ARTIFICIAL INTELLIGENCE

UNIT-V

EXPERT SYSTEMS

Expert systems - Architecture of expert systems, Roles of expert systems - Knowledge Acquisition – Meta knowledge, Heuristics. Typical expert systems - MYCIN, DART, XOON, Expert systems shells.

Learning from Observation:

- The idea behind learning is that percepts should be used not only for acting, but also for improving the agent's ability to act in the future.
- Learning takes place as the agent observes its interactions with the world and its own decision making process.
- Learning can range from trivial memorization of experience to the creation of a entire scientific theory, as exhibited like Albert Einstein.

Forms of Learning:

- Learning agent is a performance element that decides what actions to take and a learning element that modifies the performance element so that better decisions can be taken in the future.
- There are large variety of learning elements
- The design of a learning element is affected by following three major issues,
    - Which components of performance element are to be learned.
    - What feedback is available to make these components learn
    - What representation is used for the component.
- The components of these agents includes the following,
    - A direct mapping from conditions on current state to actions
    - A means to infer relevant properties of the world from the percept sequence
    - Information about the way the world evolves and about the results of possible action the agent can take
    - Utility information indicating the desirability of world states
    - Action-value information indicating the desirability of action
    - Goals that describe classes of states whose achievement maximizes the agent utilty
- Each of the component can be learned from appropriate feedback
    - For Example: - An agent is training to become a taxi driver.
    - The various components in the learning are as follows,

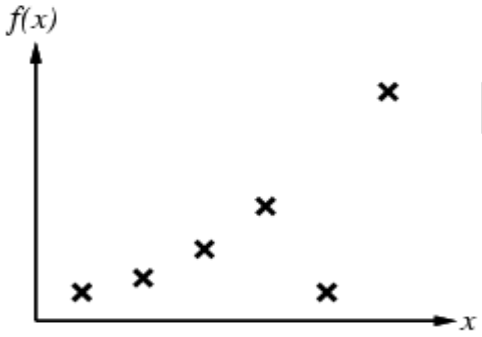
  - Everytime when the instructor shouts "Brake" the agent learn a condition – action rule for when to brake.
  - By seeing many images, agent can learn to recognize them
  - By trying actions and observing the results, agent can learn the effect of actions (i.e.) braking on a wet road – agent can experience sliding
  - The utility information can be learnt from desirability of world states, (i.e.) if the vehicle is thoroughly shaken during a trip, then customer will not give tip to the agent, which plans to become a taxi driver
- The type of feedback available for learning is also important.
- The learning can be classified into following three types.
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning
- **Supervised Learning:-**
  - It is a learning pattern, in which
    - Correct answers for each example or instance is available
    - Learning is done from known sample input and output
    - For example: - The agent (taxi driver) learns condition – action rule for braking – this is a function from states to a Boolean output (to brake or not to brake). Here the learning is aided by teacher who provides correct output value for the examples.
- **Unsupervised Learning:-**
  - It is learning pattern, in which
    - Correct answers are not given for the input.
    - It is mainly used in probabilistic learning system.
- **Reinforcement Learning:-**
  - Here learning pattern is rather than being told by a teacher.
  - It learns from reinforcement (i.e.) by occasional rewards
  - For example:- The agent (taxi driver), if he does not get a trip at end of journey, it gives him a indication that his behavior is undesirable.

Inductive Learning

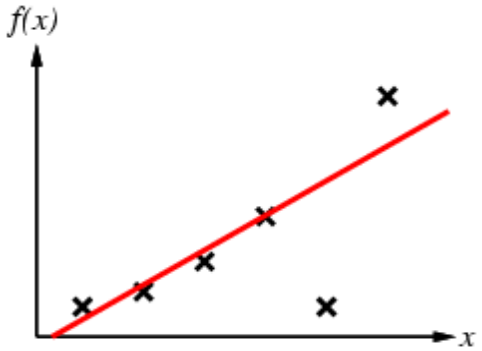
- Learn a function from example,
- For example:- *f* is target function
  An example is a pair (x, *f*(x)) where x = input and *f*(x) = output of the function is applied to x
- The pure inductive inference or induction is "given a training set of example of *f*, return a function *h* that approximates *f*.
- Where the function *h* is called hypothesis
- This is a simplified model of real learning, because it
  - Ignores prior knowledge
  - Assumes a deterministic, observable "environment".
- A good hypothesis will generalize well, i.e., able to predict based on unseen examples

Inductive learning method:-

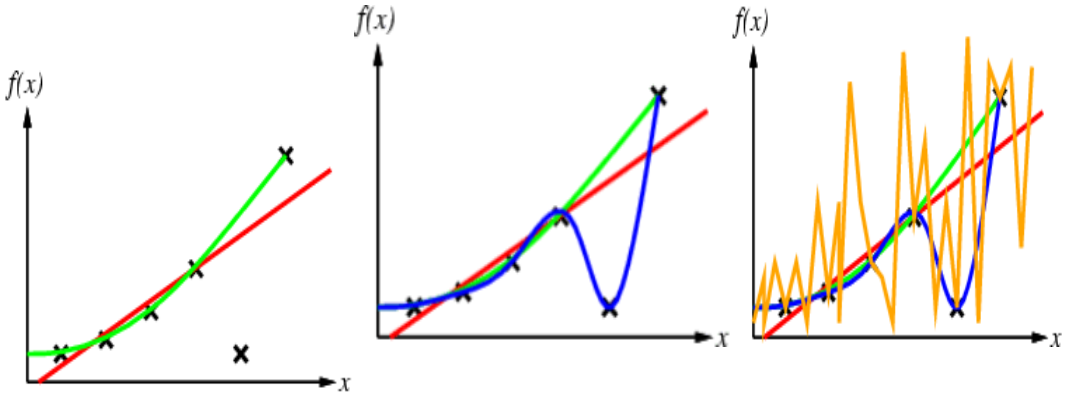
-
- Goal is to estimate real underlying functional relationship from example observations
- Construct / adjust *h* to agree with *f* on training set (*h* is consistent if it agrees with *f* on all example)
- For example:- Curve fitting example
- Given



- Linear hypothesis:



- Curve fitting with various polynomial hypothesis for the same data



- Ockham's razor : prefer simplest hypothesis consistent with the data

- Not-exactly-consistent may be preferable over exactly consistent
    - Nondeterministic behavior
    - Consistency even not always possible
- Nondeterministic functions : trade-off complexity of hypothesis / degree of fit

Decision Trees

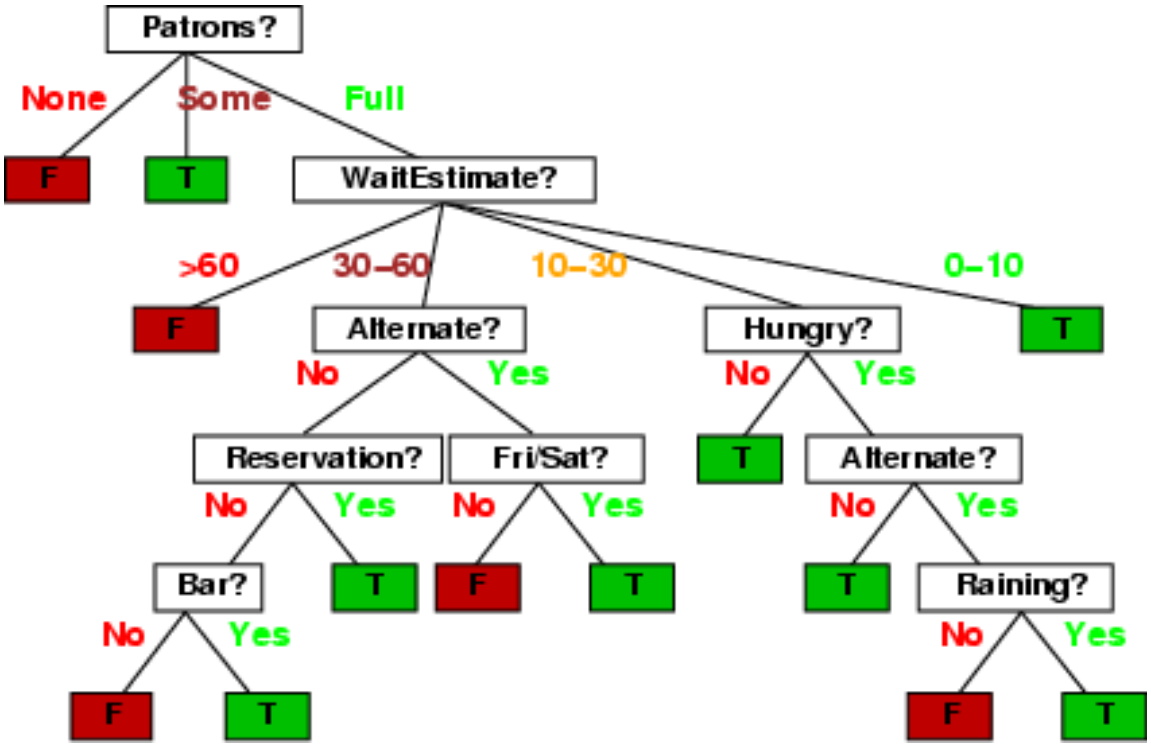
- Decision tree is one of the simplest learning algorithms.
- A decision tree is a graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.
- It can be used to create a plan to reach a goal.
- Decision trees are constructed to help with making decisions.
- It is a predictive model.

Decision trees as performance elements:-

- Each interior node corresponds to a variable; an arc to a child represents a possible value of that variable.
- A leaf represents a possible value of target variable given the values of the variables represented by the path from the root.
- The decision tree takes object or situation described by set of **attributes** as input and decides or predicts output value.
- The output value can be Boolean, discrete or continuous.
- Learning a discrete valued function is called **classification learning.**
- Learning a continuous valued function is called **regression.**
- In Boolean classification it is classified as true (positive) or false (negative).
- A decision tree reaches its destination by performing a sequence of tests.
- Each interior or internal node corresponds to a test of the variable; an arc to a child represents possible values of that test variable.
- The decision tree seems to be very for humans.
- For Example:-
    - A decision tree for deciding whether to wait for a table at a restaurant.
    - The aim here is to learn a definition for the **goal predicate.**
    - we will see how to automate the task the following attributes are decided.
        - Alternate: is there an alternative restaurant nearby?
        - Bar: is there a comfortable bar area to wait in?
        - Fri/Sat : is today Friday or Saturday?
        - Hungry: are we hungry?
        - Patrons : number of people in the restaurant [the values are None, Some, Full]
        - Price : price range [$, $$, $$$]
        - Raining: is it raining outside?
        - Reservation: have we made a reservation?
        - Type : kind of restaurant [French, Italian, Thai, Burger]
        - WaitEstimate : estimated waiting time by the host [0-10, 10-30, 30-60, >60]
        - The following table described the example by attribute values (Boolean, Discrete, Continuous) situations where I will / won't wait for a table.

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- The following diagram shows the decision tree for deciding whether to wait for a table



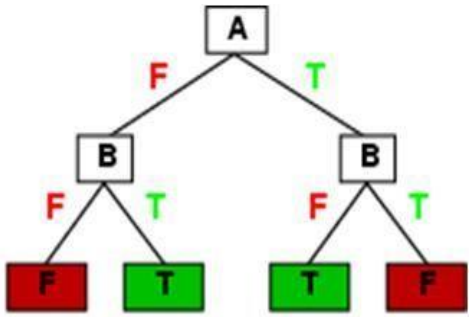
- The above decision tree does not use **price** and **type** as irrelevant.
- For example:- if the Patrons = full and the Wait Estimate = 0-10 minutes, it will be classified as positive(yes) and the person will wait for the table
- Classification of example is positive (T) or negative (F) shown in both table and in decision tree.

