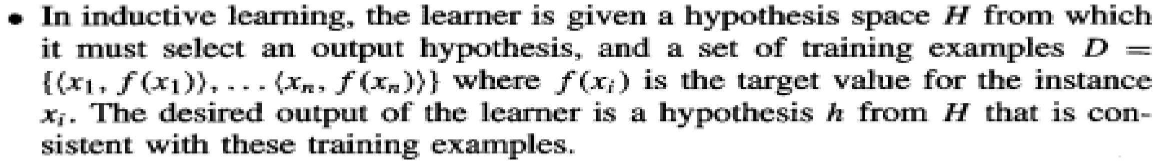
UNIT-5

- **INDUCTIVE LEARNING** is a machine learning approach where the system learns rules or patterns from observed data. In this approach, the system generalizes from specific examples to form general rules or models that can be applied to new, unseen data.



- 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 $D = \{\langle x_1, f(x_1)\rangle, \ldots \langle x_n, f(x_n)\rangle\}$ 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.

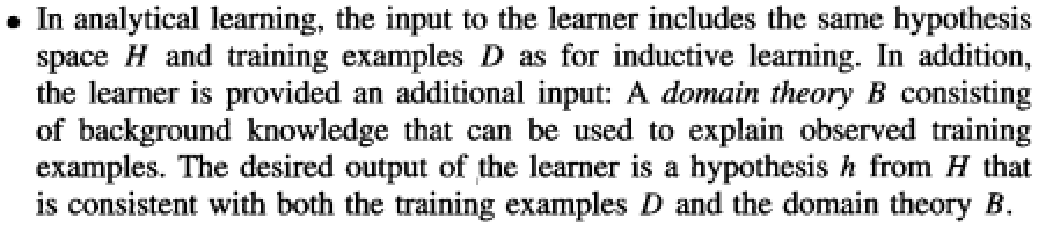
Here's an example to illustrate inductive learning:

Suppose you have a dataset of emails labelled as either "spam" or "not spam." Each email is represented by a set of features such as the presence of certain keywords, the length of the email, etc. In inductive learning, the system would analyse this dataset and learn patterns or rules that distinguish between spam and non-spam emails. For example, it might learn that emails containing words like "free," "discount," or "buy now" are more likely to be spam.

Once the system has learned these patterns from the training data, it can then use them to classify new, unseen emails as either spam or non-spam based on their features.

In this example, the inductive learning process involves generalizing from specific examples (the labelled emails in the training dataset) to form a model (the learned rules or patterns) that can be applied to new instances to make predictions.

- **ANALYTICAL LEARNING**, also known as deductive learning, is a machine learning approach where the system derives rules or models based on logical reasoning and domain knowledge rather than from observed data alone. In analytical learning, the system uses existing knowledge, principles, or theories to make predictions or decisions.



- 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.

Here's an example to illustrate analytical learning:

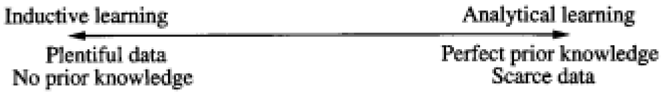
Suppose you are designing a system to predict whether a patient has a certain disease based on their symptoms. Instead of learning from a dataset of patient records, you decide to use known medical knowledge and principles to develop a diagnostic model.

You know from medical literature and expert knowledge that certain symptoms are indicative of the disease. For example, if a patient exhibits symptoms such as high fever, persistent cough, and shortness of breath, it's likely they have the disease. Additionally, you know that certain tests, such as blood tests or imaging studies, can confirm the diagnosis.

Using this existing knowledge, you construct a set of rules or logical conditions that represent the diagnostic criteria for the disease. These rules might include thresholds for certain symptoms or combinations of symptoms that are highly indicative of the disease.

Once you've developed these rules, your system can use them to analyse new patient data. When presented with a new patient's symptoms, the system applies the rules to determine the likelihood that they have the disease. If the symptoms meet the criteria outlined by the rules, the system predicts that the patient has the disease; otherwise, it predicts they do not.

In this example, the system's predictions are based on analytical reasoning and domain knowledge rather than learned patterns from data. Analytical learning is particularly useful when there is limited or no labelled data available, or when domain experts can provide valuable insights and rules for decision-making.



Inductive Learning	**Analytical Learning**
Formulate general hypotheses that fit observed training data	Formulate general hypotheses that fit domain theory while covering the observed data
The process in which the learner finds and process patterns only by processing examples	It uses prior knowledge to guide the machine learning process rather than relying solely on the data to determine the best model.
Limited sample size	Solves the problem of limited data
Methods: Decision tree learning, neural networks, GA	**Methods**: PROLOG-EBG (Explanation Based Generalization)
Requires large amount of data to derive accurate rules.	Can work with smaller data sets due to the incorporation of prior knowledge
Can fail if there exists insufficient training data or incorrect inductive bias	Analytical learning is useful in situations where the available data is limited or noisy. It can also fail when given incorrect or insufficient prior knowledge
Requires no prior knowledge	Requires domain knowledge
Uses **statistical inferences** to develop models that can learn from data and make predictions or decisions.	Uses **deductive inference** where you progress from general ideas to specific conclusions

➢ LEARNING WITH PERFECT DOMAIN THEORIES: PROLOG-EBG

PROLOG-EBG stands for "Explanation-Based Generalization in Prolog." It's an algorithm used in machine learning which combines concepts from logic programming (Prolog) and explanation-based learning (EBL).

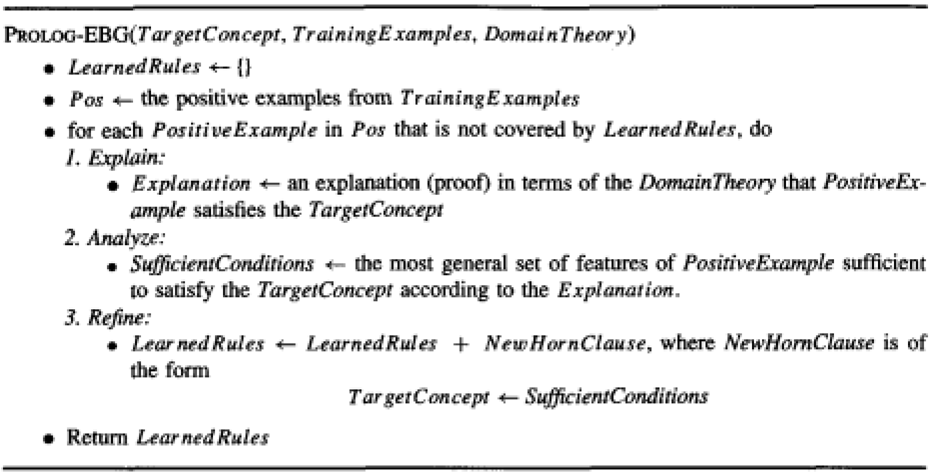
PROLOG-EBG utilizes inductive reasoning through EBL, where it learns by analyzing specific training examples and their explanations. Instead of memorizing individual examples, the system generalizes from these examples to derive more abstract rules or hypotheses. These generalizations are based on patterns observed in the examples and are used to make predictions or classifications about unseen data.

