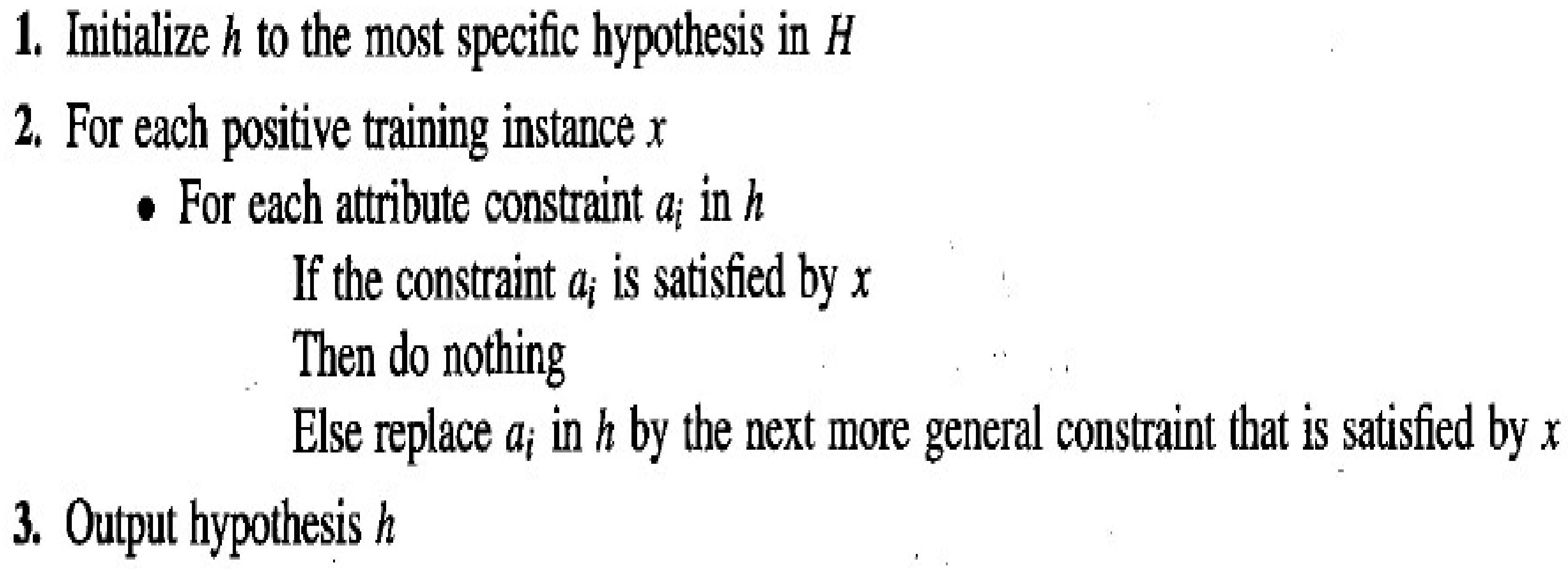
- h2 > h1 and h2 > h3
- But there is no more-general relation between h1 and h3

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS ALGORITHM

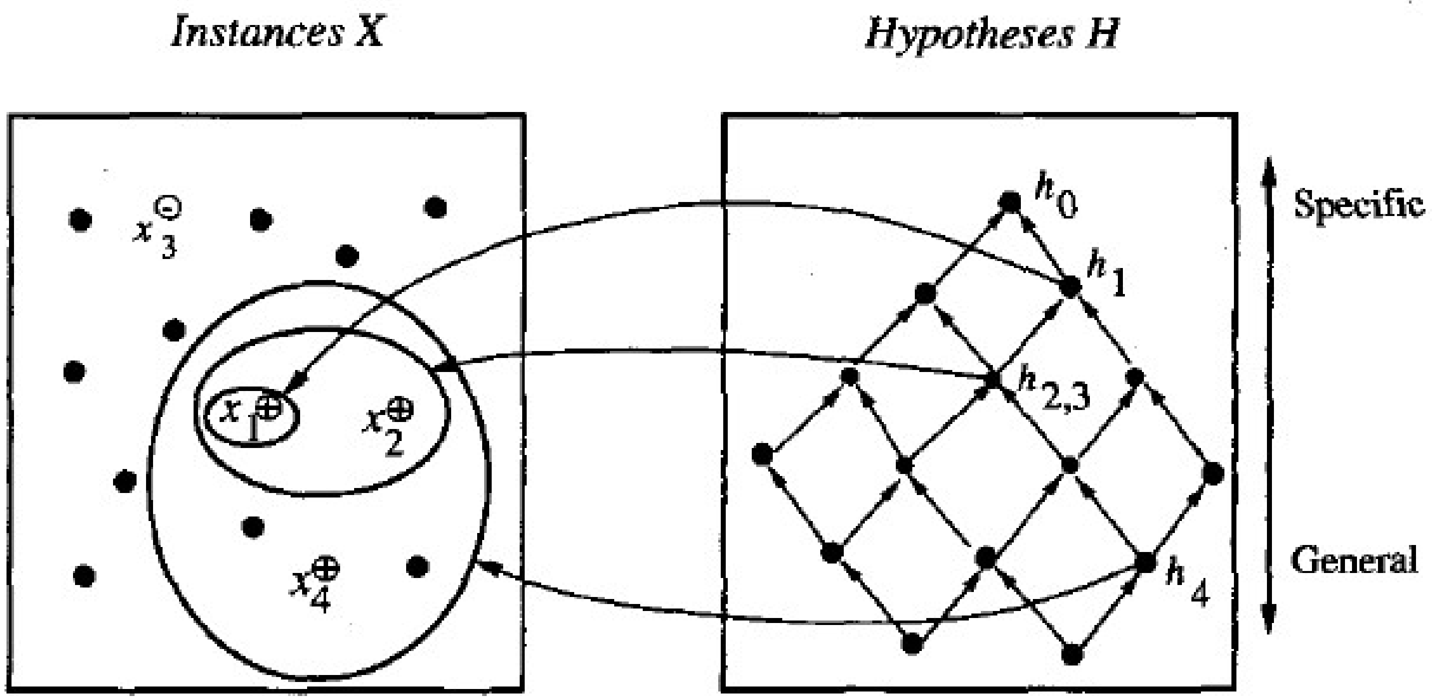
→The FIND-S algorithm illustrates how the “ more-general-than “ partial ordering can be used to organize the search for an acceptable hypothesis.

→ The search moves from hypothesis to hypothesis , searching from the most specific to progressively more general hypotheses.

→At each stage the hypothesis is the most specific hypothesis consistent with the training examples observed up to this point.



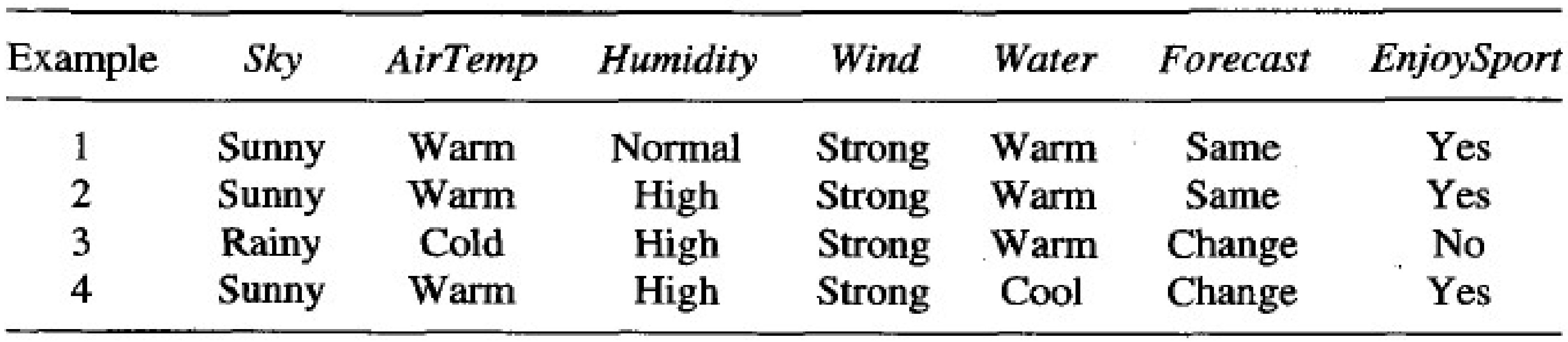
```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
     • For each attribute constraint $a_i$ in $h$
           If the constraint $a_i$ is satisfied by $x$
           Then do nothing
           Else replace $a_i$ in $h$ by the next more general constraint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



$h_0 = <\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing>$

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 ? ?* >

Example



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

Problems with FIND-S Algorithm:

1. Has the learner converged to the correct target concept?
It cannot determine if it has learnt the concept. There might be several other hypotheses that match as well – has it found the only one?
