Unit-V

Analytical Learning, Combining Inductive and Analytical Learning

- Analytical Learning
  - Introduction
  - Learning with Perfect Domain theories : PROLOG-EBG
  - Remarks on Explanation-Based Learning
  - Explanation-Based Learning of Search Control Knowledge
- Combining Inductive and Analytical Learning
  - Motivation
  - Inductive-Analytical Approaches to Learning
  - Using Prior Knowledge to Initialize the Hypothesis
  - Using Prior Knowledge to Alter the Search Objective
  - Using Prior Knowledge to Augment Search Operators

Analytical Learning - Introduction

- We have seen a variety of inductive learning methods like Decision tree learning, Neural Networks, inductive logic programming, and genetic algorithms.
- Inductive learners perform poorly when insufficient data is available.
- We need learning methods that are not subject to these fundamental bounds on learning accuracy imposed by the amount of training data available.
- If we want such methods we need to reconsider the formulation of the learning problem itself.
- We need learning algorithms that accept explicit prior knowledge as input in addition to the input training data.

Analytical Learning - Introduction

- Explanation-based learning is one such approach.
- Explanation-based learning uses prior knowledge to reduce the complexity of the hypothesis space to be searched, thereby reducing sample complexity and improving generalization accuracy of the learner.
- Let us consider the task of learning to play chess.
- Human beings can learn such target concepts using just a few examples.

Analytical Learning - Introduction

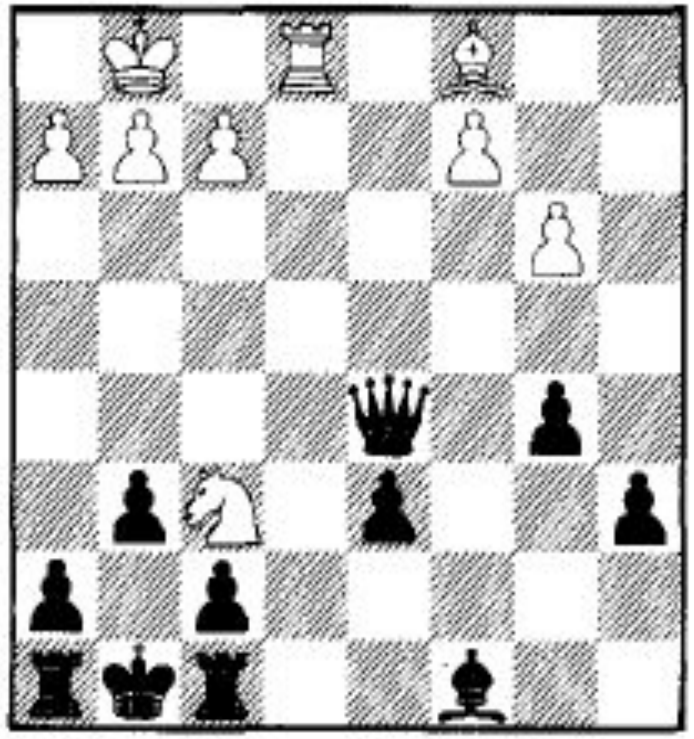


FIGURE 11.1
A positive example of the target concept "chess positions in which black will lose its queen within two moves." Note the white knight is simultaneously attacking both the black king and queen. Black must therefore move its king, enabling white to capture its queen.

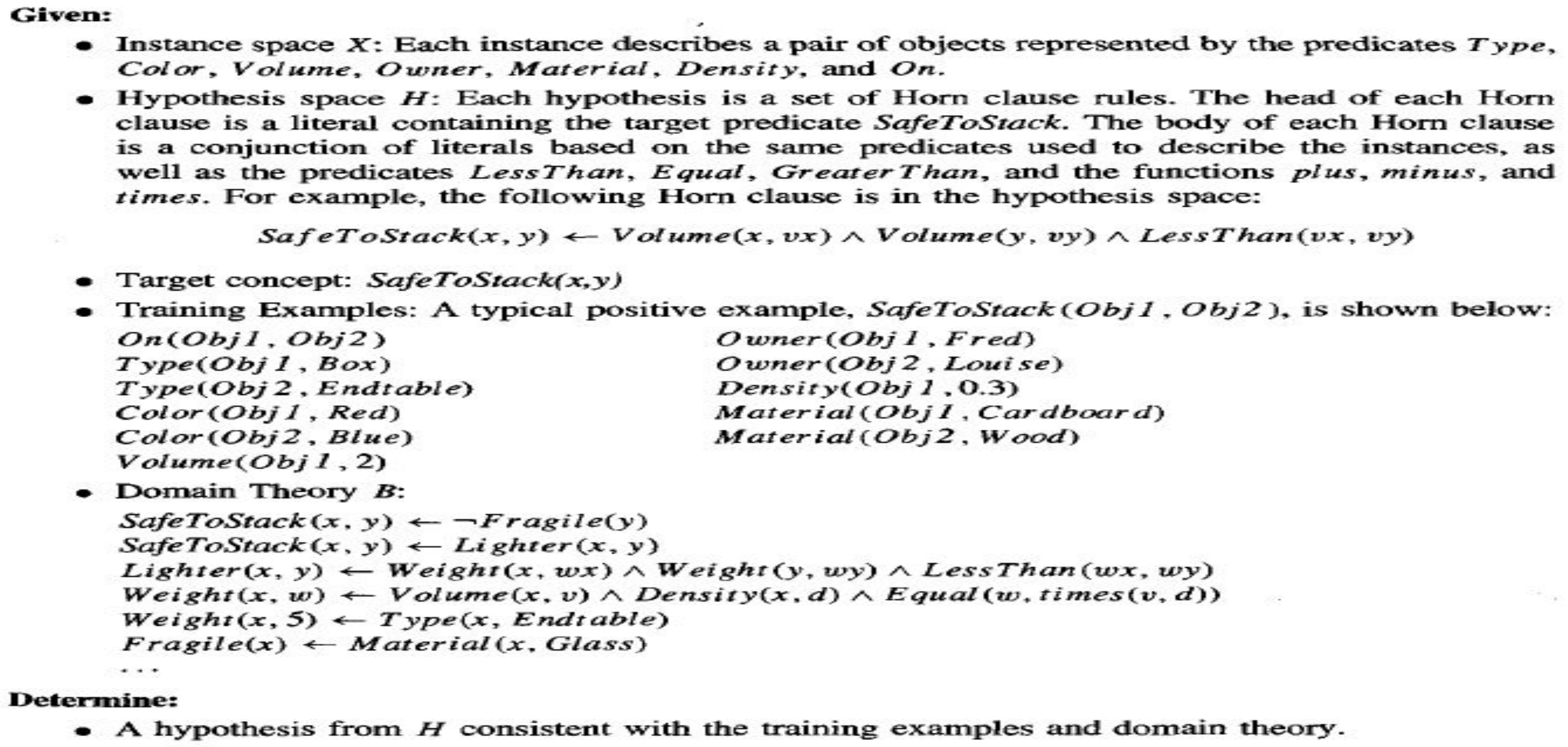
Analytical Learning - Introduction

- What exactly is the prior knowledge needed by a learner to construct the explanation in this chess example?
- It is the knowledge about the rules of the chess game as far as which are legal moves in a game of chess.
- Given just this prior knowledge it is possible ***in principle*** to calculate the optimal chess move for any board position.
- In practice this calculation can be frustratingly complex and despite the fact that we humans ourselves possess this complete, perfect knowledge of chess, we remain unable to play the game optimally.
- Here we describe learning algorithms that automatically construct and learn from such explanations.

Inductive and Analytical Learning Problems

- The essential difference between analytical and inductive learning methods is that they assume two different formulations of the learning problem:
  - In **inductive learning**, the learner is given a hypothesis space H from which it must select an output hypothesis, and a set of training examples D = $\{<x_1, f(x_1)> \ldots <x_n, f(x_n)>\}$ where $f(x_i)$ is the target value for the instance x_i. The desired output of the learner is a hypothesis h from H that is consistent with these training examples.
  - In **analytical learning**, the input to the learner includes the same hypothesis space H and training examples D as for inductive learning. In addition, the learner is provided an additional input: A ***domain theory B*** consisting of background knowledge that can be used to explain observed training examples. The desired output of ,the learner is a hypothesis h from H that is consistent with both the training examples D and the domain theory B.

An Example Analytical Learning Problem: SafeToStack(x,y)



Given:

- Instance space X: Each instance describes a pair of objects represented by the predicates *Type*, *Color*, *Volume*, *Owner*, *Material*, *Density*, and *On*.
- Hypothesis space H: Each hypothesis is a set of Horn clause rules. The head of each Horn clause is a literal containing the target predicate *SafeToStack*. The body of each Horn clause is a conjunction of literals based on the same predicates used to describe the instances, as well as the predicates *LessThan*, *Equal*, *GreaterThan*, and the functions *plus*, *minus*, and *times*. For example, the following Horn clause is in the hypothesis space:

$$SafeToStack(x, y) \leftarrow Volume(x, vx) \wedge Volume(y, vy) \wedge LessThan(vx, vy)$$

- Target concept: *SafeToStack(x,y)*
- Training Examples: A typical positive example, *SafeToStack(Obj1, Obj2)*, is shown below:

$On(Obj1, Obj2)$	$Owner(Obj1, Fred)$
$Type(Obj1, Box)$	$Owner(Obj2, Louise)$
$Type(Obj2, Endtable)$	$Density(Obj1, 0.3)$
$Color(Obj1, Red)$	$Material(Obj1, Cardboard)$
$Color(Obj2, Blue)$	$Material(Obj2, Wood)$
$Volume(Obj1, 2)$	

- Domain Theory B:

$$SafeToStack(x, y) \leftarrow \neg Fragile(y)$$
$$SafeToStack(x, y) \leftarrow Lighter(x, y)$$
$$Lighter(x, y) \leftarrow Weight(x, wx) \wedge Weight(y, wy) \wedge LessThan(wx, wy)$$
$$Weight(x, w) \leftarrow Volume(x, v) \wedge Density(x, d) \wedge Equal(w, times(v, d))$$
$$Weight(x, 5) \leftarrow Type(x, Endtable)$$
$$Fragile(x) \leftarrow Material(x, Glass)$$
$$\ldots$$

Determine:

- A hypothesis from H consistent with the training examples and domain theory.

Learning with Perfect Domain Theories: PROLOG-EBG

- We need domain theories that are correct and complete.
- A domain theory is said to be ***correct*** if each of its assertions is a truthful statement about the world.
- A domain theory is said to be ***complete*** with respect to a given target concept and instance space, if the domain theory covers every positive example in the instance space.

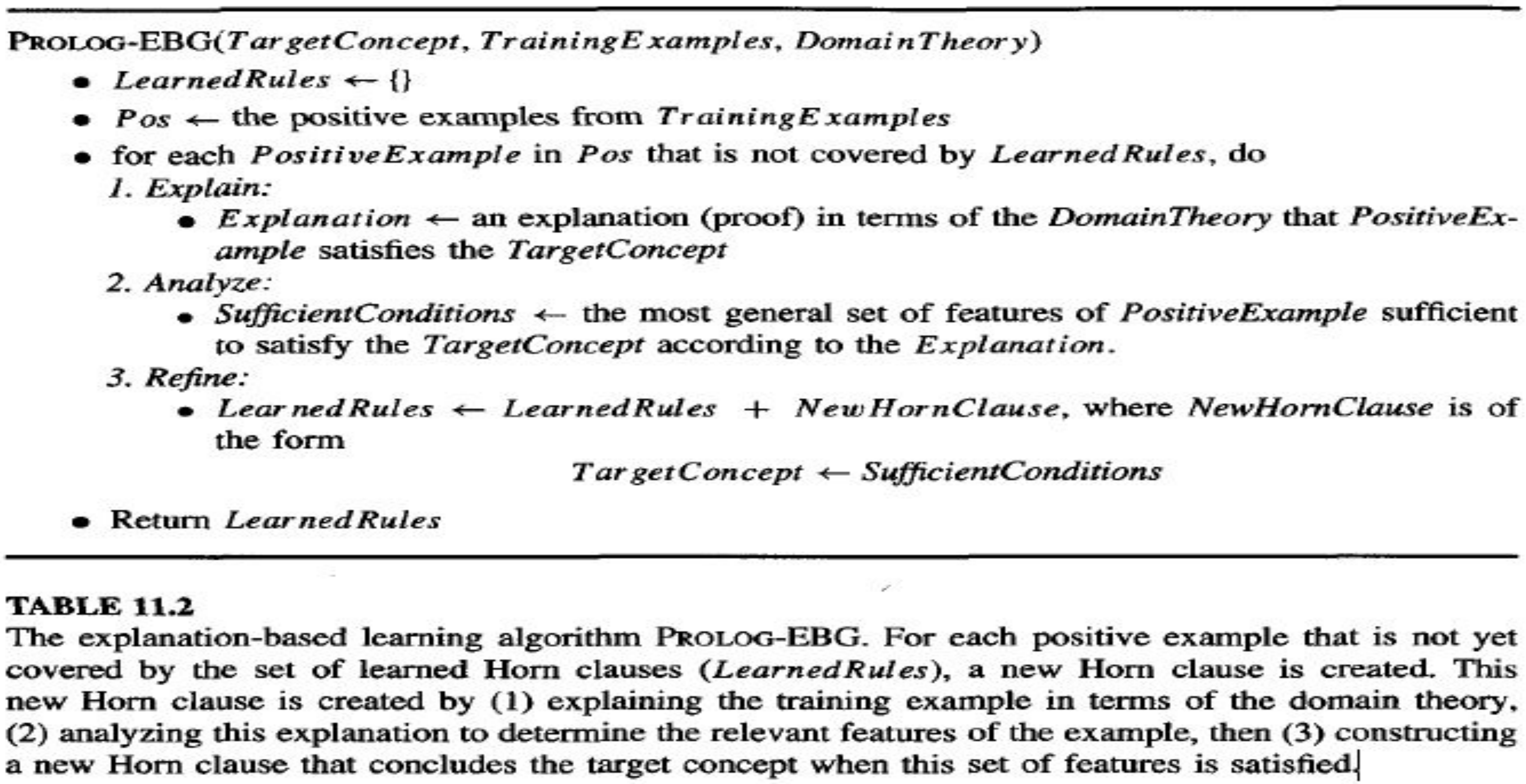
Learning with Perfect Domain Theories: PROLOG-EBG

