UNIT-3

Well Posed Learning Problem – A computer program is said to learn from experience E in context to some task T and some performance measure P, if its performance on T, as was measured by P, upgrades with experience E.
Any problem can be segregated as well-posed learning problem if it has three traits –

- Task
- Performance Measure
- Experience

Certain examples that efficiently defines the well-posed **learning problem are –**

1. To better filter emails as spam or not

- Task – Classifying emails as spam or not
- Performance Measure – The fraction of emails accurately classified as spam or not spam
- Experience – Observing you label emails as spam or not spam

2. A checkers learning problem

- Task – Playing checkers game
- Performance Measure – percent of games won against opposer
- Experience **–** playing implementation games against itself

3. Handwriting Recognition Problem

- Task – Acknowledging handwritten words within portrayal
- Performance Measure – percent of words accurately classified
- Experience – a directory of handwritten words with given classifications

4. A Robot Driving Problem

- Task – driving on public four-lane highways using sight scanners
- Performance Measure – average distance progressed before a fallacy
- Experience – order of images and steering instructions noted down while observing a human driver

5. Fruit Prediction Problem

- Task – forecasting different fruits for recognition
- Performance Measure – able to predict maximum variety of fruits
- Experience – training machine with the largest datasets of fruits images

6. Face Recognition Problem

- Task – predicting different types of faces
- Performance Measure – able to predict maximum types of faces
- Experience – training machine with maximum amount of datasets of different face images

7. Automatic Translation of documents

- Task – translating one type of language used in a document to other language
- Performance Measure – able to convert one language to other efficiently
- Experience – training machine with a large dataset of different types of languages

Design of a learning system:

Just now we looked into the learning process and also understood the goal of the learning. When we want to design a learning system that follows the learning process, we need to consider a few design choices.

The design choices will be to decide the following key components:

1. Type of training experience

2. Choosing the Target Function

3. Choosing a representation for the Target Function

4. Choosing an approximation algorithm for the Target Function

5. The final Design

We will look into the game - checkers learning problem and apply the above design choices. For a checkers learning problem, the three elements will be,

• Task T: To play checkers

• Performance measure P: Total present of the game won in the tournament.

• Training experience E: A set of games played against itself

Type of training experience:

During the design of the checker's learning system, the type of training experience available for a learning system will have a significant effect on the success or failure of the learning.

Direct or Indirect training experience:

In the case of direct training experience, an individual board states and correct move for each board state are given. In case of indirect training experience, the move sequences for a game and the final result (win, lose or draw) are given for a number of games. How to assign credit or blame to individual moves is the credit assignment problem.

1. **Teacher or Not:**

 Supervised:

The training experience will be labelled, which means, all the board states will be labelled with the correct move. So the learning takes place in the presence of a supervisor or a teacher.

 Un-Supervised:

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

 Semi-supervised:

Learner generates game states and asks the teacher for help in finding the correct move if the board state is confusing.

2. **Is the training experience good:**

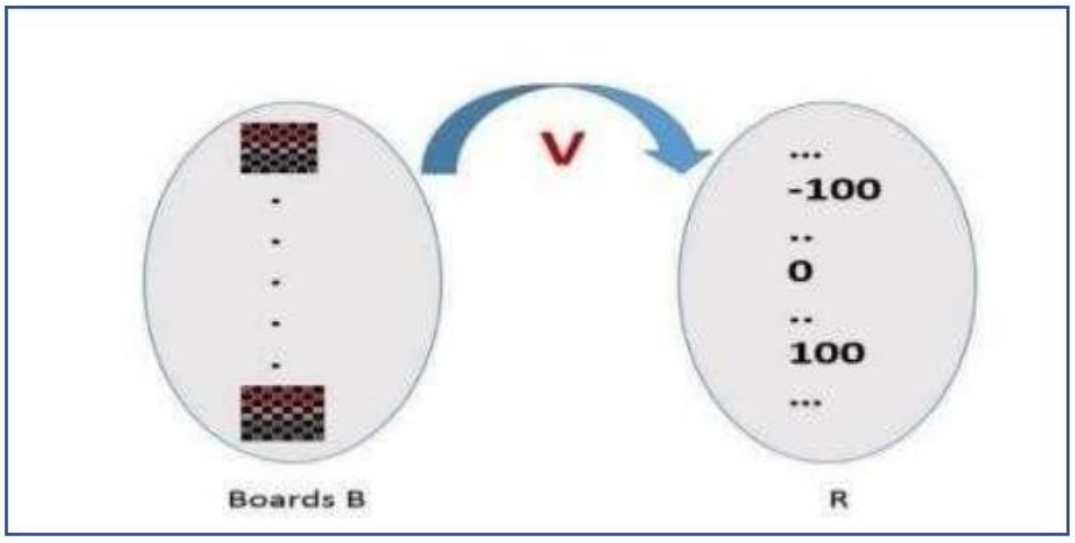
Do the training examples represent the distribution of examples over which the final system performance will be measured? Performance is best when training examples and test examples are from the same/a similar distribution.