2. Why prefer the most specific hypothesis?
Some other hypothesis might be more useful.
3. Are the training examples are consistent?
We would like to detect and be tolerant to errors and noise.
4. What if there are several maximally specific consistent hypotheses?
there is no way for Find-S to find them

Version Space

→ **Consistency of a hypotheses w.r.t a training dataset**

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.

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

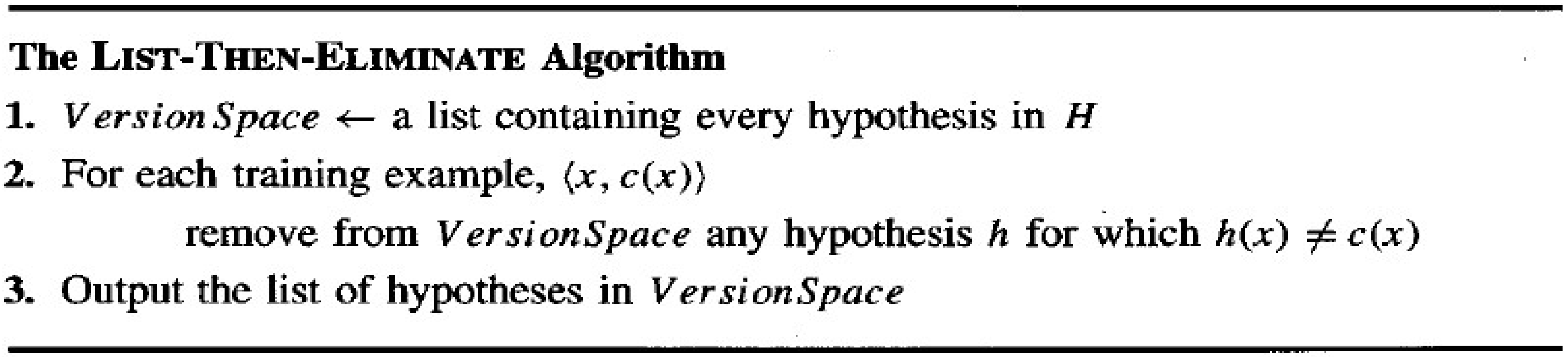
→ **Version Space:**

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

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

•The List-Then-Elimination Algorithm:

This algorithm first initializes the version space to contain all hypotheses in H, Then eliminates any hypothesis found inconsistent with any training example.



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 $VersionSpace$
```

Advantage: Guaranteed to output all hypotheses consistent with the training examples.