- Is it reasonable to assume perfect domain theories are available to the learner? There are two responses to this question:
- First there are cases in which it is feasible to provide perfect domain theory. Example-chess game.
- In many cases it is unreasonable to assume that a perfect domain theory is available. Example-SafeToStack problem.
- In the current discussion we consider the ideal case of perfect domain theories.

Learning with Perfect Domain Theories: PROLOG-EBG (Explanation Based Generalization)

- PROLOG-EBG is a sequential covering algorithm.
- When given a complete and correct domain theory, it is guaranteed to output a hypothesis that is itself correct and that covers the observed positive training examples.
- For any set of training examples, the hypothesis output by PROLOG-EBG constitutes a set of logically sufficient conditions for the target concept, according to the domain theory.

Learning with Perfect Domain Theories: PROLOG-EBG (Explanation Based Generalization)



```
PROLOG-EBG(TargetConcept, TrainingExamples, DomainTheory)
    • LearnedRules ← {}
    • Pos ← the positive examples from TrainingExamples
    • for each PositiveExample in Pos that is not covered by LearnedRules, do
      1. Explain:
           • Explanation ← an explanation (proof) in terms of the DomainTheory that PositiveEx-
             ample satisfies the TargetConcept
      2. Analyze:
           • SufficientConditions ← the most general set of features of PositiveExample sufficient
             to satisfy the TargetConcept according to the Explanation.
      3. Refine:
           • LearnedRules ← LearnedRules + NewHornClause, where NewHornClause is of
             the form
                             TargetConcept ← SufficientConditions
    • Return LearnedRules
```

TABLE 11.2
The explanation-based learning algorithm PROLOG-EBG. For each positive example that is not yet covered by the set of learned Horn clauses (*LearnedRules*), a new Horn clause is created. This new Horn clause is created by (1) explaining the training example in terms of the domain theory, (2) analyzing this explanation to determine the relevant features of the example, then (3) constructing a new Horn clause that concludes the target concept when this set of features is satisfied.

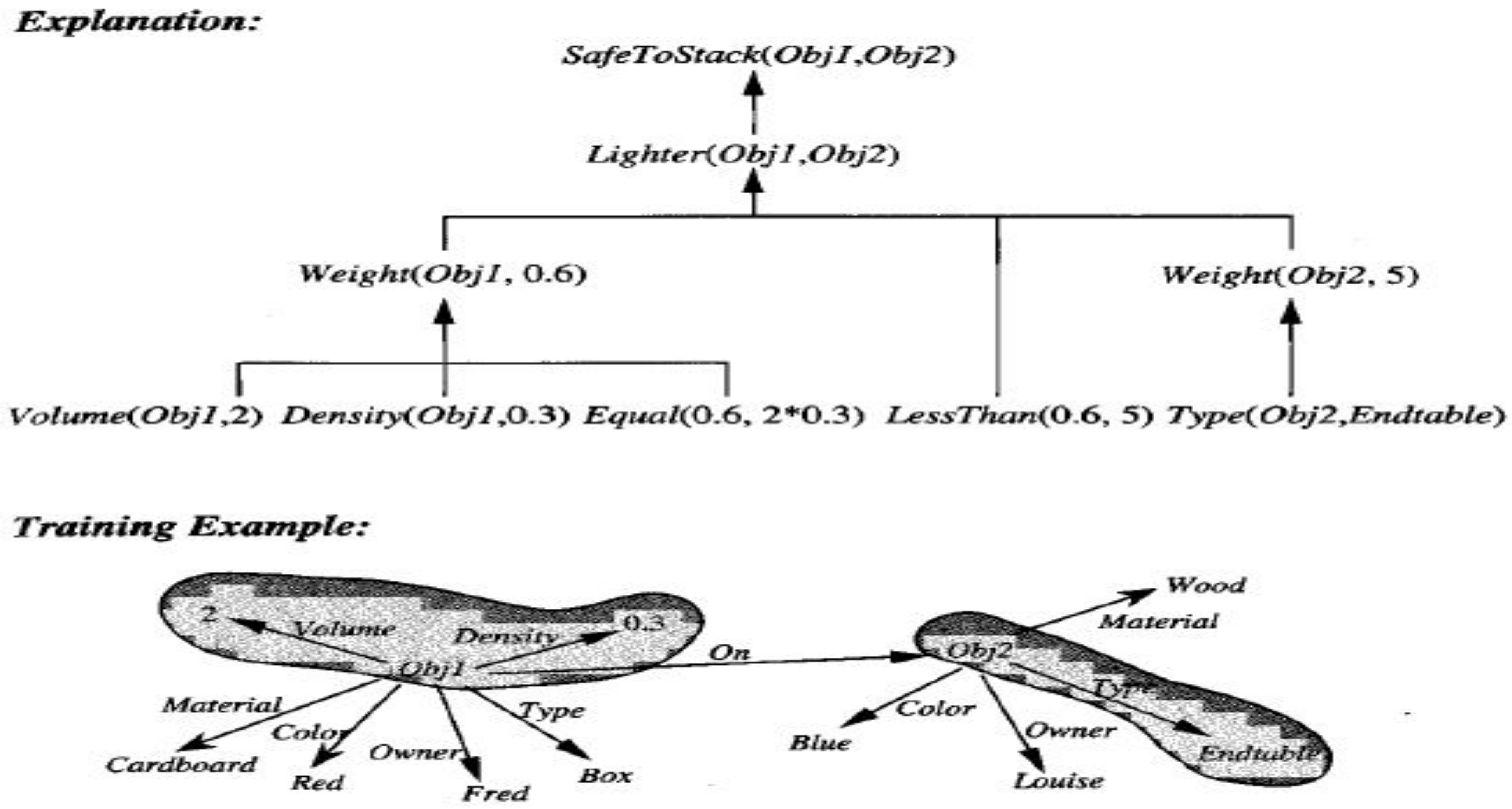
An Illustrative Trace

- Let us look at the example and domain theory of SafetoStack(x,y).
- The PROLOG-EBG is a sequential learning algorithm that covers the training data incrementally.
- For each new positive training example that is not yet covered by a learned Horn clause, it forms a new Horn clause by:
    - explaining the new positive training example,
    - analyzing this explanation to determine an appropriate generalization, and
    - refining the current hypothesis by adding a new Horn clause rule to cover this positive example, as well as other similar instances.

Explain the Training Example

- The first step in processing each novel training example is to construct an explanation in terms of the domain theory, showing how this positive example satisfies the target concept.
- When the domain theory is correct and complete this explanation constitutes a ***proof*** that the training example satisfies the target concept.

Explain the Training Example



Analyze the Explanation

- From the previous example we can form a general rule that is justified by the domain theory:

$$SafeToStack(x, y) \leftarrow Volume(x, 2) \wedge Density(x, 0.3) \wedge Type(y, Endtable)$$

- The body of the above rule includes each leaf node in the proof tree, except for the leaf nodes ***"Equal(0.6, times(2,0.3)"*** and ***"LessThan(0.6,5).“***
- **PROLOG-**EBG computes the most general rule that can be justified by the explanation, by computing the weakest preimage of the explanation, defined as follows:

Definition: The **weakest preimage** of a conclusion C with respect to a proof P is the most general set of initial assertions A, such that A entails C according to P.

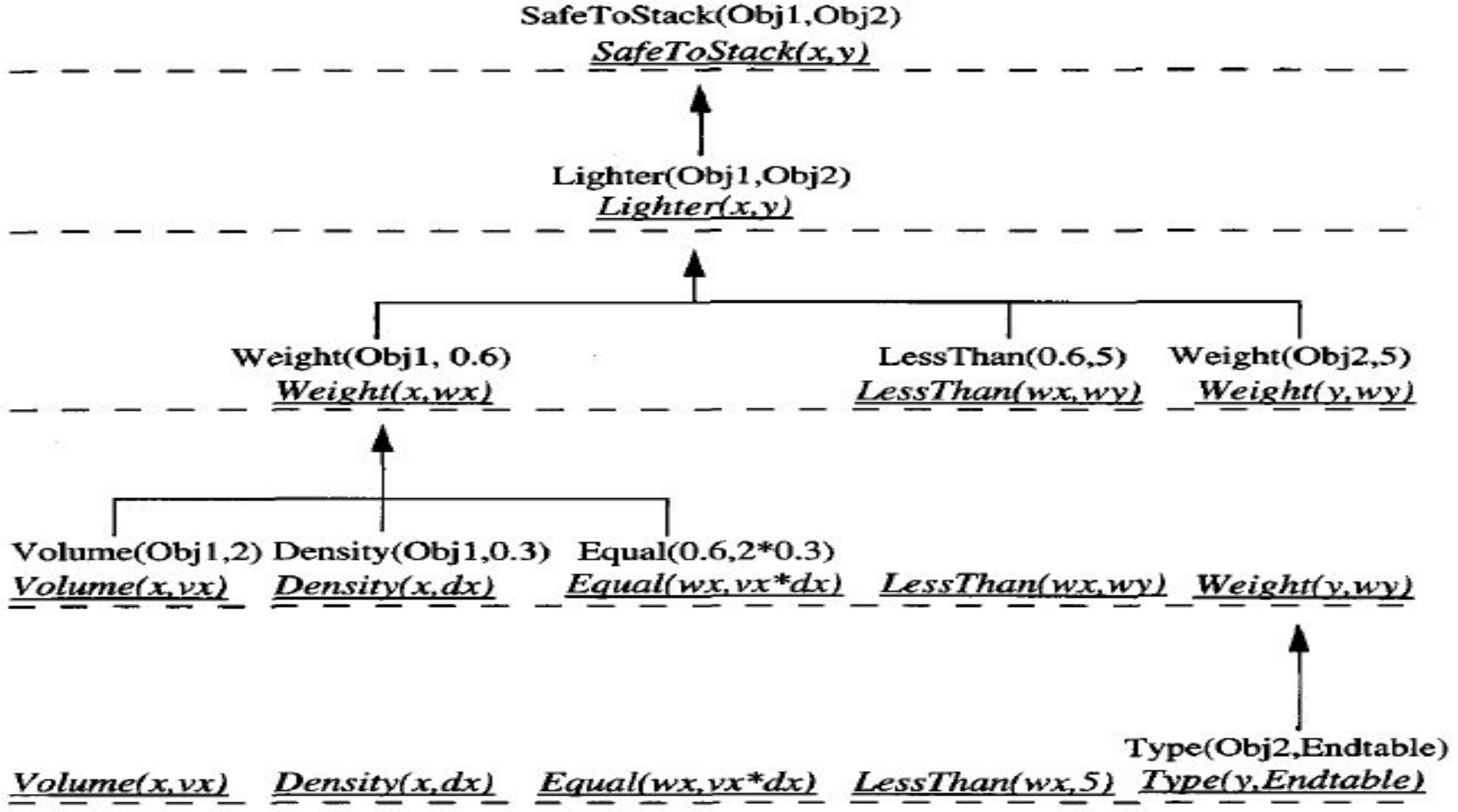
Analyze the Explanation

- The weakest preimage of the target concept SafeToStack(x,y), with respect to the explanation given earlier is given by the body of the following rule.

