UNIT IV

4. LEARNING

Learning form Observations – Forms of Learning – Inductive Learning – Learning Decision Trees – Ensemble Learning – Knowledge in Learning – Logical Formulation of Learning – Explanation based Learning – Learning Using Relevant Information – Inductive Logic Programming – Statistical Learning Methods – Learning with Complete Data – Learning with Hidden Variable – EM Algorithm – Instance based Learning – Neural Networks – Reinforcement Learning – Passive Reinforcement Learning – Active Reinforcement Learning – Generalization in Reinforcement Learning.

4.1. Learning from Observations

Studying takes position as the agent seeing its communications with the universe and its individual choice making procedure. Studying takes several varieties depending on the environment of the presentation element, the element to be enhanced, and the existing comments.

4.2. Forms of Learning

Learning agent is a performance agent that chooses what activities to be taken and a learning component that changes the performance component so that best choices can be taken in the upcoming. The plan of a learning element is influenced by 3 main issues:

- What *presentation* is utilized for the elements?
- What *feedback* is possible to learn these elements?
- Which *elements* of the performance element to be learned?

The components of those agents contain the following:

1. A straight mapping from situation on the present situation to activities.
2. A way to assume related properties of the universe from the percept-sequence.
3. Details regarding the method the universe develop and regarding the outcomes of available operations the agent may get.
4. Function details representing the importance of the global situation
5. Operation value details representing the importance of operations.
6. A target that explains classes of status whose success increases the agent's function.

The kind of comments applicable for studying is typically the most essential factor in resolving the environment of the studying problem, which the agent deals with.

The domain of machine-learning typically differentiates with 3 cases: **reinforcement, supervised, and unsupervised** learning.

Reinforcement Learning

Here learning arrangement is rather than being advised by a tutor, it learns from supplementing, i.e. by occasional rewards. The presentation of the learned knowledge just performs very imperative job in deciding how the learning algorithm should perform. The final main feature in the plan of learning methods is the possibility of earlier knowl*edge.*

Supervised Learning

1. A correct answer for each example or instance is available.
2. Learning is done from known sample input and output.

Unsupervised Learning

It is a learning pattern is which correct answers are not given for the input. It is mainly used in probabilistic learning system.

4.3. Inductive Learning

An **example** is a couple of values ‘*x’ and ‘f* (z)’, wherever ‘x’ is the entered value and ‘*f(x)’* is the output of the function tested to ‘*x’.* The job of **pure-inductive-inference** or **induction** is this: Specified a set of instances of ‘*f’,* returns a function ‘*h’* that estimates ‘f’. The function ‘h’ is known as a **hypothesis.**

For the functions which are nondeterministic, there is a predictable tradeoff between the degrees of jit to the information and the difficulty of the hypothesis. There is an exchange between the difficulty of selecting simple, reliable hypotheses inside that space and the expressiveness of a hypothesis space.

4.4. Learning Decision Trees

Decision tree introduction is the easiest and most effective variety of learning-algorithm. It provides a better introduction to the field of introductory learning, and is simple to apply.

Decision Trees as Performance Elements

A decision tree gets as entry an item or scenario defined by a group of attributes and gives back a selection the expected resultant value for the entered value. The input attributes may be continuous or discrete. At this time, we imagine distinct inputs. The resultant value also can be continuous or distinct; studying a distinct valued characteristic is known as **classification** learning, learning a continual function is known as **regression**.

A decision tree arrive its selection by executing a series of assessments. Every internal node inside the tree correlates to check of the value of one of the properties, and the branches from the node are categorized with the available values of the check. Every leaf node inside the tree gives the value to be revisit if that leaf is arrived. The decision-tree presentation appears to be most normal for peoples; certainly, several "How To" guidelines (e.g., for vehicle repair) are drafted totally as a single-decision-tree extending more than 100s of leafs.

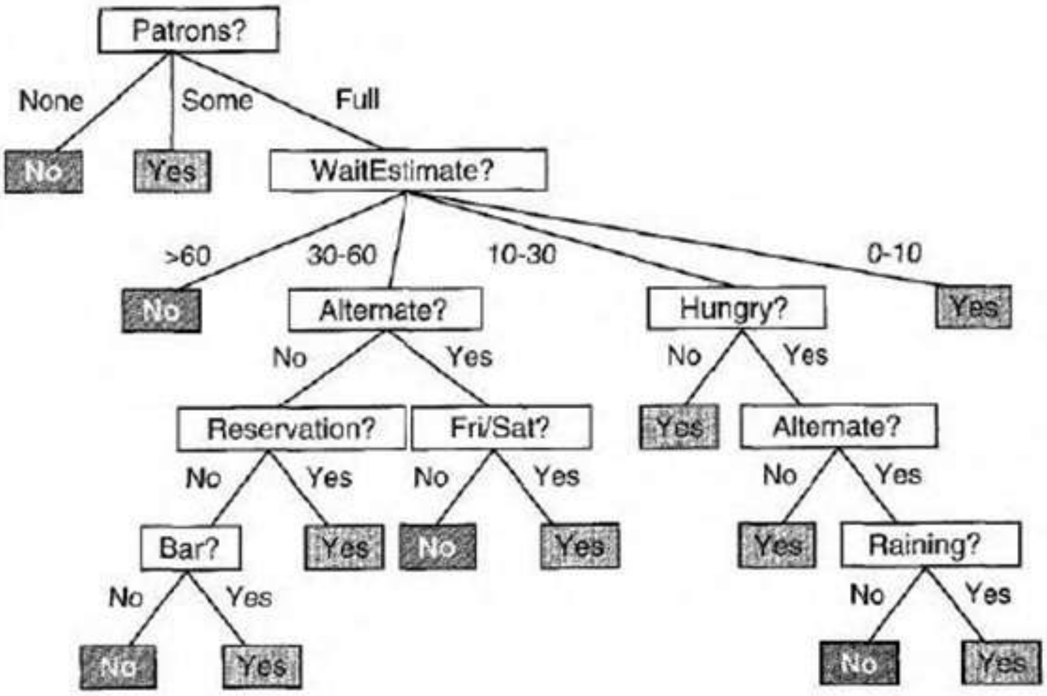


Figure: The Decision Tree for Choosing Whether to Stay for a Table

Expressiveness of Decision Trees

The relationship between the conclusion and the logical combination of attribute values the decision tree expressed as:

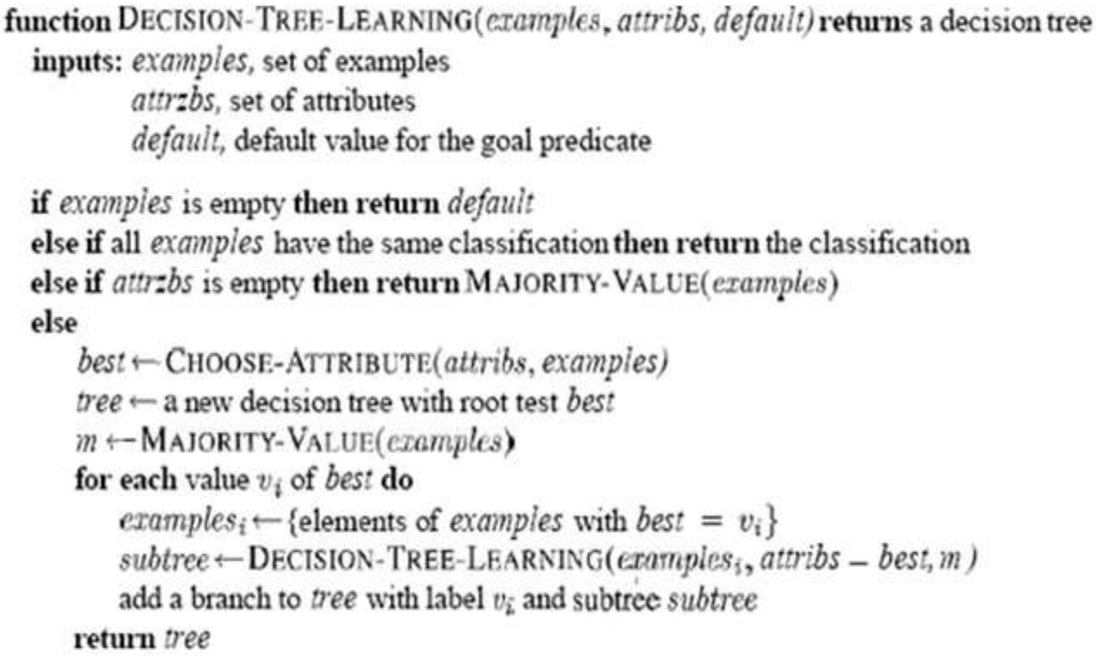
- **Propositional logic –** one variable and other predicates are unary**.**
- **Parity function -** returns one if an even number of input are one.
- **Majority function -** returns one if greater than half of its input is one.