But **inefficient**! Even in this simple example, there are 1+4·3·3·3·3 = 973 semantically distinct hypotheses.

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.

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

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.

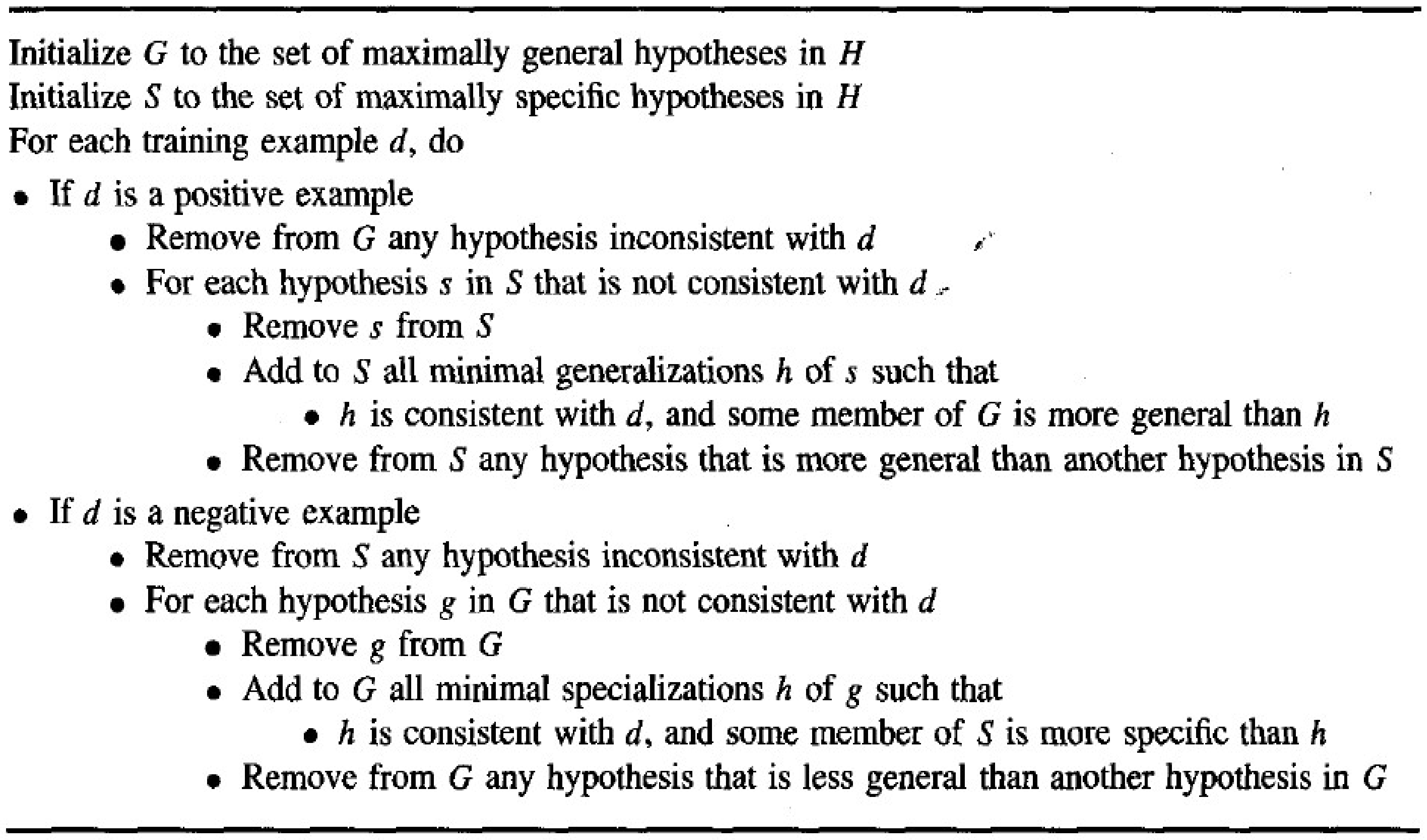
$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

G← maximally general hypothesis in H
S ← maximally specific hypothesis in H
For each training example modify G and S so that G and S are consistent with 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,

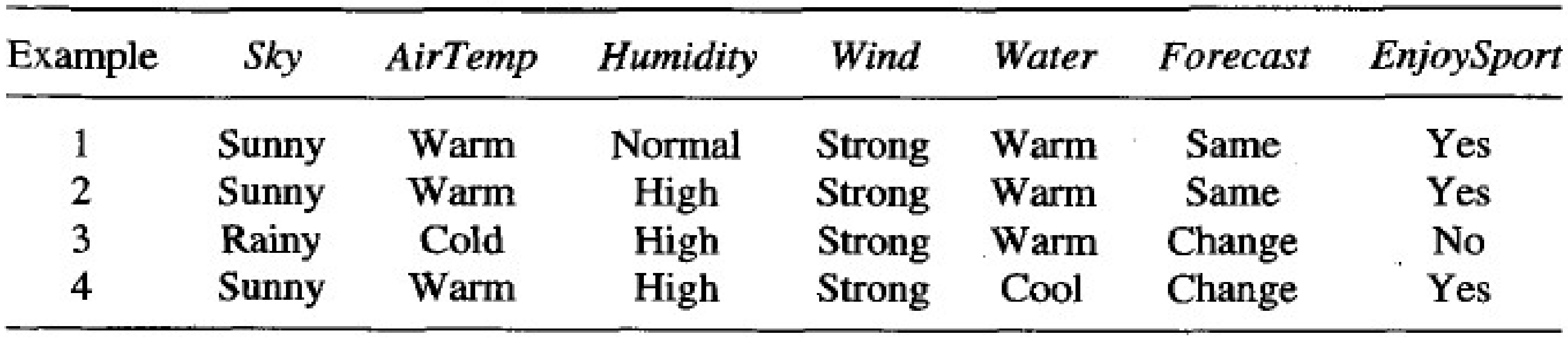
$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s)\}$$

• 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
```

Candidate-Elimination Algorithm



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

- The Candidate-elimination algorithm computes the version space containing all hypotheses from H that are consistent
- initializing the *G and S as below,* eliminate from the version space any hypotheses found inconsistent

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