3. The checker player learns by playing against oneself. Its experience is indirect. It may not encounter moves that are common in human expert play. Once the proper training experience is available, the next design step will be choosing the Target Function.

Choosing the Target Function:
When you are playing the checkers game, at any moment of time, you make a decision on choosing the best move from different possibilities. You think and apply the learning that you have gained from the experience. Here the learning is, for a specific board, you move a checker such that your board state tends towards the winning situation. Now the same learning has to be defined in terms of the target function.
Here there are 2 considerations — direct and indirect experience.
• **During the direct experience** the checkers learning system, it needs only to learn how to choose the best move among some large search space. We need to find a target function that will help us choose the best move among alternatives.

Let us call this function Choose Move and use the notation Choose Move: B
→M to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
• **When there is an indirect experience it** becomes difficult to learn such function. How about assigning a real score to the board state.

So the function be V: B →R indicating that this accepts as input any board from the set of legal board states B and produces an output a real score. This function assigns the higher scores to better board states



If the system can successfully learn such a target function V, then it can easily use it to select the best move from any board position.
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.
The (4) is a recursive definition and to determine the value of V(b) for a particular board state, it performs the search ahead for the optimal line of play, all the way to the end of the game. So this definition is not efficiently computable by our checkers playing program, we say that it is a non-operational definition.

Choosing a representation for the Target Function:

Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the function ^V that it will learn. As with earlier design choices, we again have many options. We could, for example, allow the program to represent using a large table with a distinct entry specifying the value for each distinct board state. Or we could allow it to represent 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. In general, this choice of representation involves a crucial trade off. On one hand, we wish to pick a very expressive representation to allow representing as close an approximation as possible to the ideal target function V.
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 ^V will be calculated as a linear combination of the following board features:

• x1(b) — number of black pieces on board b

• x2(b) — number of red pieces on b

• x3(b) — number of black kings on b

• x4(b) — number of red kings on b

• x5(b) — number of red pieces threatened by black • x6(b) — number of black pieces threatened by red

^V = w0 + w1 · x1(b) + w2 · x2(b) + w3 · x3(b) + w4 · x4(b) +w5 · x5(b) + w6 · x6(b)

Where w0 through w6 are numerical coefficients or weights to be obtained by a learning algorithm. Weights w1 to w6 will determine the relative importance of different board features.

Specification of the Machine Learning Problem at this time: Till now we worked on choosing the type of training experience, choosing the target function and its representation. The checkers learning task can be summarized as below.

• Task T: Play Checkers

• Performance Measure: % of games won in world tournament

• Training Experience E: opportunity to play against itself

• Target Function: V: Board → R

• Target Function Representation:

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

The first three items above correspond to the specification of the learning task, where as the final two items constitute design choices for the implementation of the learning program.

Choosing an approximation algorithm for the Target Function:

Generating training data — To train our learning program, we need a set of training data, each describing a specific board state b and the training value V_train (b) for b. Each training example is an ordered pair <b,v_train(b)>.