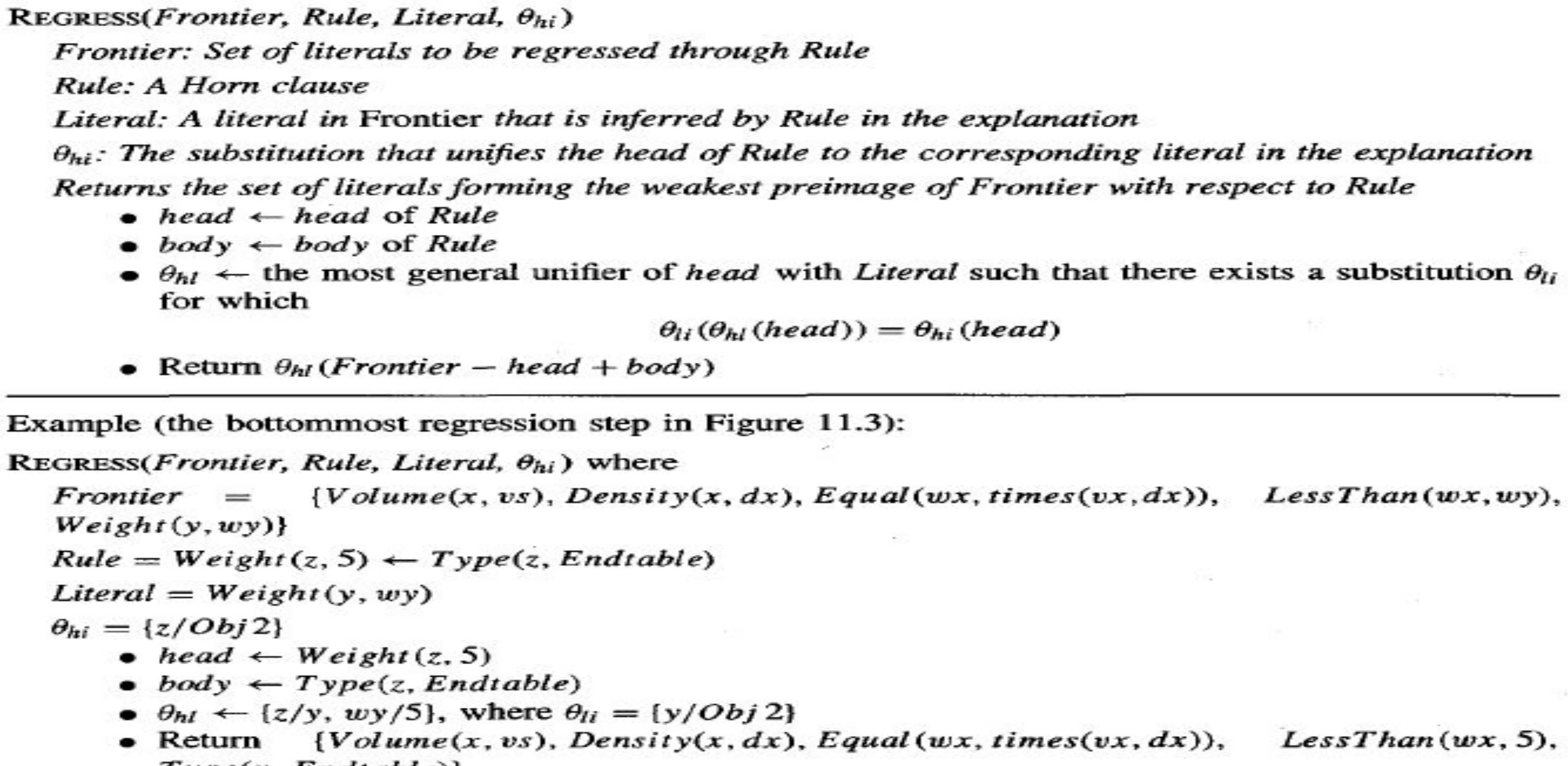
$$\begin{aligned} SafeToStack(x, y) \leftarrow\ & Volume(x, vx) \wedge Density(x, dx) \wedge \\ & Equal(wx, times(vx, dx)) \wedge LessThan(wx, 5) \wedge \\ & Type(y, Endtable) \end{aligned}$$

- This more general rule does not require the specific values for Volume and Density that were required by the previous rule.
- PROLOG-EBG computes the weakest preimage of the target concept with respect to the explanation, using a general procedure called **regression**.

Computing the Weakest Preimage of SafeToStack(x,y)- Using Regression Process



Algorithm for Regressing a set of Literals through a Single Horn Clause



```
REGRESS($Frontier, Rule, Literal, \theta_{hi}$)
    Frontier: Set of literals to be regressed through Rule
    Rule: A Horn clause
    Literal: A literal in Frontier that is inferred by Rule in the explanation
    $\theta_{hi}$: The substitution that unifies the head of Rule to the corresponding literal in the explanation
    Returns the set of literals forming the weakest preimage of Frontier with respect to Rule
        • $head \leftarrow$ head of Rule
        • $body \leftarrow$ body of Rule
        • $\theta_{hl} \leftarrow$ the most general unifier of head with Literal such that there exists a substitution $\theta_{li}$
          for which
                          $\theta_{li}(\theta_{hl}(head)) = \theta_{hi}(head)$
        • Return $\theta_{hl}(Frontier - head + body)$
```

Example (the bottommost regression step in Figure 11.3):

```
REGRESS($Frontier, Rule, Literal, \theta_{hi}$) where
    $Frontier = \{Volume(x, vs), Density(x, dx), Equal(wx, times(vx, dx)), LessThan(wx, wy),$
    $Weight(y, wy)\}$
    $Rule = Weight(z, 5) \leftarrow Type(z, Endtable)$
    $Literal = Weight(y, wy)$
    $\theta_{hi} = \{z/Obj2\}$
        • $head \leftarrow Weight(z, 5)$
        • $body \leftarrow Type(z, Endtable)$
        • $\theta_{hl} \leftarrow \{z/y, wy/5\}$, where $\theta_{li} = \{y/Obj2\}$
        • Return $\{Volume(x, vs), Density(x, dx), Equal(wx, times(vx, dx)), LessThan(wx, 5),$
          $Type(y, Endtable)\}$
```

Refine The Current Hypothesis

- At each stage, the sequential covering algorithm picks a new positive example that is not yet covered by the current Horn clauses, explains this new example, and formulates a new rule‘ according to the procedure.
- Only positive examples are covered in the algorithm as we have defined it, and the learned set of Horn clause rules predicts only positive examples.
- A new instance is classified as negative if the current rules fail to predict that it is positive.

Remarks On Explanation-Based Learning

- Discovering New Features
- Deductive Learning
- Inductive Bias In Explanation Based Learning
- Knowledge Level Learning

Remarks On Explanation-Based Learning

- The most important properties of PROLOG-EBG algorithm are as follows:
  - PROLOG-EBG produces ***justified*** general hypotheses by using prior knowledge to analyze individual examples.
  - The explanation of how the example satisfies the target concept determines which example attributes are relevant.
  - The further analysis of the explanation, regressing the target concept to determine its weakest preimage with respect to the explanation, allows deriving more general constraints on the values of the relevant features.
  - Each learned Horn clause corresponds to a sufficient condition for satisfying the target concept.
  - The generality of the learned Horn clauses will depend on the formulation of the domain theory and on the sequence in which training examples are considered.
  - PROLOG-EBG implicitly assumes that the domain theory is correct and complete.

Remarks On Explanation-Based Learning

- There are several related perspectives on explanation-based learning that help to understand its capabilities and limitations:
  - EBL as theory-guided generalization of examples.
  - EBL as example-guided reformulation of theories.
  - EBL as just restating what the learner already knows.
- In its pure form EBL involves reformulating the domain theory to produce general rules that classify examples in a single inference step.
- This kind of knowledge reformulation is sometimes referred to as ***knowledge compilation,*** indicating that the transformation is an efficiency improving one that does not alter the correctness of the system's knowledge.

Discovering New Features

- PROLOG-EBG has the ability to formulate new features that are not explicit in the description of the training examples, but they are needed to describe the general rule underlying the training examples.
- For example- the constraint on volume*density being less than 5 in the previous example.
- The learned feature is similar to the types of features represented by the hidden units of neural networks.
- Like the Backpropagation algorithm, PROLOG-EBG automatically formulates such features in its attempt to fit the training data.
- Backpropogation uses statistical process to derive hidden unit features whereas PROLOG-EBG uses analytical process

Deductive Learning

- PROLOG-EBG by calculating the weakest preimage of the explanation produces a hypothesis h that follows deductively from the domain theory B while covering the training examples D.
- PROLOG-EBG outputs a hypothesis that satisfies the following two constraints: $(\forall \langle x_i, f(x_i) \rangle \in D) \quad (h \wedge x_i) \vdash f(x_i)$

$$D \wedge B \vdash h$$

- PROLOG-EBG assumes the domain theory B entails the classification of the instances in the training data:

$$(\forall \langle x_i, f(x_i) \rangle \in D) \quad (B \wedge x_i) \vdash f(x_i)$$

Deductive Learning

- Let us compare the PROLOG-EBG learning setting to the setting for Inductive Logic Programming (ILP) discussed earlier.
- ILP is an inductive learning system where PROLOG-EBG is a deductive learning system.
- ILP uses its background knowledge B' to enlarge the set of hypotheses to be considered, whereas PROLOG-EBG uses its domain theory B to reduce the set of acceptable hypotheses.
- The ILP system outputs a hypothesis h that satisfies the following constraint:

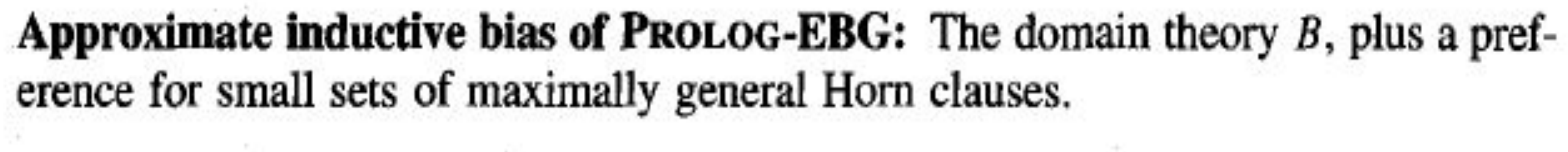
$$(\forall \langle x_i, f(x_i) \rangle \in D) \quad (B' \wedge h \wedge x_i) \vdash f(x_i)$$

Inductive Bias in Explanation Based Learning

- The importance of inductive bias is that it characterizes how the learner generalizes beyond the observed training examples.
- What is the inductive bias of PROLOG-EBG?
- In PROLOG-EBG the output hypothesis h follows deductively from $D \wedge B$.
- Given that predictions of the learner follow from this hypothesis h, it appears that the inductive bias of PROLOG-EBG is simply the domain theory B input to the learner.
- The remaining component of the inductive bias is therefore the basis by which PROLOG-EBG chooses among these alternative sets of Horn clauses entailed by the domain theory.

Inductive Bias in Explanation Based Learning

- The inductive bias can be defined as follows:



Approximate inductive bias of PROLOG-EBG: The domain theory B, plus a preference for small sets of maximally general Horn clauses.

- if we consider the larger issue of how an autonomous agent may improve its learning capabilities over time, then it is attractive to have a learning algorithm whose generalization capabilities improve as it acquires more knowledge of its domain.

Knowledge Level Learning

- By examining the PROLOG-EBG algorithm it is easy to see that ***h*** follows directly from B alone, independent of *D.*
- This seems to be more like an algorithm that is called Lemma-Enumerator.
- Lemma-Enumerator algorithm simply enumerates all proof trees that conclude the target concept based on assertions in the domain theory B.
- The only difference between LEMMA-ENUMERATOR and PROLOG-EBG is that LEMMAENUMERATOR ignores the training data and enumerates all proof trees.

Knowledge Level Learning

- Lemma-Enumerator will output a super set of horn clauses output by PROLOG-EBG.
- A few questions that arise here are:
- If its hypotheses follow from the domain theory alone, then what is the role of training data in PROLOG-EBG?
- Can PROLOG-EBG ever learn a hypothesis that goes beyond the knowledge that is already implicit in the domain theory.
- There are instances of deductive learning in which the learned hypothesis h entails conclusions that are not entailed by B.
- TO prove this we need examples where B $!\vdash$h but D$\wedge$B$\vdash$h.