The **PROLOG-EBG** algorithm is a sequential covering algorithm that considers 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:

(1) explaining the new positive training example using domain theory,

(2) analysing this explanation to extract relevant features, and

(3) Refine the hypothesis by adding rules based on the analyzed explanations.



```
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 PositiveExample 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
```

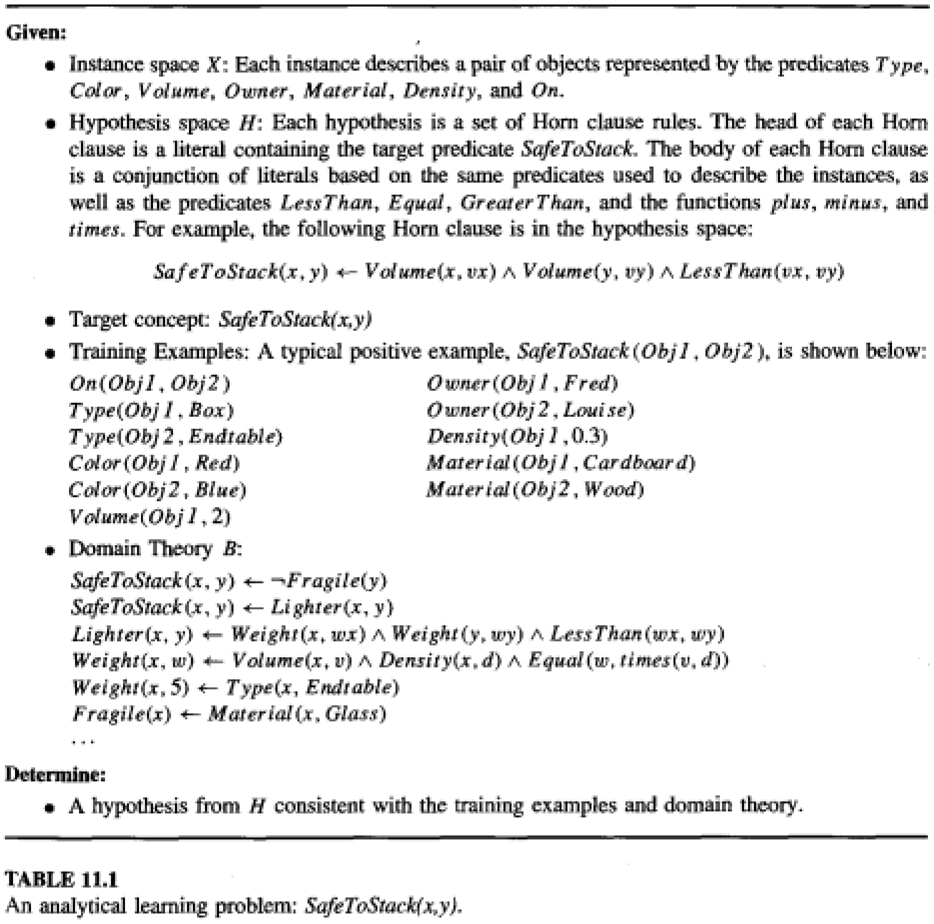
EXPLAIN THE TRAINING EXAMPLE:

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.

ANALYSING THE EXPLANATION: The algorithm then analyses the explanation to determine the relevant features of the example. It collects the features mentioned in the explanation to form a general rule that covers the positive example. This involves identifying the most general set of features sufficient to satisfy the target concept.

REFINE THE CURRENT HYPOTHESIS: The current hypothesis, consisting of the learned Horn clauses, is refined by adding a new rule based on the analyzed explanation. This process continues iteratively for each positive training example until no further positive examples remain uncovered.

Consider an example SafeToStack(Obj1,Obj2)



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:

$On(Obj1, Obj2)$
$Type(Obj1, Box)$
$Type(Obj2, Endtable)$
$Color(Obj1, Red)$
$Color(Obj2, Blue)$
$Volume(Obj1, 2)$

$Owner(Obj1, Fred)$
$Owner(Obj2, Louise)$
$Density(Obj1, 0.3)$
$Material(Obj1, Cardboard)$
$Material(Obj2, Wood)$

- Domain Theory B:

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

Determine:

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

TABLE 11.1
An analytical learning problem: *SafeToStack(x,y)*.

Consider an instance space X in which each instance is a pair of physical objects. Each of the two physical objects in the instance is described by the predicates **Color, Volume, Owner, Material, Type, and Density**, and the relationship between the two objects is described by the predicate **On**.

Given this instance space, the task is to learn the target concept "pairs of physical objects, such that one can be stacked safely on the other," denoted by the predicate **SafeToStack(x,y).** Learning this target concept might be useful, for example, to a robot system that has the task of storing various physical objects within a limited workspace.

we have chosen a hypothesis space H in which each hypothesis is a set of first-order if-then rules, or Horn clauses

For instance, the example Horn clause hypothesis shown in the table asserts that it is SafeToStack any object x on any object y, if the Volume of x is LessThan the Volume of y (in this Horn clause the variables vx and vy represent the volumes of x and y, respectively).

A typical positive training example, **SafeToStack(Obj1, Obj2)**, is also shown in the table.

To formulate this task as an analytical learning problem we must also provide a domain theory sufficient to explain why observed positive examples satisfy the target concept i.e, the domain theory must explain why certain pairs of objects can be safely stacked on one another.

The domain theory shown in the table includes assertions such as

"it is safe to stack x on y if y is not Fragile,"

"an object x is Fragile if the Material from which x is made is Glass."

The domain theory refers to additional predicates such as Lighter and Fragile, which are not present in the descriptions of the training examples, but which can be inferred from more primitive instance attributes such as Material, Density, and Volume, using other rules in the domain theory.

The domain theory shown in the table is sufficient to prove that the positive example shown there satisfies the target concept **SafeToStack.**

Let us learn the example using PROLOG-EBG

EXPLAIN THE TRAINING EXAMPLE:

The explanation, or proof, states that **it is SafeToStack Objl on Obj2 because Objl is Lighter than Obj2**. Furthermore, Objl is known to be Lighter, because its Weight can be inferred from its Density and Volume, and because the Weight of Obj2 can be inferred from the default weight of an Endtable. This explanation is derived from the domain theory, which includes rules about object weights, densities, and volumes.

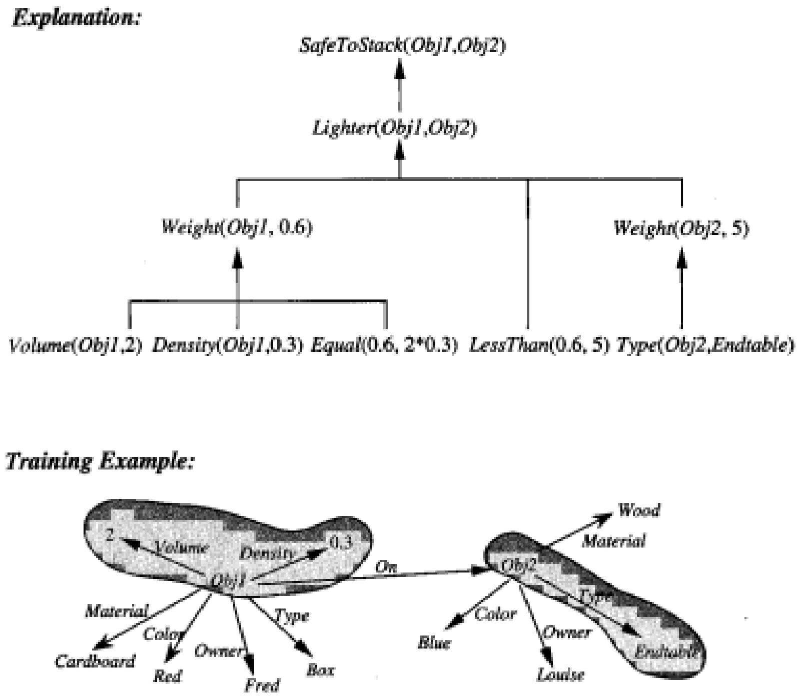


FIGURE 11.2
Explanation of a training example. The network at the bottom depicts graphically the training example *SafeToStack(Obj1, Obj2)* described in Table 11.1. The top portion of the figure depicts the explanation of how this example satisfies the target concept, *SafeToStack*. The shaded region of the training example indicates the example attributes used in the explanation. The other, irrelevant, example attributes will be dropped from the generalized hypothesis formed from this analysis.

