EID403: MACHINE LEARNING

Abhishikta Ummadi

GITAM Hyderabad

July 14, 2019

Outline

Learning Problem

- **Definition**: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.
- For any problem we must identity three features: the class of tasks,T, the measure of performance to be improved,P, and the source of experience,E.
  Well-Posed Learning Problems:
- **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

- Choosing the Training Experience
- Choosing the Target Function
- Choosing a Representation for the Target Function
- Choosing a Function Approximation Algorithm
- The Final Design

1.Choosing the Training Experience

- First step is to choose the type of training experience from which our system will learn.
- It 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 - Individual checkers board state and correct move for every step
  - Indirect feedback- Consisting of move sequences and final outcomes of various games played.
  - Credit assignment for each move which is difficult.
- learning is easy from direct feedback.

1.Choosing the Training Experience(Contd.)

- 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
  - There might be a danger that training experience might not be fully representative of situations which it will later be tested.
- Assumption: Distribution of training is identical to test data.
- Let us decide our system will be trained by plying games against itself

1.Choosing the Training Experience(Contd.)

- 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.
  - Function is difficult to learn given kind of indirect training experience available to our system.

2.Choosing the Target Function(Contd.)

- 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(Contd.)

- 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 : Ideal target 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(Contd.)

- 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(Contd.)

- Our learning program will represent $\hat{V}(b)$ as a linear function of the form

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

where w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.

3.Choosing a Representation for the Target Function(Contd.)

- 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 $->$ R
  - Target function representation:
    $\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$

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)
  $V_{train(b)} = +100$

$$<< x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0 >, +100 >$$

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, $V_{train}(b)$).
- What we mean by best fit to the training data?
- One approach: Define best hypothesis or set of weights to minimize squared error between training values and values predicted by hypothesis $\hat{V}$.
- If E is small $\rightarrow$ most probable hypothesis

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

4.2.ADJUSTING THE WEIGHTS(Contd.)

- 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.

$$\text{For each training example} < b, V_{train}(b) >$$

- use the current weights to calculate $\hat{v}(b)$
- for each weight w_i, update it as
  $w_i \leftarrow w_i + \eta(V_{train}(b) - \hat{v}(b))x_i$

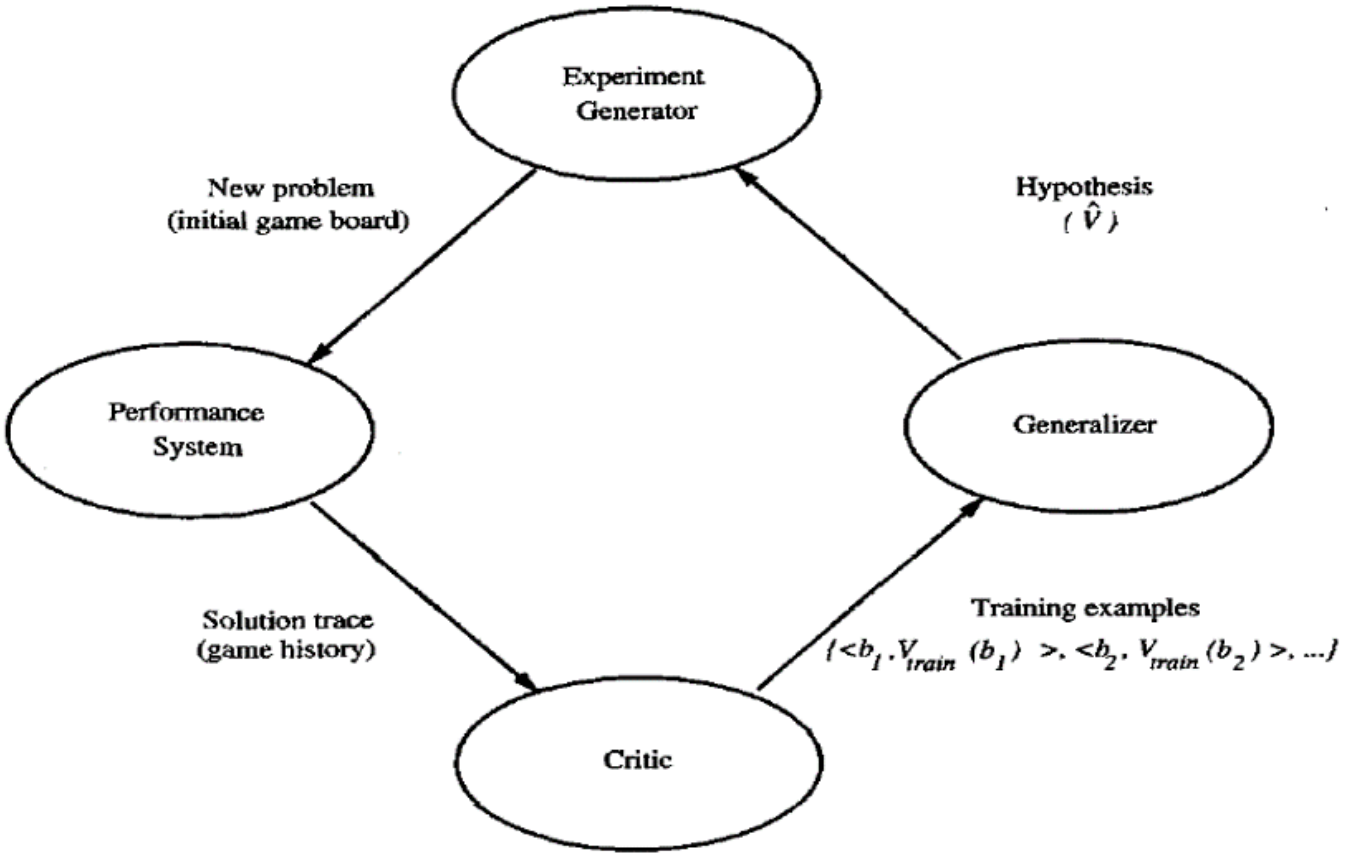
- Here n is a small constant (e.g., 0.1) that moderates the size of the weight update.