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

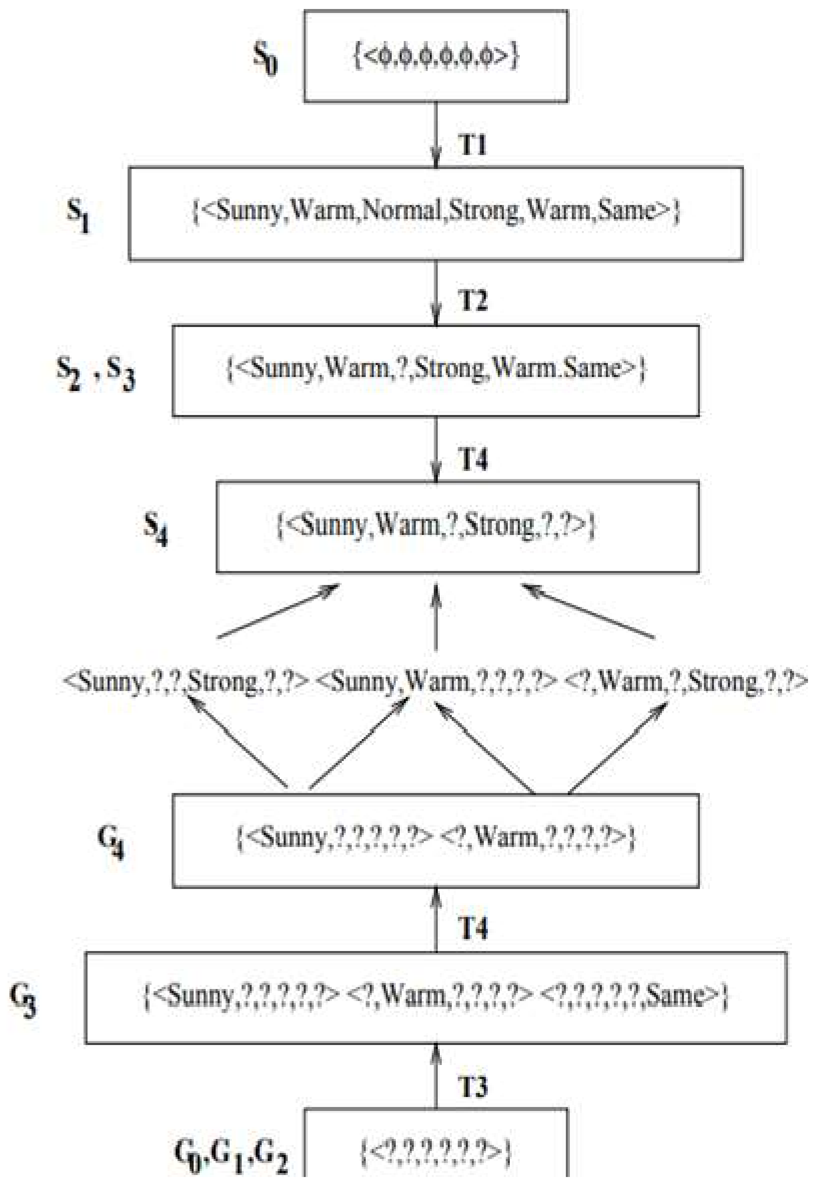
Training Examples:

T1: $\langle$*Sunny, Warm, Normal, Strong, Warm, Same*$\rangle$, *Yes*

T2: $\langle$*Sunny, Warm, High, Strong, Warm, Same*$\rangle$, *Yes*

T3: $\langle$*Rainy, Cold, High, Strong, Warm, Change*$\rangle$, *No*

T4: $\langle$*Sunny, Warm, High, Strong, Cool, Change*$\rangle$, *Yes*



S0
{<ϕ,ϕ,ϕ,ϕ,ϕ,ϕ>}
T1
S1
{<Sunny,Warm,Normal,Strong,Warm,Same>}
T2
S2 , S3
{<Sunny,Warm,?,Strong,Warm.Same>}
T4
S4
{<Sunny,Warm,?,Strong,?,?>}
<Sunny,?,?,Strong,?,?> <Sunny,Warm,?,?,?,?> <?,Warm,?,Strong,?,?>
G4
{<Sunny,?,?,?,?,?> <?,Warm,?,?,?,?>}
T4
G3
{<Sunny,?,?,?,?,?> <?,Warm,?,?,?,?> <?,?,?,?,?,Same>}
T3
G0,G1,G2
{<?,?,?,?,?,?>}

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

REMARKS ON VERSION SPACES AND CANDIDATE ELIMIATION

- **Will Candidate-Elimination Algorithm Converge to Correct Hypothesis?**
- **What Training Example Should the Learner Request Next?**
- **How Can Partially Learned Concepts Be Used?**

Will Candidate-Elimination Algorithm Converge to Correct Hypothesis?