Expressiveness of decision trees

- Decision trees can express any function of the input attributes
- E.g., for Boolean functions, truth table row path to leaf
- The following table shows the truth table of A XOR B

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F

- The following diagram shows the decision tree of XOR gate



- There is a consistent decision tree for any training set with one path to leaf for each example [unless *f* nondeterministic in *x*] but it probably won't generalize to new examples
- Applying Ockham's razor : smallest tree consistent with examples
- Able to generalize to unseen examples
    - No need to program everything out / specify everything in detail

'true' tree = smallest tree?

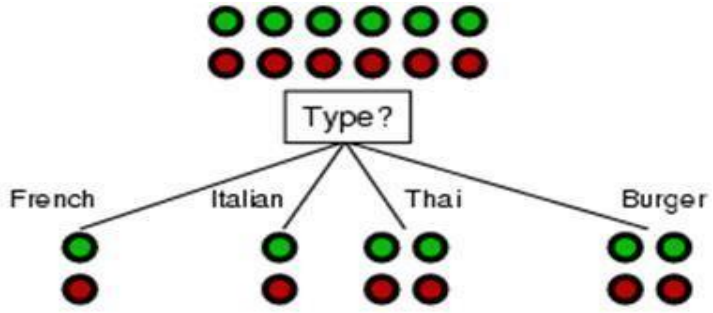
Advantages of Decision Tree:

- They are simple to understand and interpret
- They require little data preparation
- If uses a white box model.
- It is possible to validate a model using statistical tests, hence robust.
- Perform well with large data in a short time.

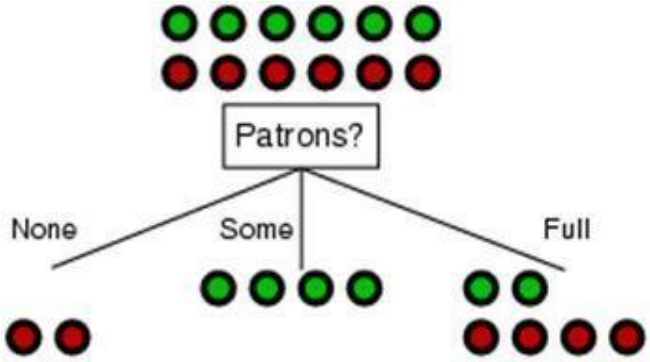
Decision tree learning:

- Unfortunately, finding the 'smallest' tree is intractable in general
- New aim : find a 'smallish' tree consistent with the training examples
- Idea : [recursively] choose 'most significant' attribute as root of [sub]tree
- 'Most significant' : making the most difference to the classification
- Idea : a good attribute splits the examples into subsets that are [ideally] 'all positive' or 'all negative'
- Patrons? is a better choice

- The following diagram shows the splitting the examples by testing on attributes



- The above diagram Splitting on Type brings us no nearer to distinguishing between positive and negative examples
- The below diagram Splitting on Patrons does a good job of separating positive and negative examples



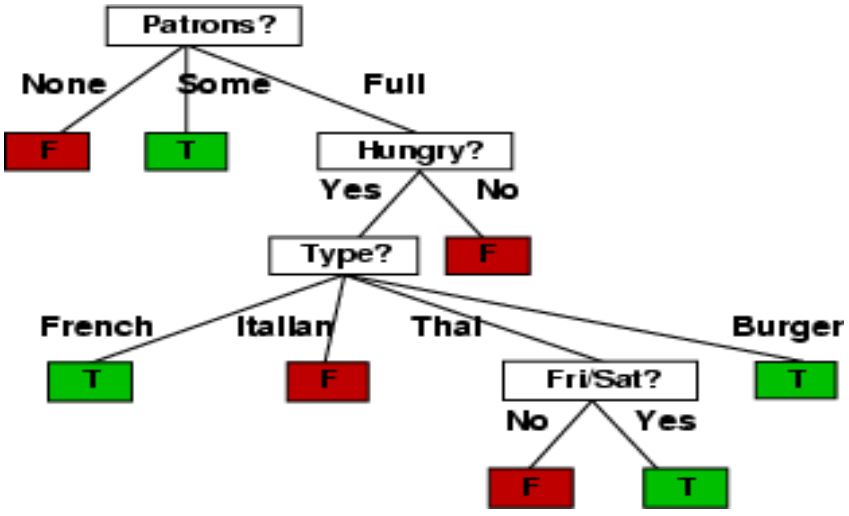
- The following table shows the Decision Tree Learning Algorithm,

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- The following tree shows the decision tree induced from the training data set as follows,

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- substantially simpler solution than 'true' tree
- More complex hypothesis isn't justified by small amount of data



Using Information theory:

- Information content [entropy] :
  - $I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log 2\ P(v_i)$
  - For a training set containing *p* positive examples and *n* negative examples

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

- Specifies the minimum number of bits of information needed to encode the classification of an arbitrary member

Information Gain:

- Chosen attribute *A* divides training set *E* into subsets *E*1, … , *Ev* according to their values for *A*, where *A* has *v* distinct values

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

- Information gain [IG] : expected reduction in entropy caused by partitioning the examples

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

- Information gain [IG] : expected reduction in entropy caused by partitioning the examples

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

- Choose the attribute with the largest IG
- For Example:- For the training set : $p = n = 6$, I(6/12, 6/12) = 1 bit
- Consider Patrons? and Type? [and others]

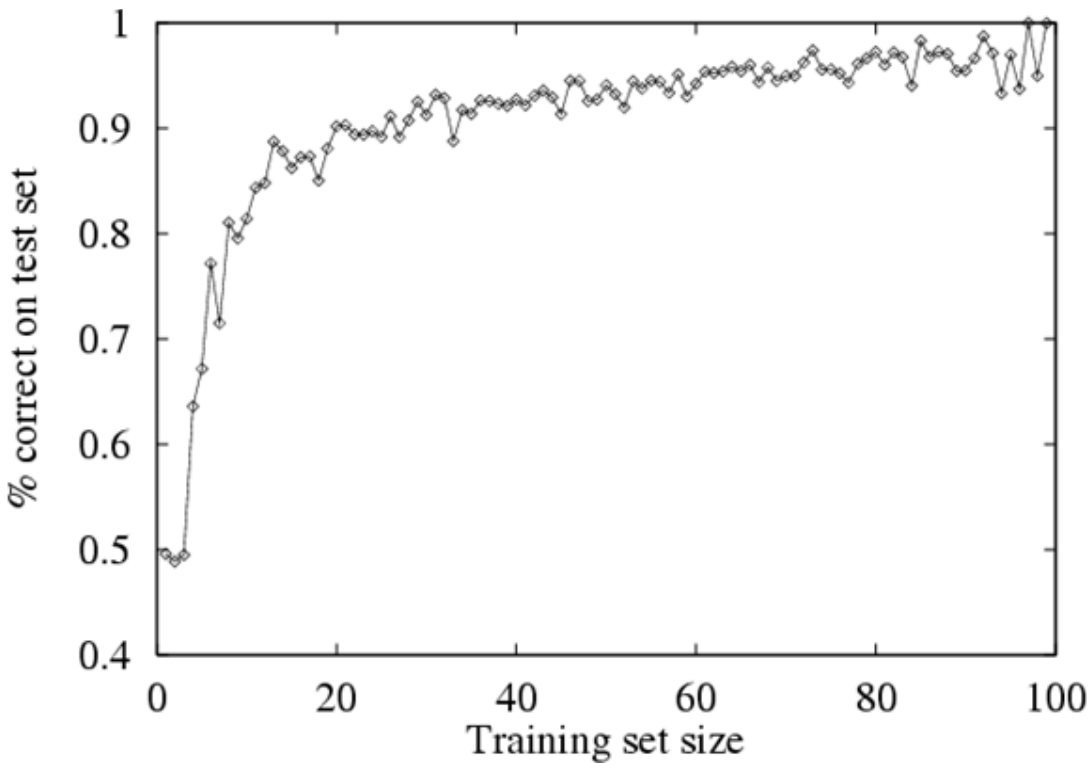
$$IG(Patrons) = 1 - \left[\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right)\right] = .0541 \text{bits}$$

$$IG(Type) = 1 - \left[\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right)\right] = 0 \text{ bits}$$

- Patrons has the highest IG of all attributes and so is chosen as the root

Assessing the performance of the learning Algorithm:

- A learning algorithm is good if it produces hypothesis that do a good job of predicting the classification of unseen examples.
- Obviously, a prediction is good if it turns out to be true, so we can assess the quality of a hypothesis by checking its predictions against the correct classification once we know it.
- We do this on a set of examples known as the **test set.**
- The following are the steps to assess the performance,
    1. Collect a large set of examples
    2. Divide it into two disjoint sets: the **training set** and the **test set**
    3. Apply the learning algorithm to the training set, generating a hypothesis *h.*
    4. Measure the percentage of examples in the test set that are correctly classified *h.*
    5. Repeat steps 1 to 4 for different sizes of training sets and different randomly selected training sets of each size.
- The result of this procedure is a set of data that can be processed to give the average prediction quality as a function of the size of the training set.
- This function can be plotted on a graph, giving what is called the **learning curve** for the algorithm on the particular domain.
- The following diagram shows the learning curve for DECISION-TREE-LEARNING with the above attribute table example.



- In the graph the training set grows, the prediction quality increases.
- Such a curves are called **happy graphs**.