5.The Final Design

- The final design of our checkers learning system can be 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)

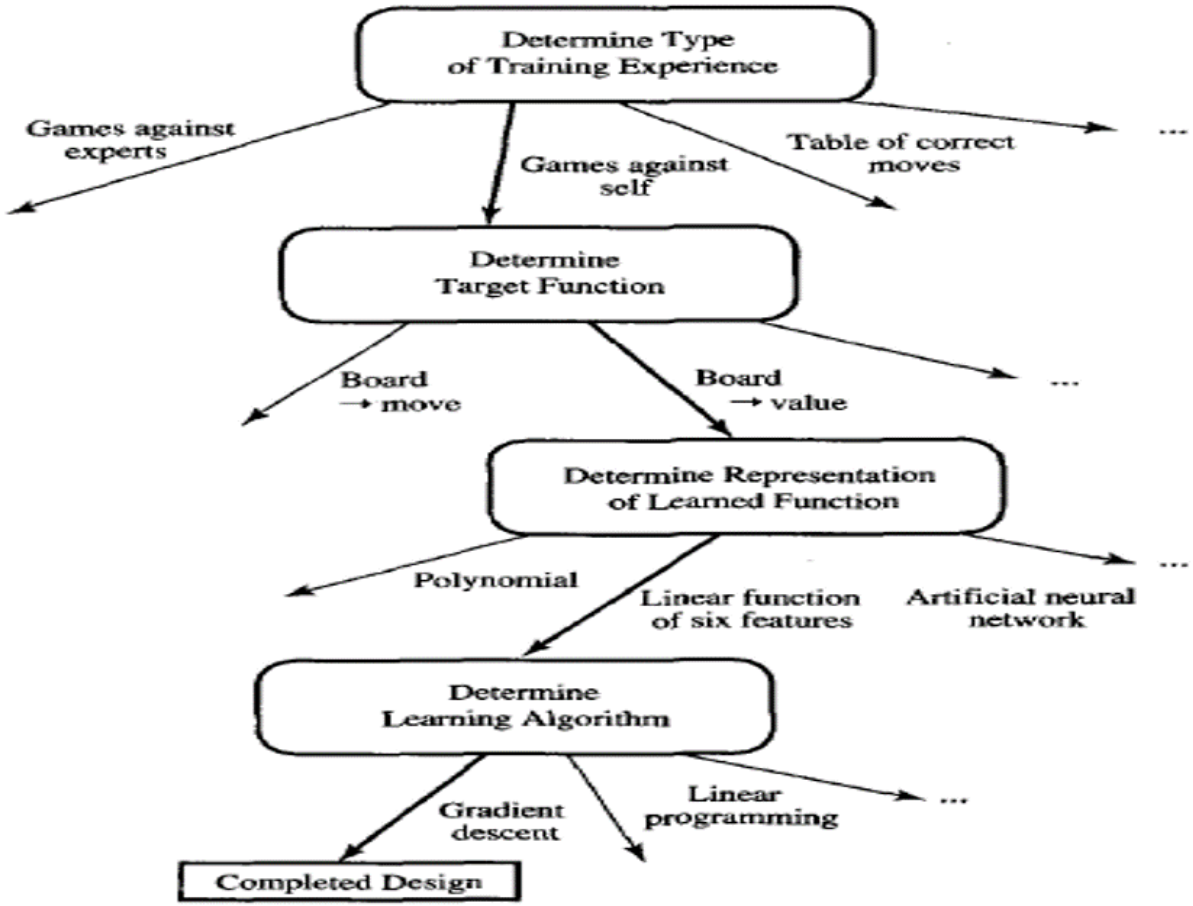
5.The Final Design

Figure 1: Final Design of Checkers Learning Problem



5.The Final Design

Figure 2: Summary of choices in designing the checkers learning problem



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?

Concept Learning Task

- Much of learning involves acquiring general concepts from specific training examples.
- Each general concept can be viewed as describing some subset of objects or events defined over a larger set.
  - Example: Let there be a set of unique instances X.
    $X = \{duck, rabbit, chicken, elephant, planet, money, cow, dog\}$
  - Let there be a set of unique labels L. Where $L = \{0, 1\}$
  - A concept C is a subset of X. e.g. C = mammals=
    $\{rabbit, elephant, cow, dog\}$
- **Concept learning**: Inferring a Boolean-valued function from training examples of its input. [Example: A function that returns 1 (or "true") only for elements in the concept e.g. C(rabbit) = 1, C(duck) = 0].

* Example Task: Learning the concept "Days on which my friend Aldo enjoys his favorite water sport"

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

Figure 3: Set of example days, each represented by a set of attributes

EnjoySport- This attribute indicates whether or not Aldo enjoys water sport on this day.
Task- To Learn to predict the value of EnjoySport for an arbitrary day, based on values of its other attributes.

What hypothesis representation shall we provide to learn in this case?