ANALYSING THE EXPLANATION: From the explanation, we extract the relevant features. In this case, the relevant features might include the weights, densities, and volumes of the objects involved.

By collecting just, the features mentioned in the leaf nodes of the explanation and substituting variables x and y for Objl and Obj2, 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)."** We omit these two because they are by definition always satisfied, independent of x and y.

Along with this learned rule, the program can also provide its justification: The explanation of the training example forms a proof for the correctness of this rule.

The above rule constitutes a significant generalization of the training example, because it omits many properties of the example (e.g., the Color of the two objects) that are irrelevant to the target concept. However, an even more general rule can be obtained by more careful analysis of the explanation. 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.

This is the most general rule that can be justified by the explanation

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

Notice this more general rule does not require the specific values for Volume and Density that were required by the first rule.

PROLOG-EBG computes the weakest preimage of the target concept with respect to the explanation, using a general procedure called **regression.**

The **regression procedure** operates on a domain theory represented by an arbitrary set of Horn clauses. It works iteratively backward through the explanation, first computing the weakest preimage of the target concept with respect to the final proof step in the explanation, then computing the weakest preimage of the resulting expressions with respect to the preceding step, and so on. The procedure terminates when it has iterated over all steps in the explanation, yielding the weakest precondition of the target concept with respect to the literals at the leaf nodes of the explanation.

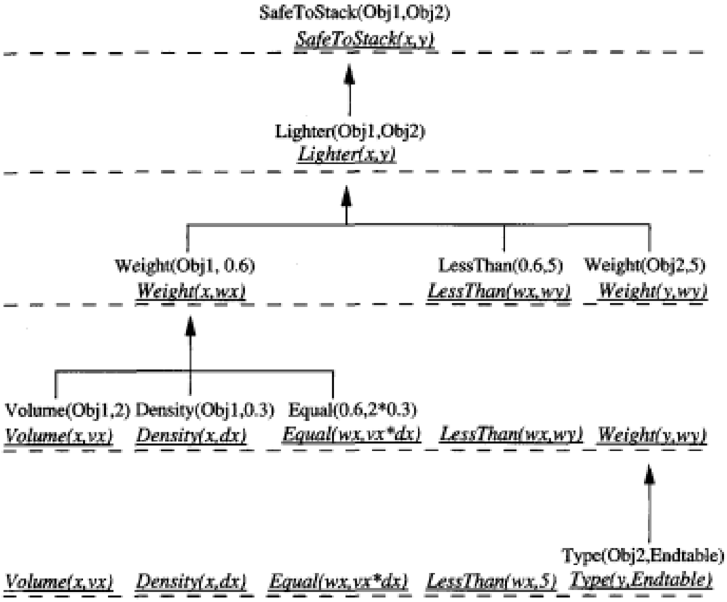
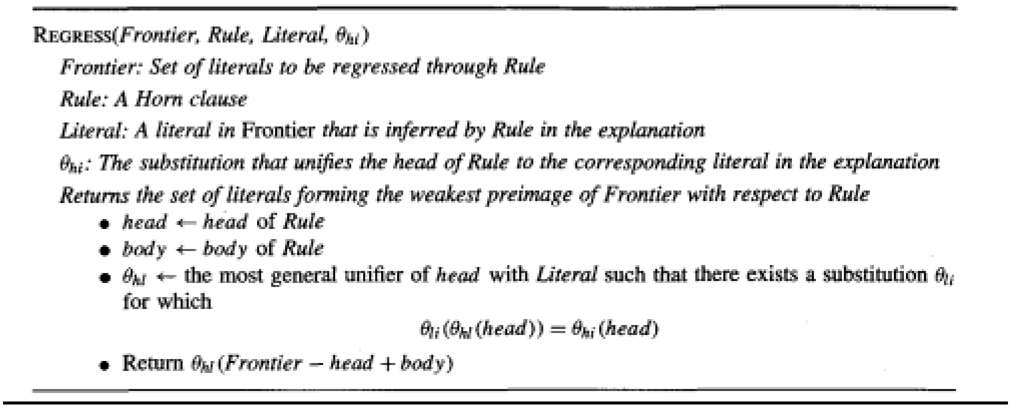


FIGURE 11.3
Computing the weakest preimage of *SafeToStack*$(Obj1, Obj2)$ with respect to the explanation. The target concept is regressed from the root (conclusion) of the explanation, down to the leaves. At each step (indicated by the dashed lines) the current frontier set of literals (underlined in italics) is regressed backward over one rule in the explanation. When this process is completed, the conjunction of resulting literals constitutes the weakest preimage of the target concept with respect to the explanation. This weakest preimage is shown by the italicized literals at the bottom of the figure.

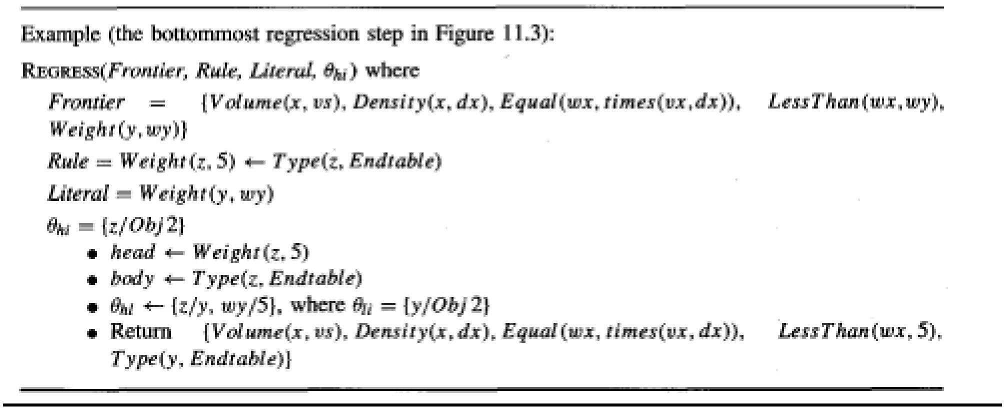
This final set of literals, shown at the bottom of figure, forms the body of the final rule.