Knowledge Level Learning

- Let us look at the example of PlayTennis.
- Let us assume each day is described only by a single attribute Humidity.
- Let the domain theory B be a single assertion “If Ross likes to play tennis when the humidity is x, then he will also like to play tennis when the humidity is lower than ***x***”
- The above assertion can be represented as:

$$
\begin{array}{ll}
(\forall x) & \text{IF } ((PlayTennis = Yes) \leftarrow (Humidity = x)) \\
 & \text{THEN } ((PlayTennis = Yes) \leftarrow (Humidity \leq x))
\end{array}
$$

- This does not provide any information as to which examples are which are negative.

Knowledge Level Learning

- Once the learner observes a positive example where Humidity=.30 then the domain theory along with the positive example form the hypothesis:

$$(PlayTennis = Yes) \leftarrow (Humidity \leq .30)$$

- The learned hypothesis in this case entails predictions that are not entailed by the domain theory alone.
- The phrase ***knowledge-level learning*** is sometimes used to refer to this type of learning, in which the learned hypothesis entails predictions that go beyond those entailed by the domain theory.

Knowledge Level Learning

- The set of all predictions entailed by a set of assertions Y is often called the **deductive closure** of **Y.**
- The key distinction here is that in knowledge-level learning the deductive closure of B is a proper subset of the deductive closure of B + h.
- Let us look at another example where the assertions are known as determinations.
- Determinations assert that some attribute of the instance is fully determined by certain other attributes, without specifying the exact nature of the dependence.

Knowledge Level Learning

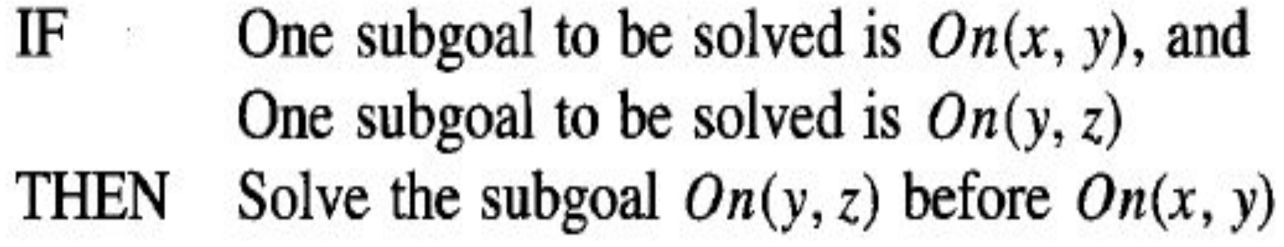
- Let the target concept be “People who speak Portuguese”.
- The domain theory is a single assertion “the language spoken by a person is determined by their nationality”
- This does not classify any instances until we see an example.

Explanation Based Learning of Search Control Knowledge

- The practical applicability of the PROLOG-EBG algorithm is restricted by its requirement that the domain theory be correct and complete.
- One important class of learning problems where this requirement is easily satisfied is learning to speed up complex search programs.
- Playing games such as chess involves searching through a vast space of possible moves and board positions to find the best move.
- Many practical scheduling and optimization problems are easily formulated as large search problems, in which the task is to find some move toward the goal state.

Explanation Based Learning of Search Control Knowledge

- One system that employs explanation-based learning to improve its search is PRODIGY.
- For example, one target concept is "the set of states in which sub-goal A should be solved before sub-goal B.“
- An example of a rule learned by PRODIGY for this target concept in a simple block-stacking problem domain is:

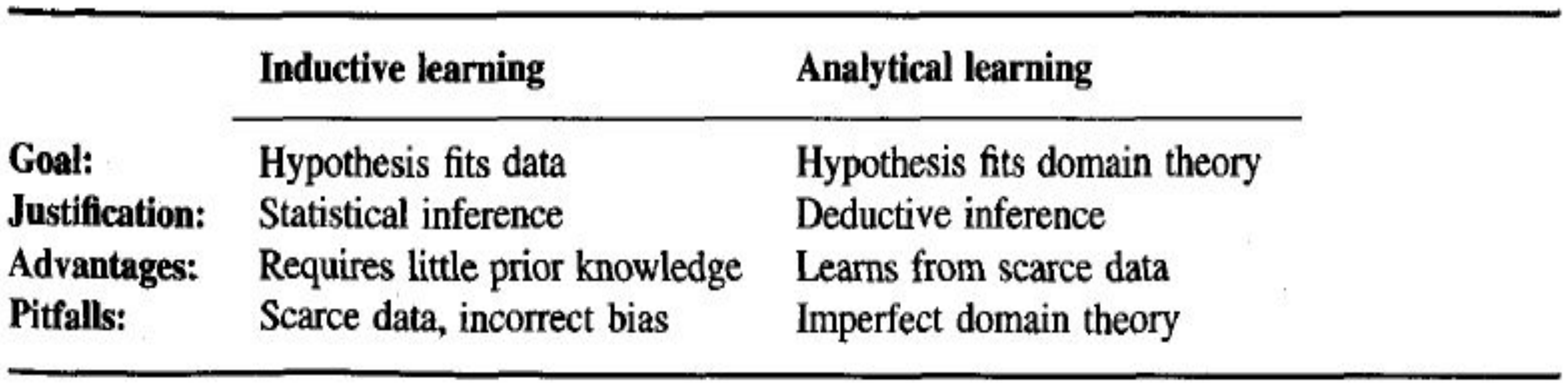


IF One subgoal to be solved is $On(x, y)$, and
One subgoal to be solved is $On(y, z)$
THEN Solve the subgoal $On(y, z)$ before $On(x, y)$

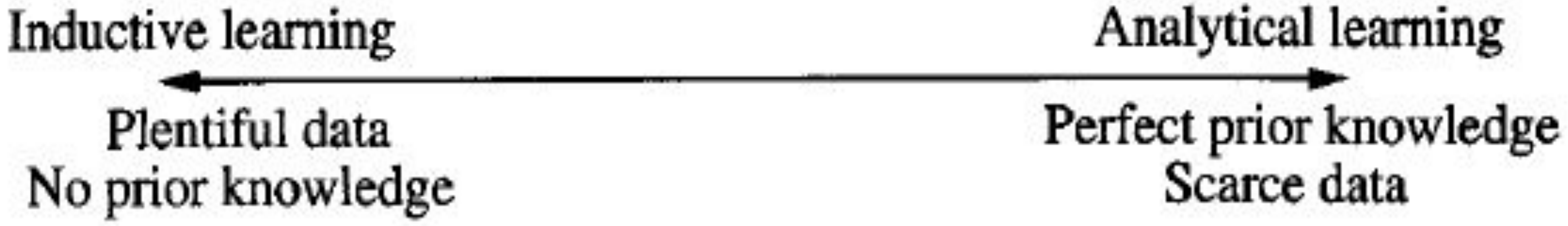
Combining Inductive and Analytical Learning - Motivation

- In the previous discussions we have seen two different paradigms of machine learning: inductive learning and analytical learning.
- Combining these two learnings offer the possibility of more powerful learning methods.
- Purely analytical learning methods can be misleading if the prior knowledge is incomplete.
- Purely inductive learning methods can be misleading if the training data is insufficient.

Combining Inductive and Analytical Learning - Motivation



	Inductive learning	Analytical learning
Goal:	Hypothesis fits data	Hypothesis fits domain theory
Justification:	Statistical inference	Deductive inference
Advantages:	Requires little prior knowledge	Learns from scarce data
Pitfalls:	Scarce data, incorrect bias	Imperfect domain theory



Combining Inductive and Analytical Learning - Motivation

- Here we consider the question of how to combine the two into a single algorithm that captures the best aspects of both.
- The most important question is “What kinds of learning algorithms can be devised that make use of approximate prior knowledge together with available data, to form general hypotheses?”
- Our interest here lies in domain-independent algorithms that employ explicit domain-dependent knowledge.

Combining Inductive and Analytical Learning - Motivation

- Some specific properties we would like from such a learning method include:
  - Given no domain theory, it should learn at least as effectively as purely inductive methods.
  - Given a perfect domain theory, it should learn at least as effectively as purely analytical methods.
  - Given an imperfect domain theory and imperfect training data, it should combine the two to outperform either purely inductive or purely analytical methods.
  - It should accommodate an unknown level of error in the training data.
  - It should accommodate an unknown level of error in the domain theory.

Inductive-Analytical Approaches to Learning

- The Learning Problem
- Hypothesis Space Search

The Learning Problem

- To summarize, the learning problem considered here:
- **Given:**
  - A set of training examples D, possibly containing errors
  - A domain theory B, possibly containing errors
  - A space of candidate hypotheses H
- **Determine:**
  - A hypothesis that best fits the training examples and domain theory.
- What precisely shall we mean by "the hypothesis that best fits the training examples and domain theory?
- In particular, shall we prefer hypotheses that fit the data a little better at the expense of fitting the theory less well, or vice versa?

The Learning Problem

- Let us define the error ***$error_B(h)$ o***f ***h*** with respect to a domain theory B to be the probability that ***h*** will disagree with B on the classification of a randomly drawn instance.
- We can attempt to characterize the desired output hypothesis in terms of these errors.
- For example, we could require the hypothesis that minimizes some combined measure of these errors, such as

$$\underset{h \in H}{\operatorname{argmin}} \; k_D error_D(h) + k_B error_B(h)$$

- It is not clear what values to give to K_D and K_B .

The Learning Problem

- An alternative perspective on the question of how to weigh prior knowledge and data is the Bayesian perspective.
- Bayes theorem computes this posterior probability based on the observed data ***D,*** together with prior knowledge in the form of ***P(h), P(D),*** and ***P(Dlh).***

Hypothesis Space Search

- How can the domain theory and training data best be combined to constrain the search for an acceptable hypothesis?
- One way to understand the range of possible approaches is to return to our view of learning as a task of searching through the space of alternative hypotheses.
- Here we explore three different methods for using prior knowledge to alter the search performed by purely inductive methods:
  - Use prior knowledge to derive an initial hypothesis from which to begin the search (Knowledge Based ANN).
  - Use prior knowledge to alter the objective of the hypothesis space search (Explanation Based NN).
  - Use prior knowledge to alter the available search steps (First Order Combined Learner).

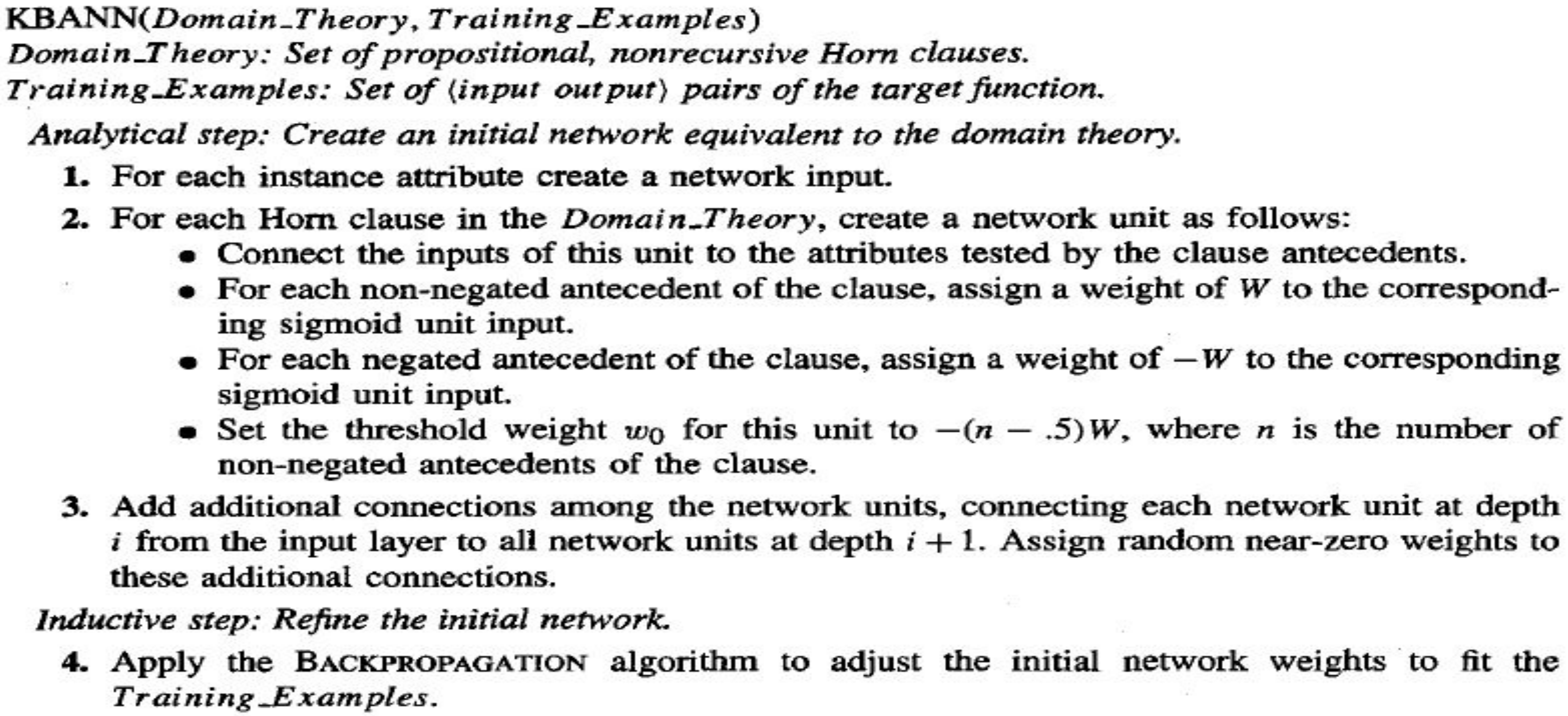
Using Prior Knowledge to Initialize the Hypothesis

- One approach to using prior knowledge is to initialize the hypothesis to perfectly fit the domain theory, then inductively refine this initial hypothesis as needed to fit the training data.
- This approach is used by the KBANN (Knowledge-Based Artificial Neural Network) algorithm to learn artificial neural networks.
- In KBANN an initial network is first constructed so that for every possible instance, the classification assigned by the network is identical to that assigned by the domain theory.
- The Backpropagation algorithm is then employed to adjust the weights of this initial network as needed to fit the training examples.