- Let each hypothesis be vector of six constraints specifying value of six attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast
- For each attribute, 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 "ø" that no value is acceptable.
- For example, to illustrate the hypothesis that Aldo enjoys his favorite sport only on cold days with high humidity is represented by

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

Concept Learning Task

- Most general hypothesis- Every day is a positive example - $<?, ?, ?, ?, ?, ? >$
- Most specific possible hypothesis- No day is a positive example - $< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$

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), "$\emptyset$" (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.

Notations

Set of Instances: Set of items over which concept is defines. Denoted by X.

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

Target Concept: The concept or function to be learned is called the target concept. Denoted by c.

- c can be any Boolean valued function defined over the instances X; i.e.

$$c : X -> \{O, 1\}$$

  In current example, target concept corresponds to value of attribute EnjoySports i.e

- c(x) = 1 if EnjoySport = Yes,
- c(x) = 0 if EnjoySport = No.

Notations

Training Examples:

- When learning target concept, learner is presented a set of training example each consisting of an instance x from X along with target value c(x).
- Instances for which $c(x) = 1$: positive examples or members of the target concept.
- Instances for which $c(x) = 0$: negative examples or nonmembers of the target concept.
- Training example- ordered pair (x, c(x))
- D : denotes the set of available training examples.

Notations

Hypotheses:

- H to denote the set of all possible hypotheses.
- Each hypothesis h in H represents a Boolean-valued function defined over X; i.e.

$$h : X -> \{O, 1\}$$

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

The Inductive Learning Hypothesis

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 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.
- For example, Consider instances X and hypothesis H in EnjoySport learning task. Given that Sky has three possible values and other attributes have two possible values, instance space X contains exactly $3.2.2.2.2.2 = 96$ distinct instances.
- Similar calculations show that there are $5.4.4.4.4.4 = 5120$ syntactically distinct hypotheses within H.(adding ? and $\varnothing$)
- Every hypothesis containing one or more "$\varnothing$" symbols represents the empty set of instances; i.e. it classifies every instance as negative. Therefore, number of Semantically distinct hypothesis is $1+(4.3.3.3.3.3)=973$.

General-to-Specific Ordering of Hypotheses

- We can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis.
- Consider two hypothesis:

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

- h2 classifies more instances as positive(few constraints)
- 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

- more_general_than: 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_to: Given hypotheses h_j and h_k, h_j is more_general_than_or_equal_to h_k if and only if any instance that satisfies h_k also satisfies h_j.
- Definition: Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is more_general_than_or_equal_to h_k (written $h_j \geq g\ h_k$) if and only if

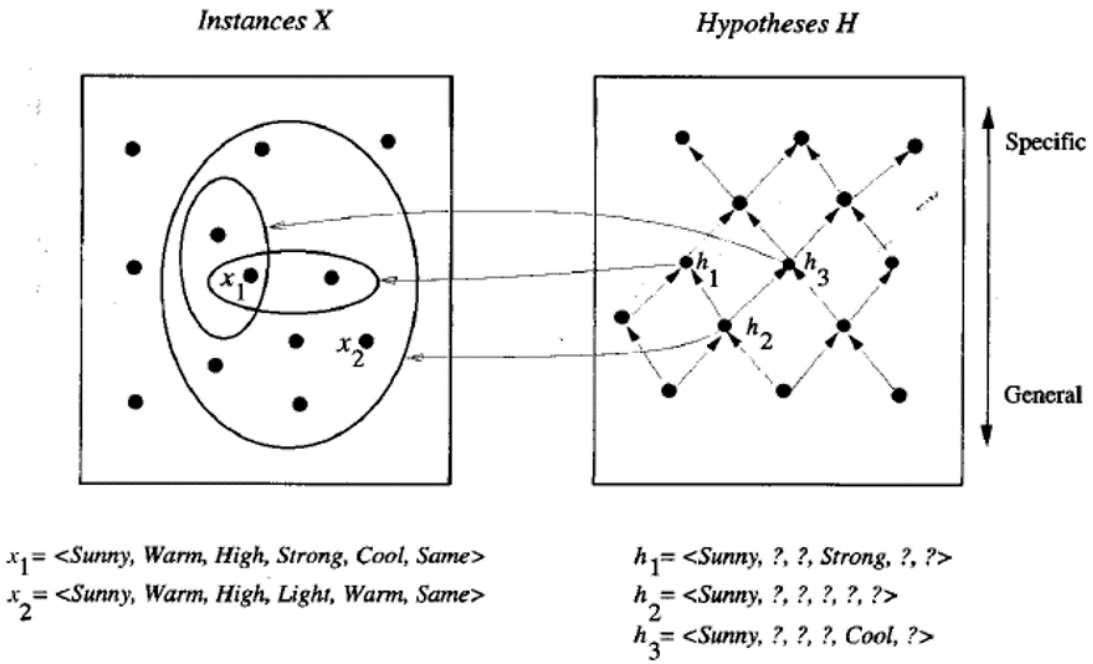
$$(\forall x \epsilon X)[(h_k(x)) = 1 \rightarrow (h_j(x)) = 1]$$

- h_j is (strictly) more_general_than h_k (written $h_j >g\ h_k$) if and only if

$$(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$$

- Some times we say that h_j is more_specific_than h_k when h_k is more_general_than h_j.