Explanation Based Learning:

- Explanation-based learning is a method for extracting general rules from individual observations
- Human appear to learn quite a lot from example
- Basic idea: Use results from one examples problem solving effort next time around.
- when an agent can utilize a worked example of a problem as a problem-solving method, the agent is said to have the capability of **explanation-based learning (EBL).**
- This is a type of **analytic learning.**
- The advantage of explanation-based learning is that, as a **deductive mechanism,** it requires only a single training example (inductive learning methods often require many training examples)
- To utilize just a single example most **EBL** algorithms require all of the following,
    - The training example
    - A Goal concept
    - An Operationality Criteria
    - A Domain theory
- An **EBL** accepts four kinds of input as follows,
    - **A training example:-** what the learning sees in the world
    - **A goal concept:-** a high level description of what the program is supposed to learn
    - **An operational criteria:-** a description of which concepts are usable
    - **A domain theory:-** a set of rules that describe relationships between objects and actions in a domain

- The domain theory has two types as,
  - **Explanation: -** the domain theory is used to prune away all unimportant aspects of the training example with respect to the goal concept.
  - **Generalisation: -** the explanation is generalized as far possible while still describing the goal concept

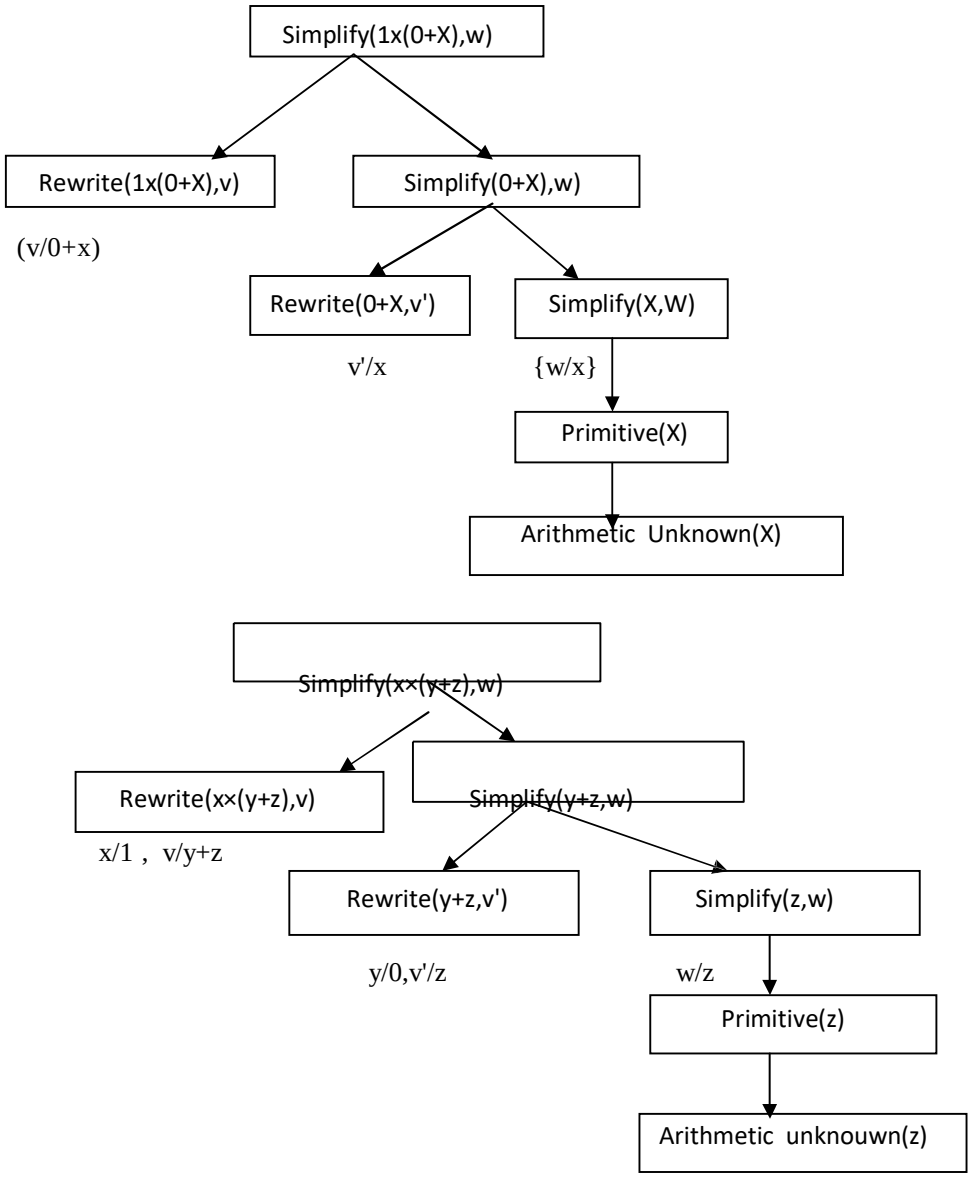
- **For Example:-**
  - Cary Larson once drew a cartoon in which a bespectacled caveman, Zog, is roasting a lizard on the end of a pointed stick.
  - He is watched by an amazed crowd of less intellectual contemporaries.
  - In this case, the caveman generalize by explaining the success of the pointed stick which supports the lizard and keeps the hand away from the fire.
  - This explanation can infer a general rule: that any long, rigid, sharp object can be used to toast small, soft bodies.
  - This kind of generalization process is said to be **Explanation based Learning.**
  - The **EBL** procedure is very much domain theory driven with the training example helping to focus the learning.
  - Entailment constraints satisfied by **EBL** is
    - *Hypothesis* $\wedge$ *Descriptions* |= *Classifications*
    - *Background* |= *Hypothesis*

Extracting rules from examples:

- **EBL** is a method for extracting general rules from individual observations.
- The basic idea is first to construct an explanation of the observation using prior knowledge.
- Consider the problem of differentiating and simplifying the algebraic expressions.
- If we differentiate the expression X^2 with respect to X, we obtain 2X.
- The proof tree for Derivative(X^2, X) = 2X is too large to use, so we will use a simpler problem to illustrate the generalization method.
- Suppose our problem is to simplify 1 x (0 + X).
- The knowledge base includes the following rules
  - *Rewrite(u, v)* $\wedge$*Simplify(v, w)* $\Rightarrow$ *Simplify(u, w)*
  - *Primitive(u)* $\Rightarrow$ *Simplify(u, u)*
  - *ArithmeticUnknown(u)* $\Rightarrow$ *Primitive(u)*
  - *Number(u)* $\Rightarrow$ *Primitive(u)*
  - *Rewrite(1 x u, u)*
  - *Rewrite(0 x u, u)*

- **EBL Process Working**
- The EBL work as follows
  1. Construct a proof that the goal predicate applies to the example using the available background knowledge
  2. In parallel, construct a generalized proof tree for the variabilized goal using the same inference steps as in the original proof.
  3. Construct a new rule whose left hand side consists of leaves of the proof tree and RHS is the variabilized goal.
  4. Drop any conditions that are true regardless of the values of the variables in the goal.

- In the diagram, the first tree shows the proof of original problem instance, from which we can derive
  - *ArithmeticUnknown(z) = Simplify(1 x (0 + z), z)*
- The second tree shows the problem where the constants are replaced by variables as generalized proof tree.



Improving efficiency:

- The generalized proof tree mentioned above gives or yields more than one generalized rule.
- For example if we terminate, or **PRUNE,** the growth of the right hand branch in the tree when it reached the *primitive* step, we get the rule as,
  - *Primitive(z)* $\Rightarrow$ *Simplify(1 X (0 + z), z)*

- This rule is a valid as, but more general than, the rule using *ArithmeticUnknow*, because it covers cases where z is a number.
- After pruning the step,
  - *Simplify (y + z, w)*, yielding the rule
  - *Simplify (y + z, w)* $\Rightarrow$ *Simplify (1 X(y + z), w)*
- The problem is to choose which of these rules.
- The choice of which rule to generate comes down to the question of efficiency.
- There are three factors involved in the analysis of efficiency gains from **EBL** as,
  - Adding large number of rules can slow down the reasoning process, because the inference mechanism must still check those rules even in case where they not a solution. It increases the **branching factor** in the search space.
  - To compensate the slowdown in reasoning, the derived rules must offer significant increase in speed for the cases that they do not cover. This increase occurs because the derived rules avoid dead ends but also because they short proof also.
  - Derived rule is as general as possible, so that they apply to the largest possible set of cases.

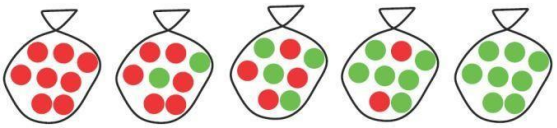
Statistical Learning Methods:

- Agents can handle uncertainty by using the methods of probability and decision theory.
- But they must learn their probabilistic theories of the world from experience.
- The learning task itself can be formulated as a process of probabilistic inference.
- A Bayesian view of learning is extremely powerful, providing general solutions to the problem of noise, overfitting and optimal prediction.
- It also takes into account the fact that a less than omniscient agent can never be certain about which theory of the world is correct, yet must still make decisions by using some theory of the world.

Statistical Learning

- The key concepts of statistical learning are **Data** and **Hypotheses.**
- **Data** are evidence (i.e.) instantiations of some or all of the random variables describing the domain.
- **Hypotheses** are probabilistic theories of how the domain works, including logical theories as a special case.
- **For Example:-**
  - The favorite surprise candy comes in two flavors as Cherry and Lime
  - The manufacturer has a peculiar sense of humor and wraps each piece of candy in the same opaque wrapper, regardless of flavor.
  - The candy is sold in very large bags of which there are known to be five kinds-again, indistinguishable from the outside:

    h1: 100% cherry candies
    h2: 75% cherry candies + 25% lime candies
    h3: 50% cherry candies + 50% lime candies
    h4: 25% cherry candies + 75% lime candies
    h5: 100% lime candies



- Given a new bag of candy the random variable **H** (for hypotheses) denotes the type of tile bag, with possible values h1 through h5. **H** is not directly observable.
- As the pieces of candy are opened and inspected, data are revealed as D1, D2…Dn in which each D is a random variable with possible values Cherry and Lime.
- **The basic task** faced by the agent is to predict the flavor of the next piece of candy

Bayesian Learning:

- Bayesian Learning calculates the probability of each hypothesis, given the data and makes predictions by using all the hypotheses, weighted by their probabilities.
- In this way learning is reduced to probabilistic inference.
- Let D be all data, with observed value d, then probability of a hypothesis h_i, using Bayes rule

$$P(h_i|d) = a\ P(d \mid h_i)P(h_i)$$

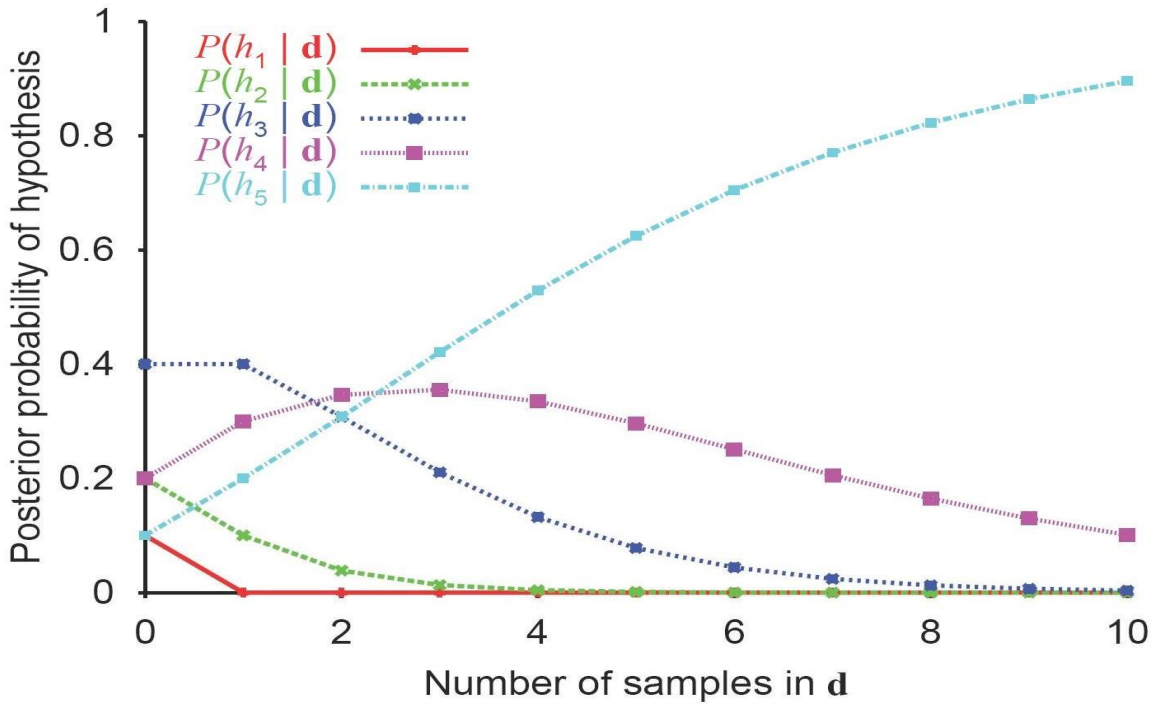
- For prediction about quantity X :