Using Prior Knowledge to Initialize the Hypothesis

- If the domain theory is correct, the initial hypothesis will correctly classify all the training examples and there will be no need to revise it.
- If the initial hypothesis is found to imperfectly classify the training examples, then it will be refined inductively to improve its fit to the training examples.
- The intuition behind KBANN is that even if the domain theory is only approximately correct, initializing the network to fit this domain theory will give a better starting approximation to the target function than initializing the network to random initial weights.

The KBANN Algorithm



KBANN(*Domain_Theory*, *Training_Examples*)
Domain_Theory: Set of propositional, nonrecursive Horn clauses.
Training_Examples: Set of ⟨input output⟩ pairs of the target function.

Analytical step: Create an initial network equivalent to the domain theory.

1. For each instance attribute create a network input.
2. For each Horn clause in the *Domain_Theory*, create a network unit as follows:
    - Connect the inputs of this unit to the attributes tested by the clause antecedents.
    - For each non-negated antecedent of the clause, assign a weight of W to the corresponding sigmoid unit input.
    - For each negated antecedent of the clause, assign a weight of $-W$ to the corresponding sigmoid unit input.
    - Set the threshold weight w_0 for this unit to $-(n - .5)W$, where n is the number of non-negated antecedents of the clause.
3. Add additional connections among the network units, connecting each network unit at depth i from the input layer to all network units at depth $i + 1$. Assign random near-zero weights to these additional connections.

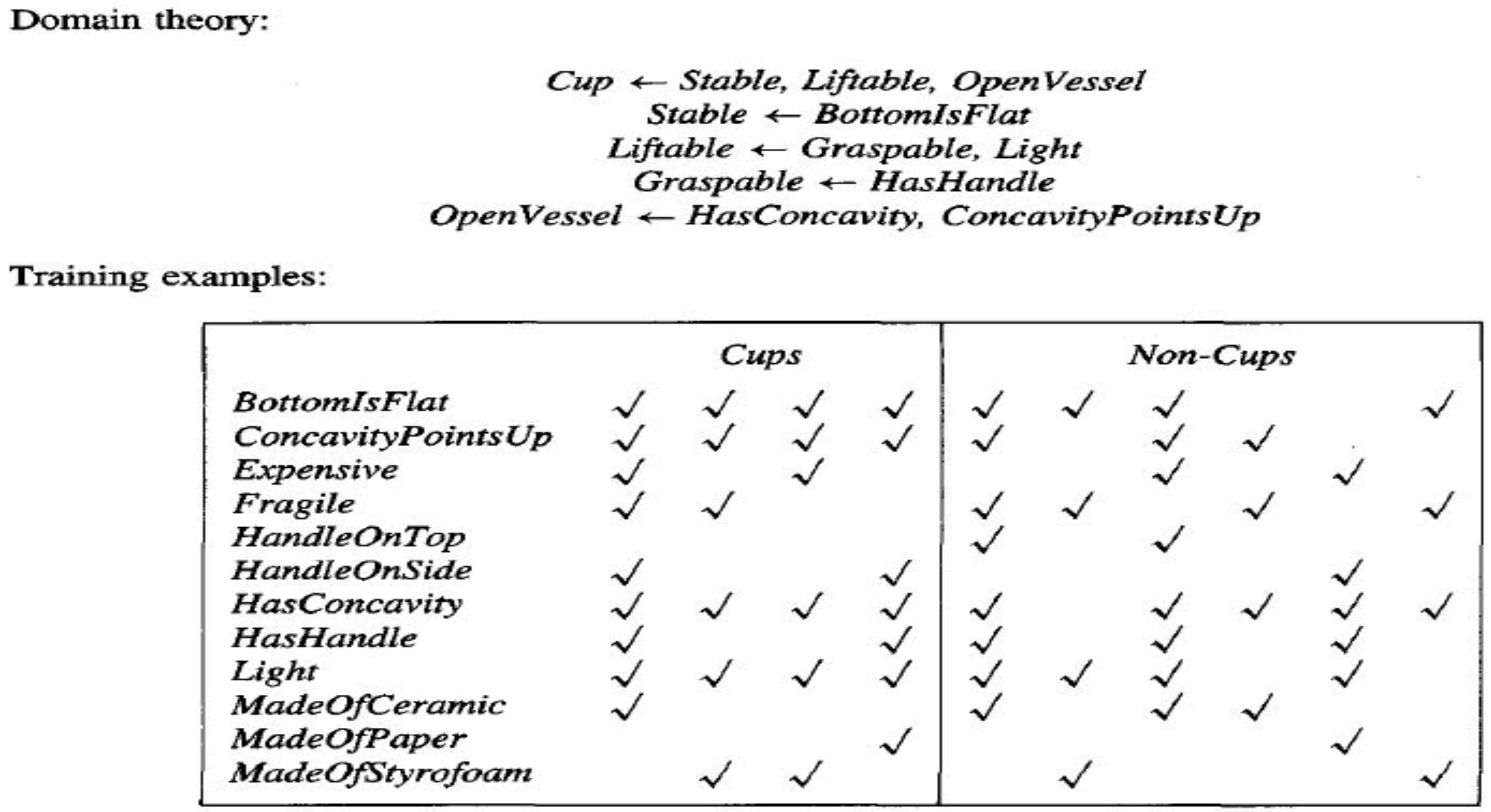
Inductive step: Refine the initial network.

4. Apply the BACKPROPAGATION algorithm to adjust the initial network weights to fit the *Training_Examples*.

The KBANN Algorithm

- Given
  - A set of training examples
  - A domain theory consisting of non-recursive, propositional Horn clauses
- Determine
  - An artificial neural network that fits the training examples, biased by the domain theory
- The two stages of the KBANN algorithm are first to create **an** artificial neural network that perfectly fits the domain theory and second to use the backpropogation algorithm to refine this initial network to fit the training examples.

An Illustrative Example



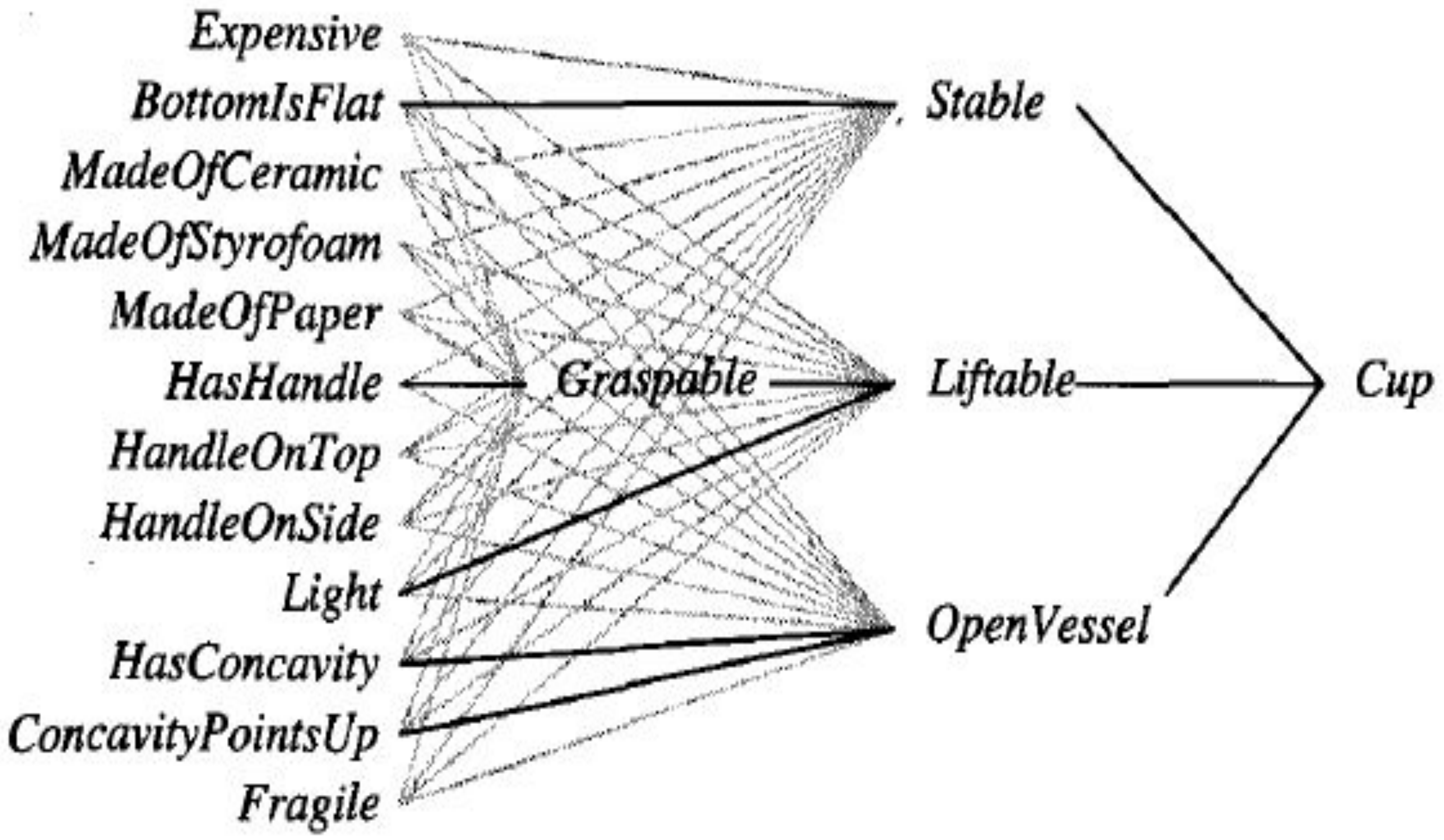
Domain theory:

Cup ← *Stable, Liftable, OpenVessel*
Stable ← *BottomIsFlat*
Liftable ← *Graspable, Light*
Graspable ← *HasHandle*
OpenVessel ← *HasConcavity, ConcavityPointsUp*