General-to-Specific Ordering of Hypotheses Example



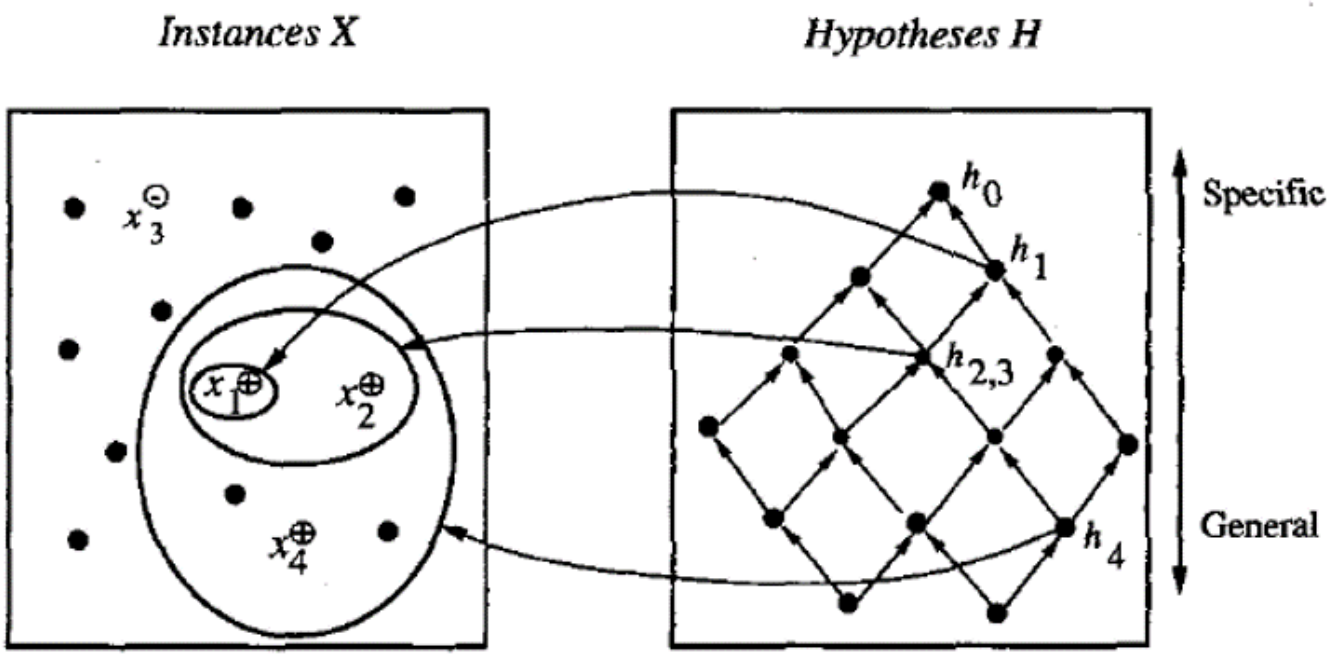
- The first box represents the set of all instances X. Each point is an instance. The circles are hypothesis. The points inside a circle correspond to the instances that hypothesis classifies as positive.
- The second box represents an ordered set of all the possible hypothesis.
- The top will contain the hypothesis that classifies all examples as negative while the bottom as positive.

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS 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

- How can we use $> g$ partial ordering to organize search for a hypothesis consistent with observed training examples?
- One way is to begin with most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example.

Hypothesis Space Search by FIND-S



x_1 = <Sunny Warm Normal Strong Warm Same>, +

x_2 = <Sunny Warm High Strong Warm Same>, +

x_3 = <Rainy Cold High Strong Warm Change>, -

x_4 = <Sunny Warm High Strong Cool Change>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>

h_1 = <Sunny Warm Normal Strong Warm Same>

h_2 = <Sunny Warm ? Strong Warm Same>

h_3 = <Sunny Warm ? Strong Warm Same>

h_4 = <Sunny Warm ? Strong ? ? >

- 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.
- Problems with FIND-S Algorithm:
  - Has the learner converged to the correct target concept?
  - Why prefer the most specific hypothesis?
  - Are the training examples consistent?
  - What if there are several maximally specific consistent hypotheses?

Version Space and The Candidate-Elimination Algorithm

Second approach to concept learning, CANDIDATE-ELIMINATION ALGORITHM that addresses several limitations of Find-S.

- Though Find-S output a hypothesis from H that is consistent with training examples, this is just one of many hypotheses from H that might fit training data equally well.
- **Candidate-Elimination Algorithm** is to output a description of *set of all hypotheses consistent with training examples*. It does without explicitly enumerating all its members.
- It is accomplished by using more_general_than partial ordering.
- Both Find-S and Candidate Elimination algorithm perform poorly when given noisy training data.

Representation

- To define candidate-elimination algorithm, we begin with 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.

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

Candidate-Elimination Algorithm represents set of all hypotheses *consistent* with observed training examples. This subset is called **Version space** w.r.t hypothesis space H and training examples D.

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

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

The List-Then-Eliminate Algorithm

- One way to represent version space is to simply list all of its members.

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 $VersionSpace$

- This algorithm first initializes the version space to contain all hypotheses in H, Then eliminates any hypothesis found inconsistent with any training example.
- List-Then-Eliminate Algorithm can be applied whenever hypothesis space H is finite.

More compact representation for version spaces

- In Candidate Elimination Algorithm, version space is represented by its *most general and least general members*.
- These members form *general and specific boundary sets* that delimit the version space within partially ordered hypothesis space.
- Candidate Elimination Algorithm represents version space by storing only its **most general members, labeled G and most specific, labeled S**.

More compact representation for version spaces

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.

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

- Version space is the set of hypotheses contained in G, plus those in S, plus those that lie between G and S in partial order hypothesis space.
- It is stated using a theorem.