• The version space learned by the Candidate-Elimination Algorithm will
converge toward the hypothesis that correctly describes the target
concept, provided
– There are no errors in the training examples, and
– there is some hypothesis in H that correctly describes the target concept.
• What will happen if the training data contains errors?
– The algorithm removes the correct target concept from the version space.
– S and G boundary sets eventually converge to an empty version space if sufficient additional training data is available.
– Such an empty version space indicates that there is no hypothesis in H consistent with all observed training examples.
• A similar symptom will appear when the training examples are correct,
but the target concept cannot be described in the hypothesis
representation.
– e.g., if the target concept is a disjunction of feature attributes and the hypothesis space supports only conjunctive descriptions

What Training Example Should the Learner Request Next?

- Convergence can be speeded up by presenting the data in a strategic order. The best examples are those that satisfy exactly half of the hypotheses in the current version space.

– E.g. T5: {Sunny,Warm,Normal,Light,Warm,Same} satisfies 3 hypotheses in previous example

∗ If T5 positive, S generalised, 3 hypotheses eliminated

∗ If T5 negative, G specialised, 3 hypotheses eliminated

– Optimal query strategy is to request examples that exactly split version space

– converge in $\lceil \log_2 |V S| \rceil$ steps. However, this is not always possible

Using Partially Learned Concepts

Version-Spaces can be used to assign certainty scores to the classification of new examples. Voting provides the most probable classification of the new instance

$$\begin{tikzcd}
 & S:\{\langle Sunny, Warm, ?, Strong, ?, ?\rangle\} & \\
\langle Sunny, ?, ?, Strong, ?, ?\rangle \arrow[ur] & \langle Sunny, Warm, ?, ?, ?, ?\rangle \arrow[u] & \langle ?, Warm, ?, Strong, ?, ?\rangle \arrow[ul] \\
 & G:\{\langle Sunny, ?, ?, ?, ?, ?\rangle, \langle ?, Warm, ?, ?, ?, ?\rangle\} \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

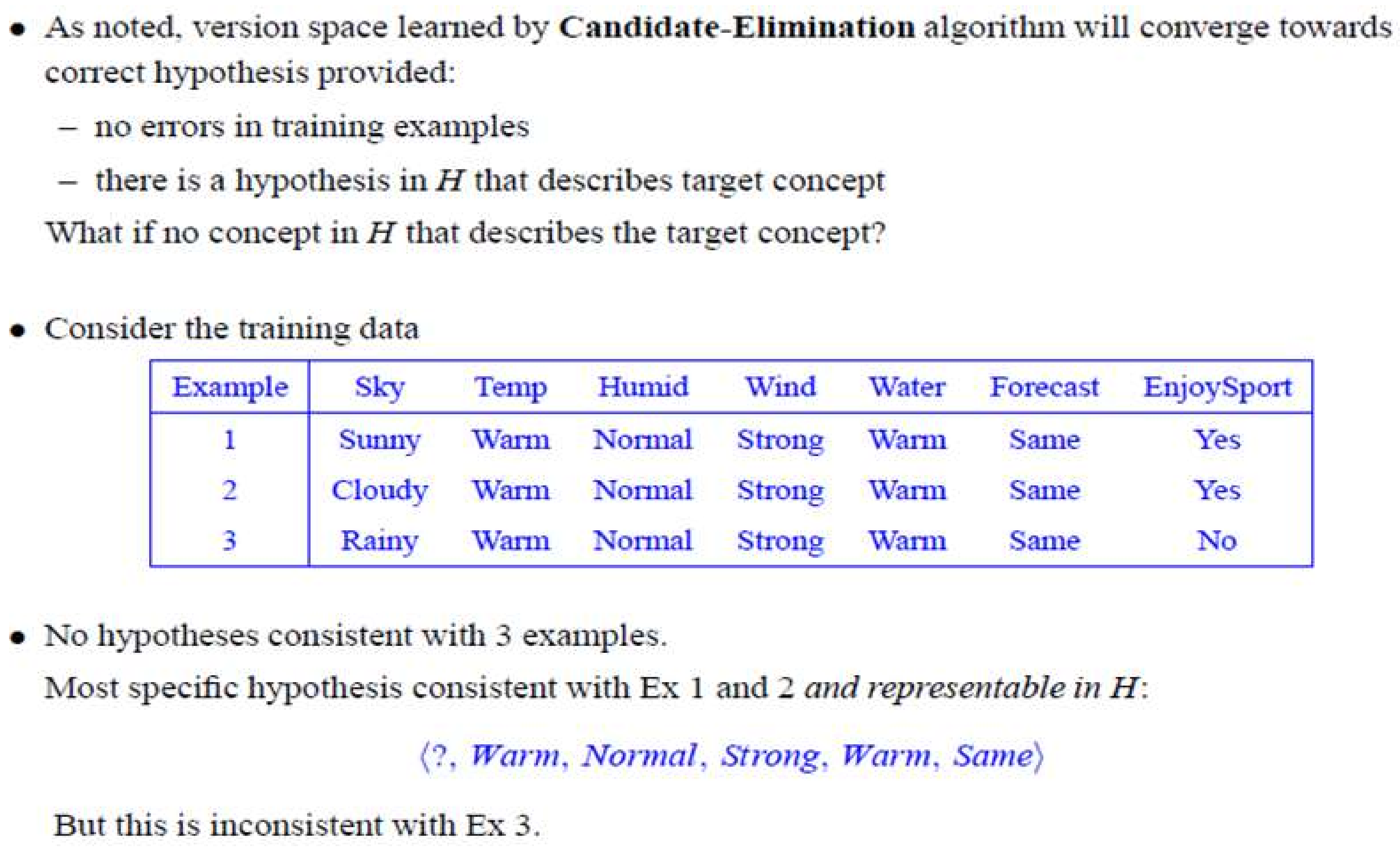
New instance	Votes: Pos	Neg
$x = \langle Sunny, Warm, Normal, Strong, Cool, Change\rangle$	6	0
$x = \langle Rainy, Cold, Normal, Light, Warm, Same\rangle$	0	6
$x = \langle Sunny, Warm, Normal, Light, Warm, Same\rangle$	3	3
$x = \langle Sunny, Cold, Normal, Strong, Warm, Same\rangle$	2	4