The truth-table contains two n rows, for the reason that every input case is defined by n-attributes. If it receives two n bits to describe the function, then there are twenty two n dissimilar functions on 'n' Attributes.

Inducing Decision Trees from Examples

An instance for a Boolean-decision tree contains of a vector of giving attributes, 'X' and a single-Boolean resultant value is *'y'.*

DECISION TREE LEARNING algorithm is displayed in the bellow Figure:



```
function DECISION-TREE-LEARNING(examples, attribs, default) returns a decision tree
  inputs: examples, set of examples
          attrzbs, set of attributes
          default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attrzbs is empty then return MAJORITY-VALUE(examples)
  else
     best ← CHOOSE-ATTRIBUTE(attribs, examples)
     tree ← a new decision tree with root test best
     m ← MAJORITY-VALUE(examples)
     for each value $v_i$ of best do
        $examples_i$ ← {elements of examples with best = $v_i$}
        subtree ← DECISION-TREE-LEARNING($examples_i$, attribs − best, m)
        add a branch to tree with label $v_i$ and subtree subtree
     return tree
```

Figure: Decision tree Learning Algorithm

The performance of learning algorithm depends on,

- Choosing attribute tests.
- Data set for training and testing without noise and over lifting.

Selecting Attribute Tests

The method utilized in decision-tree learning for choosing attributes is planned to reduce the depth of the last tree. The plan is to choose the attribute that moves as distant as available near giving a precise arrangement of the examples. A best attribute splits the examples into groups, which are all negative or positive.

CHOOSE-ATTRIBUTE Function

The measure could contain its highest value whenever the attribute is best and its least value whenever the attribute is of not usable at all.

In common, if the available solution ‘vi’ contain probabilities “P(vi)”, which the knowledge content ‘I’ of the real solution is specified by

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1} -P(v_i) \log_2 P(v_i)$$

Evaluating the Work of the Learning Algorithm

A learning algorithm is best, if it gives suggestion that perform a better work of estimating the categorizations of not seen examples. It is most suitable to accept the following methods:

1. Gather a huge group of examples.
2. Separate it into 2 not joint groups: the **training group** and the **test group.**
3. Apply the learning algorithm to the training group, producing theories.
4. Determine the percentage of instances in the test group, which is properly arranged by ‘h’.
5. Replicate steps two to four for dissimilar sizes of training groups and dissimilar at random chosen training groups of all size.

The outcome of this process is a dataset that may be prepared to provide the average forecast excellence as a function of the size of the training group. This function may be designed on a chart, offering what is termed the **learning-curve** for the algorithm on the appropriate field.

Noise and Over Fitting

Noise: more than two examples with the similar explanation in attributes but dissimilar classifications.

Over lifting: When there is a large group of probable hypothesis tree result with meaningless regularity in the data. To overcome the problem of over lifting decision tree pruning or cross-validation can be applied.

Improving the Appropriateness of Decision Trees

The decision tree induction in applied on variety of problems, to address the following issues:

1. Missing data.
2. Multi-valued attributes.
3. Numerical valued and Continuous input attributes.
4. Repeated value output attributes.

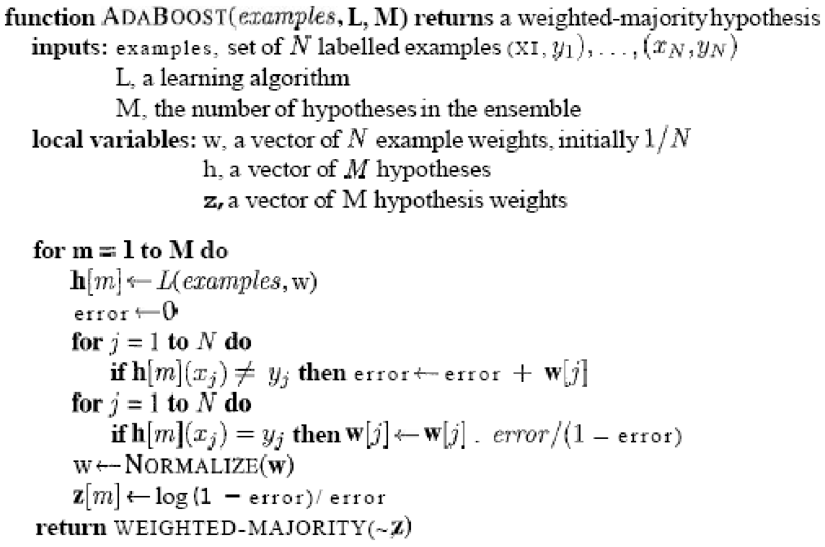
4.5. Ensemble Learning

The intention of **ensemble learning** process is to choose an entire group, or **ensemble** of hypotheses from joining their predictions and the hypothesis space.

The most popular ensemble learning methods are:

1. Boosting
2. Bagging

Boosting: Most generally utilized ensemble approach is known as **boosting.** To know how it process, "WEIGTHTED-TRAINING" require initially defining the thought of a **weighted training set.** In such a training group, every example has a related weight "wJ>0". The maximum weight of an instance, the greater is the import connected to it throughout the studying of a hypothesis. It is sincere to change the learning algorithms. We contain observed so far to work with weighted training groups; Boosting begins with w=1 for every examples (i.e., a general training group). From this group, it produces the primary hypothesis-*hl.* This hypothesis can analyze a few of the training examples properly and a few improperly. We could similar to the later hypothesis to perform well on the misclassified instances, so we maximize their weights while minimizing the weights of the properly classified examples. From this latest weighted training group, we produce hypothesis-*h2.* The procedure carries on in this method upto we have produced 'M' hypotheses, where 'M' is an entry to the boosting-algorithm. The last ensemble hypothesis is a weighted popular grouping of all the 'M' hypotheses, all weighted-corresponding to how can it worked on the training group.



```
function ADABOOST(examples, L, M) returns a weighted-majority hypothesis
  inputs: examples, set of $N$ labelled examples $(x_1, y_1), \dots, (x_N, y_N)$
          L, a learning algorithm
          M, the number of hypotheses in the ensemble
  local variables: w, a vector of $N$ example weights, initially $1/N$
                   h, a vector of $M$ hypotheses
                   z, a vector of M hypothesis weights

  for m = 1 to M do
      h[m] ← L(examples, w)
      error ← 0
      for $j$ = 1 to $N$ do
          if $h[m](x_j) \neq y_j$ then error ← error + w[j]
      for $j$ = 1 to $N$ do
          if $h[m](x_j) = y_j$ then w[j] ← w[j] . error/(1 − error)
      w ← NORMALIZE(w)
      z[m] ← log (1 − error)/ error
  return WEIGHTED-MAJORITY(~z)
```

Figure: The ADABOOST Alternative of the Boosting Process for Ensemble Learning

4.6. Knowledge in Learning

This easy knowledge free image of inductive learning continued upto the before time 1980s.

The recent method is to propose agents that previously recognize a bit and attempting to study a few most these cannot noise similar to an extremely deep insight, but it creates entirely a different way we plan the agents. It can also contain a few relevance to our hypothesis regarding how knowledge itself working. The common thought is displayed schematically in the bellow figure.

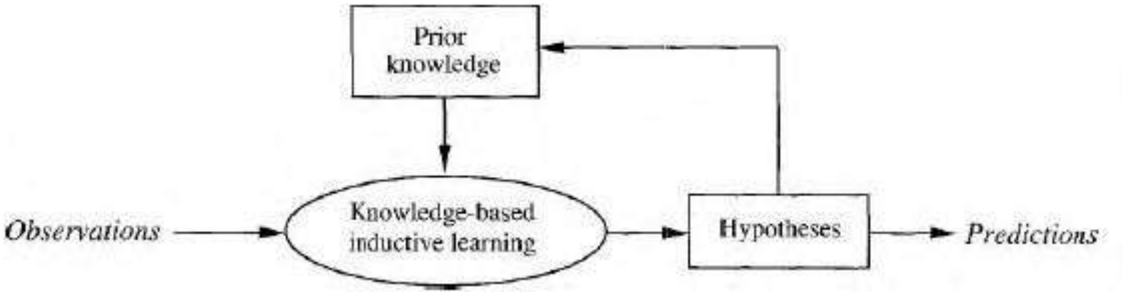


Figure: A cumulative-learning Procedure Utilizes, and Inserts to its Store of Background Information Over Time

4.7. Logical Grouping of Learning

Pure inductive learning is a procedure of discovering a theory that acknowledges with the noticed instances. We focus on this description to the case wherever the theory is characterized by a group of logical statements.