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

The heart of the regression procedure is the algorithm that at each step regresses the current frontier of expressions through a single Horn clause from the domain theory



```
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 ← head of Rule
     • body ← body of Rule
     • $\theta_{hl}$ ← 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,* θ_{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 $\{Volume(x, vs), Density(x, dx), Equal(wx, times(vx, dx)), LessThan(wx, 5), Type(y, Endtable)\}$

REFINEMENT: Based on the analysed explanation, we refine our hypothesis by adding a rule that states it is safe to stack an object (Obj1) on another object (Obj2) if Obj1's weight is less than Obj2's weight. This rule captures the general condition derived from the specific example and the domain theory.

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:

PROLOG-EBG conducts a detailed analysis of individual training examples to determine how best to **generalize** from the **specific example to a general Horn clause hypothesis**. The following are the key properties of this algorithm.

1. PROLOG-EBG follows a bottom-up learning approach, where it starts with specific examples and progressively generalizes them to form more general rules.
2. PROLOG-EBG builds a knowledge base by representing the learned rules and their associated explanations. This knowledge base can be used for further reasoning and inference, enabling the algorithm to make predictions or classify new instances.
3. PROLOG-EBG produces generalized hypotheses by using prior knowledge to analyse individual examples, unlike purely inductive methods.
4. The attributes mentioned in the explanation of how an example satisfies the target concept are considered relevant.
5. Further analysis of the explanation helps regress the target concept to determine its weakest preimage, leading to more general constraints on relevant feature values.
6. Each learned Horn clause serves as a sufficient condition for satisfying the target concept. The set of learned Horn clauses covers positive training examples and instances sharing the same explanations.
7. The generality of learned Horn clauses depends on the formulation of the domain theory and the sequence in which training examples are considered.
8. PROLOG-EBG implicitly assumes that the domain theory is correct and complete. If the domain theory is incorrect or incomplete, the resulting learned concept may also be incorrect.

EBL Perspectives:

EBL as theory-guided generalization: It rationalizes generalization from examples based on a given domain theory.

EBL as example-guided theory reformulation: PROLOG-EBG can be seen as reformulating the domain theory into operational rules capable of classifying instances.

EBL as restating existing knowledge: In some cases, learning involves reformulating existing knowledge into a more operational form.

Discovering new features:

The ability of PROLOG-EBG to automatically formulate new features not explicitly present in the training examples is a significant aspect of its learning process. This capability allows the model to capture complex relationships between attributes and derive meaningful insights from the data.

Deductive Learning: PROLOG-EBG is a deductive, rather than inductive, learning process. That is, by calculating the **weakest preimage** of the explanation **it produces a hypothesis h that follows deductively from the domain theory B, while covering the training data D.**

PROLOG-EBG outputs a hypothesis h that satisfies the following two constraints:

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

$$D \wedge B \vdash h$$

where the training data D consists of a set of training examples in which x_i is the i^{th} training instance and f (x_i) is its target value (f is the target function). The first constraint describes the hypothesis h correctly predict the target value f (x_i) for each instance xi in the training data

The second constraint describes the impact of the domain theory in PROLOG-EBL: The output hypothesis is further constrained so that it must follow from the domain theory and the data.

Using similar notation, we can state the type of knowledge that is required by PROLOG-EBG for its domain theory. PROLOG-EBG assumes the domain theory B entails the classifications of the instances in the training data

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

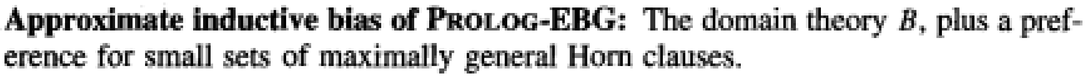
This constraint on the domain theory B assures that an explanation can be constructed for each positive example.

Comparison with ILP (Inductive logic programming

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. ILP systems output 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 inductive bias of a learning algorithm is a set of assertions that, together with the training examples, deductively entail subsequent predictions made by the learner. The importance of inductive bias is that it characterizes how the learner generalizes beyond the observed training examples.



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

Knowledge level learning: PROLOG-EBG can entail predictions beyond those entailed by the domain theory alone, known as knowledge-level learning.

The hypothesis h output by PROLOG-EBG follows deductively from the domain theory B and training data D. In fact, by examining the PROLOG-EBG algorithm it is easy to see that h follows directly from B alone, independent of D.

The **Lemma Enumerator** is an imaginary algorithm that enumerates all possible proof trees based on assertions in the domain theory (B). For each proof tree, the algorithm calculates the weakest preimage and constructs a Horn clause, similar to how PROLOG-EBG operates**. The only difference between LEMMA-ENUMERATOR and PROLOG-EBG is that LEMMA ENUMERATOR ignores the training data and enumerates all proof trees**

➢ EXPLANATION BASED LEARNING OF SEARCH CONTROL KNOWLEDGE:

Explanation-Based Learning (EBL) of search control knowledge refers to the process of using explanations derived from past problem-solving experiences to improve future problem-solving efficiency in search-based systems.

Explanation-based learning (EBL) has been particularly successful in addressing the challenge of learning to control search in complex problem-solving domains.

One significant application area of EBL is in speeding up search processes in games like chess, as well as in various optimization and scheduling problems. In such domains, the problem typically involves searching through a vast space of possible moves or configurations to find an optimal solution or move. EBL can help improve the efficiency of this search process by learning from past experiences

Let’s explore how we can formulate the problem of learning search control using **explanation-based learning (EBL**). consider a general search problem with the following components:

1. **Search Space (S):**
   - Represents the set of possible search states.
   - These states define the space in which we perform our search.
2. **Legal Search Operators (O):**
   - Consists of operators that transform one search state into another.
   - These operators guide the search process.
3. **Goal Predicate (G):**
   - A predicate defined over the search states.
   - Indicates which states are goal states.

The problem is to find a sequence of operators that transforms an arbitrary initial state $\mathbf{s_i}$ to a final state $\mathbf{s_f}$ satisfying the goal predicate **G.**

Now, let’s discuss how we can apply EBL to enhance search control:

- **Target Concepts for Operators:**
    - For each operator o in O, we can learn a separate target concept.
    - Specifically, we might attempt to learn: “the set of states for which operator o leads toward a goal state.”
- **Internal Structure of the Problem Solver:**
    - The choice of target concepts depends on the problem solver’s internal structure.
    - For example, if the problem solver follows a means-ends planning approach (establishing and solving subgoals), we could learn target concepts like: “the set of planning states in which subgoals of type A should be solved before subgoals of type B.”