$$P(X|d) = \sum P(X|d,h_i)P(h_i|d) = \sum P(X|h_i)P(h_i|d)$$

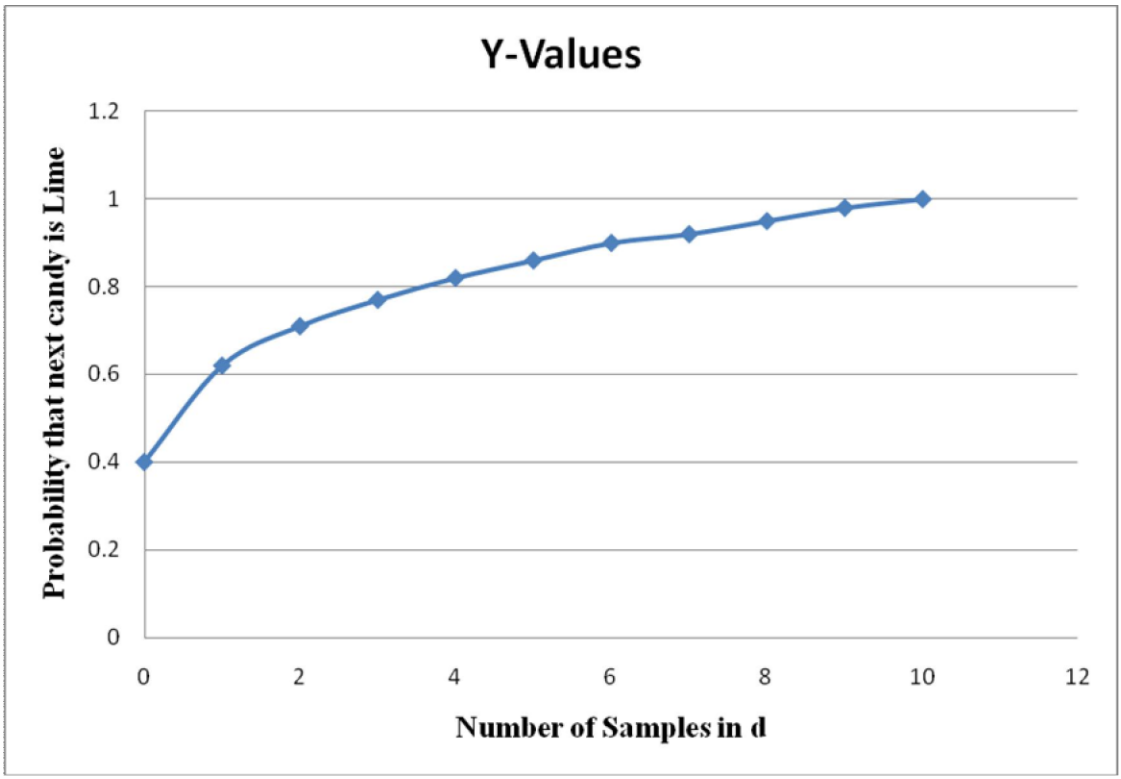
- Where it is assumed that each hypothesis determines a probability distribution over **X.**
- This equation shows that predictions were weighted averages over the predictions of the individual hypothesis
- The key quantities in the Bayesian approach are the
  - Hypothesis ***Prior,*** $P(h_i)$
  - **Likelihood** of the data under each hypothesis, $P(d \mid h_i)$
- For candy example, assume the time being that the prior distribution over h1,….h5 is given by (0.1,0.2,0.4,0.2,0.1), as advertised by the manufacturer.
- The likelihood of the data is calculated under the assumption that the observations are i..i..d, that is i= independently, i= identically and d= distributed So that

$$P(d \mid h_i) = \prod_j P(d_j \mid h_i)$$

- The following figure shows how the posterior probabilities of the five hypotheses change as the sequence of 10 Lime is observed.
- Notice that the probabilities start out at their prior values. So h1 is initially the most likely choice and remains so after 1 Lime candy is unwrapped.
- After 2 Lime candies are unwrapped, h1 is most likely; after 3 or more, h5 is the most likely.



- The following figure shows the predicted probability that the next candy is Lime as expected, it increases monotonically toward 1



Characteristics of Bayesian Learning:

- The true hypothesis eventually dominates the Bayesian prediction. For any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will vanish, because the probability of generating uncharacteristic data indefinitely is vanishingly small.
- More importantly, the Bayesian prediction is optimal, whether the data set is small or large.

- For real learning problems, the hypothesis space is usually very large or infinite.
- In most cases choose the approximation or simplified methods.

Approximation

- Make predictions based on a single most probable hypothesis *hi* that maximizes P(hi|d).
- This is often called a maximum a posteriori or MAP hypothesis.
- Predictions made according to an MAP hypothesis h_{MAP} are approximately Bayesian to the extent that $P(X|d) \approx P(X| h_{MAP})$.
- In candy example, $h_{MAP} = h_5$ after three lime candies in a row, so the MAP learner then predicts that the fourth candy is lime with probability 1.0 a much more dangerous prediction than the Bayesian prediction of 0.8 shown in the above graphs.
- As more data arrive, the MAP and Bayesian predictions become closer, because the competitors to the MAP hypothesis become less and less probable.
- **Finding MAP hypothesis is much easier than Bayesian Learning is more advantage.**

Learning with Complete Data:

- The statistical learning method begins with **parameter learning with complete data.**
- A **parameter learning** task involves finding the numerical parameter for the probability model.
- The structure of the model is fixed.
- Data are complete when each data point contains values for every variable in the probability model.
- Complete data simplify the problem of learning the parameters of complex model.

Maximum Likelihood Parameter Learning: Discrete Models

- Suppose we buy a bag of lime and cherry candy from a manufacturer whose lime-cherry proportion are completely unknown.
- The fraction can be anywhere between 0 and 1.
- The parameter in this case is θ, which is the proportion of cherry candies, and the hypothesis is h_θ.
- The proportion of lime is (1-θ).
- We assume all the proportions are known a priori then Maximum Likelihood approach can be applied.
- If we model the situation in Bayesian network, we need just one random variable called **Flavor** it has values cherry and lime.
- The probability of cherry is θ`.
- If we unwrap *N* candies, of which *C* are cherries and *L=N-C* are limes.
- The likelihood of the particular set is,

$$P(d / h_\theta) = \prod_{j=1}^{N} P(dj \setminus h_\theta) = \theta^c .(1-\theta)^L$$

- The maximum-likelihood hypothesis is given by the value of e` that maximizes the expression.
- It can be obtained by maximizing the **log likelihood.**

$$L1(d \mid h_\theta) = P(d \mid h_\theta) = \sum_{J=1}^{N} \log(P(d_j \setminus h_\theta) = c \log\theta + L \log(1-\theta)$$

- To find the ML value of θ differentiate wrt θ and then equate resulting to zero

$$\frac{JL(d\setminus h\theta)}{d\theta} = \frac{c}{\theta} - \frac{L}{1-\theta} \quad , \quad \frac{c}{\theta} - \frac{L}{1-\theta} = 0, \ \theta - \frac{c}{c+l} \quad , \quad \theta - \frac{c}{N}$$ where c+l = N

- The standard method for maximum likelihood parameter learning is given by
  - Write down an expression for the likelihood of the data as a function of the parameters
  - Write down the derivative of the log likelihood with respect to each parameter.
  - Find the parameter values such that the derivatives are zero
- The most important fact is that, with complete data, the maximum-likelihood parameter learning problem for a Bayesian network

Maximum Likelihood Parameter Learning: Continuous Models

- Continuous variables are ubiquitous (everywhere) in real world applications.
- Example of Continuous probability model is linear-Gaussian model.
- The principles for maximum likelihood learning are identical to discrete model.
- Let us take a simple case of learning the parameters of a Gaussian density function on a single variable.
- The data are generated as follows

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

- Parameters of this model μ = mean and σ = Standard deviation.
- Let the observed values be $x_1, x_2, \ldots\ldots x_N$
- Then the log likelihood is given as

$$L = \sum_{j=1} \log \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{-2\sigma^2}} = N = (-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1} \frac{(x_j-\mu)^2}{2\sigma^2}$$

- Setting the derivatives to zero as usual, we obtain

$$\frac{\partial L}{\partial \mu} = \frac{1}{\sigma^2} \sum_{j=1}^{N} (x - \mu) = 0 \qquad \Rightarrow \mu = \frac{\sum_j x_j}{N}$$

$$\frac{\partial L}{\partial \sigma} = \frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^{N} (x_j - \mu)^2 = 0 \qquad \Rightarrow \sigma = \sqrt{\frac{\sum_j (x_j - \mu)^2}{N}}$$

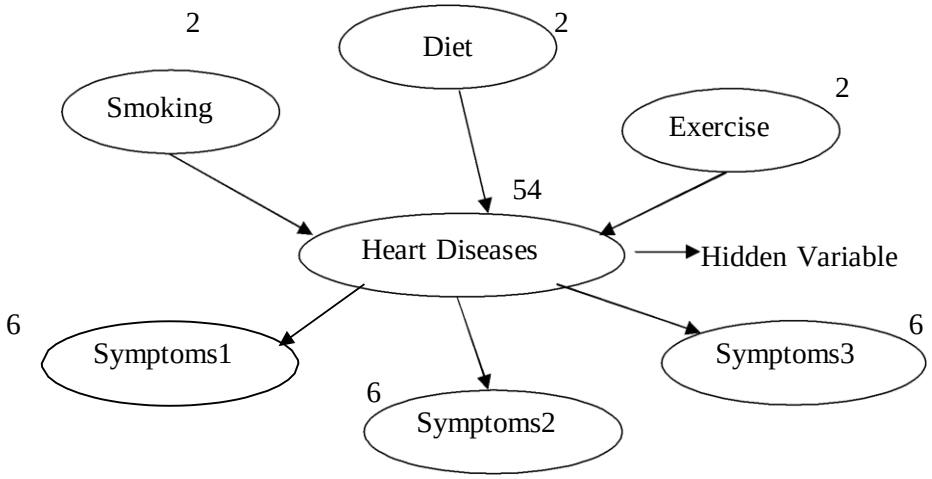
- Maximum likelihood value of the mean is the simple average.

- Maximum likelihood value of the standard deviation is the square root of the simple variance.