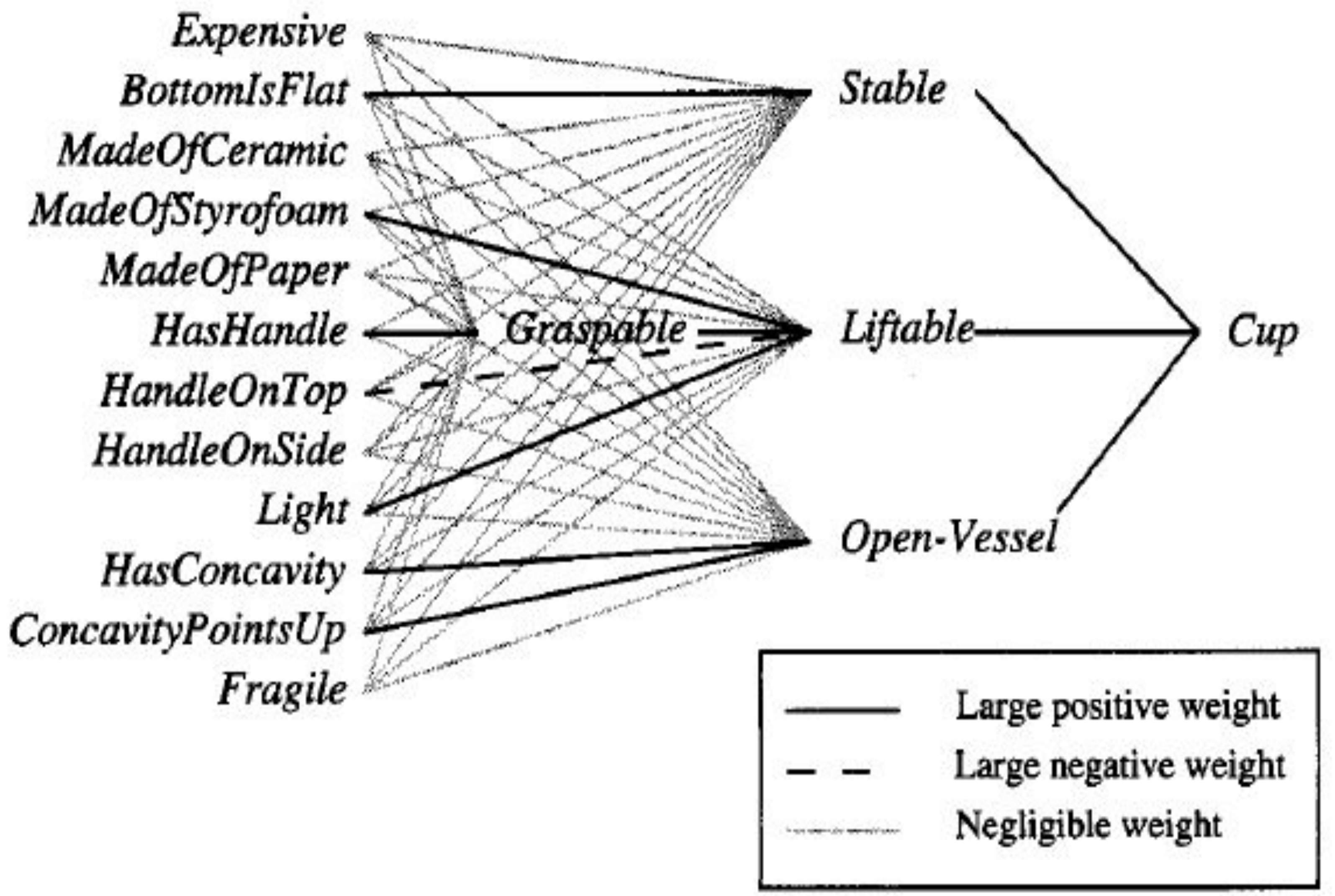
Training examples:

	Cups				*Non-Cups*					
BottomIsFlat	✓	✓	✓	✓	✓	✓	✓			✓
ConcavityPointsUp	✓	✓	✓	✓	✓		✓	✓		
Expensive	✓		✓				✓		✓	
Fragile	✓	✓			✓	✓		✓		✓
HandleOnTop					✓		✓			
HandleOnSide	✓			✓					✓	
HasConcavity	✓	✓	✓	✓	✓		✓	✓	✓	✓
HasHandle	✓			✓	✓		✓		✓	
Light	✓	✓	✓	✓	✓	✓	✓		✓	
MadeOfCeramic	✓				✓		✓	✓		
MadeOfPaper				✓					✓	
MadeOfStyrofoam		✓	✓			✓				✓

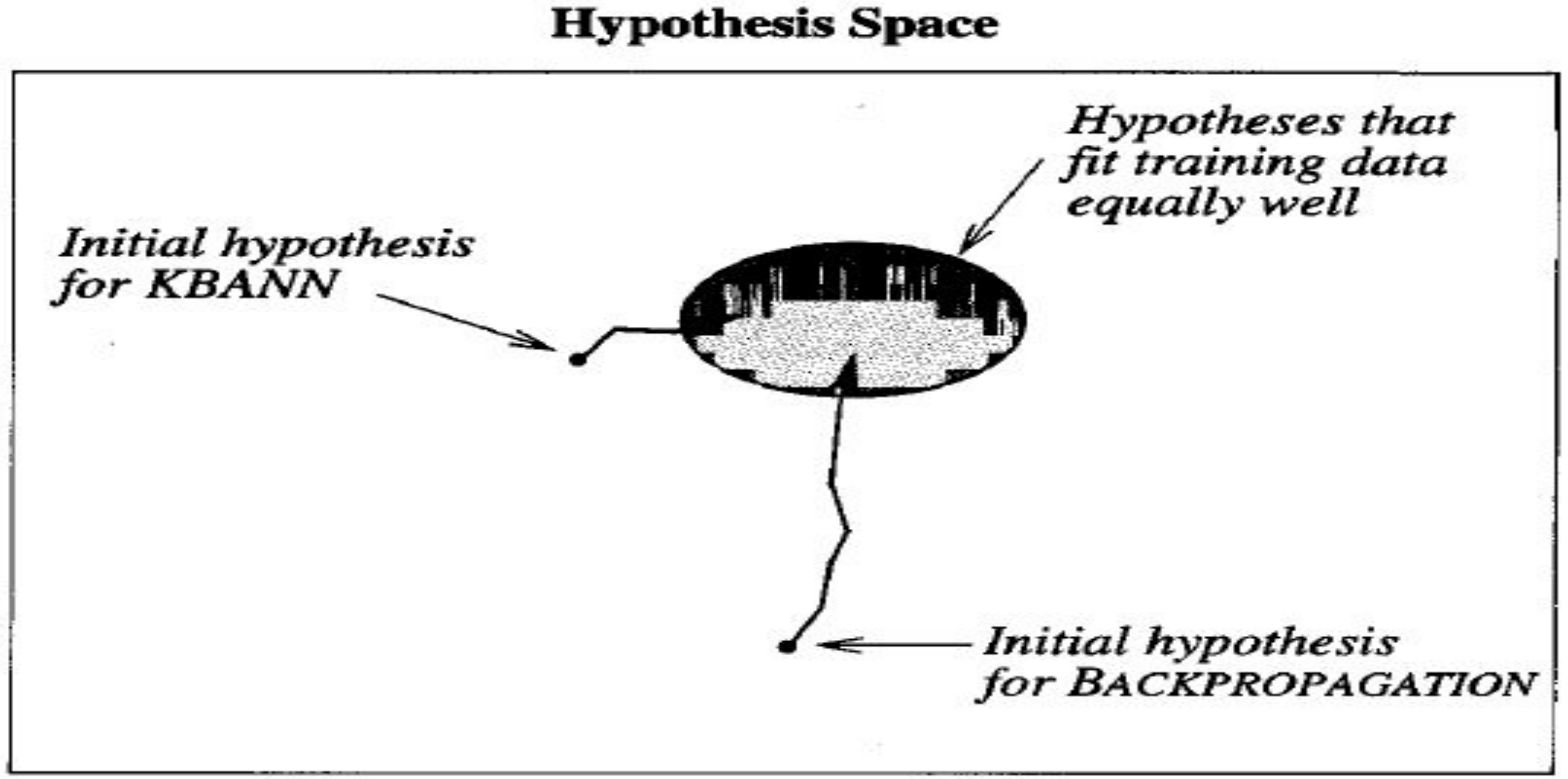
An Illustrative Example



An Illustrative Example



Remarks


Hypothesis Space
Hypotheses that fit training data equally well
Initial hypothesis for KBANN
Initial hypothesis for BACKPROPAGATION

Remarks

- Limitations of KBANN include the fact that it can accommodate only propositional domain theories; that is, collections of variable-free Horn clauses.
- It is also possible for KBANN to be misled when given highly inaccurate domain theories, so that its generalization accuracy can deteriorate below the level of backpropogation.
- It and related algorithms have been shown to be useful for several practical problems.

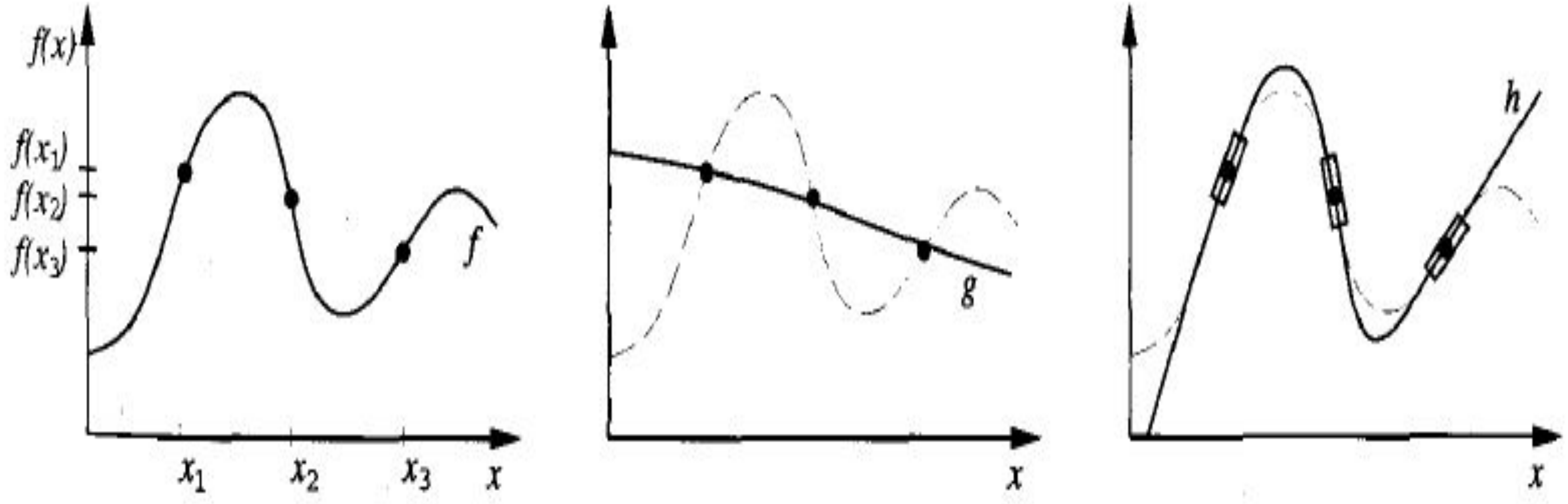
Using Prior Knowledge to Alter the Search Objective

- An alternative way of using prior knowledge is to incorporate it into the error criterion minimized by gradient descent, so that the network must fit a combined function of the training data and domain theory.
- Here we consider prior knowledge in the form of known derivatives of the target function.
- In training a neural network to recognize handwritten characters we can specify certain derivatives of the target function in order to express our prior knowledge that "the identity of the character is independent of small translations and rotations of the image.“
- We describe the TangentProp algorithm, which trains a neural network to fit both training values and training derivatives.

The TangentProp Algorithm

- TangentPro accommodates domain knowledge expressed as derivatives of the target function with respect to transformations of its inputs.
- The TangentProp algorithm assumes various training derivatives of the target function are also provided.
- If each instance ***xi*** is described by a single real value, then each training example may be of the form $\langle x_i, f(x_i), \frac{\partial f(x)}{\partial x}|_{x_i} \rangle$. $\frac{\partial f(x)}{\partial x}|_{x_i}$ denotes the derivative of the target function f with respect to x, evaluated at point $x=x_i$
- Let us consider a simple learning task.