Examples of Hypotheses

The restaurant-learning problem: Learning a regulation for selecting regardless if to wait for a table. Instances are explained by an **attributes** like FrilSat, Bar, Alternate, and so on.

For example, a decision-tree declares that the target estimate is correct of an object if any branches noted to *true* is fulfilled. Every hypothesis forecast that certain group of instances; namely, these which gratify its applicant meaning shall be instances of the target estimate. This group is known as the **extension** of the estimate. 2 theories with dissimilar add-ons are then logically incompatible with one another, for the reason that they oppose on their forecasts for no less than 1 example. Logically, it is precisely similar to the decision rule of conclusion. We may describe inductive-learning in a logical situation as a procedure of regularly removing theories, which are incompatible with the instances, reducing the probabilities.

Current Best Hypothesis Search (CBHS)

The thought beyond the **CBHS** is to alter it as latest examples appear in order to continue consistency and to continue a single hypothesis. The hypothesis states it could be negative but it is really positive. The expansion of the hypothesis should be improved to contain it. It is known as **generalization;** the expansion of the hypothesis should be reduced to keep out the instance. It is known as **specialization.**

Define the **CURRENT-BEST LEARNING** algorithm:

```
function CURRENT-BEST-LEARNING(examples) returns a hypothesis

  H ← any hypothesis consistent with the first example in examples
  for each remaining example in examples do
     if e is false positive for H then
        H ← choose a specialization of H consistent with examples
     else if e is false negative for H then
        H ← choose a generalization of H consistent with examples
     if no consistent specialization/generalization can be found then fail
  return H
```

Figure: The current-best hypothesis learning algorithm. It looks for a reliable hypothesis and backtracks whenever no reliable generalization/specialization may be detected.

We described specialization and simplification as process that alter the **expansion** of a hypothesis. Currently we required to decide precisely how they may be executing as syntactic procedures that alter the person description connected with the hypothesis, so that a program may take them away. This is completed by initial mentioning that simplification. Specialization is also logical associations among hypotheses. If hypothesis-HI with explanation ‘C1’ is a simplification of hypothesis-H2 with explanation ‘C2’, next we should contain *t*xC2 (x)=+ C l(x). Then in order to build a simplification of ‘*Hz’,* we just required discovering an explanation ‘Cl’, which is logically implicated by ‘C2’. This is simply completed. For example, if C2(x) is an *Alternate(x)Patrons(x,Some),* next one probable simplification is specified by *Cl(x) Patrons(x, Some).* This is known as dropping **conditions.**

In such situations, the program should retrack to an earlier selection point. The recent better learning algorithm and its alternates have been utilized in several machine learning methods, beginning with PatrickWinston's(1970) arch-learning program.

With a huge count of instances and a huge space, though, a few complications happen:

1. The search procedure can involve a big deal of back-tracking.
2. Inspecting all the earlier instances another time for all change is highly expensive.

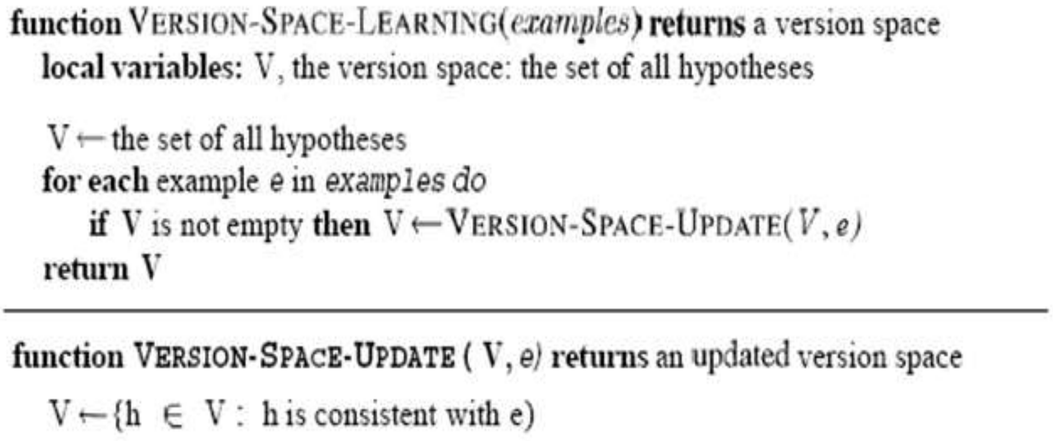
Hypothesis space may be a twice as exponentially big place.

Smallest Commitment Search

Retracking happens for the reason that the present better hypothesis method has to select a specific hypothesis as its better estimate even if it could not contain sufficient records still to be of the selection.

$$H1 \vee H2 \vee H3 \ldots \vee H\text{n}.$$

The group of hypotheses continuing is known as the **version-space,** and the learning algorithm is known as the candidate elimination algorithm or version space learning algorithm.



```
function VERSION-SPACE-LEARNING(examples) returns a version space
  local variables: V, the version space: the set of all hypotheses

  V ← the set of all hypotheses
  for each example e in examples do
      if V is not empty then V ← VERSION-SPACE-UPDATE(V, e)
  return V
--------------------------------------------------------------------
function VERSION-SPACE-UPDATE ( V, e) returns an updated version space
  V ← {h ∈ V : h is consistent with e)
```

Figure: The Version Space Learning Algorithm

4.8. Explanation based Learning (EBL)

EBL is a process for extract common rules from separate review. The method of **memorization** has extended been utilized in computer science to accelerate programs by storing the outcomes of computing. The essential thought of memo functions is to build up a data base of output/input pairs; whenever the function is called, it initially verifies the data base to observe regardless if, it may avoid resolving the problem from scrape. EBL obtains a best deal after, by generating common rules that wrap a complete class of instances.

Extracting General Rules from Examples

The constraint can include the Background information, in inclusion to the Hypothes*is* and the noticed Classifications and Explanations. In the instance of lizard warming, the caveman simplify by describing the victory of the pointed attach: it supports the lizard whilst putting the hand aside from the fire. From this clarification, they may collect a common rule: any sharp, stiff, long object may be utilized to soft-bodied, warm little edibles. This type of simplification procedure known as **EBL.**

"The agent cannot really study anything exactly latest from the case".

We may simplify this instance to occur with the residual limitation:

Background Hypothesis explanations categorizations.

In ILP systems, earlier awareness plays 2 key functions in decreasing the difficulty of learning:

1. Any hypothesis created should be reliable with the earlier awareness also as with the latest inspections; the successful hypothesis space range is decreased. To contain just these hypotheses, which are reliable with what is previously recognized?

2. For any specified group of inspections, the range of the hypothesis necessary to build a clarification for the inspections may be a lot decreased, for the reason that the previous awareness can be accessible to assist the latest rules in clarifying the inspections. The slighter the hypothesis, the simpler it is to discover.

The primary **EBL** procedure performs as follows:

1. Specified an instance, build evidence that the target predicate executes to the instance by utilizing the existing background awareness.

2. In parallel, build a simplified evidence tree for the mobilized target by applying the similar conclusion steps in the primary evidence.

3. Build a latest rule whose left side contains the ranges of the evidence tree and whose right side is the mobilized target (once using the required binding from the derived evidence).

4. Leave any situations, which are true anyway of the variables values in the target.

Enhancing Efficiency

There are 3 things concerned in the study of efficiency increases from EBL:

1. Inserting huge number of rules may drop the logic procedure; for the reason that the deduction method should yet to verify these rules still in cases wherever they perform disturbance give up a result. In other way, it raises the **Branching Factor** in the search space.
2. To recompense for the drop in logic, the resulting rules should propose important raises in velocity for the cases, which they make wrap. These raises come regarding mostly for the reason that the derivative rules keep away from last points otherwise it could be taken, because they cut down the evidence itself.
3. Obtained rules could be as common as probable, SCI, which they use to the biggest probable group of cases.

By simplifying from previous instance problems, EBL builds the database more proficient for the variety of problems, which it is sensible to imagine.

4.9. Learning Using Relevant Information

Statements state a strict form of connection: specified language, nationality is completely decided. Language is a meaning of nationality. These statements are known as **determinations** or **Functional Dependencies.** They happen so generally in definite varieties of requests (e.g., describing data base design) that a particular statement applied to draft them. We accept the document of Davies(1985):

$$\text{Language }(x, 1)+\text{Nationality }(x,n)$$

Determining the Hypothesis Space

Determinations state an enough basis language from which to build hypotheses relating to the goal predicate. This sentence may be confirmed by viewing that a specified resolution is rationally the same to a sentence that the proper explanations of the goal predicate is one of the groups of each explanation expressible by applying the predicates on the left part of the resolution.