INDUCTIVE BIAS - Fundamental Questions for Inductive Inference

The 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?

Inductive Bias -A Biased Hypothesis Space



- As noted, version space learned by **Candidate-Elimination** algorithm will converge towards correct hypothesis provided:
  - no errors in training examples
  - there is a hypothesis in H that describes target concept

  What if no concept in H that describes the target concept?

- Consider the training data

Example	Sky	Temp	Humid	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Cloudy	Warm	Normal	Strong	Warm	Same	Yes
3	Rainy	Warm	Normal	Strong	Warm	Same	No

- No hypotheses consistent with 3 examples.

  Most specific hypothesis consistent with Ex 1 and 2 *and representable in* H:

$$\langle ?, \mathit{Warm}, \mathit{Normal}, \mathit{Strong}, \mathit{Warm}, \mathit{Same} \rangle$$

  But this is inconsistent with Ex 3.

PROBLEM: We have biased the learner to consider only conjunctive hypotheses. We require a more expressive hypothesis space.

- Need more expressive hypothesis representation language.

  E.g. allow disjunctive or negative attribute values:

$$\begin{aligned} Sky &= Sunny \vee Cloudy \\ Sky &\neq Rainy \end{aligned}$$

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.

– Every possible subset of the instances X -> ***the power set of X.***

- What is the size of the hypothesis space H (the power set of X) ?

– In EnjoySport, the size of the instance space X is 96.

– The size of the power set of X is $2^{|X|}$ -> The size of H is 296

– Our conjunctive hypothesis space is able to represent only 973of these hypotheses.

-> a very biased hypothesis space

- Let the hypothesis space H to be the power set of X.

– A hypothesis can be represented with disjunctions, conjunctions, and negations of
our earlier hypotheses.

– The target concept "Sky = Sunny or Sky = Cloudy" could then be described as

<Sunny, ?, ?, ?, ?, ?> <Cloudy, ?, ?, ?, ?, ?>

An Unbiased Learner

- **NEW PROBLEM:** our concept learning algorithm is now completely unable to generalize beyond the observed examples.

– three positive examples (xl,x2,x3) and two negative examples (x4,x5) to the learner.

– S : { x1 V x2 V x3 } and G : { not (x4 V x5) }

-> NO GENERALIZATION

– Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves.

Inductive Bias – 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.