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

 V_train(b) ← ^V(Successor(b))

Final Design for Checkers Learning system:

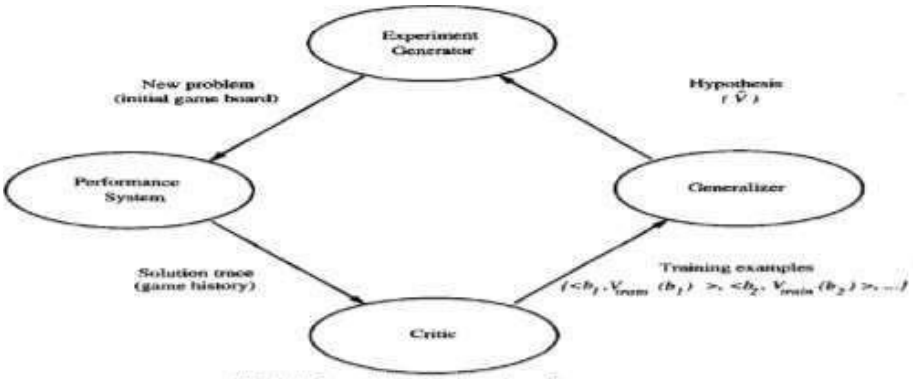
The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems.

1. The performance System: Takes a new board as input and outputs a trace of the game it played against itself.

2. The Critic: Takes the trace of a game as an input and outputs a set of training examples of the target function.

3. The Generalizer: Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.

4. The Experiment Generator: Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.



Final design of the checkers learning program.

Issues in Machine Learning:

Our checkers example raises a number of generic questions about machine learning. The field of machine learning, and much of this book, 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?

Concept Learning

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

Learning involves acquiring general concepts from specific training examples. Example: People continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.

 Each such concept can be viewed as describing some subset of objects or events defined over a larger set

 Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

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

A CONCEPT LEARNING TASK

Consider the example task of learning the target concept "Days on which *Aldo* enjoyshis favorite water sport”

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

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

The task is to learn to predict the value of ***EnjoySport*** for an arbitrary day, based on the values of its other attributes?

What hypothesis representation is provided to the learner?

 Let's consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.

 Let each hypothesis be a vector of six constraints, specifying the values of the six attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*.

For each attribute, the hypothesis will either

 Indicate by a "?' that any value is acceptable for this attribute,

 Specify a single required value (e.g., Warm) for the attribute, or

 Indicate by a "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***).

The hypothesis that ***PERSON*** enjoys his favorite sport only on cold days with high humidity is represented by the expression

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

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

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

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

(Φ, Φ, Φ, Φ, Φ, Φ)

Notation

 The set of items over which the concept is defined is called the ***set of instances***, which is denoted by X.

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 is denoted by c. c can be any Boolean valued function defined over the instances X

c: X→ {O, 1}

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).
 Instances for which c(x) = 1 are called ***positive examples***, or members of the target concept.

 Instances for which c(x) = 0 are called ***negative examples***, or non-members of the target concept.

 The ordered pair (x, c(x)) to describe the training example consisting of the instance x and its target ***concept value c(x).***

 D to denote the set of available training examples

 The symbol ***H*** to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis ***h*** in ***H*** represents a Boolean- valued function defined over ***X***

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.
 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),
 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), “Φ” (no value is acceptable), or a specific value.

 Target concept *c*: *EnjoySport* : X → {0, l}
 Training examples *D*: Positive and negative examples of the target function

 Determine:
 A hypothesis h in H such that *h(x) = c(x)* for all *x* in X.

Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

CONCEPT LEARNING AS SEARCH

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

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

Example:

Consider the instances X and hypotheses H in the *EnjoySport* learning task. The attribute Sky has three possible values, and *AirTemp, Humidity, Wind, Water, Forecast* each have two 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.

Every hypothesis containing one or more "Φ" symbols represents the empty set of instances; that is, it classifies every instance as negative. 1 + (4.3.3.3.3.3) = 973. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

h1 = (Sunny, ?, ?, Strong, ?, ?)

h2 = (Sunny, ?, ?, ?, ?, ?)

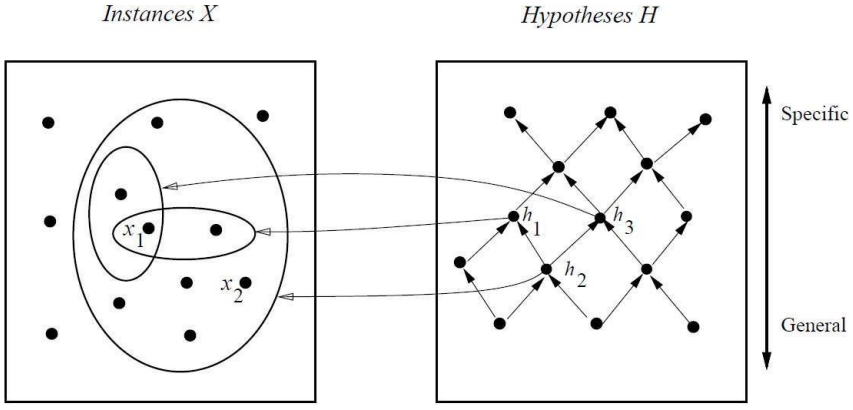
 Consider the sets of instances that are classified positive by hl and by h2.

 h2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by hl will also be classified positive by h2. Therefore, h2 is more general than hl.