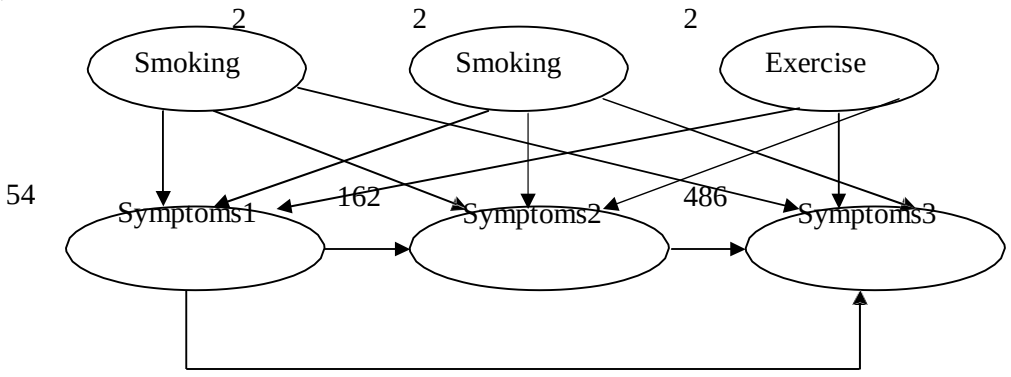
Learning with Hidden Variables:

1. Many real world problems have hidden variables (or) latent variables which are not observable in the data that are available for learning.
2. For Example:- Medical record often include the observed symptoms, treatment applied and outcome of the treatment, but seldom contain a direct observation of disease itself.

Assumed the diagnostic model for heart disease. There are three observable predisposing factors and 3 observable symptoms. Each variable has 3 possible values (none, moderate and severe)

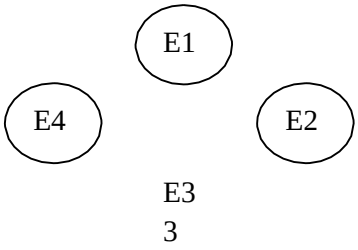


If hidden is removed the total number of parameters increases from 78 (54 + 2 + 2 + 2 + 6 + 6 + 6) to 708

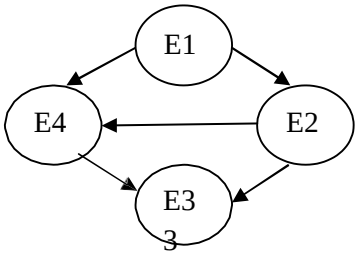


3. Hidden variables can dramatically reduce the number of parameters required to specify the Bayesian network, there by reduce the amount of data needed to learn the parameters.
4. It also includes estimating probabilities when some of the data are missing.
5. The reason we learn Bayesian network with hidden variable is that it reveals interesting structures in our data.
6. Consider a situation in which you can observe a whole bunch of different evidence variables, Er through En. They are all different symptoms that a patient might have.

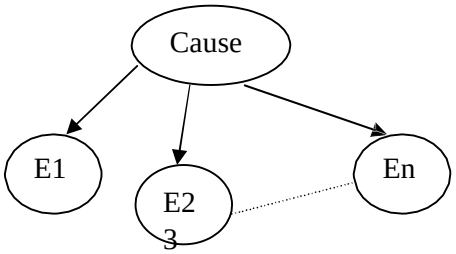
Different systems:



If the variables are conditionally dependent on one another, we will get a highly connected graph that representing the entire joint distribution between the variables.



7. The model can be made simpler by introducing an additional "cause" node. It represents the underlying disease state that was causing the patient symptoms.



This will have O(n) parameters, because the evident variables are conditionally independent given the causes.

8. Missing data
   - Imagine that we have 2 binary variables A and B that are not independent. We try to estimate the Joint distribution.

A	B
1	**1**
1	**1**
0	**0**
0	**0**
0	**0**
0	**H**
0	**1**
1	**0**

- It is done by counting how many were (true, true) and how may (false, false) and divide by the total number of cases to get maximum likelihood estimate.
- The above data set has some data missing (denoted on "H"). There's no real way to guess the value during our estimation problem.
- The missing data items can be independent of the value it would have had. The data can be missed if there is a fault in the instrument used to measure.
- For Example:- Blood pressure instrument fault, so blood pressure data can be missing)

9. We can ignore missing values and estimate parameters

Estimate Parameters

	$\underline{0}$ A	1 A
$0\bar{B}$	3/7	1/7
1 B	1/7	2/7

	0 $\bar{A}$	1 A
$0B$	0.429	0.143
1 B	0.143	0.285

We can consider H = 0 (or) H = 1

$\therefore \log \Pr(D / M) = \log(\Pr/ D, H = 0 / M) + \Pr(D, H = 1 / M))$

= 3 log 0.429 + 2 log 0.143 + 2 log 0.285 + log (0.429 + 0.143)

= -9.498 Maximum likelihood score

10. We also try to fit it with best value.

For the above cause consider H=0, Estimated parameters as follows,

	0 $\bar{A}$	1 A
$0\bar{B}$	4/8	1/8
1 B	1/8	2/8

	$\underline{0}$ A	1 A
$0\bar{B}$	0.5	0.125
1 B	0.125	0.25

$\therefore \log \Pr(D / M) = \log(\Pr/ D, H = 0 / M) + \Pr(D, H = 1 / M))$

= 3 log 0.5 + 2 log 0.125 + 2 log 0.25 + log (0.5 + 0.125)

= -9.481

There is an improvement in likelihood value.

11. We will employ some soft assignment technique. we fill the value of the missing variable by using our knowledge of the joint distribution over A, B and compute a distribution over H.

	0 $\bar{A}$	1 A
$0\bar{B}$	0.25	0.25
1 B	0.25	0.25

Initial guess Uniform distribution.

Compute probability distribution over H

$\Pr(H / D, \theta_0) = \Pr(H / D^6, \theta_0)$ because it refers to 6th case in the observed data in the table.

= $\Pr(H / D^6, \theta_0)$

= $\Pr(B / \neg A, \theta_0)$

because missing variable is B and the observed one is not A, we need the probability of B given not A.

$\Pr(B / \neg A, \theta_0) = \Pr(\neg A, B / \theta_0) / \Pr(\neg A / \theta_0)$

$= \frac{0.25}{0.5} = 0.5$

H = 0 probability is 0.5

H = 1 probability is 0.5

A	B
1	**1**
1	**1**
0	**0**
0	**0**
0	**0**
0	**0,0.5** **1,0.5**
0	**1**
1	**0**

Now maximum likelihood estimation using expected counts.

So expected parameter is

	0 $\overline{A}$	1 A
$0\overline{B}$	3.5/8	1/8
1 B	1.5/8	2/8

	0 $\overline{A}$	1 A
$0\overline{B}$	0.4375	0.125
1 B	0.1875	0.25

New estimate is

$\Pr(H / D, Q1) = \Pr(\neg, B / Q1) / \Pr(\neg A / Q1)$

$= \frac{0.1875}{0.625}$

So the new table is = 0.3

A	B
1	**1**
1	**1**
0	**0**
0	**0**

0	**0**
0	**0,0.7** **1,0.3**
0	**1**
1	**0**

	$\frac{0}{A}$	$\frac{1}{A}$
$0\bar{B}$	3.7/8	1/8
1 B	1.3/8	2/8

	$\frac{0}{A}$	$\frac{1}{A}$
$0\bar{B}$	0.4625	0.125
1 B	0.1625	0.25

$\therefore$ theta2 is θ_2 is

$$\Pr(H / D,\theta_2) = \Pr(\neg Ar, B / \theta_2) / \Pr(\neg A / \theta_2)$$

$$= \frac{0.1625}{0.625} = 0.26$$

log likelihood is increasing

$$\log \Pr(0 / \theta_0) = -10.3972$$

$$\log \Pr(D / \theta_1) = -9.4760$$

$$\log \Pr(D / \theta_2) = -9.4524$$

Since all values are negative it is in increasing order.

$\therefore$ We have to choose the best value

12. The above iterative process is called **EM** algorithm.
    a. The basic idea in EM algorithm is to pretend that we know the parameters of the model and then to infer the probability that each data point belongs to each component.
    b. After that we refit the components of the data, where each component is fitted to the entire data set with each point weighted by probabilit y that it belongs to the component.
    c. This process is iterated until it converges.
    d. We are completing the data by inferring probability distributions over the hidden variable.

13. EM Algorithm

a. want to find θ to maximize PR(D/θ)
   To find theta (θ) that maximizes the probability of data for given theta (θ)
b. Instead find θ, $\tilde{P}$ to maximize, where $\tilde{P}$ = P tilde

$$g(\theta,\tilde{P}) = \sum_H \tilde{P}(H)\log(\Pr(D,H/\theta)/\tilde{P}(H))$$

$$= E\tilde{p}\log\Pr(D,H/\theta) - \log\tilde{P}(H)$$

   Where, $\tilde{P}(H)$ = Probability distribution over hidden variables, H= Hidden Variables

c. Find optimum value for g
   - ✓ holding θ fixed and optimizing $\tilde{P}$

- ✓ holding $\tilde{P}$ fixed and optimizing θ
- ✓ and repeat the procedure over and again

d. g has some local and global optima as PR(D/ θ)

e. Example:-

i. Pick initial θ_0

ii. Probability of hidden variables given the observed data and the current model. Loop until it converges

$$\tilde{P}_{t+1}(H) = \Pr(H / D, \theta_t)$$

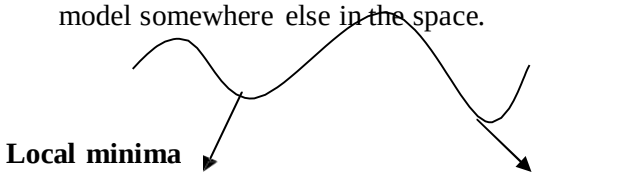
$$\tilde{P}_{t+1} = \overset{\arg\max}{\theta} \underset{r\tilde{P}t+1}{E} \log \Pr(D, H / \theta)$$

We find the maximum likelihood model for the "expected data" using the distribution over H to generate expected counts for different case.

iii. Increasing likelihood.

iv. Convergence is determined (but difficult)

v. Process with local optima i.e., sometimes it converges quite effectively to the maximum model that's near the one it started with, but there's much better model somewhere else in the space.



Local minima

optimum Value

EM for Bayesian Network:

Let us try to apply EM for Bayesian Networks.

1. Our data is a set of cases of observations of some observable variables i.e. D = Observable Variables
2. Our hidden variables will actually be the values of the hidden node in each case. H = Values of hidden variable in each case
   For Example:- If we have 10 data case and a network with one hidden node, then we have 10 hidden variables on missing pieces of data.
3. Assume structure is known
4. Find maximum likelihood estimation of CPTSs that maximize the probability of the observed data D.
5. Initialize CPT's to anything (with no 0's)

Filling the data

1. Fill in the data set with distribution over values for hidden variables
2. Estimate Conditional probability using expected counts.
   We will compute the probability distribution over H given D and theta (θ), we have 'm' different hidden variables, one for the value of node H in each of the m data cases.

$$\tilde{P}_{t+1}(H) = \Pr(H / D, \theta_t)$$

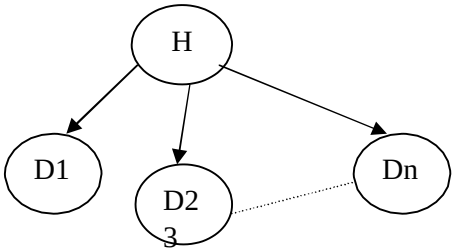
$$= \prod_m \Pr(H^m / D^m, \theta_t)$$

3. Compute a distribution over each individual hidden variable
4. Each factor is a call to bayes net inference

5. For Example:-

a. Consider a simple case with one hidden node

All nodes are binary



0_1	0_2	…….	D_n	$\Pr(H^m / D^m, \theta_t)$
1	**1**	…….	**0**	**0.9**
0	**1**	…….	**0**	**0.2**
0		…….	**1**	**0.1**
1	**1**	…….	**1**	**0.2**
1	**1**	…….	**1**	**0.5**

$\Pr(H^m / D^m, \theta_t)$ = Bayes net inference

b. We use bayes net inference to compute for each case in our data set, the probabilit y that H would be true, given the values of the observed variables.

c. To compute expected count i.e., the expected number of times H is true, we will add up the probability of H.

$$E(H) = \sum_m \Pr(H^m / D^m, \theta_t)$$

$$= 1.9(0.9 + 0.2 + 0.1 + 0.2 + 0.5)$$

d. To get the expected number of times that H and D2 are true, we find all the cases in which D2 is true, and add up their probabilities of H being true.

$$E(H) = \sum_m \Pr(H^m / D^m, \theta_t)$$

$$= 1.9$$

$$E(H \wedge D2) = \sum_m \Pr(H^m / D^m, \theta_t) I(D^m_2)$$

$$= 0.9 + 0.2 + 0.2 + 0.5$$

$$= 1.8$$

$$\Pr(D2 / H) = \frac{1.8}{1.9}$$ Probability of D2 given H

$$= 0.9473$$

Instance Based Learning:-

- A **parametric learning** method is simple and effective.
- In parametric learning method when we have little data or data set grows larger then the hypothesis is fixed.
- Instance based model represents a distribution using the collection of training instances.
- Thus the number of parameter grows with the training set.
- **Non Parametric learning** methods allows the hypothesis complexity to grow with the data.
- **Instance based Learning or Memory based learning** is a non-parametric model because they construct hypothesis directly from the training set.
- The simplest form of learning is **memorization**.