Learning and Using Relevance Information

```
function MINIMAL-CONSISTENT-DET($E, A$) returns a set of attributes
    inputs: E, a set of examples
            $A$, a set of attributes, of size n

    for $i \leftarrow 0, \ldots,$ n do
        for each subset $A_i$ of $A$ of size $i$ do
            if CONSISTENT-DET?($A_i$, E) then return $A_i$
```

```
function CONSISTENT-DET?($A$, E) returns a truth-value
    inputs: A, a set of attributes
            E, a set of examples
    local variables: H. a hash table

    for each example e in E do
        if some example in H has the same values as e for the attributes $A$
            but a different classification then return false
        store the class of e in H, indexed by the values for attributes $A$ of the example e
    return true
```

Figure: Algorithm for Discovering a Smallest Consistent Determination

4.10. Inductive Logic Programming(ILP)

ILP joins inductive process with the capacity of initial order representation, focusing on the representation of hypothesis as logic program. It has increased status for 3 causes. Initially ILP proposes an accurate method to the common data based inductive- learning problem. Second it proposes entire algorithms for inducing common initial order hypothesis from instances that should then learn effectively in fields wherever the attribute-based algorithms are tough to execute.

Example

The common record based-induction problem is to "solve" the consequence restriction

Background A Hypothesis A Descriptions = classifications

For the unspecified theories, specified the Background awareness and instances defined by Classifications and Descriptions.

The equivalent explanations are as follows:

Father (Philip, Charles) Father (Philip, Anne) . . .

Mother (Mum, Margaret) Mother (Mum, Elizabeth) . . .

Married (Diana, Charles) Married (Elixabeth, Philip) . .

Male (Philip) Male (Charles) . . .

Female (Beatrice) Female (Margaret:) . . .

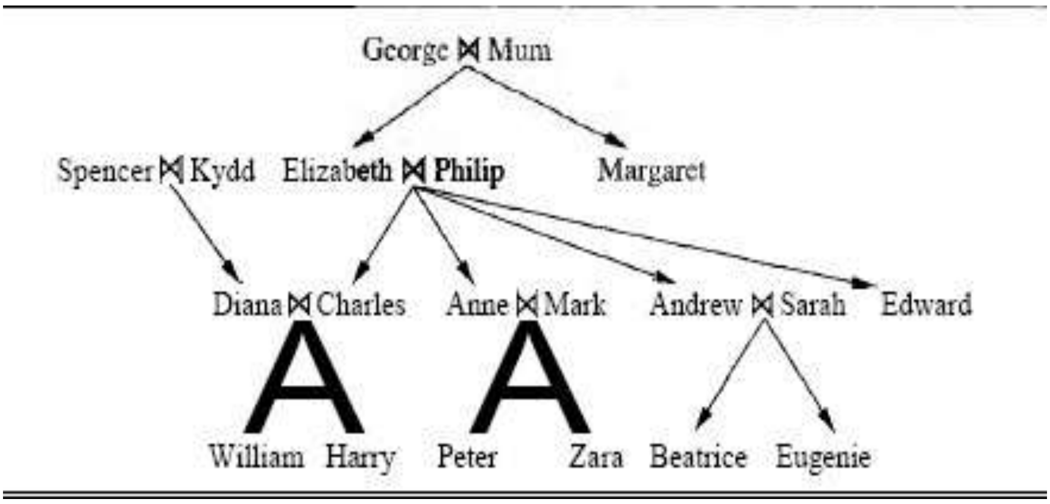


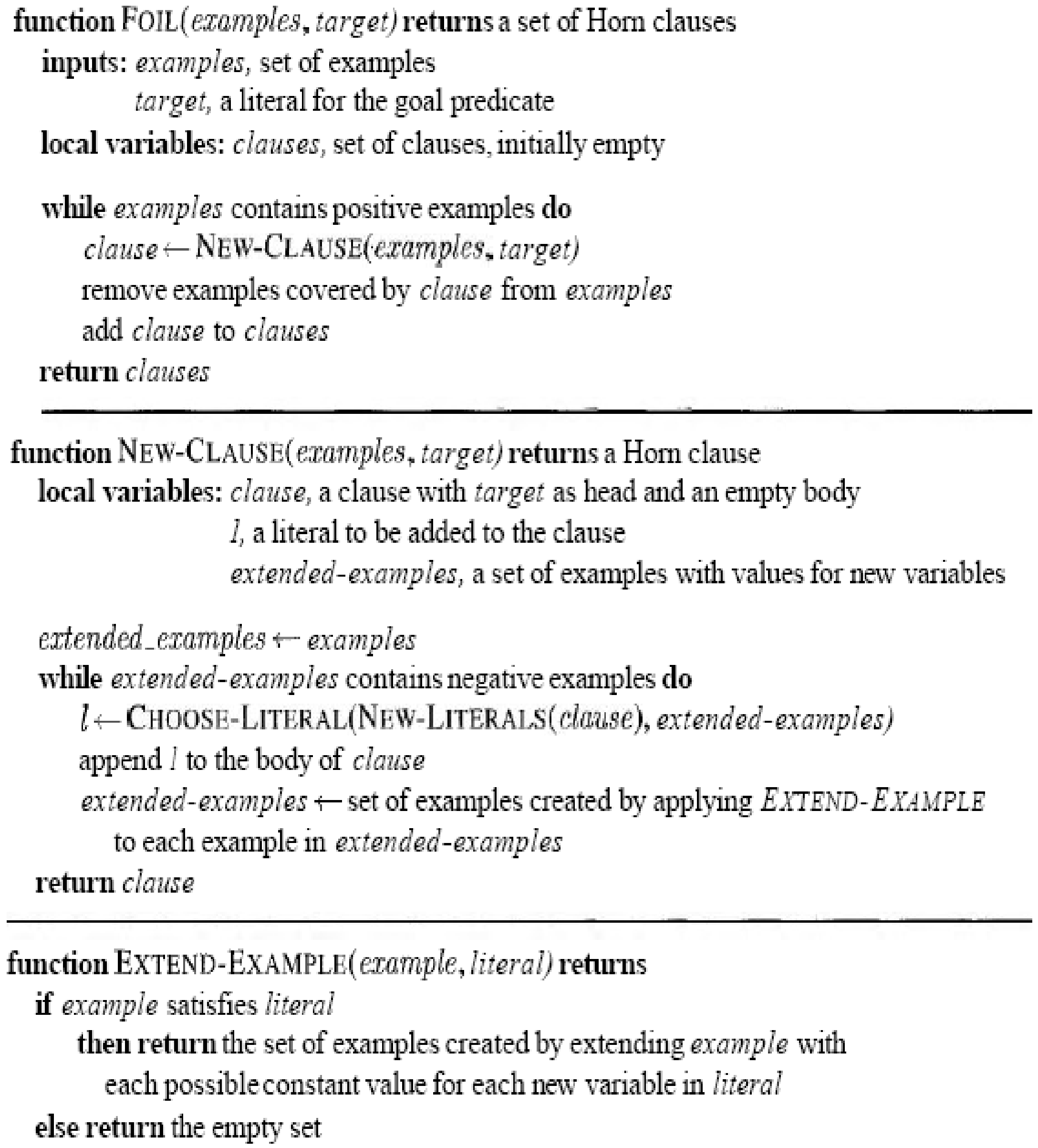
Figure: A Typical Family Tree

The goal of an Inductive Learning program is to occur with a group of statements for the theories such that the consequence limitation is fulfilled. Assume, for the instant, which the agent does not have background awareness: Background is unfilled.

Attribute Based Learning algorithms are unqualified of studying related predicates*.*

Top Down Inductive Learning Approaches

The initial method to ILP performs by beginning with a extremely common rule and regularly specialising it, so that it fix the information. It is basically what occurs in Decision Tree studying, wherever a Decision Tree is slowly developed upto it is reliable with the considerations. To perform ILP, we apply first-order literals rather than of attributes, and the theories is a group of sections rather than of a Decision Tree.



```
function FOIL(examples, target) returns a set of Horn clauses
    inputs: examples, set of examples
                target, a literal for the goal predicate
    local variables: clauses, set of clauses, initially empty

    while examples contains positive examples do
        clause ← NEW-CLAUSE(examples, target)
        remove examples covered by clause from examples
        add clause to clauses
    return clauses
```

```
function NEW-CLAUSE(examples, target) returns a Horn clause
    local variables: clause, a clause with target as head and an empty body
                     l, a literal to be added to the clause
                     extended-examples, a set of examples with values for new variables

    extended_examples ← examples
    while extended-examples contains negative examples do
        l ← CHOOSE-LITERAL(NEW-LITERALS(clause), extended-examples)
        append l to the body of clause
        extended-examples ← set of examples created by applying EXTEND-EXAMPLE
            to each example in extended-examples
    return clause
```

```
function EXTEND-EXAMPLE(example, literal) returns
    if example satisfies literal
        then return the set of examples created by extending example with
            each possible constant value for each new variable in literal
    else return the empty set
```

Figure: Drawing of the FOIL Algorithm for Learning Groups of First-order

Inductive Learning with Opposite Inference

The secondary major way to ILP involves reversing the common inferable evidence procedure.