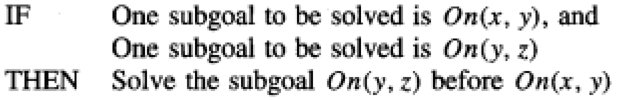
One example of a system that employs EBL for search control is

PRODIGY(PROblem-solving by Directed GREedy search of Instantiated knowledge and Yielding”), a domain-independent planning system i.e., PRODIGY accepts the definition of a problem domain in terms of the state space and operators and then solves problems by finding a sequence of operators leading to a goal state.

PRODIGY uses a means-ends planner to find solutions i.e., it decomposes problems into subgoals, solves them, and combines their solutions into a solution for the full problem.

PRODIGY integrates EBL by defining target concepts appropriate for control decisions. For instance, it learns rules based on target concepts like "the set of states in which subgoal A should be solved before subgoal B or which operator to consider for solving a subgoal.

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 $On(x, y)$

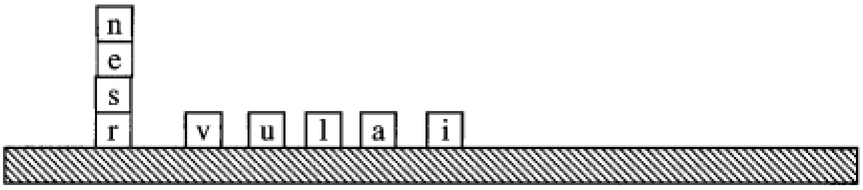


fig: Block stacking problem

The goal is to stack the blocks so that they spell the word "universal."

PRODIGY would decompose this problem into several subgoals to be achieved, including On(U, N), On(N, I), etc. Notice the above rule matches the subgoals On(U, N) and On(N, I), and recommends solving the subproblem On(N, I) before solving On(U, N).

The justification for this rule (and the explanation used by PRODIGY to learn the rule) is that if we solve the subgoals in the reverse sequence, we will encounter a conflict in which we must undo the solution to the On(U, N) subgoal in order to achieve the other subgoal On(N, I). PRODIGY learns by first encountering such a conflict, then explaining to itself the reason for this conflict and creating a rule such as the one above.

A second example of a general problem-solving architecture that incorporates a form of explanation-based learning is the **SOAR**(State, Operator, And Result) **system.** SOAR employs a variant of EBL called **chunking**, **which extracts general conditions under which explanations apply. It learns by explaining situations where its search strategy leads to inefficiencies and uses these explanations to guide future decision-making.**

Both PRODIGY and SOAR leverage EBL to enhance their problem-solving capabilities, they differ in their specific architectures, problem-solving approaches, and application domains. PRODIGY focuses more on domain-independent planning and explicit rule learning, while SOAR emphasizes cognitive modelling and the extraction of general problem-solving strategies.

➢ INDUCTIVE-ANALYTICAL APPROACHES TO LEARNING:

It refers to a category of learning methods that combine both inductive and analytical reasoning to acquire knowledge from data. These approaches aim to leverage both bottom-up induction from examples and top-down analytical reasoning based on existing knowledge or theories.

Consider the learning problem:

Given a set of training examples (D) and a domain theory (B), the task is to determine a hypothesis (h) from a space of candidate hypotheses (H) that best fits both the training examples and the domain theory.

The error of a hypothesis is defined with respect to both the training data and the domain theory.

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

Where, errorD(h) is defined to be the proportion of examples from the training data D that are misclassified by h.

errorB(h) of h with respect to a domain theory B to be the probability that h will disagree with domain theory B on the classification of a randomly drawn instance.

However, determining appropriate values for the weights kD and kB is challenging.

- If we have a very poor theory and a great deal of reliable data, it will be best to weight errorD(h) more heavily.
- Given a strong theory and a small sample of very noisy data, the best results would be obtained by weighting errorB(h). when the quality of data or theory is not known, finding the right balance become unclear.

  Various strategies, such as empirical estimation or sensitivity analysis, may be employed to explore different weightings and their impact on the learned hypotheses.

Bayesian perspective: The Bayesian perspective provides an alternative approach for weighting prior knowledge and data in learning tasks. The Bayesian perspective computes the posterior probability of a hypothesis given the observed training data (training examples) and prior knowledge (domain theory). but bayes is applicable only when accurate prior distributions are known.

Hypothesis space search: Hypothesis space search refers to the process in machine learning where algorithms explore and evaluate different hypotheses to find the one that best fits the observed data. the key components involved in hypothesis space search are

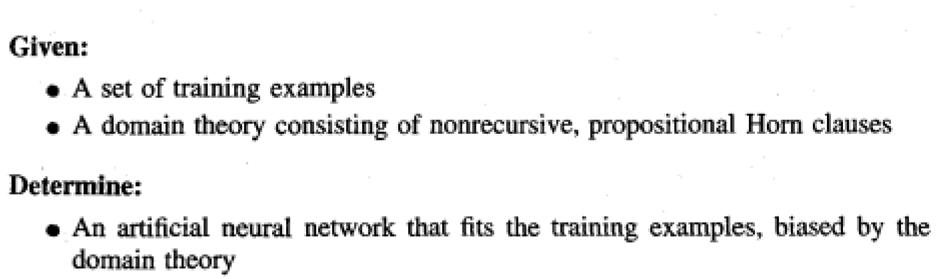
- **H: Hypothesis space:** set of all possible hypotheses or models
- **h_0: Initial hypothesis**
- **O: Set of search operators:** operations used to navigate the hypothesis space. Search operators define how one hypothesis can be transformed into another hypothesis. Common search operators include mutation, crossover, pruning, and refinement.
- **G: Goal criterion:** specifies the objective or criteria used to evaluate and compare different hypotheses. The goal criterion can vary depending on the specific task and learning algorithm. It could involve minimizing prediction error, maximizing likelihood, or optimizing some other performance metric.

There are three different methods for incorporating prior knowledge into the search process when learning hypothesis.

- **Use prior knowledge to derive an initial hypothesis from which to begin the search**: In this approach, the domain theory B is utilized to construct an initial hypothesis $h0$ that is consistent with B. This initial hypothesis is then refined inductively using a standard inductive method. For instance, the KBANN system employs this approach to learn artificial neural networks. By starting the search with a hypothesis consistent with the domain theory, the final output hypothesis is more likely to align with the theory.
- **Use prior knowledge to alter the objective of the hypothesis space search:** Here, the goal criterion G is adjusted to require that the output hypothesis fits not only the training examples but also the domain theory. An example of this approach is the EBNN system, which modifies the criterion used in inductive learning of neural networks. Instead of solely minimizing the squared error over the training data, EBNN optimizes a criterion that incorporates the error of the learned network relative to the domain theory.
- **Use prior knowledge to alter the available search steps:** In this method, the set of search operators O is modified based on the domain theory. For instance, the FOCL system learns sets of Horn clauses by expanding the set of search alternatives available during hypothesis revision. By allowing the addition of multiple literals in a single search step when warranted by the domain theory, FOCL enables "macro-moves" that can significantly change the search trajectory compared to the original inductive algorithm.

- **USE PRIOR KNOWLEDGE TO DERIVE AN INITIAL HYPOTHESIS FROM WHICH TO BEGIN THE SEARCH:**

The KBANN (Knowledge-Based Artificial Neural Network) algorithm utilizes an approach that initializes the hypothesis to perfectly fit the domain theory and then refines it inductively based on the training data.

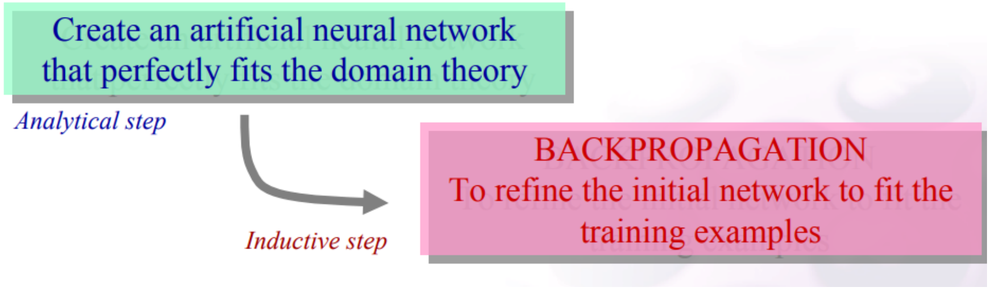


Given:

- A set of training examples
- A domain theory consisting of nonrecursive, propositional Horn clauses

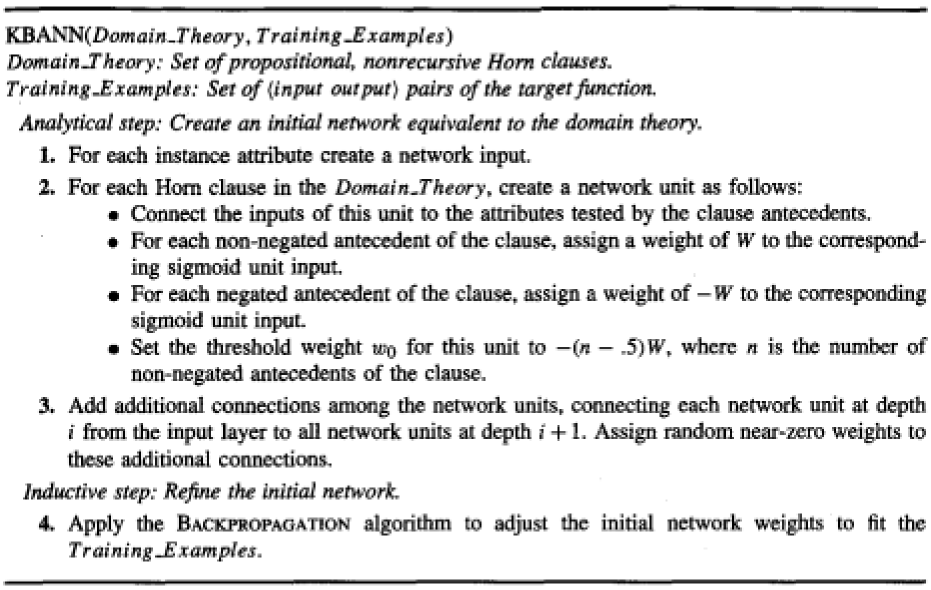
Determine:

- An artificial neural network that fits the training examples, biased by the domain theory



Create an artificial neural network
that perfectly fits the domain theory
Analytical step
BACKPROPAGATION
To refine the initial network to fit the training examples
Inductive step

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 BACKPROPAGATION algorithm to refine this initial network to fit the training examples

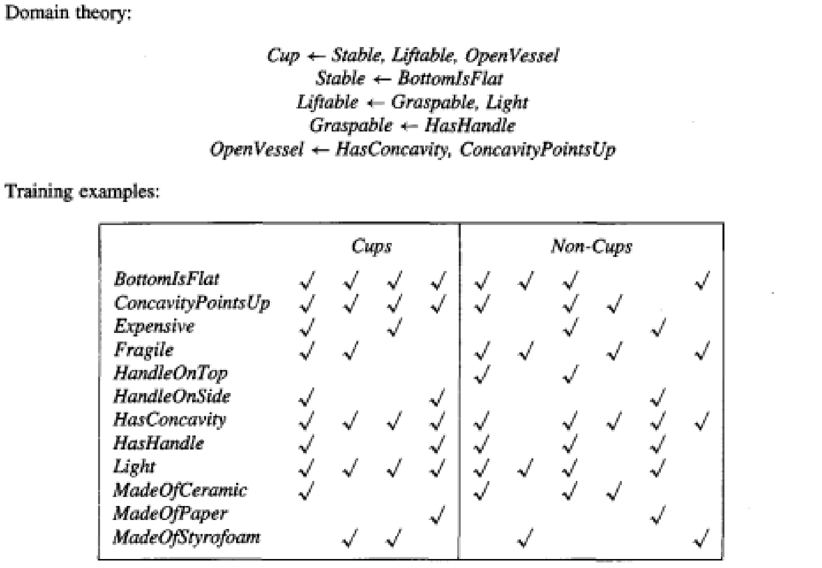