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,

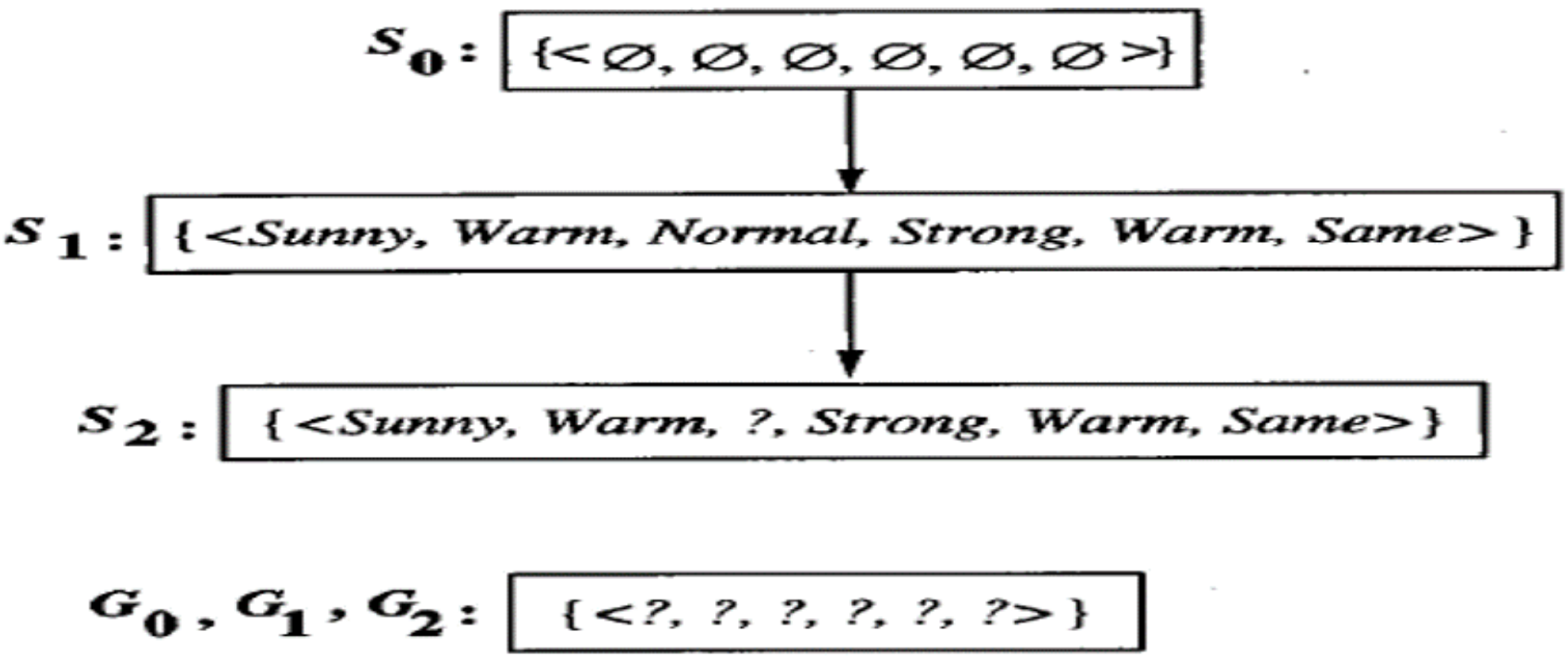
$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s)\}$$

- 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
- $G_0 \leftarrow \{\langle ?, ?, ?, ?, ?, ?\rangle\}$ $S_0 \leftarrow \{\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\rangle\}$

Candidate-Elimination Algorithm

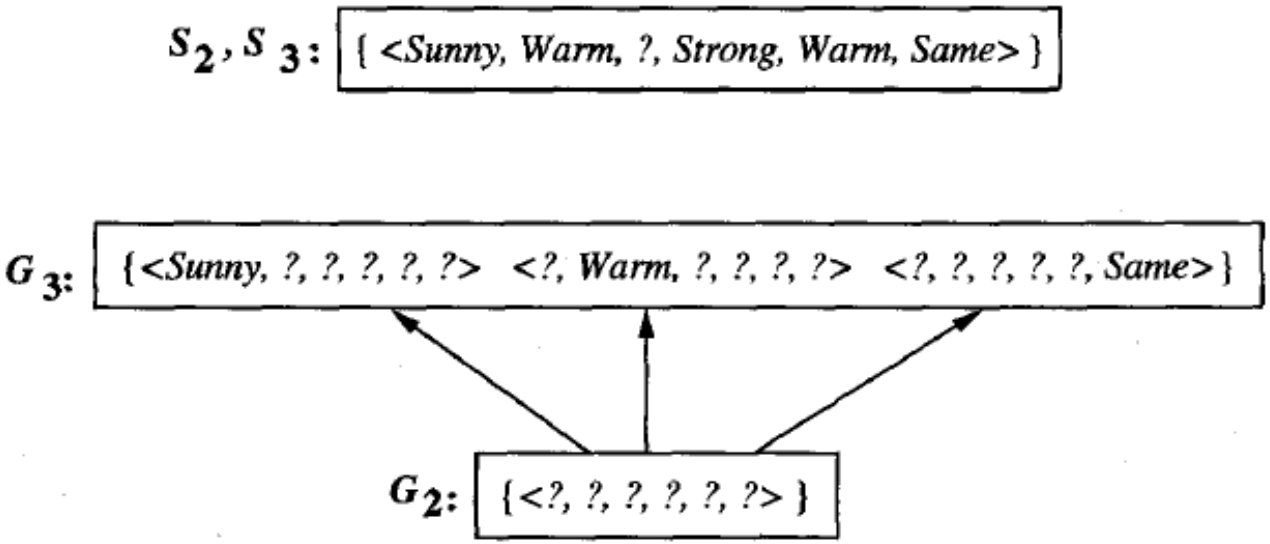
Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