Opposite declaration is based on the monitoring that if the instance Classification goes after from *Background* A *Hypothesis A Descriptions,* next one should be capable to confirm this reality by declaration. ***Twelve*** numbers of methods to educating the search have been attempted in applying ILP methods:

1. Too many selections should be removed
2. The proof strategy can be restricted.
3. The representation language can be restricted,
4. Inference may be completed with model inspection instead theorem confirming.
5. Inference may be completed with ground proposal sections instead in first-order logic.

Preparing Selections with Inductive-logic-programming

An inverse-resolution process, which reverses an entire resolution approach, it is a standard, an entire algorithm for studying first-order hypothesis.

4.11. Statistical Learning Methods

Bayesian learning easily computes the probability of all hypotheses, specified the information, and prepares forecasts on that basis. That is, the forecasts are prepared by applying *each* hypothesis, weighted by their possibilities instead by utilizing only a single "better" hypothesis.

The key measures in the Bayesian method are the hypothesis earlier P(hi*),* and the possibility of the information beneath all hypotheses, P *(* dJhi *).*

4.12. Learning with Entire Information

A statistical learning process starts with the easiest work: parameter learning with entire information. A parameter learning work contains discovering the arithmetic parameters for a possibility model, whose configuration is permanent.

Maximum Probability Parameter Learning: Discrete Models

Actually, while we have placed out a single standard process for most probability parameter learning:

1. List a declaration for the possibility of the information as a meaning of the parameter(s).
2. List the copied of the log possibility with regard to all parameter.
3. Discover the values of parameter such the copies are 0.

An important problem with most probability studying in common: -*when the information group is less sufficient that a few actions have not yet been watched for example, no cherry confections the most probability theory allots zero likelihood to those actions".*

The most significant point is that, *with entire records, the most probability parameter learning problem for a Bayesian system disintegrates into independent learning problems, single for every parameter.* The subsequent point is the parameter esteems for a variable, specified its parents are only the watched recurrences of the values of a variable for every configuration of the values of a parent. As in the past, we should be mindful to keep away from 0s when the records group is less.

Naive Bayes Models

Possibly the regular Bayesian system method utilized in AI is the **naïveBayes** method. In this method, the variable of a class-C is the root and the variables of an attribute-*Xi* are the ranges. The method is "naive7' on the grounds that it accepts that the attributes are restrictively free of one another, specified the class.

Most Possibly Parameter Learning Continuous Methods

Continuous Probability method acting as the **linear_Gaussian** method. The standards for most possibility learning are equal to the separate case. Let us start with an easy case: parameter learning of a Gaussian_Density function on one variable. That is the information is created as well as follows:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The parameter of the mean of this model is 'Y' and the standard-deviation is '*a'.*

The weight (yj-(B1 xj+02)) is the **mistake** for (zj,yj), which is, the dissimilarity among the real value "y_j" and the expected value (1xj+$2) - SOE is the popular **Sum of Squared Errors(SSE).** It is the weight, which is reduced by the regular **LinearRegression** method. At present we should know why reducing the SSE specifies the most probability continuous process, *given that the information is created with GaussianNoise of* per*manent variation.*

Bayesian Parameter Learning

The Bayesian method to learning parameter positions a theory earlier above the probable parameters values and improves this sharing as record enter. This expression of guess and learning generates this obvious that Bayesian-learning needs no more "standards of learning". In addition, *there is a real meaning, only a single algorithm for learning,* i.e., the algorithm of inference for BayesianNetworks.

Learning Bayes Networks

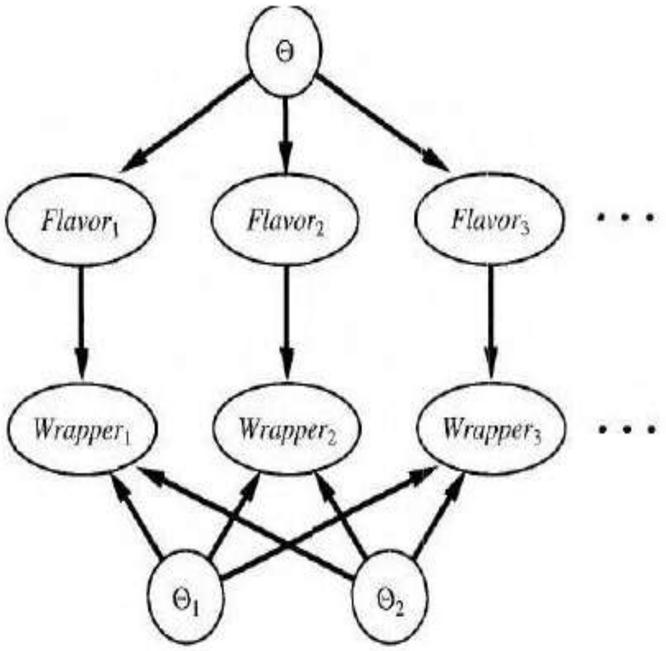


Figure: Bayesian network, which matches to a Bayesian learning procedure. Posterior distributions for the parameter variables e_0, e_1, and e_2 should be deduced from their earlier distributions and the proof in the *Flavor*, and the *Wrapper variables*.

There are 2 different approaches for determining whenever best designs have been selected.

The primary is to check regardless if the provisional freedom statements implied in the design are really fulfilled in the record.

4.13. Learning with Hidden Variable

Several genuine problems contain unseen variables (sometimes termed as hidden variables) that are unrecognizable in the record that are accessible for learning.

Ex: health data frequently contain the noticed symptoms, the treatment apply, and possibly the result of the treatment, however they rarely have a straight examination of the illness itself.

Thus, *latent variables should noticeably decrease the number of parameters necessary to state a Bayesian network.*

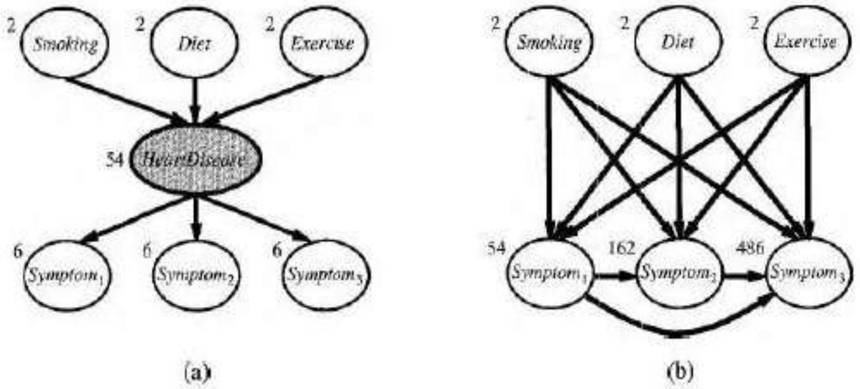


Figure: (a) an easy investigative network for heart illness, which is supposed to be a unseen variable. Every variable has 3 probable values and is tagged with the number of independent-parameters in its conditional sharing; the entire number is 78. (b) The corresponding network with *Heart illness* detached. Note that the sign variables are no long-lasting conditionally autonomous specified their parents. This network needs "708" parameters.

Unsupervised Clustering: Learning Combinations of Gaussians

Unsupervised clustering is the difficulty of discriminating several kinds in a group of elements. The difficulty is unsupervised for the reason that the group tags are unspecified.

Clustering assumes that the records are created from a **mixture sharing** 'P'. Such sharing contains '*k'* **elements,** all of which is a sharing of itself.

A data position is created by initially selecting a component and then creating an easy from that element. C is a random-variable indicate the element, with the value 1*K;* afterward the mixture sharing is specified:

$$P(\mathbf{x}) = \sum_{i=1}^{k} P(C=i)\ P(\mathbf{x}|C=i)\ ,$$

where 'x' mentions, the attributes value for a data position.