```
KBANN(Domain_Theory, Training_Examples)
Domain_Theory: Set of propositional, nonrecursive Horn clauses.
Training_Examples: Set of ⟨input output⟩ pairs of the target function.
 Analytical step: Create an initial network equivalent to the domain theory.
   1. For each instance attribute create a network 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.
```

Example: consider the simple learning problem. Each instance describes a physical object in terms of the material from which it is made, whether it is light, etc. **The task is to learn the target concept Cup defined over such physical objects.**

The following figure describes a set of training examples and a domain theory for the Cup target concept. Notice the domain theory defines a Cup as an object that is Stable, Liftable, and an OpenVessel.

The domain theory also defines each of these three attributes in terms of more primitive attributes, terminating in the primitive, operational attributes that describe the instances. Note the domain theory is not perfectly consistent with the training examples. For example, the domain theory fails to classify the second and third training examples as positive examples. KBANN uses the domain theory and training examples together to learn the target concept more accurately than it could from either alone.



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		✓	✓			✓				✓

TABLE 12.3
The *Cup* learning task. An approximate domain theory and a set of training examples for the target concept *Cup*.

Step1: The initial neural network (feed forward network h) is constructed to match the predictions of the domain theory(B).

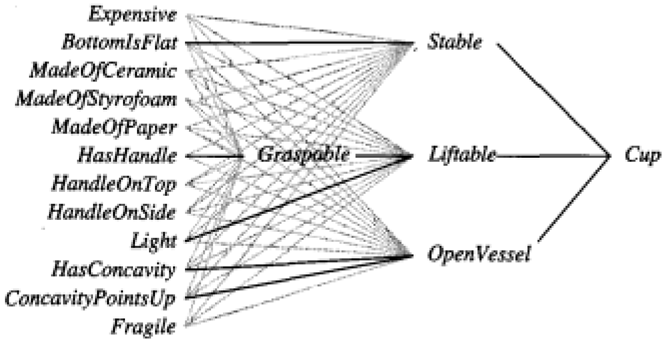


FIGURE 12.2
A neural network equivalent to the domain theory. This network, created in the first stage of the KBANN algorithm, produces output classifications identical to those of the given domain theory clauses. Dark lines indicate connections with weight *W* and correspond to antecedents of clauses from the domain theory. Light lines indicate connections with weights of approximately zero.

Step2: The refined network(h), obtained after applying BACKPROPAGATION, adjusts weights to fit the training examples(D) better.

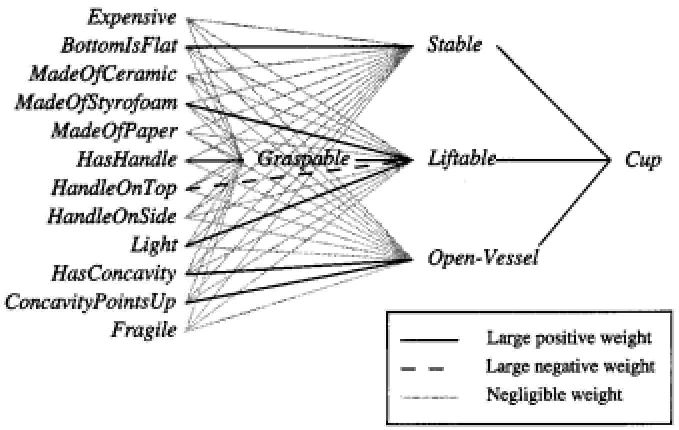


FIGURE 12.3
Result of inductively refining the initial network. KBANN uses the training examples to modify the network weights derived from the domain theory. Notice the new dependency of *Liftable* on *MadeOfStyrofoam* and *HandleOnTop*.

Advantages of KBANN:

- Generalizes more accurately than back propagation when given an approximately correct domain theory and when training data is scarce
- Initializes the hypothesis

Limitations of KBANN:

- Accommodate only propositional domain theories
- Misled when given highly inaccurate domain theories

➢ USING PRIOR KNOWLEDGE TO ALTER THE SEARCH OBJECTIVE

The TANGENTPROP algorithm offers an alternative approach to incorporating prior knowledge into the training process of neural networks.

Overview:

Prior knowledge: Incorporates domain knowledge expressed as derivatives of the target function with respect to input transformations. Prior knowledge can be expressed quite naturally for certain applications such as recognizing hand written characters.

Training with derivatives: Extends the training process to fit both training values and training derivatives.

Enhanced Generalization: By considering training derivatives, the algorithm aims to generalize more accurately, especially in cases where prior knowledge about input transformations is available.

TANGENTPROP algorithm

TANGENTPROP algorithm is designed for training neural networks by incorporating training derivatives alongside training values, aiming to improve generalization accuracy. It is particularly useful when prior knowledge, expressed as derivatives of the target function, is available.

TANGENTPROP follows a series of steps similar to standard backpropagation but includes an additional consideration for training derivatives. While backpropagation minimizes the error between predicted and actual outputs, TANGENTPROP goes further by also minimizing the error between the predicted derivatives and the actual derivatives of the target function.

Example: Consider a scenario where we aim to recognize handwritten characters. We have a dataset of images of handwritten digits and their labels, along with training derivatives indicating invariance to small translations. The TANGENTPROP algorithm is applied to train a neural network using this data. It involves forward propagation, error computation, backpropagation with a focus on training derivatives, gradient descent update, and convergence check.

Consider a learning task involving an instance space X and target function f.

Up to now we have assumed that each training example consists of a pair **(x_i, f (x_i))** that describes some instance **x_i** and its training value **f(x_i)**. The **TANGENTPROP** algorithm assumes various training derivatives of the target function are also provided. For example, if each instance **x_i** is described by a single real value, then each training example may be of the form $\left\langle x_i, f(x_i), \frac{\partial f(x)}{\partial x} \mid x_i \right\rangle$. Here $\frac{\partial f(x)}{\partial x} \mid x_i$, denotes the derivative of the target function f with respect to x, evaluated at the point x = xi.