Given hypotheses h_j and h_k, h_j is more-general-than or- equal do h_k if and only if any instance that satisfies h_k also satisfies hi

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 $h_j \geq h_k$) if and only if

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



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

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

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

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

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

In the figure, 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 h2 subsumes the subset characterized by hl , hence h2 is more - general– than h1

Find -S Algorithm: Finding a maximally specific Hypothesis

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 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

 The first step of FIND-S is to initialize h to the most specific hypothesis in H
h - (Ø, Ø, Ø, Ø, Ø, Ø)

1. **Consider the first training example**

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

Observing the first training example, it is clear that hypothesis *h* is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next *more general constraint* that fits the example

h1 = <Sunny Warm Normal Strong Warm Same>

2. Consider the second training example

x2 = <Sunny, Warm, High, Strong, Warm, Same>, +

The second training example 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

h2 = <Sunny Warm ? Strong Warm Same>

3. Consider the third training example

x3 = <Rainy, Cold, High, Strong, Warm, Change>, -

Upon encountering the third training the algorithm makes no change to h. The FIND-S algorithm simply ignores every negative example.

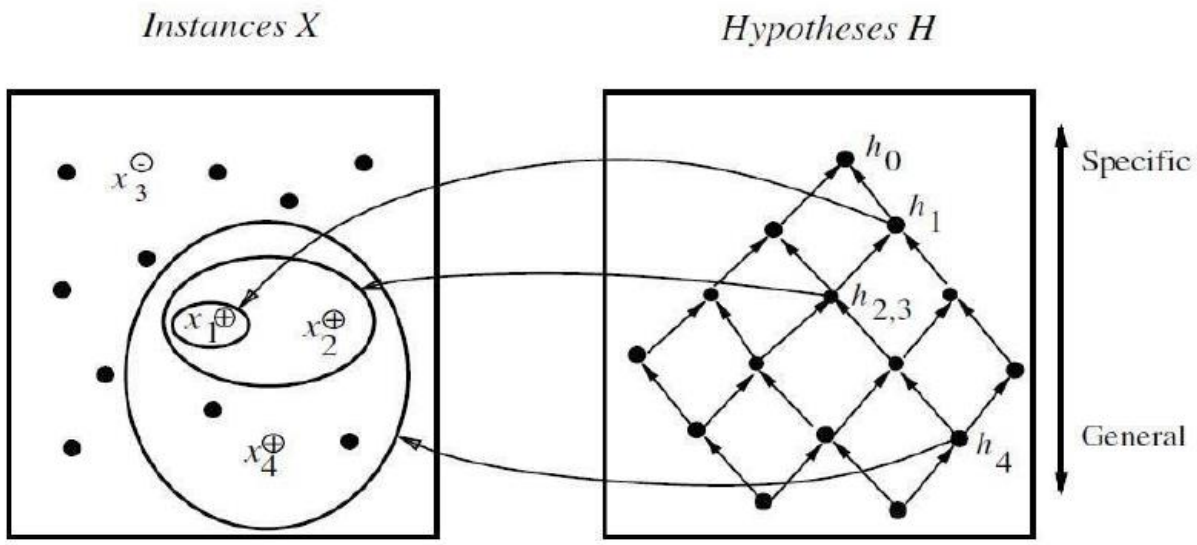
h3 = < Sunny Warm ? Strong Warm Same>

4. Consider the fourth training example

x4 = <Sunny Warm High Strong Cool Change>, +

The fourth example leads to a further generalization of h

h4 = < Sunny Warm ? Strong ? ? >



$h_0 = \langle \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$

$x_1 = \langle Sunny\ Warm\ Normal\ Strong\ Warm\ Same \rangle, +$ $h_1 = \langle Sunny\ Warm\ Normal\ Strong\ Warm\ San$