For the Gaussians mixture, we initialise the mixture approach parameters randomly and after that repeat the bellow 2 steps:

A. **E-step**: calculates the probabilities *pij=P(C=*i*lxj),* the probability that datum *"xj"* are created by an element 'i'. By Bayes' rule, we contain *P (C=**I**) and pij=aP(xj*IC=*i).* The expression *P (C*=I) is only the weight parameter for the 'ith' Gaussian and the expression *P(xj*IC=i) is only the probability of the 'ith' Gaussian at *"xj"*. Describe *pi=Cjpij.*

B. **M-step**: calculate the latest covariance, mean and element as follows:

$$\mu_i \leftarrow \sum_j p_{ij}\mathbf{x}_j / p_i$$

$$xi + Cpii\,(xj - pi)\,(xi - Pi)\,/\,pi\,Wi + pi\,.$$

The E-step, should be observing as calculating the values predictable is *'p'* of the unseen **pointer-variables** *'Z'*, where *'Z'* is one if daturn-**x3** are created by the 'ith' element and zero or else.

Learning Bayesian Networks with Hidden Variables

The message from this instance is that *the constraint upgrades for Bayesian Network learning with variables of unseen are openly accessible from the outcomes of deduction on every instance. In addition simply* local *posterior-probabilities are required for every constraint.*

Learning of Hidden Markov Models (HMMs)

Our last program of EM includes learning the changeover possibilities in HMMs. This model should be presented by a Dynamic Bayes Network with one distinct situation variable. Every data position includes of a study series of limited length, so the problem is to study the changeover possibilities from a group of study series.

The Common Structure of the EM Algorithm

We noticed multiple cases of the EM algorithm. All includes calculating predictable values of the variables of variables for every case and then calculating the constraints, utilizing the values of predictable as if they are the values of experiential. Let '**x**' be the entire values of experiential in each of the case let '**Z**' indicate each of the unseen variable for each of the case and let **eight** be the entire elements for the probability method. The EM algorithm is:

$$\boldsymbol{\theta}^{(i+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\mathbf{z}} P(\mathbf{Z}=\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(i)}) L(\mathbf{x}, \mathbf{Z}=\mathbf{z}|\boldsymbol{\theta})$$

This expression is the EM algorithm in a shell of nut.

Learning Bayes Network Structures with Hidden Variables

In the easiest instance, the unseen variables are indexed besides the experimental variables; while their values aren't experimental, the learning algorithm is informed that they be present and should discover a position for them in the grid formation. The final approach should be applied by including latest change selections in the structural search: additionally to changing connections, the algorithm should insert or remove an unseen variable or alter its parity.

One probable development is the so-referred to as **structure** EM algorithm that functions in greatly the similar method as regular parametric EM excluding. The algorithm should upgrade the design in addition to the considerations.

4.14. Instance based Learning

Parametric Learning systems are frequently effective and easy, but suppose a selected constrained family of methods frequently restraint what is occurrence inside the actual world, from where the records appear. In comparison to nonparametric and parametric learning process permit the hypothesis difficulty to grow with the records.

Two extremely easy relations of nonparametric **memory-based learning** or **example-based learning** process, so known as for the reason that they make hypotheses honestly from the preparation examples itself.

Nearest Neighbor Models (NNM)

The major concept of **NNM** is that the property of every specific entry point 'x' is probable to be equal to these of objects inside the neighborhood of x.

The k-NNM learning algorithm is quite easy to apply, needs slight in the method of altering, and frequently executes very fine. This is a great factor to attempt initially on a latest learning problem. For huge record groups, still, we need an competent method for discovering the nearby neighbors of a question factor 'x' easily computing the gap to each factor might get lengthy. A change of inventive strategies had been planned to build this step competent by pre processing the preparation records. Unfortunately, maximum of those strategies do longer size properly with the measurement of the space (i.e., the characteristics count).

Kernel Models

In a kernel model, we saw all preparation examples as creating a small weight function of its own. The weight guesses as an entire is simply the distributed addition of the whole small kernel functions. A preparation example at 'xi' can create a kernel_function K(x,xi) that allocates a possibility to every factor-x inside the space. Therefore, the weight

guesses is one. P (x)=*NC*~(xxi,). The kernel function commonly depends just on the *distance-**D(x,xi)*** from ***'x'*** to the example ***'xi'.*** The great trendy kernel function is the Gaussian. For minimalism, we will anticipate spherical Gaussians with general difference 'w' next to all axes, i.e.

$$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{(w^2\sqrt{2\pi})^d} e^{-\frac{D(\mathbf{x},\mathbf{x}_i)^2}{2w^2}} ,$$

The dimensions count in 'x' is 'd'.

Supervised learning with kernels is completed by picking a *weighted* grouping of *each of* the forecasts from the preparation examples.

4.15. Neural Networks

A neuron is one of the cells in the brain whose major task is the dissemination, procedure, and collection of electrical signals. The brain's data working capability is an idea to appear initially from *networks* of like neurons. For this cause, a few of the initial '**A1'** process tried to generate Artificial **Neural Networks.** (Additional names for the area contain **neural computation, parallel distributed processing,** and **connectionism.)**

Elements in Neural Networks

Neural networks are collected of nodes or **elements** linked by directed **links. A** link from element 'j' to element 'i' provide to spread the **activation-**a_j from '**j**' to '*i'.* Every connection just had a numerical **weight-**W_j linked with this that decides sign and power of WEIGHT the link. Every element 'i' initially calculates a weighted addition of its entries: '*N*' Next, it gives an **activation function-**g to this addition to get the result: Note that we can incorporated a **bias-weight-***Wo, 'i'* linked to a permanent entry a_o=*-1.*

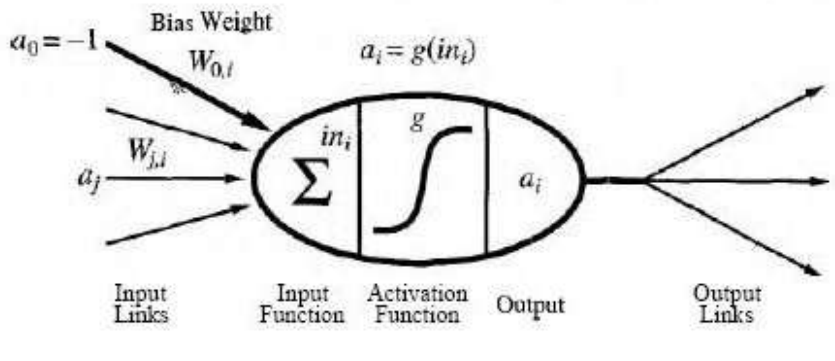


Figure: An Easy Mathematical Approach for a Neuron

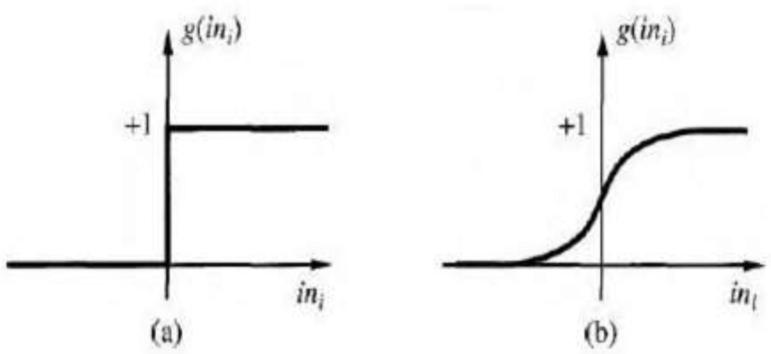


Figure 4.13 (a) The threshold activation-function, which yields '1' when the input is positive value and zero if not. (b) The sigmoid-function 1 /(1+*e* -").

Figure demonstrates how the Boolean tasks NOT, AND and OR should be signified by threshold elements with appropriate weights. It is essential for the reason that it denotes we should utilize those elements to generate a network to calculate any Boolean task of the entries.

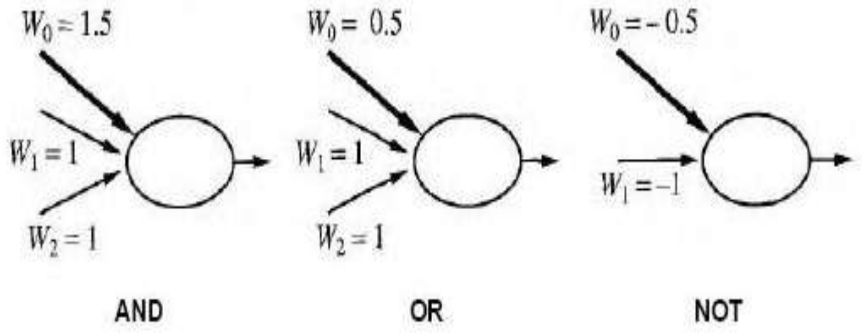


Figure: Elements with a threshold activation-function should work as logic-gates, specified proper input and bias-weights.