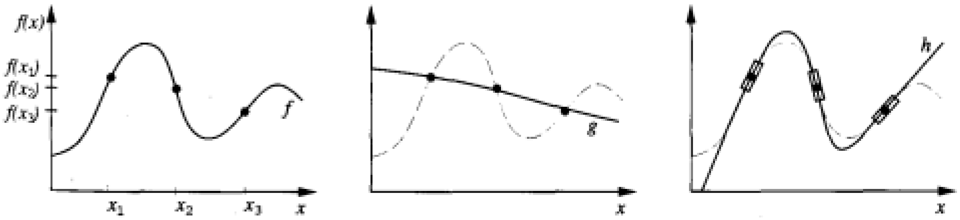


FIGURE 12.5
Fitting values and derivatives with TANGENTPROP. Let f be the target function for which three examples $\langle x_1, f(x_1)\rangle$, $\langle x_2, f(x_2)\rangle$, and $\langle x_3, f(x_3)\rangle$ are known. Based on these points the learner might generate the hypothesis g. If the derivatives are also known, the learner can generalize more accurately h.

The task is to learn the target function f shown in the leftmost plot of the figure, based on the three training examples **(x_l, f (x_l)), (x_2, f (x_2)), and (x_3, f (x_3))**. Given these three training examples, the **BACKPROPAGATION** algorithm can be expected to hypothesize a smooth function, such as the function g depicted in the middle plot of the figure. The rightmost plot shows the effect of providing training derivatives, or slopes, as additional information for each training example $\left\langle x_i, f(x_i), \frac{\partial f(x)}{\partial x} \mid x_i \right\rangle$. By fitting both the training values **f (x_i)** and these training derivatives $\frac{\partial f(x)}{\partial x} \mid x_i$,the learner has a better chance to correctly generalize from the sparse training data. we considered only simple kinds of derivatives of the target function. In fact, **TANGENTPROP** can accept training derivatives with respect to various transformations of the input x

Consider, for example, the task of learning to recognize handwritten characters. In particular, 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 $s(\alpha, x)$, which rotates the image x by α degrees. In other words, we can assert the following training derivative for every training instance x_i

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

where f is the target function and $s(\alpha, x)$ is the image resulting from applying the transformation s to the image x_i. In **TANGENTPROP** these training derivatives are incorporated into the error function that is minimized by gradient descent.

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

where x_i denotes the i^{th} training instance, f denotes the true target function, and f denotes the function represented by the learned neural network.

In the **TANGENTPROP** algorithm, an additional term is introduced into the error function to address discrepancies between the training derivatives and the actual derivatives of the learned neural network function. This allows **TANGENTPROP** to accommodate multiple transformations, such as rotational and translational invariance of character identity.

Each transformation, denoted as $s_j(\alpha, x)$ is characterized by a continuous parameter α where S_j is a differentiable function satisfying $s_j(0, x) = x$ (meaning the identity function when the transformation parameter is zero, representing no transformation). For example, for a rotation of zero degrees, the transformation function remains the identity function.

For each transformation $s_j(\alpha, x)$, **TANGENTPROP** evaluates the squared error between the specified training derivative and the actual derivative of the learned neural network. This results in a modified error function, denoted as **E**, which consists of two terms:

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

where **μ** is a constant provided by the user to determine the relative importance of fitting training values versus fitting training derivatives. Notice the first term in this definition of **E** is the original squared error of the network versus training values, and the second term is the squared error in the network versus training derivatives.

Advantages of TANGENTPROP:

- By incorporating training derivatives, **TANGENTPROP** can provide more accurate generalization
- The ability to account for transformations and invariances in the input data can make **TANGENTPROP** more robust to variations and noise in the dataset.

Limitations of TANGENTPROP:

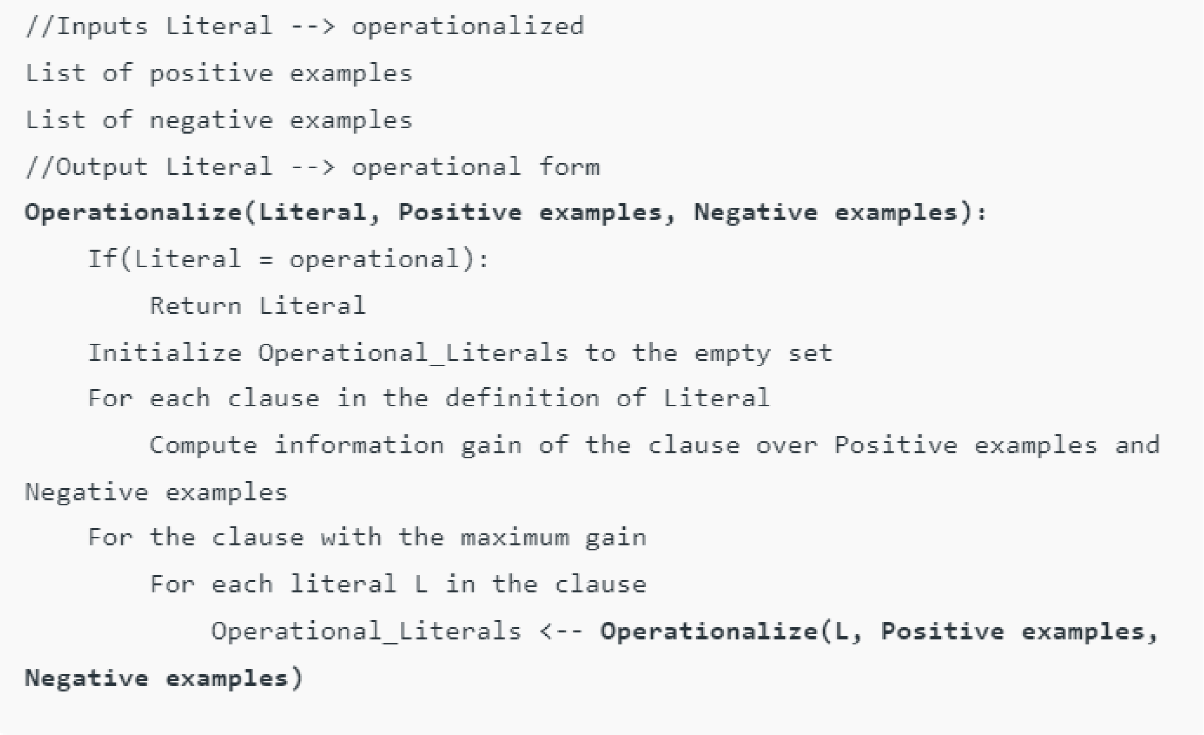
- Requirement of prior knowledge in the form of derivatives
- Complexity in transformation specification
- Incorporating training derivatives into the error function increases computational complexity, especially when dealing with high-dimensional data or large datasets.

➢ USING PRIOR KNOWLEDGE TO AUGMENT SEARCH OPERATORS

The **FIRST ORDER COMBINED LEARNER (FOCL) ALGORITHM** is an extension of the purely inductive, **FOIL** Algorithm. It uses domain theory to further improve the search for the best-rule and greatly improves accuracy. It incorporates the methods of **Explanation-Based learning (EBL) into the existing methods of FOIL**.

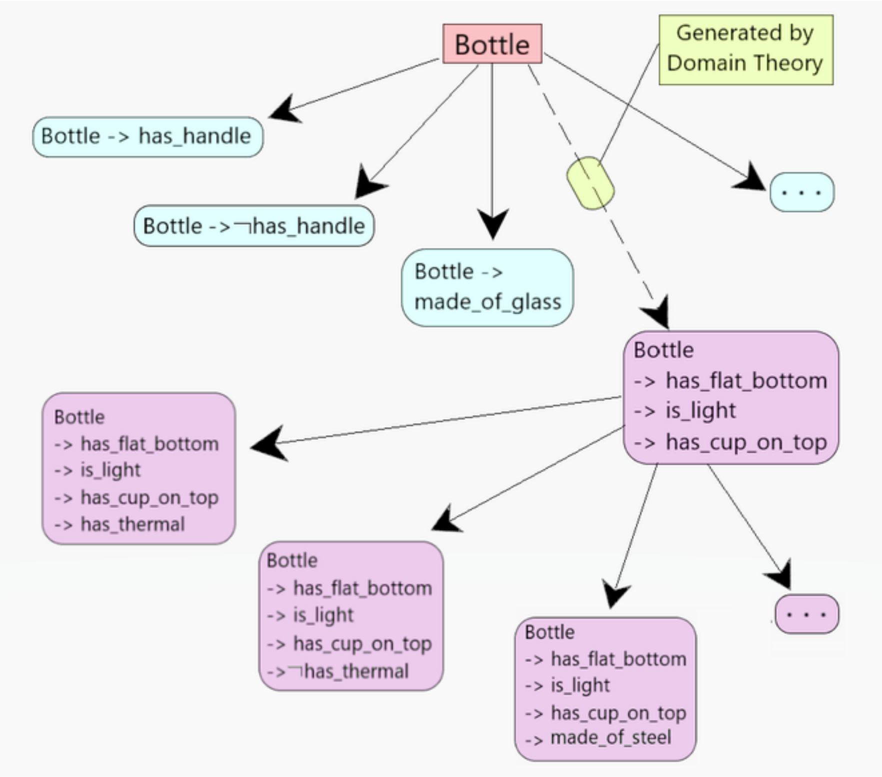
The goal of **FOCL,** like **FOIL**, is to create a rule, that covers all the positive examples and none of the negative examples.

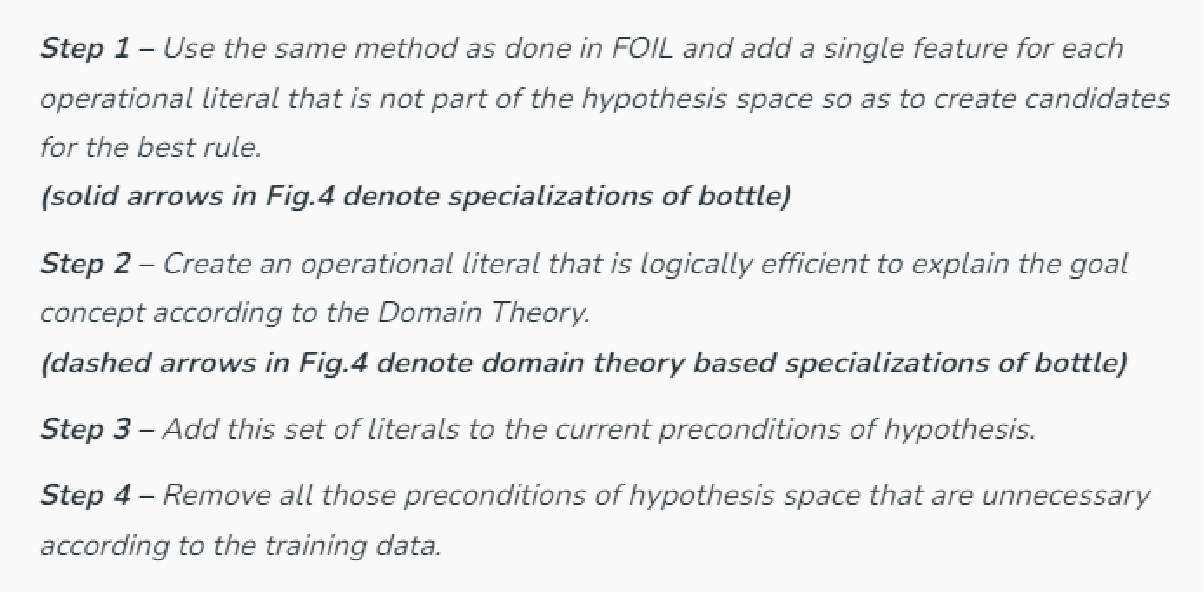
Algorithm