$x_2 = \langle Sunny\ Warm\ High\ Strong\ Warm\ Same \rangle, +$ $h_2 = \langle Sunny\ Warm\ ?\ Strong\ Warm\ Same \rangle$

$x_3 = \langle Rainy\ Cold\ High\ Strong\ Warm\ Change \rangle, -$ $h_3 = \langle Sunny\ Warm\ ?\ Strong\ Warm\ Same \rangle$

$x_4 = \langle Sunny\ Warm\ High\ Strong\ Cool\ Change \rangle, +$ $h_4 = \langle Sunny\ Warm\ ?\ Strong\ ?\ ? \rangle$

The key property of the FIND-S algorithm

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

2. FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct.

Limitations/Questions still Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all ***hypotheses consistent with the training examples***

Surprisingly, the CANDIDATE-ELIMINATION ALGORITHM computes the description of this set without explicitly enumerating all of its members. This is accomplished by again using the ***more-general-than*** partial ordering, this time to maintain a compact representation of the set of consistent hypotheses and to incrementally refine this representation as each new training example is encountered.

The CANDIDATE-ELIMINATION ALGORITHM has been applied to problems such as learning regularities in chemical mass spectroscopy and learning control rules for heuristic search

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.

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

Note difference between definitions of *consistent* and *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.

 An example *x* is said to ***consistent*** with hypothesis ***h*** iff ***h(x)*** = ***c(x)***

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-ELIMINATlON 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-THENELIMINATION Algorithm

1. ***VersionSpace c*** a list containing every hypothesis in H
2. For each training example, (x, c(x))
   remove from ***VersionSpace*** any hypothesis h for which h(x) ≠ c(x)
3. Output the list of hypotheses in ***VersionSpace***

The LIST-THEN-ELIMINATE Algorithm

 List-Then-Eliminate works in principle, so long as version space is finite.

 However, since it requires exhaustive enumeration of all hypotheses in practice it is not feasible.

A More Compact Representation for Version Spaces

The 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 the partially ordered hypothesis space.

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.

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

As long as the sets ***G*** and ***S*** are well defined 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.

Version Space Representation Algorithm

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 →{O, 1} be an arbitrary target concept defined over

X, and let D be an arbitrary set of training examples {(x, c(x))). For all X, H, c, and D such that S and G are well defined,

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

Proof. To prove the theorem it suffices to show that

1. Every h satisfying the right hand side of the above expression is in $\boldsymbol{VS_{H,D}}$

2. Every member of $\boldsymbol{VS_{H,D}}$ satisfies the right-hand side of the expression

Sketch of proof:

1. let g, h, s be arbitrary members of G, H, S respectively with $g \geq_g h \geq_g s$
   - By the definition of ***S,*** s must be satisfied by all positive examples in D. Because $h \geq_g s$, h must also be satisfied by all positive examples in D.
   - By the definition of ***G,*** g cannot be satisfied by any negative example in D, and because $g \geq_g h$ h cannot be satisfied by any negative example in D. Because h is satisfied by all positive examples in D and by no negative examples in D, h is consistent with D, and therefore h is a member of $VS_{H,D}$.
2. It can be proven by assuming some h in $VS_{H,D}$, that does not satisfy the right-hand side of the expression, then showing that this leads to an inconsistency

Candidate Elimination Learning Algorithm:

The Candidate Elimination Algorithm 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

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

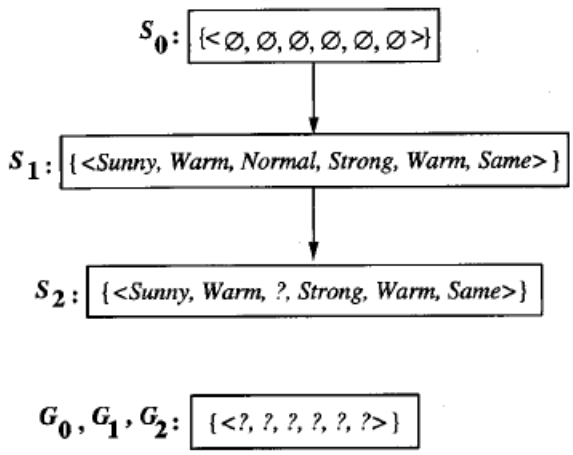
These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than ***So*** and more specific than ***Go.*** As each training example is considered, the ***S*** and ***G*** boundary sets are generalized and specialized, respectively, to eliminate 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.