- When an object is observed or the solution to a problem is found, it is stored in memory for future use.
- Memory can be thought of as a lookup table.
- When a new problem is encountered, memory is searched to find if the same problem has been solved before.
- If an exact match for the search is required, learning is slow and consumes very large amounts of memory.
- However, approximate matching allows a degree of generalization that both speeds learning and saves memory.
- For Example:- " If we are shown an object and we want to know if it is a chair, then we compare the description of this new object with descriptions of "typical" chairs that we have encountered before.
- If the description of the new object is "close" to the description of one of the stored instances then we may call it a chair.
- Obviously, we must defined what we mean by "typical" and "close".
- |To better understand the issues involved in learning prototypes, we will briefly describe three experiments in **Instance based learning (IBL)** by Aha, Kibler and Albert (1991).
- IBL learns to classify objects by being shown examples of objects, described by an attribute/value list, along with the class to which each example belongs.
- **Experiment 1:-**
  - In the first experiment (**IB1**), to learn a concept simply required the program to store every example.
  - When an unclassified object was presented for classification by the program, it used a simple **Euclidean distance measure** to determine the **nearest neighbor** of the object and the class given to it was the class of the neighbor.
  - The simple scheme works well, and is tolerant to some noise in the data.
  - Its major disadvantage is that it requires a large amount of storage capacity.
- **Experiment 2:-**
  - The second experiment (**IB2**) attempted to improve the space performance of **IB1**.
  - In this case, when new instances of classes were presented to the program, the program attempted to classify them.
  - Instances that were correctly classified were ignored and only incorrectly classified instances were stored to become part of the concept.
  - This scheme reduced storage dramatically, it was less noise tolerant than the first.
- **Experiment 3:-**
  - The third experiment (**IB3**) used a more sophisticated method for evaluating instances to decide if they should be kept or not.
  - IB3 is similar to IB2 with the following additions.
  - IB3 maintains a record of the number of correct and incorrect classification attempts for each saved instance.
  - This record summarized an instances classification performance.
  - IB3 uses a significance test to determine which instances are good classifiers and which ones are believed to be noisy.
  - The latter are discarded from the concept description.
  - This method strengthens noise tolerance, while keeping storage requirements down.

Neural Network:-

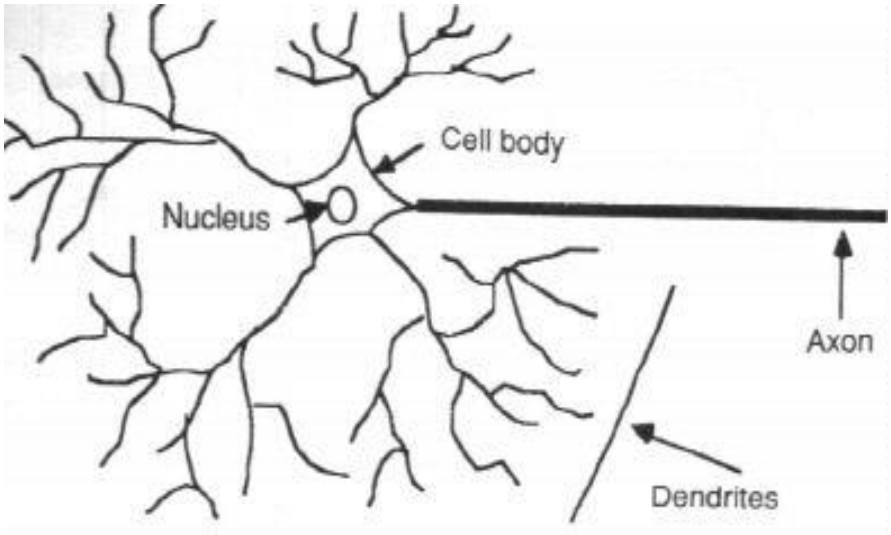
- A neural network is an interconnected group of neurons.

- The prime examples are biological neural networks, especially the human brain.
- In modern usage the term most often refers to ANN (Artificial Neural Networks) or neural nets for short.
- An **Artificial Neural Network** is a mathematical or computational model for information processing based on a connections approach to computation.
- It involves a network of relatively simple processing elements, where the global behavior is determined by the connections between the processing elements and element parameters.
- In a neural network model, simple nodes (neurons or units) are connected together to form a network of nodes and hence the term "**Neural Network**"

The biological neuron Vs Artificial neuron:-

Biological Neuron:-

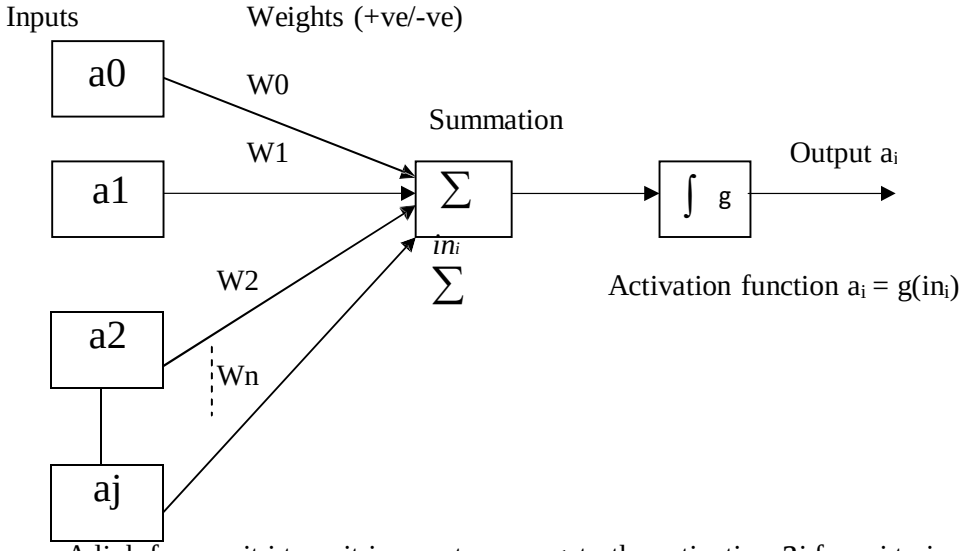
- The brain is a collection of about 10 million interconnected neurons shown in following figure.



- Each neuron is a cell that uses biochemical reactions to receive, process and transmit information.
- A neurons dendrites tree is connected to a thousand neighboring neurons.
- When one of those neurons fire, a positive or negative charge is received by one of the dendrites.
- The strengths of all the received charges are added together through the processes of spatial and temporal summation.
- Spatial summation occurs when several weak signals are converted into a single large one, while temporal summation converts a rapid series of weak pulses from one source into one large signal.
- The aggregate input is then passed to the soma (cell body).
- The soma and the enclosed nucleus don't play a significant role in the processing of incoming and outgoing data.

Artificial Neuron (Simulated neuron):-

Artificial Neurons are composed of nodes or units connected by directed links as shown in following figure.



- A link from unit j to unit i serve to propagate the activation aj from j to i.
- Each link also has a numeric weight Wj, i associated with it, which determines the strength and sign of the connection.
- Each unit i first computes a weighted sum of its inputs

$$in_i = \sum_{j=0}^{n} Wj, ia_j$$

- Then it applies an activation function g to this sum to derive the output.

$$a_i = g(in_i) = g(\sum_{j=0}^{n} Wj, ia_j)$$

- A simulated neuron which takes the weighted sum as its input and sends the output 1, if the sum is greater than some adjustable threshold value otherwise it sends 0.
- The activation function g is designed to meet two desires,
  - The unit needs to be "active" (near +1) when the "right" inputs are given and "inactive" (near 0) when the "wrong" inputs are given.
  - The activation needs to be non linear, otherwise the entire neural network collapses into a simple linear function.
- There are two activation functions,
  - Threshold function
  - Sigmoid function

Comparison between Real neuron and Artificial neuron (or) Simulated neuron:-

	Computers (Artificial neuron)	**Human brain (Real neuron)**
Computational Units	1 CPU, 10^5 gates	10^{11} neurons
Storage Units	10^9 bits RAM, 10^{11} bits disk	10^{11} neurons, 10^{14} Synapses
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec
Neuron updates/Sec	10^5	10^{14}

- The above table shows the comparison based on raw computational sources available to computer and human brain.
- The following table shows the comparison based on structure and working method.

Real neuron	Simulated neuron (Artificial neuron)
The character of real neuron is not modeled	The properties are derived by simply adding up the weighted sum as its input
Simulation of dendrites is done using electro chemical reaction	A process output is derived using logical circuits
Billion times faster in decision making process	Million times faster in decision making process
More fault tolerant	Less fault tolerant
Autonomous learning is possible	Autonomous learning is not possible

Abstract properties of neural networks:-

- They have the ability to perform distributed computation
- They have the ability to learn.
- They have the ability to tolerate noisy inputs

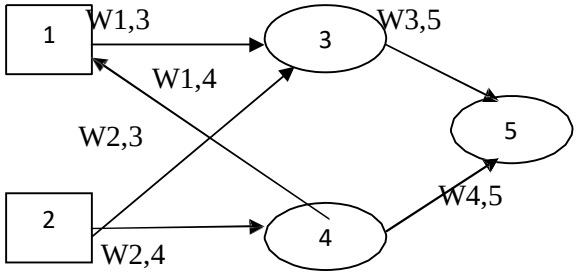
Neural network Structures:-

- The arrangement of neurons into layers and the connection patterns within and between layers is called the network structures.
- They are classified into two categories depends on the connection established in the network and the number of layers.
    - Acyclic (or) Feed-forward network
        - Single layer feed-forward network
        - Multilayer feed-forward network
    - Cyclic (or) Recurrent networks
- The following table shows the difference between Feed-forward network and Recurrent network,

Feed-Forward network	Recurrent network
Unidirectional Connection	Bidirectional Connection
Cycles not exist	Cycles exist
A layered network, backtracking is not possible	Not a layered network, backtracking is not possible
Computes a function of the input values that depends on the weight settings, no internal state other than the weight settings	Internal state stored in the activation levels of the units.
Example:- Simple layering Models	Example:- Brain
A model used for simple reflex agent	A model used for complex agent design

Feed-Forward network:-

- A feed-forward network represents a function of its current input; thereby it has no internal state other than the weights themselves.
- Consider the following network, which has two hidden input units and an output unit.