The TangentProp Algorithm



The TangentProp Algorithm

- Let us look at another example.
- Assume the input x corresponds to an image containing a single handwritten character, and the task is to correctly classify the character.
- In this task, we might be interested in informing the learner that "the target function is invariant to small rotations of the character within the image."
- In order to express this prior knowledge to the learner, we first define a transformation s(α, x), which rotates the image x by α degrees.
- We can assert the following training derivative for every training instance x_i.

$$\frac{\partial f(s(\alpha, x_i))}{\partial \alpha} = 0$$

The TangentProp Algorithm

- As we know the backpropogation algorithm performs gradient descent to attempt to minimize the sum of squared errors.

$$E = \sum_i (f(x_i) - \hat{f}(x_i))^2$$

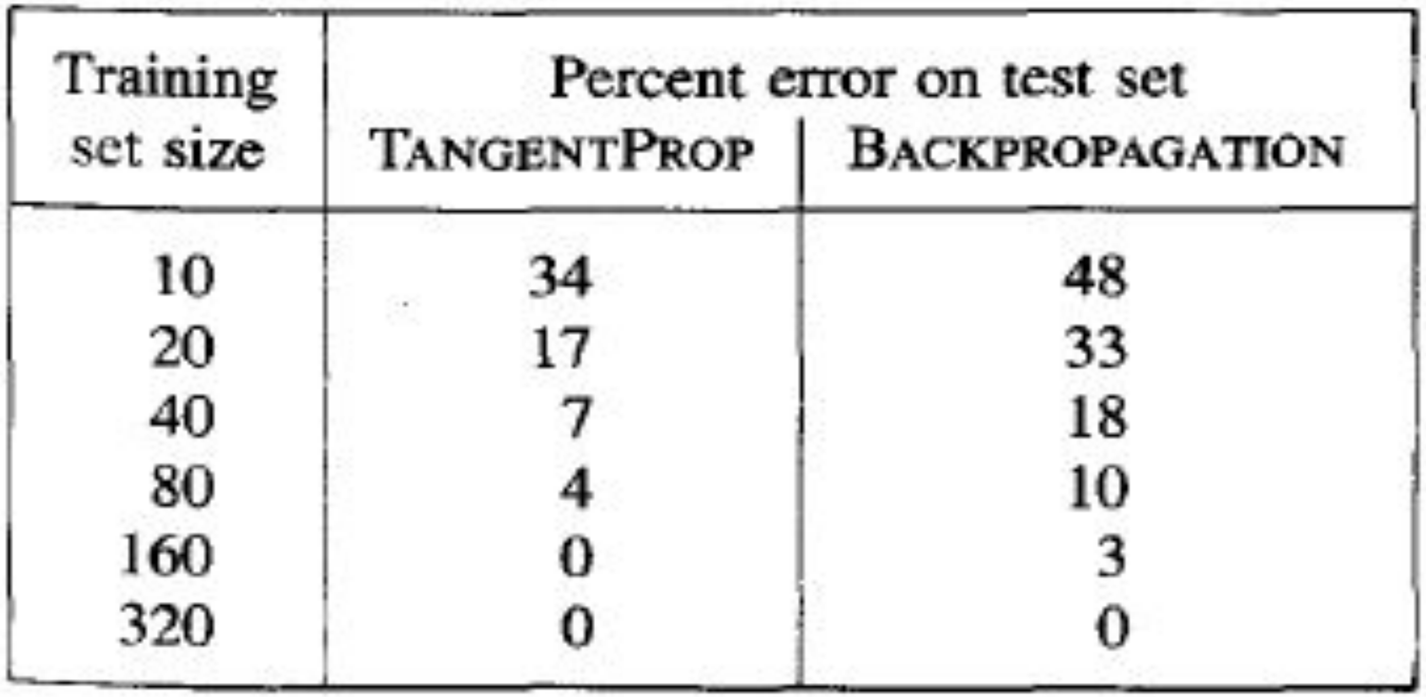
- In TangentProp an additional term is added to the error function to penalize discrepancies between the trainin4 derivatives and the actual derivatives of the learned neural network function f .

$$E = \sum_i \left[(f(x_i) - \hat{f}(x_i))^2 + \mu \sum_j \left(\frac{\partial f(s_j(\alpha, x_i))}{\partial \alpha} - \frac{\partial \hat{f}(s_j(\alpha, x_i))}{\partial \alpha} \right)^2_{\alpha=0} \right]$$

Where μ is a constant provided by the user to determine the relative importance of fitting training values versus fitting training derivatives.

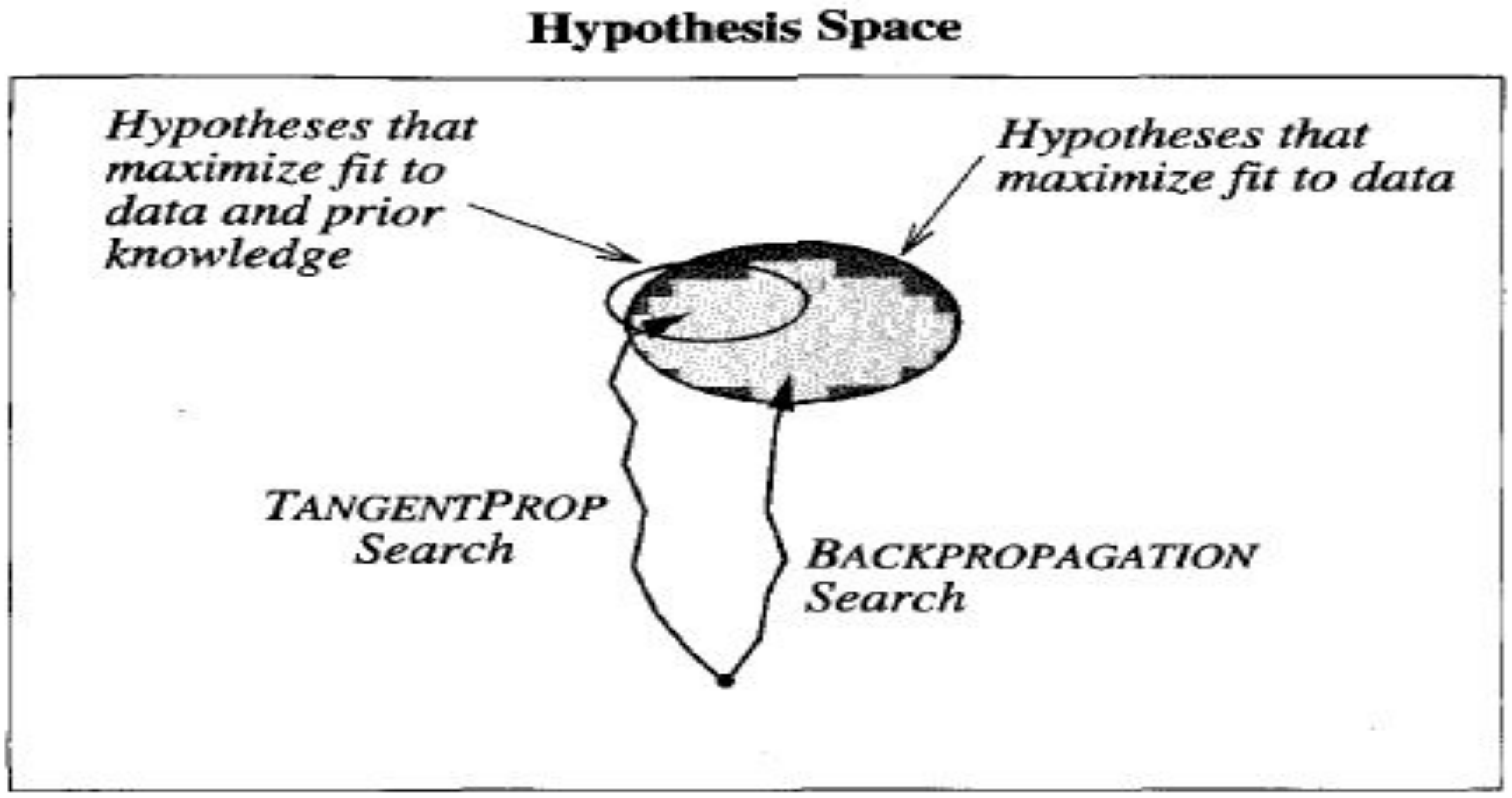
An Illustrative Example

- Here are the results comparing the generalization accuracy of TangentPro and inductive backpropogation for the problem of recognizing handwritten characters. The results are provided by Simard (1992).



Training set size	Percent error on test set TANGENTPROP	BACKPROPAGATION
10	34	48
20	17	33
40	7	18
80	4	10
160	0	3
320	0	0

Remarks


Hypothesis Space
Hypotheses that maximize fit to data and prior knowledge
Hypotheses that maximize fit to data
TANGENTPROP Search
BACKPROPAGATION Search

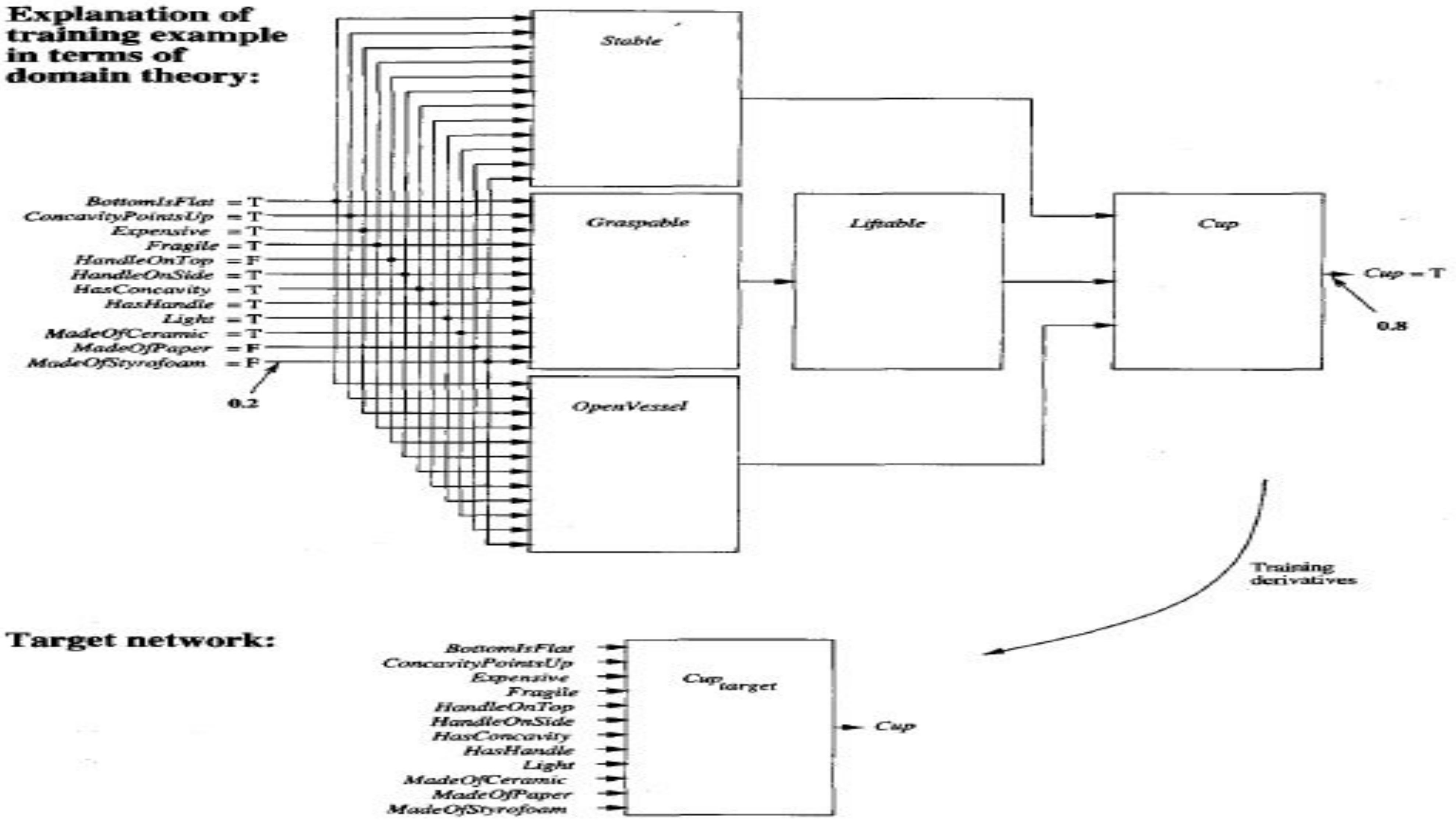
The EBNN Algorithm

- The EBNN (Explanation-Based Neural Network learning) algorithm builds on the TANGENTPROP algorithm in two significant ways:
  - First, instead of relying on the user to provide training derivatives, EBNN computes training derivatives itself for each observed training example.
  - Second, EBNN addresses the issue of how to weight the relative importance of the inductive and analytical components of learning.
- The value of μ is chosen independently for each training example, based on a heuristic that considers how accurately the domain theory predicts the training value for this particular example.

The EBNN Algorithm

- The inputs to EBNN include:
  - a set of training examples of the form (xi,f(xi)) with no training derivatives provided, and
  - a domain theory analogous to that used in explanation-based learning and in **KBANN,** but represented by a set of previously trained neural networks rather than a set of Horn clauses.
- The output of EBNN is a new neural network that approximates the target function f.
- Fitting the training examples (xi,f(xi)) constitutes the inductive component of learning, whereas fitting the training derivatives extracted from the domain theory provides the analytical component.