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

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

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

An Illustrative Example:



Training examples:

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

FIGURE 2.4
CANDIDATE-ELIMINATION Trace 1. S_0 and G_0 are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the S boundary to become more general, as in the FIND-S algorithm. They have no effect on the G boundary.

CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H

G_0 ⟨?, ?, ?, ?, ?, ?⟩

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

S_0 ⟨∅, ∅, ∅, ∅, ∅, ∅⟩

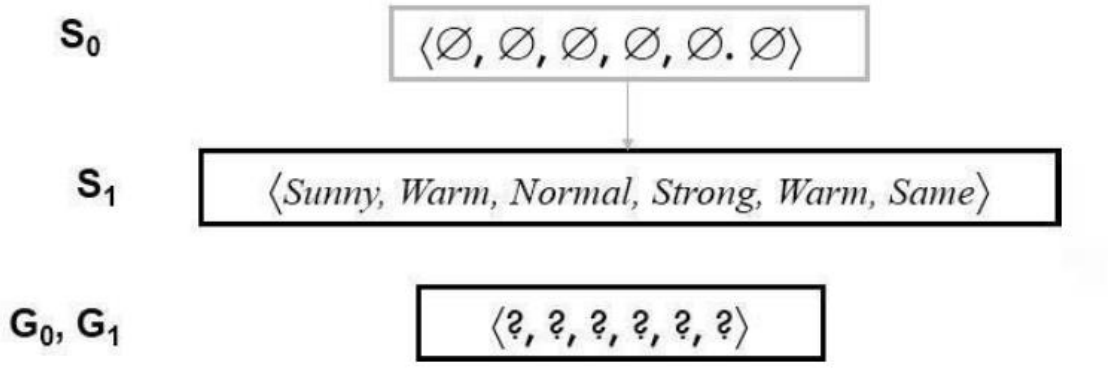
• When the first training example is presented, the CANDIDATE-ELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.

 The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example

 No update of the G boundary is needed in response to this training example because G_o correctly covers this example

For training example d,

⟨Sunny, Warm, Normal, Strong, Warm, Same ⟩ +

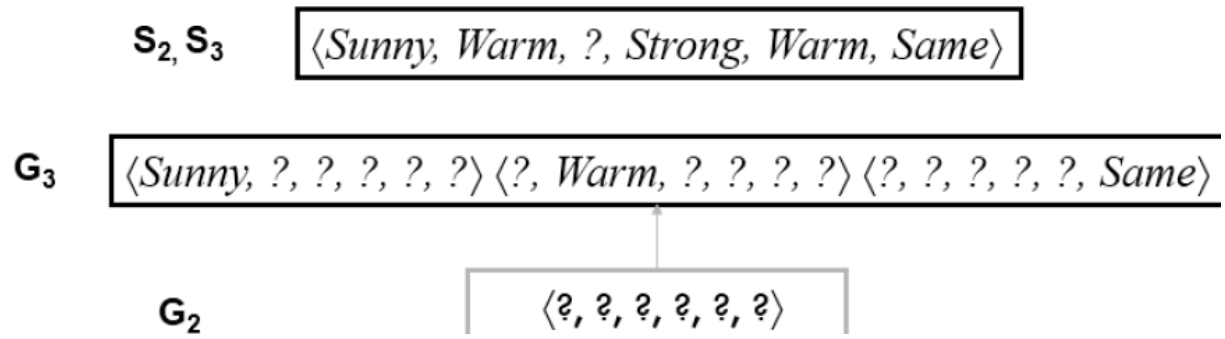


• When the second training example is observed, it has a similar effect of generalizing S further to S_2, leaving G again unchanged i.e., $G_2 = G_1 = G_0$

• Consider the third training example. 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