- 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

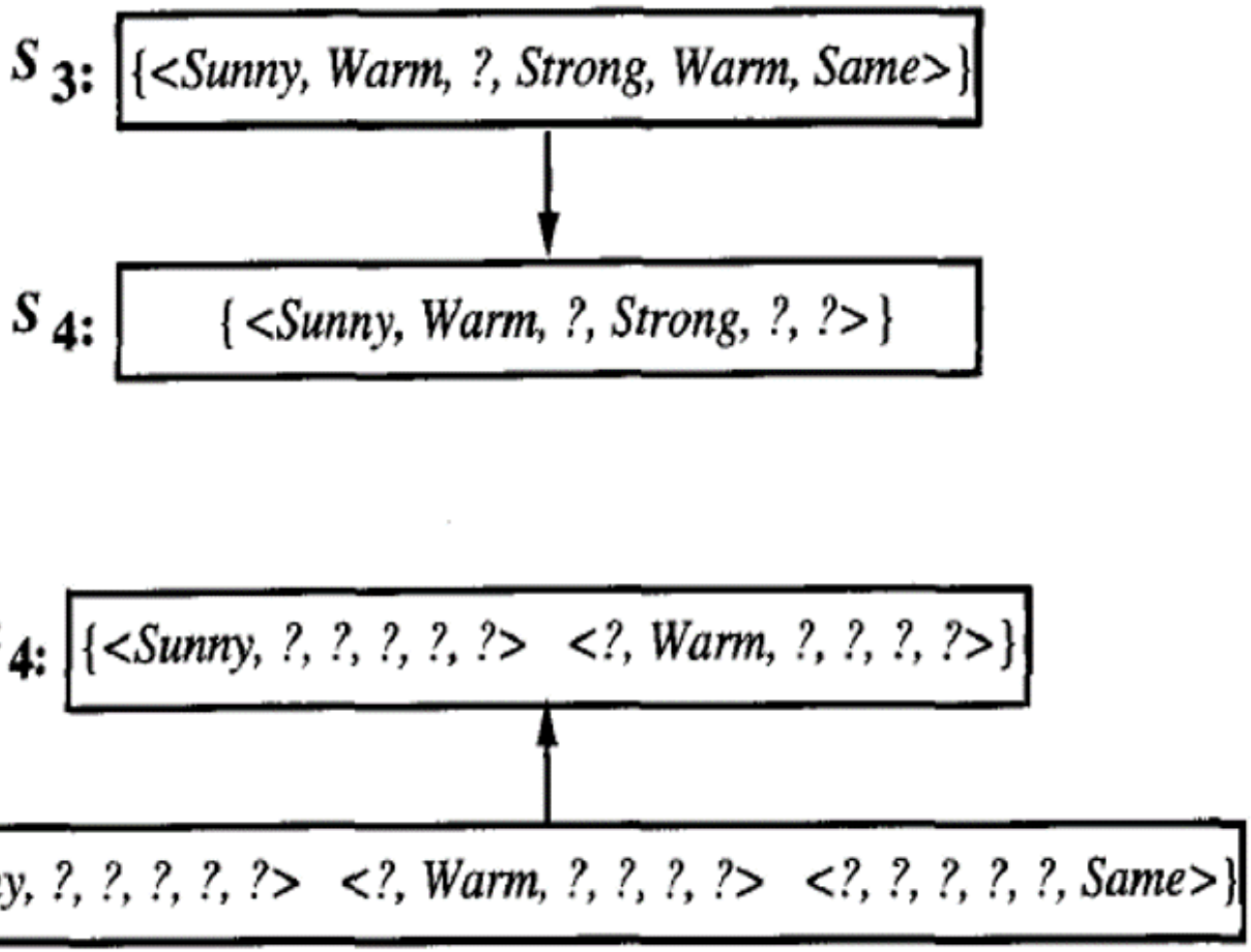


1. <Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Yes

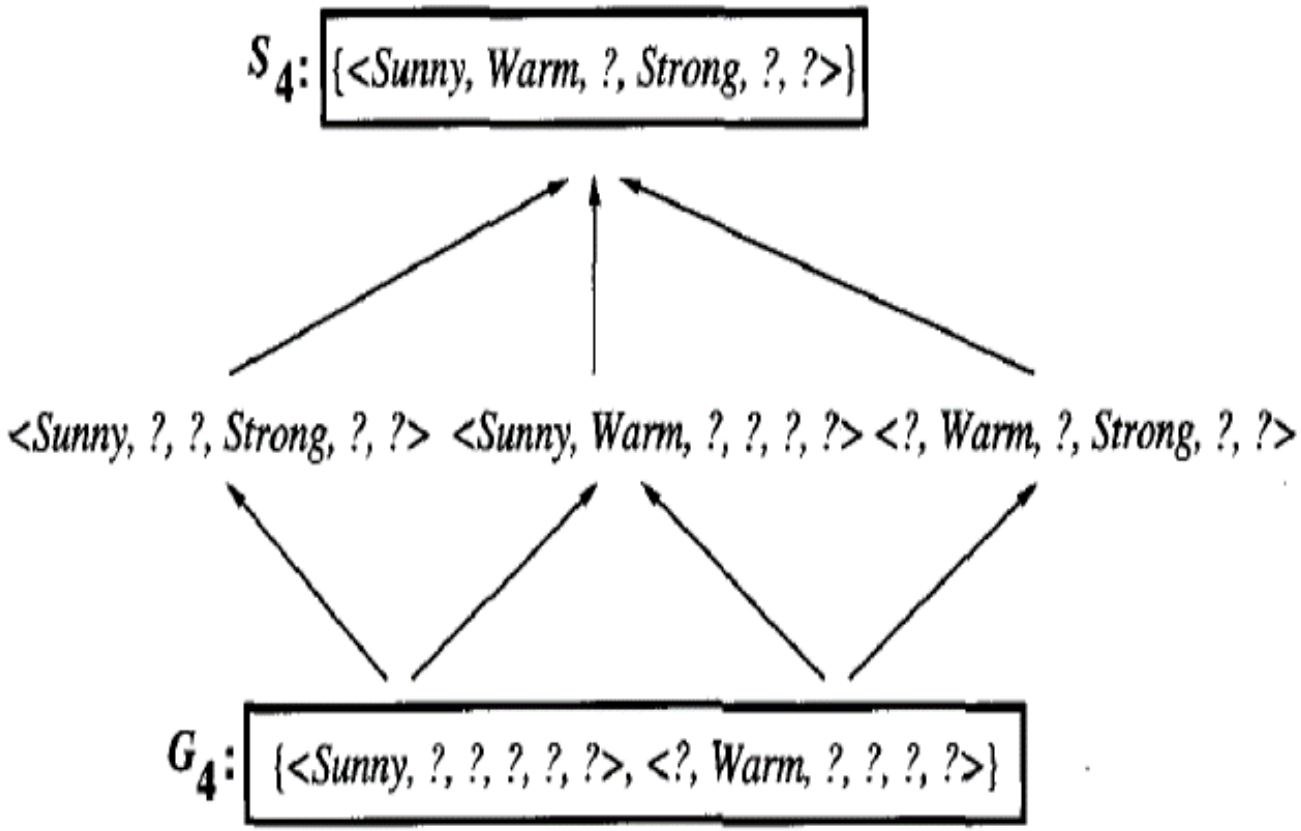
2. <Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes



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



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



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.

Properties

- If there is a consistent hypothesis then the algorithm will converge to S = G = h when enough examples are provided.
- False examples may cause the removal of the correct h.
- If the examples are inconsistent, S and G become empty.
- This can also happen, when the concept to be learned is not in H.

REMARKS ON VERSION SPACES AND CANDIDATE-ELIMINATION

- **1.Will the Candidate-elimination Algorithm Converge to the Correct Hypothesis?**
  Hypothesis that correctly describes the target concept, provided
  - There are no errors in the training examples,
  - There is some hypothesis in H that correctly describes the target concept
- **2.What Training Example Should the Learner Request Next?**
  Learner is allowed to self-construct new instances and obtain correct classifications from an external oracle(e.g. teacher)
  Such instances are referred as *query*.
- **3.How Can Partially Learned Concepts Be Used?**

New instances to be classified

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	?

Classification:

- Classify a new example as positive or negative, if all hypotheses in the version space agree in their classification.
- Otherwise:
  – Rejection or
  – Majority vote

INDUCTIVE BIAS

- The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.
- 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. Fundamental questions for inductive inference:
- What if target concept not contained in hypothesis space?
- Should we include every possible hypothesis? How does this influence the generalisation ability to generalize unobserved instances?