Network Structures

There are 2 major types of Neural Network structures: acyclic or feed forward networks and cyclic or recurrent networks. An acyclic system signifies a task of its present input; therefore, it had no internal status excluding the weights itself. A repeated network, any further way, feeds its results back into one its own inputs.

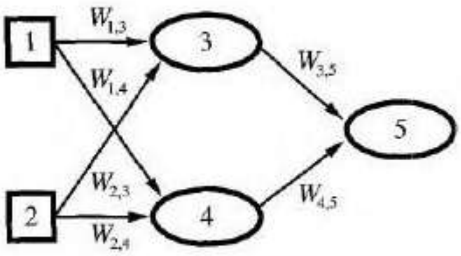


Figure: An easy neural-network with 2 inputs, one unseen layer of two elements, and one output.

A neural-network should be utilized for regression or classification.

Feed-forward structures are regularly ordered in **layers,** so every element obtains input just from elements in the instantly earlier layer.

Single Layer Feed Forward Neural Networks (Perceptrons)

A network with each input linked directly to the outputs is known as a **Single Layer Neural Network,** or a **perceptron** network. While every output element is autonomous of the furthers, every weight influences just 1 of the outputs.

In common, *threshold perceptrons should signify just linearly divisible functions.*

In spite of their partial communicative power, threshold perceptrons had a few benefits.

In specific, *there is an easy learning algorithm, which can specify a threshold perceptron to a few limitedly divisible preparation groups.*

The entire algorithm is exposed in Figure

```
function PERCEPTRON-LEARNING(examples, network) returns a perceptron hypothesis
  inputs: examples, a set of examples, each with input x = $x_1, \ldots, x_n$ and output $y$
          network, a perceptron with weights $W_j$, $j = 0 \ldots n$, and activation function $g$

  repeat
     for each $e$ in examples do
         $in \leftarrow \sum_{j=0}^{n} W_j \, x_j[e]$
         $Err \leftarrow y[e] - g(in)$
         $W_j \leftarrow W_j + \alpha \ x \ Err \times g'(in) \times x_j[e]$
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(network)
```

Figure: The grade descending learning algorithm for perceptrons, guessing a diverse activation function 'g'.

Observe that *the weight upgrade vector for highest possibility studying in sigmoid-perceptrons is basically the same to the upgrade vector for squared error reduction.*

Multilayer Feed-forward Neural Networks

The benefit of inserting unseen layers is that it increases the space of hypotheses that the network should signify.

Learning-algorithms for multi-layer systems are equal to the perceptron-learning algorithm. The back transmission procedure should be outlined as given:

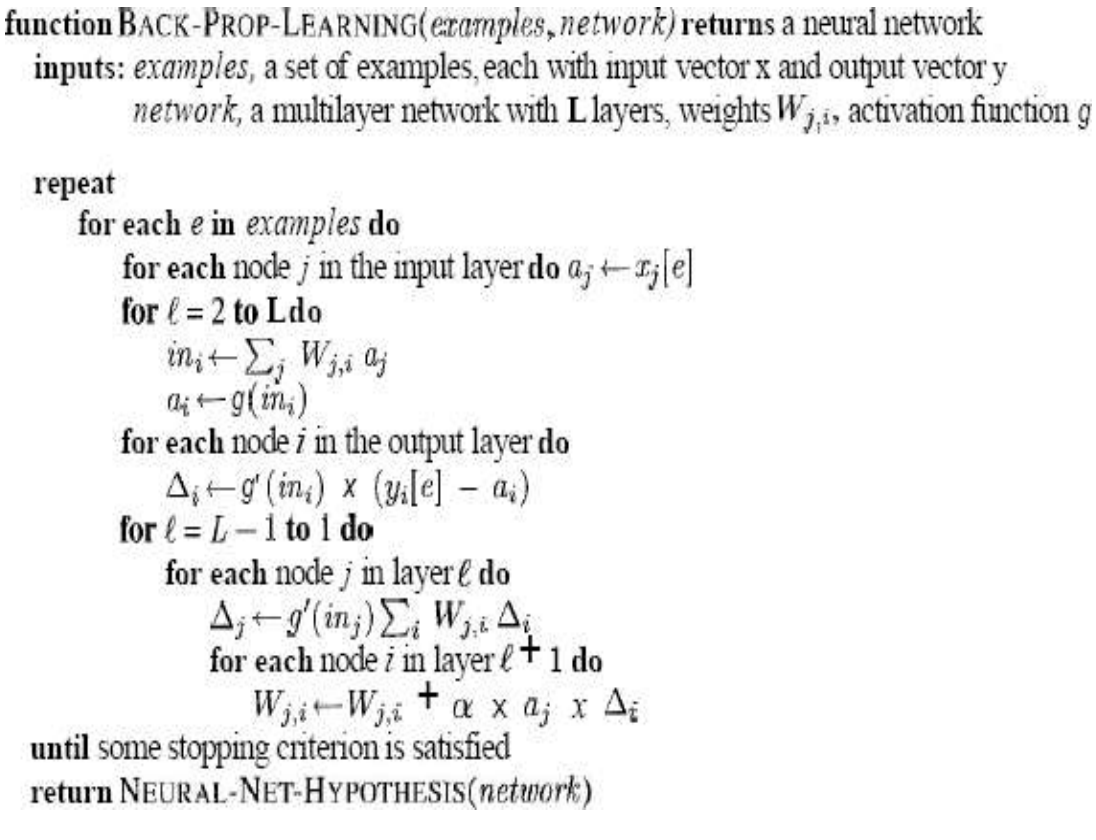
Calculate the 'A' values for the outcome elements by utilizing the experimental error.

Beginning with outcome layer, replicate the subsequent for every layers in the network, till the initially unseen layer is attained:

a) Transmit the 'A' values backwards to the earlier layer.

b) Upgrade the weights among the 2 layers.

The complete algorithm is exposed in Figure



```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights $W_{j,i}$, activation function g

  repeat
     for each e in examples do
        for each node j in the input layer do $a_j \leftarrow x_j[e]$
        for $\ell$ = 2 to L do
           $in_i \leftarrow \sum_j W_{j,i} a_j$
           $a_i \leftarrow g(in_i)$
        for each node i in the output layer do
           $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
        for $\ell = L - 1$ to 1 do
           for each node j in layer $\ell$ do
              $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$
              for each node i in layer $\ell + 1$ do
                 $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(network)
```

Figure: The Back Transmission Algorithm for Learning in Multi-layer Networks

Learning Neural Network Structures

We need to observe networks that are not entirely linked, and then we require locating a few valuable search methods through the huge space of probable link topologies. The best Brain Damage algorithm starts with a entirely linked network and eliminates links from it. Once the network is skilled for the primary time, a data-theoretic method recognizes a best choice of links that should be removed. The network is re-trained, and if its presentation had not minimized next the procedure is do again. Additionally to eliminating links, it is just likely to eliminate elements that are not giving more to the outcome.

4.16. Reinforcement Learning

The problem is *without a few feedbacks regarding what is fine and what is poor; the agent shall contain no grounds for selecting which go to build.* The agent requires knowing that a little fine has occurred when it succeeds and that a little poor has occurred when it drops. This type of comment is known as a **reward** or **reinforcement.**

The job of **Reinforcement Learning** is to apply experimental rewards to study a best strategy for the environment.

Three of the agent designs:

a. **Q-learning** agent studies a Q-function or an **action-value** function, specifying the estimated value of obtaining a specified operation in a specified condition.
b. **Utility-based agent** studies a utility task on positions and applies it to choose operations that increase the estimated result utility.
c. **Reflex agent** studies a strategy, which maps directly from status to operations.

Kinds of reinforcements learning.

1. Active reinforcement learning
2. Passive reinforcement learning

4.17. Passive Reinforcement Learning

To keep objects easy, we begin with the instance of a PassiveLearning agent by utilizing a status based presentation in a completely evident situation. In PassiveLearning, the agent's strategy-T is permanent: in status-s, it forever performs the operation (s), its target is just to study how the best strategy is to study the Utility_Function UT(s)?

The major dissimilarity is that the PassiveLearning agent is unrecognized the **transition_Model** T(s,a,s'), that gives the likelihood of success states' from states later than performing action-a or does not recognize the **reward_Function** R(s) that gives the reward for every status.