For training example d, ⟨Rainy, Cold, High, Strong, Warm, Change⟩ –

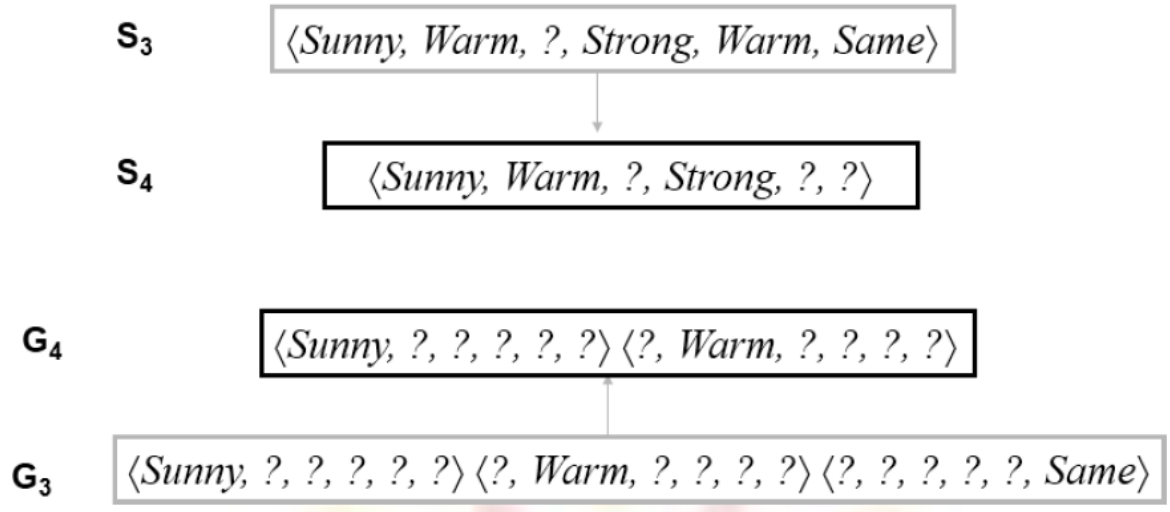


Given that there are six attributes that could be specified to specialize G_2, why are there only three new hypotheses in G_3?

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.

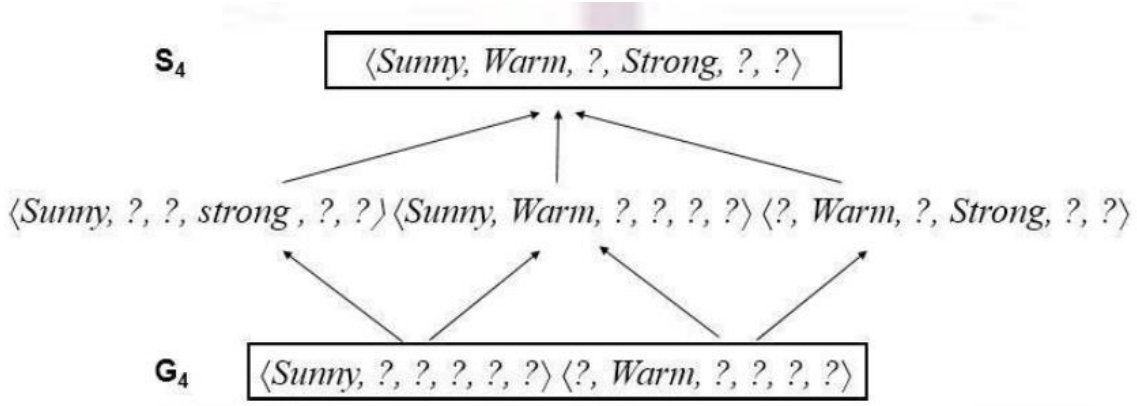
- Consider the fourth training example.

For training example d, ⟨Sunny, Warm, High, Strong, Cool Change⟩ +



- This positive example 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

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.



INDUCTIVE BIAS

The fundamental questions for inductive inference

1. What if the target concept is not contained in the hypothesis space?
2. Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?
3. How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?
4. How does the size of the hypothesis space influence the number of training examples that must be observed?

These fundamental questions are examined in the context of the CANDIDATE-ELIMINTION algorithm

A Biased Hypothesis Space

- Suppose the target concept is not contained in the hypothesis space *H*, then obvious solution is to enrich the hypothesis space to include every possible hypothesis.
- Consider the ***EnjoySport*** example in which the hypothesis space is restricted 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."