- How does this influence number of training examples that must be observed?

A Biased Hypothesis space

in EnjoySport example,

- Hypothesis space include only conjunctions of attribute values. So, this representation is unable to represent disjunctive target concepts such as "Sky=Sunny or Sky=Cloudy".
- For example, given following training examples of disjunctive hypothesis, our algorithm would find that there are "zero hypothesis" in version space.

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

Most specific Hypothesis consistent with first two examples in give hypothesis space H is

$$S_2 : \langle ?, Warm, Normal, Strong, Cool, Change \rangle$$

- Although it is most specific, it is already overly general. It incorrectly covers 3rd example
- We have biased the learner to consider only conjunctive hypotheses.
- In such cases, we require more expressive hypothesis space.

An Unbiased Learner

- The obvious solution is to provide a hypothesis space capable of representing every teachable concept. i.e. It is capable of representing every possible subset of the instances X.
- Set of all subsets of a set X is called Power Set of X.
- The number of distinct subsets that can be defined over a set X containing $|X|$ elements(ie size of power set of X) is $2^{|X|}$.

Reformulate EnjoySport in unbiased way by defining new hypothesis space H' that can represent every subset of instances:

- One way to define such an H' is to allow arbitrary dis-junctions, conjunctions and negations of our hypothesis.

$$\langle Sunny, ?, ?, ?, ?, ? \rangle \vee \langle Cloudy, ?, ?, ?, ?, ? \rangle$$

This hypothesis space unfortunately raises a new, equally difficult problem: Concept learning algorithm now is completely unable to generalize beyond observed examples.

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

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

  G boundary will consist of hypothesis that rules out only negative examples.