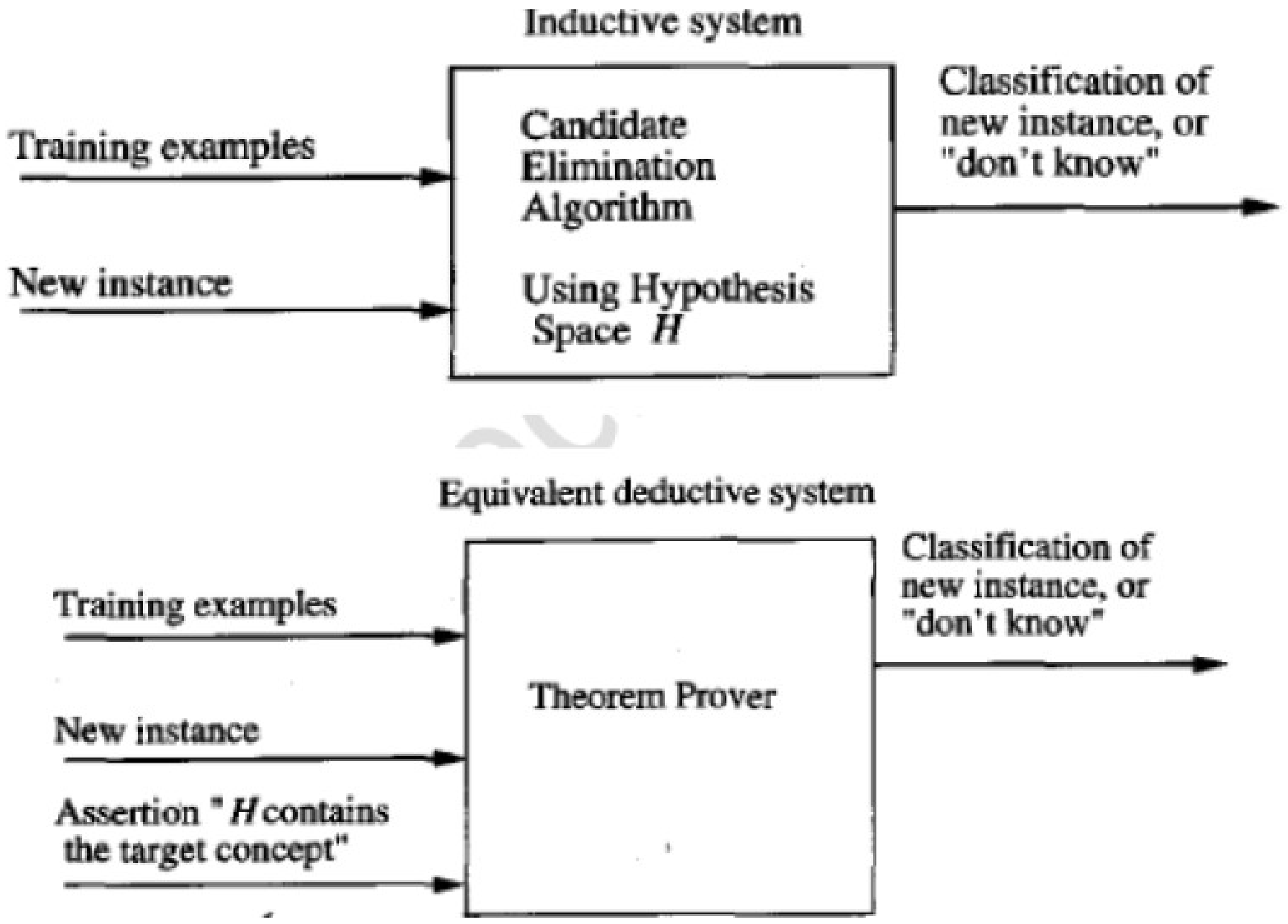
- **Inductive Leap: A learner should be able to generalize training data** using prior assumptions in order to classify unseen instances.
- The generalization is known as **inductive leap and our prior** assumptions are the **inductive bias of the learner.**
- Inductive Bias (prior assumptions) of Candidate-Elimination Algorithm s that the target concept can be represented by a conjunction of attribute values, the target concept is contained in the hypothesis space and training examples are correct.

Inductive Bias:

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

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

the notation ***y |- z*** *indicates that z follows deductively from y (i.e., that z* is provable from *y). 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.


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 – Three Learning Algorithms

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.

Inductive Bias: No inductive bias

CANDIDATE-ELIMINATION: 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.

Inductive Bias: the target concept can be represented in its hypothesis space.

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.

Inductive Bias: the target concept can be represented in its hypothesis space, and all instances are negative instances unless the opposite is entailed by its other knowledge.