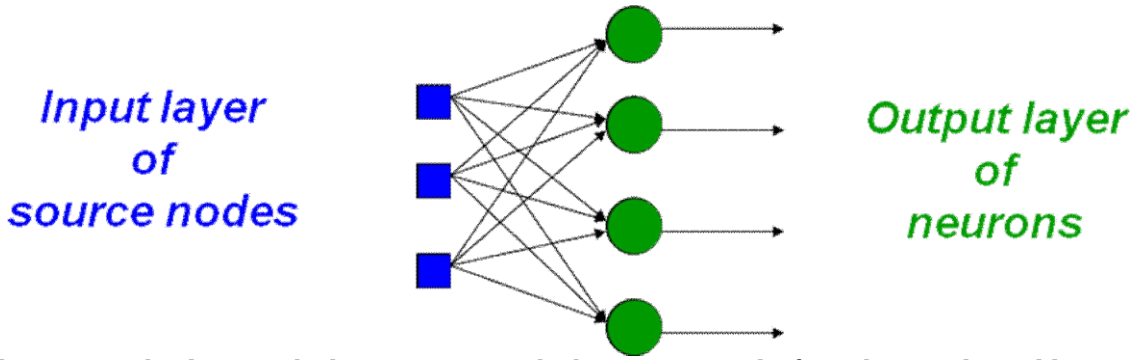


- Given an input vector x = (x1, x2), the activations of the input units are set to (a1,a2) = (x1,x2) and the network computes

$$a_5 = g(W_{3,5}a_3 + W_{4,5}a_4) = g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))$$

Single Layer feed-forward network:-

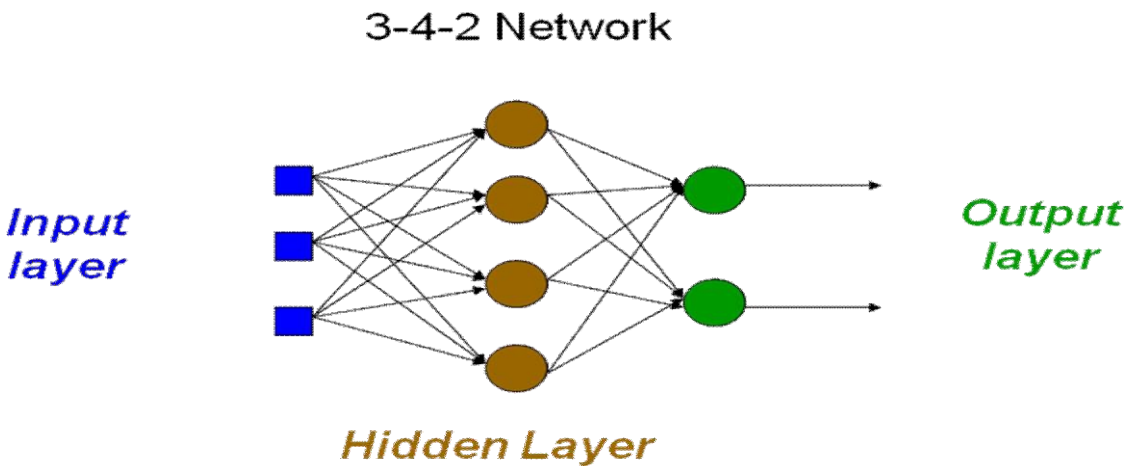
- A single layer network has one layer of connection weights.
- The following figure shows the single layer feed forward network.



- The units can be distinguished as input units, which receive signals from the outside world, and output units, from which the response of the network can be read.
- The input units are fully connected to output units but are not connected to other input units.
- They are generally used for pattern classification.

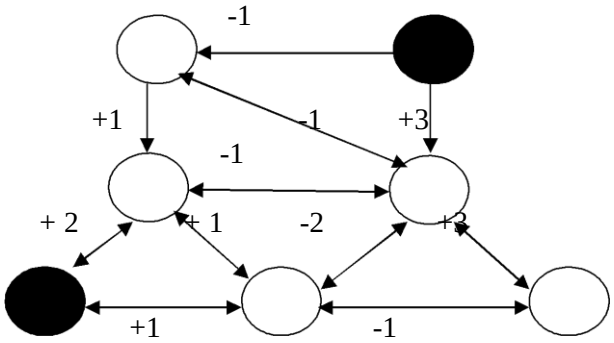
Multi Layer feed-forward network:-

- A multi layer network with one or more layers of nodes called hidden nodes.
- Hidden nodes connected between the input units and the output units.
- The below figure shows the multilayer feed-forward network.
- Typically there is a layer of weights between two adjacent levels of units.
- The network structure has 3 input layer, 4 hidden layer and 2 output layer.
- Multilayer network can solve more complicated problems than single layer networks.
- In this network training may be more difficult.



Recurrent network:-

- Each node is a processing element or unit, it may be in one of the two states (Black-Active, White-Inactive) units are connected to each other with weighted symmetric connection.
- A positive weighted connection indicates that the two units tend to activate each other.
- A negative connection allows an active unit to deactive neighboring unit.
- The following diagram shows the simple recurrent network which is a Hopfield network,



- **Working method:-**
  - A random unit is chosen.
  - If any of its neighbors are active, the unit computes the sum of the weights on the connections to those active neighbors.
  - If the sum is positive, the unit becomes active, otherwise it become inactive.
- **Fault tolerance:-** If a new processing element fails completely, the network will still function properly.

Learning Neural network structures:-

- It is necessary to understand how to find the best network structure.
- If a network is too big is chosen, it will be able to memorize all the examples by forming a large lookup table, but will not generalize well to inputs that have not been seen before.
- There are two kinds of networks must be considered namely,
  - Fully connected network
  - Not Fully connected network
- **Fully Connected networks:-**

- If fully connected networks are considered, the only choices to be made concern the number of hidden layers and their sizes.
- The usual approach is to try several and keep the best.
- The cross validation techniques are needed to avoid peeking at the test set.

- **Not Fully Connected network:-**
  - If not fully connected networks are considered, then find some effective search method through the very large space of possible connection topologies.
- **Optimal Brain damage Algorithm:-**
  - The following are the steps involved in brain damage algorithm,
    1. Begin with a fully connected network
    2. Remove connections from it.
    3. After the network is trained for the first time, an information theoretic approach identifies an optimal selection of connections that can be dropped.
    4. Then the network is trained.
    5. If its performance has not decreased then the process is repeated.
    6. In addition to removing connections, it is also possible to remove units that are not contributing much to the result.
- **Tiling Algorithm:-**
  - It is an algorithm, which is proposed for growing a larger network from a smaller one.
  - it resembles decision-list learning.
  - The following are the steps involved in tiling algorithm,
    1. Start with a single unit that does its best to produce the correct output on as many of the training examples as possible.
    2. Subsequent units are added to take care of the examples that the first unit got wrong.
    3. The algorithm adds only as many units as are needed to cover all the examples.

Advantages of Neural Networks:-

- The neural network learns well, because the data were generated from a simple decision tree in the first place.
- Neural networks are capable of far more complex learning tasks of course.
- There are literally tens of thousands of published applications of neural networks

Reinforcement Learning

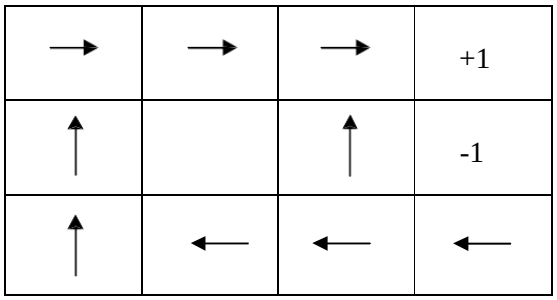
Reinforcement:

- Reinforcement is a feedback from which the agent comes to know that something good has happened when it wins and that something bad has happened when it loses. This is also called as reward.
- For Examples:-
  - In chess game, the reinforcement is received only at the end of the game.
  - In ping-pong, each point scored can be considered a reward; when learning to crawl, any forward motion is an achievement.
- The framework for agents regards the reward as part of the input percept, but the agent must be hardwired to recognize that part as a reward rather than as just another sensory input.
- Rewards served to define optimal policies in Markov decision processes.
- An optimal policy is a policy that maximizes the expected total reward.

- The task of reinforcement learning is to use observed rewards to learn an optimal policy for the environment.
- Learning from these inforcements or rewards is known as reinforcement learning
- In reinforcement learning an agent is placed in an environment, the following are the agents
  - Utility-based agent
  - Q-Learning agent
  - Reflex agent
- The following are the **Types of Reinforcement Learning,**
  - **Passive Reinforcement Learning**
  - **Active Reinforcement Learning**

Passive Reinforcement Learning

- In this learning, the agent's policy is fixed and the task is to learn the utilities of states.
- It could also involve learning a model of the environment.
- In passive learning, the agent's policy $\prod$ is fixed (i.e.) in state s, it always executes the action $\prod(s)$.
- Its goal is simply to learn the utility function $U^{\prod}(s)$.
- For example: - Consider the 4 x 3 world.
- The following figure shows the policy for that world.



- The following figure shows the corresponding utilities

0.812	0.868	0.918	+1
0.762		0.560	-1
0.705	0.655	0.611	0.388

- Clearly, the passive learning task is similar to the policy evaluation task.
- The main difference is that the passive learning agent does not know
  - Neither the transition model T(s, a,s'), which specifies the probabilit y of reaching state's from state s after doing action a;
  - Nor does it know the reward function R(s), which specifies the reward for each state.

- The agent executes a set of trials in the environment using its policy $\prod$.
- In each trial, the agent starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal states, (4,2) or (4,3).
- Its percepts supply both the current state and the reward received in that state.
- Typical trials might look like this:

$(1,1)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (2,3)_{-0.4} \rightarrow (3,3)_{-0.4} \rightarrow (4,3)_{+1}$
$(1,1)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (2,3)_{-0.4} \rightarrow (3,3)_{-0.4} \rightarrow (3,2)_{-0.4} \rightarrow (3,3)_{-0.4} \rightarrow (4,3)_{+1}$
$(1,1)_{-0.4} \rightarrow (2,1)_{-0.4} \rightarrow (3,1)_{-0.4} \rightarrow (3,2)_{-0.4} \rightarrow (4,2)_{-1}$

- Note that each state percept is subscripted with the reward received.
- The object is to use the information about rewards to learn the expected utility $U^{\prod}(s)$ associated with each nonterminal state s.
- The utility is defined to be the expected sum of (discounted) rewards obtained if policy is $\prod$ followed, the utility function is written as

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

- For the 4 x 3 world set $\gamma = 1$

Direct utility estimation:-

- A simple method for direct utility estimation is in the area of adaptive control theory by Widrow and Hoff(1960).
- The idea is that the utility of a state is the expected total reward from that state onward, and each trial provides a sample of this value for each state visited.
- Example:- The first trial in the set of three given earlier provides a sample total reward of 0.72 for state (1,1), two samples of 0.76 and 0.84 for (1,2), two samples of 0.80 and 0.88 for (1,3) and so on.
- Thus at the end of each sequence, the algorithm calculates the observed reward- to-go for each state and updates the estimated utility for that state accordingly.
- In the limit of infinitely many trails, the sample average will come together to the true expectations in the utility function.
- It is clear that direct utility estimation is just an instance of supervised learning.
- This means that reinforcement learning have been reduced to a standard inductive learning problem.
- **Advantage:-** Direct utility estimation succeeds in reducing the reinforcement learning problem to an inductive learning problem.
- **Disadvantage:-**
  - It misses a very important source of information, namely, the fact that the utilities of states are not independent
    - **Reason:-** The utility of each state equals its own reward plus the expected utility of its successor states. That-is, the utility values obey the Bellman equations for a fixed policy

      $$U^{\pi}(s) = R(s) + \lambda \sum_{s} T(s, \pi(s), s`) U^{\pi}(s`)$$

  - It misses opportunities for learning
    - **Reason:-** It ignores the connections between states
  - The algorithm often converges very slowly.