```
//Inputs Literal --> operationalized
List of positive examples
List of negative examples
//Output Literal --> operational form
Operationalize(Literal, Positive examples, Negative examples):
    If(Literal = operational):
        Return Literal
    Initialize Operational_Literals to the empty set
    For each clause in the definition of Literal
        Compute information gain of the clause over Positive examples and
Negative examples
    For the clause with the maximum gain
        For each literal L in the clause
            Operational_Literals <-- Operationalize(L, Positive examples,
Negative examples)
```

Working of the Algorithm


Bottle
Generated by Domain Theory
Bottle -> has_handle
Bottle ->¬has_handle
Bottle -> made_of_glass
. . .
Bottle -> has_flat_bottom -> is_light -> has_cup_on_top
Bottle -> has_flat_bottom -> is_light -> has_cup_on_top -> has_thermal
Bottle -> has_flat_bottom -> is_light -> has_cup_on_top ->¬has_thermal
Bottle -> has_flat_bottom -> is_light -> has_cup_on_top -> made_of_steel
. . .



***Step 1** – Use the same method as done in FOIL and add a single feature for each operational literal that is not part of the hypothesis space so as to create candidates for the best rule.*
(solid arrows in Fig.4 denote specializations of bottle)

***Step 2** – Create an operational literal that is logically efficient to explain the goal concept according to the Domain Theory.*
(dashed arrows in Fig.4 denote domain theory based specializations of bottle)

***Step 3** – Add this set of literals to the current preconditions of hypothesis.*

***Step 4** – Remove all those preconditions of hypothesis space that are unnecessary according to the training data.*

Let us consider the example shown in Fig

- First, FOCL creates all the candidate literals that have the possibility of becoming the best-rule (all denoted by solid arrows). Something we have already seen in the FOIL algorithm. In addition, it creates several logically relevant candidate literals of its own. (the domain theory)

- Then, it selects one of the literals from the domain theory whose precondition matches with the goal concept. If there are several such literals present, then it just selects one which gives the **most information** related to the goal concept.

- Now, all those literals that removed unless the affect the classification accuracy over the training examples. This is done so that the domain theory doesn’t overspecialize the result by addition irrelevant literals. This set of literals is now added to the preconditions of the current hypothesis.

- Finally, one candidate literal which provides the maximum information gain is selected out two specialization methods. (FOIL and domain theory)