The utility is described to be the predictable addition of (not counted) rewards gained if the strategy is followed. This expression is shown as:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) \mid \pi, s_0 = s\right]$$

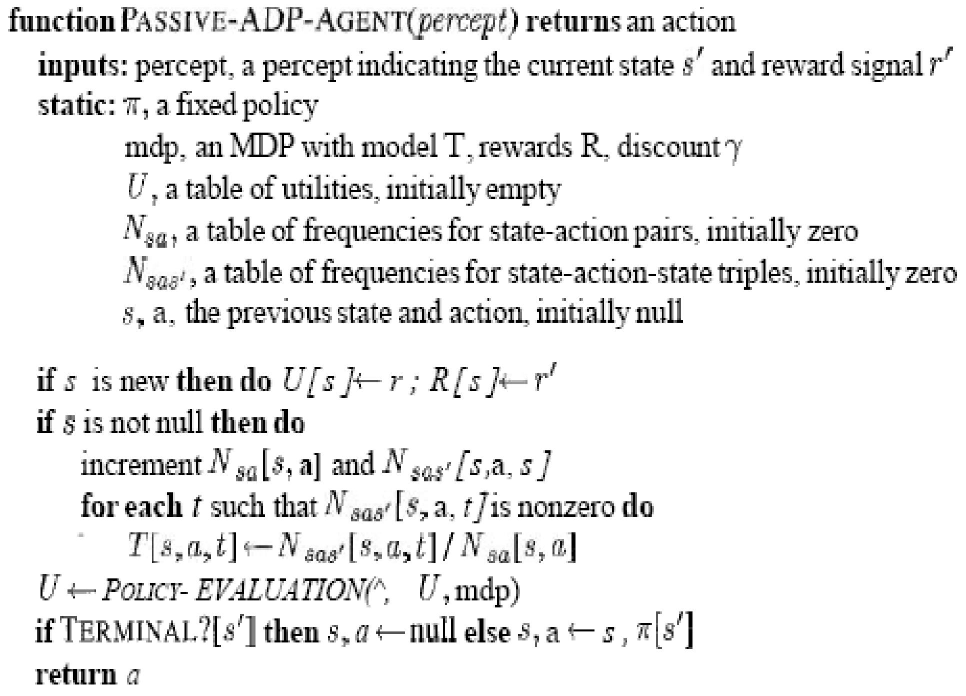
Direct Utility Estimation (DUE)

A simplest process for **DUE** was innovated in the late 1950s in the field of **Adaptive Control Theory.** The thought is that the utility of a THEORY condition is the predictable entire reward from that condition forward, and every test gives a *model* of this value for every condition is looked up.

Adaptive Dynamic Programming (AIDP)

An **AIDP** agent performs by studying the Transition method of the situation as it moves next to and resolving the matching Markov Decision procedure by applying a dynamic programming system.

The complete agent program for a passive-ADP agent is represented in the Figure:



```
function PASSIVE-ADP-AGENT(percept) returns an action
    inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
    static: $\pi$, a fixed policy
            mdp, an MDP with model T, rewards R, discount $\gamma$
            $U$, a table of utilities, initially empty
            $N_{sa}$, a table of frequencies for state-action pairs, initially zero
            $N_{sas'}$, a table of frequencies for state-action-state triples, initially zero
            $s$, a, the previous state and action, initially null

    if $s$ is new then do $U[s] \leftarrow r$ ; $R[s] \leftarrow r'$
    if $s$ is not null then do
        increment $N_{sa}[s, a]$ and $N_{sas'}[s, a, s]$
        for each $t$ such that $N_{sas'}[s, a, t]$ is nonzero do
            $T[s,a,t] \leftarrow N_{sas'}[s,a,t] / N_{sa}[s,a]$
    $U \leftarrow$ POLICY-EVALUATION(^, $U$, mdp)
    if TERMINAL?[$s'$] then $s, a \leftarrow$ null else $s$, a $\leftarrow s$ , $\pi[s']$
    return $a$
```

Figure: A Passive-reinforcement-learning agent-based on ADP

```
function PASSIVE-TD-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  static: n-, a fixed policy
          $U$, a table of utilities, initially empty
          $N_s$, a table of frequencies for states, initially zero
          $s$, a, $r$, the previous state, action, and reward, initially null

  if $s'$ is new then $U[s'] \leftarrow r'$
  if $s$ is not null then do
     increment $N_s[s]$
     $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if TERMINAL?[$s'$] then $s, a, r \leftarrow$ null else $s$, a, $r \leftarrow s, \pi[s'], r'$
  return $a$
```

Figure: A passive-reinforcement-learning agent that learns utility guesses by applying temporal difference.

Temporal Difference Learning (TDL)

It is probable to contain the best of both worlds; which is, one should estimated the restriction expressions displayed previous with not resolving them for each probable conditions. *The major thing is to apply the experimental moves to change the experimental conditions values so that they accept with the restriction expressions.*

4.18. Active Reinforcement Learning

A passive-learning agent had permanent strategies that determine its performance. An active-agent should choose what actions to obtain.

Exploration

EXPLORATION maximizes its reward as mirrored in its present utility approximates and **exploration** to increase its lifelong success. Pure utilization threats receiving fixed in a track. Pure exploration to upgrade one's data is not usable if one not at all sets that information into procedure.

The following expression performs this:

$$U^{+}(s) \leftarrow R(s) + \gamma \max_{a} f\left(\sum_{s'} T(s,a,s')U^{+}(s'),\ N(a,s)\right)$$

at this point, f(u,n) is known as the **exploration function.**

Learning an Action-Value Function

Q-learning at learns an action-value presentation in place of learning utilities. A key property: *A TD agent, which learns a Q-function, performs without requirement a method* ***for*** *either action selection or learning.*

```
function Q-LEARNING-AGENT(percept) returns an action
    inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
    static: $Q$, a table of action values index by state and action
            $N_{sa}$, a table of frequencies for state-action pairs
            $s, a, r$, the previous state, action, and reward, initially null

    if $s$ is not null then do
        increment $N_{sa}[s, a]$
        $Q[a,s] \leftarrow Q[a,s] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[a',s'] - Q[a,s])$
    if TERMINAL?[$s'$] then $s, a, r \leftarrow$ null
    else $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[a',s'], N_{sa}[s',a']), r'$
    return $a$
```

```
function Q-LEARNING-AGENT(percept) returns an action
    inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
    static: $Q$, a table of action values index by state and action
            $N_{sa}$, a table of frequencies for state-action pairs
            $s, a, r$, the previous state, action, and reward, initially null

    if $s$ is not null then do
        increment $N_{sa}[s, a]$
        $Q[a,s] \leftarrow Q[a,s] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[a',s'] - Q[a,s])$
    if TERMINAL?[$s'$] then $s, a, r \leftarrow$ null
    else $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[a',s'], N_{sa}[s',a']), r'$
    return $a$
```

Figure: An Exploratory Q-learning Agent

4.19. Generalization in Reinforcement Learning

Function estimation creates it realistic to present utility tasks for huge situation spaces, but it is not the primary advantage. *The compression attained by a task estimated permits the learning-agent to situates it is not visited and to simplification it is visited.*

Question Bank

Unit - IV

Part - A

1. Differentiate supervised learning with unsupervised learning?
2. What is meant by reinforcement learning?
3. Define decision tree with an example.
4. Define regression.
5. Define over fitting.
6. What is meant by cross validation?
7. List the general uses of decision tree learning system?
8. what is meant by boosting?
9. Define the terms false negative and false positive for the hypothesis.
10. What is meant by memorization?
11. What is meant by functional dependencies or determinations?
12. Define Baye‘s rule.
13. What is meant by Artificial Neural Network(ANN)?
14. What is need of activation functions in neural network?
15. Differentiate real neuron with simulated neuron.
16. Differentiate feed forward with recurrent network.
17. What is meant by learning rate?
18. Differentiate passive learning with active learning.
19. Write short notes on Q-functions.
20. Write short notes on perception.
21. Differentiate nearest-neighbor method with Kernal method.
22. What is meant by cumulative learning?
23. List some of the applications of learning.
24. Define learning.
25. Explain the different kinds of learning methods.

Part - B

1. How the decision-tree-learning is done? Explain with an example and algorithm.
2. Explain the ensemble learning with boosting algorithm.
3. Explain the version space/candidate elimination algorithm with an example.
4. How to extract general rules from an example? Explain.
5. How learning with complete data is achieved?
6. Explain in detail EM algorithm.
7. Explain the back propagation-algorithm of learning in a multi-layer neural-network.
8. Explain passive reinforcement learning agent with

   (i) Adaptive Dynamic Programming (ADP) (ii) Temporal Difference (TD)