- The following three training examples of disjunctive hypothesis, the algorithm would find that there are zero hypotheses in the version space

⟨Sunny Warm Normal Strong Cool Change⟩	Y
⟨Cloudy Warm Normal Strong Cool Change⟩	Y
⟨Rainy Warm Normal Strong Cool Change⟩	N

- If Candidate Elimination algorithm is applied, then it end up with empty Version Space. After first two training example

  S= ⟨? Warm Normal Strong Cool Change⟩

- This new hypothesis is overly general and it incorrectly covers the third negative training example! So H does not include the appropriate c.
- In this case, a more expressive hypothesis space is required.

An Unbiased Learner

- The 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 representing every possible subset of the instances X.
- The set of all subsets of a set X is called the power set of X
- In the ***EnjoySport*** learning task the size of the instance space X of days described by the six attributes is 96 instances.
- Thus, there are 2^{96} distinct target concepts that could be defined over this instance space and learner might be called upon to learn.
- The conjunctive hypothesis space is able to represent only 973 of these - a 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
- The target concept "Sky = Sunny or Sky = Cloudy" could then be described as

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

The Futility of Biased Learning

Inductive inference states that:

A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances. In fact, the only reason that the CANDIDATE-ELIMINAaTlgIOorNith m 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

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

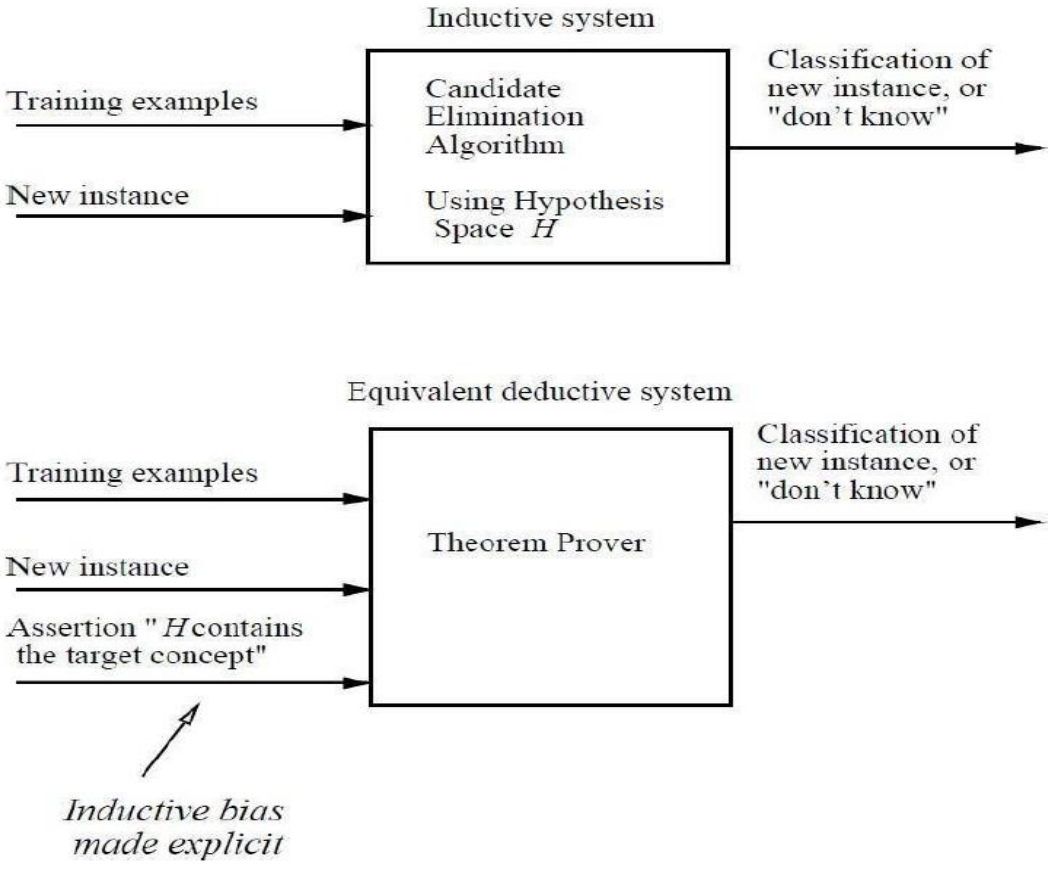
$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)] \quad (2.1)$$

The below figure explains

 Modelling 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 modelling 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.