The EBNN Algorithm



The EBNN Algorithm

- EBNN uses a minor variant of the TangentProp algorithm to train the target network to fit the following error function.

$$E = \sum_i \left[(f(x_i) - \hat{f}(x_i))^2 + \mu_i \sum_j \left(\frac{\partial A(x)}{\partial x^j} - \frac{\partial \hat{f}(x)}{\partial x^j} \right)^2_{(x=x_i)} \right]$$

where

$$\mu_i \equiv 1 - \frac{|A(x_i) - f(x_i)|}{c}$$

- Here x_i denotes the ith training instance and A(x) denotes the domain theory prediction for input x.
- The superscript notation x^j denotes the jth component of the vector x.
- The coefficient c is a normalizing constant whose value is chosen to assure that for all i, **0** *<= μ<=* ***1.***

Remarks

- The EBNN algorithm uses a domain theory expressed as a set of previously learned neural networks, together with a set of training examples, to train its output hypothesis.
- For each training example EBNN uses its domain theory to explain the example, then extracts training derivatives from this explanation.
- For each attribute of the instance, a training derivative is computed that describes how the target function value is influenced by a small change to this attribute value, according to the domain theory.
- These training derivatives are provided to a variant of TANGENTPROP which fits the target network to these derivatives and to the training example values.

Remarks

- Fitting the derivatives constrains the learned network to fit dependencies given by the domain theory, while fitting the training values constrains it to fit the observed data itself.
- The weight μ_i placed on fitting the derivatives is determined independently for each training example, based on how accurately the domain theory predicts the training value for this example.
- EBNN has been shown to be an effective method for learning from approximate domain theories in several domains.

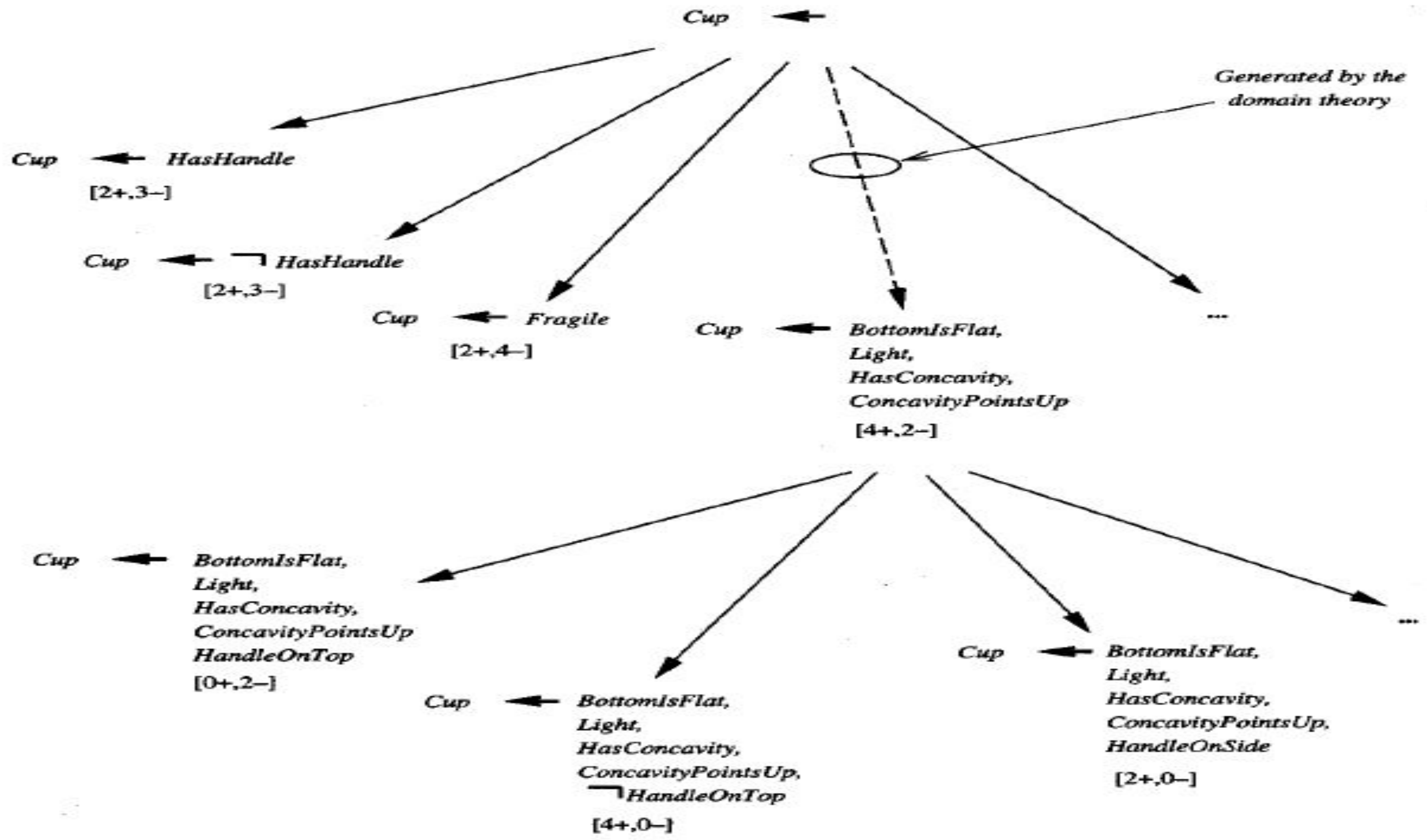
Using Prior Knowledge to Augment Search Operators

- Here we consider a third way of using prior knowledge to alter the hypothesis space search: using it to alter the set of operators that define legal steps in the search through the hypothesis space.
- We use First Order Combined Learner (FOCL) to understand the approach.

The FOCL Algorithm

- FOCL is an extension of the purely inductive FOIL system.
- The difference between FOIL and FOCL lies in the way in which candidate specializations are generated during the general-to-specific search for a single Horn clause.
- FOIL generates each candidate specialization by adding a single new literal to the clause preconditions.
- FOCL uses this same method for producing candidate specializations, but also generates additional specializations based on the domain theory.

The FOCL Algorithm


Cup ←
Generated by the domain theory
Cup ← HasHandle
[2+,3–]
Cup ← ¬ HasHandle
[2+,3–]
Cup ← Fragile
[2+,4–]
Cup ← BottomIsFlat, Light, HasConcavity, ConcavityPointsUp
[4+,2–]
...
Cup ← BottomIsFlat, Light, HasConcavity, ConcavityPointsUp HandleOnTop
[0+,2–]
Cup ← BottomIsFlat, Light, HasConcavity, ConcavityPointsUp, ¬HandleOnTop
[4+,0–]
Cup ← BottomIsFlat, Light, HasConcavity, ConcavityPointsUp, HandleOnSide
[2+,0–]
...

The FOCL Algorithm

- At each point in its general-to-specific search, FOCL expands its current hypothesis h using the following two operators:
  - For each operational literal that is not part of h, create a specialization of h by adding this single literal to the preconditions. This is also the method used by FOIL to generate candidate successors. The solid arrows in the figure denote this type of specialization.
  - Create an operational, logically sufficient condition for the target concept according to the domain theory. Add this set of literals to the current preconditions of h. Finally, prune the preconditions of h by removing any literals that are unnecessary according to the training data. The dashed arrow in the figure denotes this type of specialization.

The FOCL Algorithm

- The detailed procedure for the second operator above is as follows.
- FOCL first selects one of the domain theory clauses whose head matches the target concept.
- If there are several such clauses, it selects the clause whose body have the highest information gain relative to the training examples of the target concept. For example, in the domain theory and training data of the figure.
- There is only one such clause:

Cup□ Stable, Liftable, OpenVessel

The FOCL Algorithm

- The domain theory clause ***Stable* □*BottomIsFlat*** is used to substitute the operational ***BottomIsFlat*** for the un-operational ***Stable.***
- This process of "unfolding" the domain theory continues until the sufficient conditions have been restated in terms of operational literals.
- The final clause is:

BottomIsFlat , ***HasHandle, Light, HasConcavity*** , ***ConcavityPointsUp***

- Now this sufficient condition is pruned. Pruning HasHandle improves the performance:

BottomZsFlat , ***Light, HasConcavity*** , ***ConcavityPointsUp***

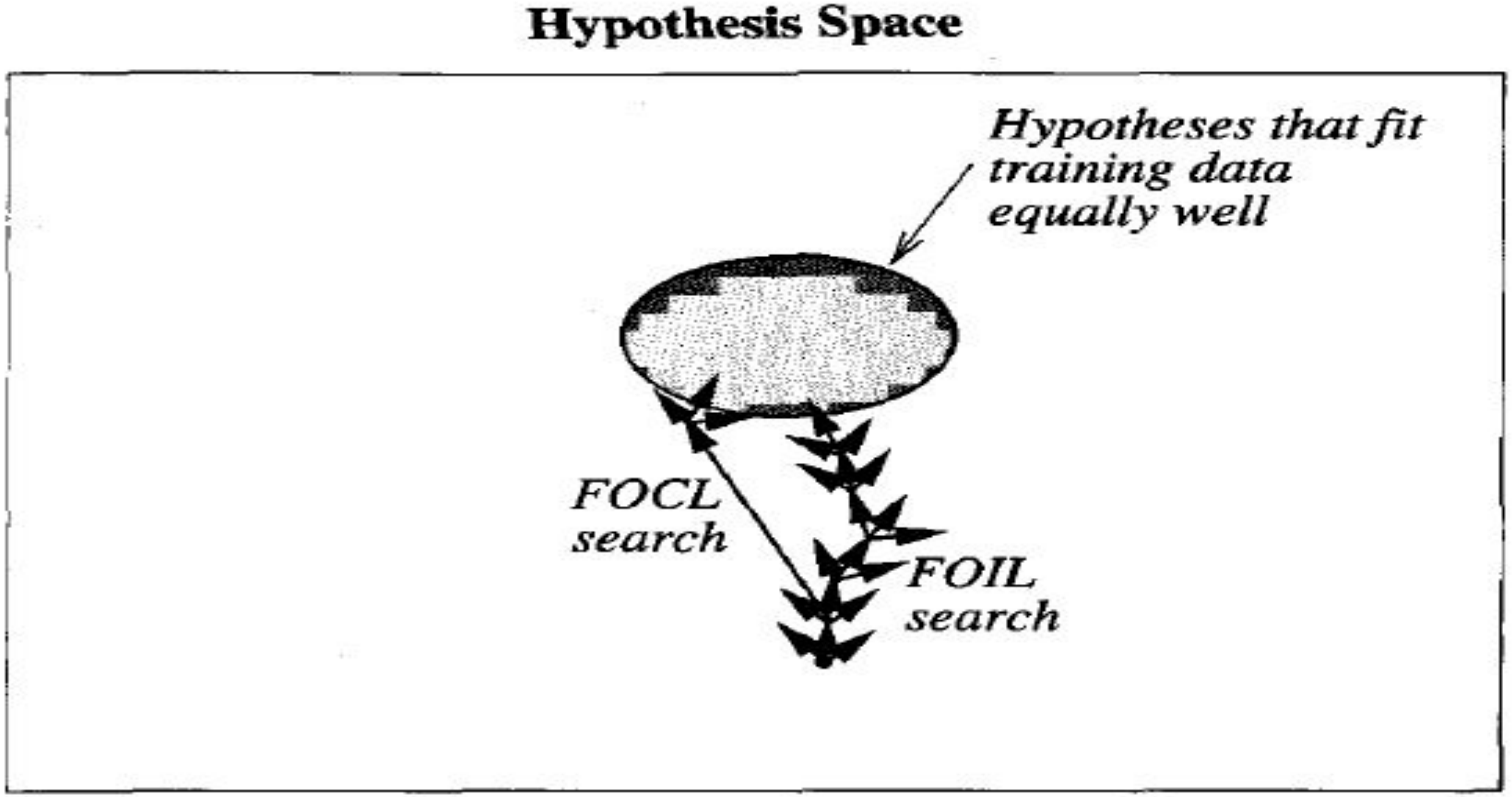
The FOCL Algorithm

- FOCL learns horn clauses of the form:

$$C\square o_i \wedge o_b \wedge o_f$$

Where c is the target concept, o_i is an initial conjugation of operational literals added one at a time by the first syntactic operator, o_b is a conjunction of operational literals added in a single step based on the domain theory, and o_f is the final conjunction of operational literals added one at a time by the first syntactic operator. Any of these three sets of literals may be empty.

Remarks


Hypothesis Space
Hypotheses that fit training data equally well
FOCL search
FOIL search