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

- S = $\{s\}$, with s = disjunction of positive examples
- G = $\{g\}$, with g = Negated disjunction of negative examples

The Futility of Bias-Free Learning

Inductively Inferred:

- Consider general setting in which an arbitrary learning algorithm L is provided an arbitrary set of training data $D_c = \{< x, c(x) >\}$ of some arbitrary target concept c.
- After training, L is asked to classify a new instance x_i. Let $L(x_i, D_c)$ denote classification(i.e.positive or negative) that L assigns to x_i after learning from training data D_c.
  - $D_c \wedge x_i \succ L(x_i, D_c)$
    where $y \succ z$ denotes z is inductively inferred from y.
- For example: Let L be candidate elimination algorithm, D_c be training data of EnjoySport, x_i be first instance from data.
  $L(x_i, D_c) = (EnjoySport = yes)$

Follow Deductively:

- Consider a concept learning algorithm L for set of training 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 classification assigned to instance x_i by L after training on 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

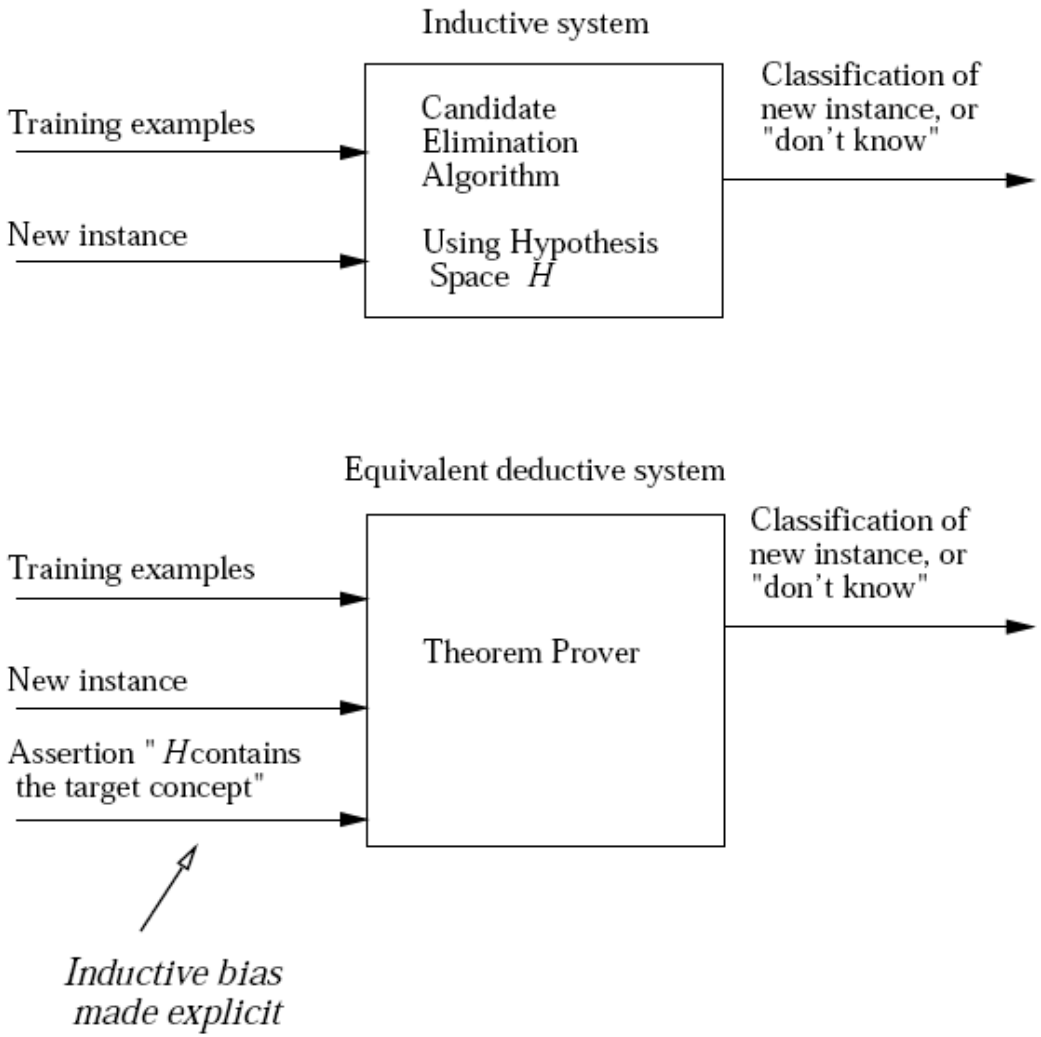
$$(\forall x_i \epsilon X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

where $y \vdash z$ indicates that z follows deductively from y. (i.e. z is provable from y.)

Inductive Bias for Candidate Elimination

- Assume instance x_i and training set D_c
- The algorithm computes the version space.
- x_i is classified by unanimous voting (using the instances in the version space)
- This way $L(x_i, D_c)$ is computed.
- Now assume that the underlying concept c is in H. This means that c is a member of its version space.
- EnjoySport = k implies that all members of VS, including c vote for class k. Because unanimous voting is required, $k = c(x_i)$.
- This is also the output of the algorithm $L(x_i, D_c)$
- *The inductive bias of the Candidate Elimination Algorithm is:* **c is in H:** $c \epsilon H$.

Inductive Systems and Equivalent Deductive Systems



Three Learners with Different Biases

- Three Learning algorithms, which are listed from weakest to strongest bias:
  - ROTE LEARNER: Stores examples in memory, Classify x iff it matches previously observed examples. Otherwise, system refuses to classify new instances.
  - Candidate elimination algorithm: c is in H.
  - Find-S: c is in H and that all instances are negative examples unless opposite is entailed by its training data.
- A good generalisation capability of course depends on the appropriate choice of the inductive bias!