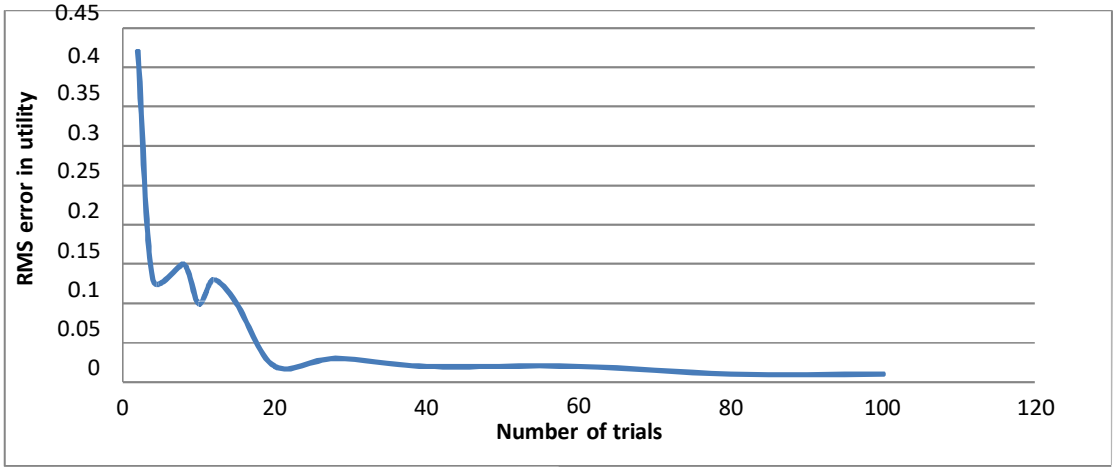
- **Reason:-** More broadly, direct utility estimation can be viewed as searching in a hypothesis space for U that is much larger that it needs to be, in that it includes many functions that violate the Bellman equations.

Adaptive Dynamic programming:-

- Agent must learn how states are connected.
- Adaptive Dynamic Programming agent works by learning the transition model of the environment as it goes along and solving the corresponding Markov Decision process using a dynamic programming method.
- For passive learning agent, the transition model $T(s,\pi(s),s`)$ and the observed rewards R(S) into Bellman equation to calculate the utilities of the states.
- The process of learning the model itself is easy, because the environment is fully observable i.e. we have a supervised learning task where the input is a state-action pair and the output is the resulting state.
- We can also represent the transition model as a table of probabilities.
- The following algorithm shows the passive ADP agent,

Function PASSIVE-ADP-AGENT(percept) **returns** an action
Inputs: percept,a percept indicating the current state s' and reward signal r'
Static: π a,fixed policy
Mdb,an MDP with model T,rewards R,discount γ
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']←r' ; R[s']←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←VALUE-DETERMINATION(π,U,mdb)
If TERMINALS?[s']**then** s,a←null else s,a←s',π[s']
return a

- Its performance on the 4 * 3 world is shown in the following figure.
- The following figure shows the root-mean square error in the estimate for U(1,1), averaged over 20 runs of 100 trials each.



- **Advantages:-**
  - It can converges quite quickly
    - **Reason:-** The model usually changes only slightly with each observation, the value iteration process can use the previous utility estimates as initial values.
  - The process of learning the model itself is easy
    - **Reason:-** The environment is fully observable. This means that a supervised learning task exist where the input is a state-action pair and the output is the resulting state.
  - It provides a standard against which other reinforcement learning algorithms can be measured.
- **Disadvantage:-**
  - It is intractable for large state spaces

Temporal Difference Learning:-

- In order to approximate the constraint equation $U^{\pi}(S)$, use the observed transitions to adjust the values of the observed states, so that they agree with the constraint equation.
- When the transition occurs from S to S^1, we apply the following update to $U^{\pi}(S)$

$$U^{\pi}(S) \leftarrow U^{\pi}(S) + \alpha(R(S) + \alpha U^{\pi}(S^1) - U^{\pi}(S))$$

- Where α = learning rate parameter.
- The above equation is called Temporal difference or TD equation.
- The following algorithm shows the passive reinforcement learning agent using temporal differences,

Function PASSIVE-TD-AGENT(precept)**returns** an action
Inputs:percept,a percept indicating the current state s' and reward signal r'
Static:π,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']←r'
If s is not null **then do**
Increment N_s[s]
U[s]←U[s] + α(N_s[s])(r + γU[s'] - U[s])
If TERMINAL?[s']**then** s,a,r←null **else** s,a,r←s',π[s'],r'
return a

- **Advantages:-**
  - It is much simpler
  - It requires much less computation per observation
- **Disadvantages:-**
  - It does not learn quite as fast as the ADP agent
  - It shows much higher variability
- The following table shows the difference between ADP and TD approach,

ADP Approach	TD Approach
ADP adjusts the state to agree with all of the successors that might occur, weighted by their probabilities	TD adjusts a state to agree with its observed successor
ADP makes as many adjustments as it needs to restore consistency between the utility estimates U and the environment model T	TD makes a single adjustment per observed transition

- The following points shows the relationship between ADP and TD approach,
  - Both try to make local adjustments to the utility estimates in order to make each state "agree" with its successors.
  - Each adjustment made by ADP could be seen, from the TD point of view, as a result of a "pseudo-experience" generated by simulating the current environment model.
  - It is possible to extend the TD approach to use an environment model to generate several "pseudo-experiences-transitions that the TD agent can imagine might happen, given its current model.
  - For each observed transition, the TD agent can generate a large number of imaginar y transitions. In this way the resulting utility estimates will approximate more and more closely those of ADP- of course, at the expense of increased computation time.

Active Reinforcement learning:-

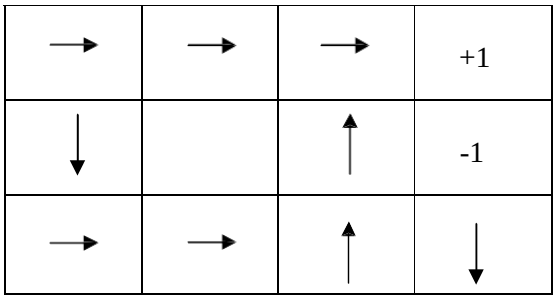
- A passive learning agent has a fixed policy that determines its behavior.
- ***"An active agent must decide what actions to do"***
- An ADP agent can be taken an considered how it must be modified to handle this new freedom.
- The following are the **required modifications:-**
  - First the agent will need to learn a complete model with outcome probabilities for all actions. The simple learning mechanism used by PASSIVE-ADP-AGENT will do just fine for this.
  - Next, take into account the fact that the agent has a choice of actions. The utilities it needs to learn are those defined by the *optimal policy.*

$$U(s) = R(s) + \gamma \max_a \sum_{s`} T(s, a, s`)U(s`)$$

  - These equations can be solved to obtain the utility function U using he value iteration or policy iteration algorithms.
  - Having obtained a utility function U that is optimal for the learned model, the agent can extract an optimal action by one-step look ahead to maximize the expected utility;
  - Alternatively, if it uses policy iteration, the optimal policy is already available, so it should simply execute the action the optimal policy recommends.

Exploration:-

- Greedy agent is an agent that executes an action recommended by the optimal policy for the learned model.
- The following figure shows the suboptimal policy to which this agent converges in this particular sequence of trials.



- The agent does not learn the true utilities or the true optimal policy! what happens is that, in the 39th trial, it finds a policy that reaches +1 reward along the lower route via (2,1), (3,1),(3,2), and (3,3).

- After experimenting with minor variations from the 276th trial onward it sticks to that policy, never learning the utilities of the other states and never finding the optimal route via (1,2),(1.3) and (2,3).
- Choosing the optimal action cannot lead to suboptimal results.
- The fact is that the learned model is not the same as the true environment; what is optimal in the learned model can therefore be suboptimal in the true environment.
- Unfortunately, the agent does not know what the true environment is, so it cannot compute the optimal action for the true environment.
- Hence this can be done by the means of **Exploitation.**
- The greedy agent can overlook that actions do more than provide rewards according to the current learned model; they also contribute to learning the true model by affecting the percepts that are received.
- An agent therefore must make a trade-off between **exploitation** to maximize its reward and **exploration** to maximize its long-term well being.
- Pure exploitation risks getting stuck in a rut.
- Pure exploitation to improve ones knowledge id of no use if one never puts that knowledge into practice.

GLIE Scheme:-

- To come up with a reasonable scheme that will eventually lead to optimal behavior by the agent a GLIE Scheme can be used.
- A GLIE Scheme must try each action in each state an unbounded number of times to avoid having a finite probability that an optimal action is missed because of an unusually bad series of outcomes.
- An ADP agent using such a scheme will eventually learn the true environment model.
- A GLIE Scheme must also eventually become greedy, so that the agents actions become optimal with respect to the learned (and hence the true) model.
- There are several GLIE Scheme as follows,
    - The agent can choose a random action a fraction 1/t of the time and to follow the greedy policy otherwise.
        - Advantage:- This method eventually converges to an optimal policy
        - Disadvantage:- It can be extremely slow
    - Another approach is to give some weight to actions that the agent has not tried very often, while tending to avoid actions that are believed to be of low utility. This can be implemented by altering the constraint equation, so that it assigns a higher utility estimate to relatively UP explored state-action pairs.
- Essentially, this amounts to an optimistic prior over the possible environments and causes the agent to behave initially as if there were wonderful rewards scattered all over the place.

Exploration function:-

- Let U^+ denotes the optimistic estimate of the utility of the state s, and let N(a,s) be the number of times action a has been tried in state s.
- Suppose that value iteration is used in an ADP learning agent; then rewrite the update equation to incorporate the optimistic estimate.
- The following equation does this,

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left[\sum_{s`} T(s,a,s`)U^+(s`), N(a,s)\right]$$

- Here f(u ,n) is called the **exploration** function.
- It determines how greed is trade off against curiosity.

- The function f(u, n) should be increasing in u and decreasing in n.
- The simple definition is
  f(u, n) = R^+ in $n<N_c$
  u otherwise
  where R^+ = optimistic estimate of the best possible reward obtainable in any state and N_c is a fixed parameter.
- The fact that U^+ rather than U appears on the right hand side of the above equation is very important.
- If U is used, the more pessimistic utility estimate, then the agent would soon become unwilling to explore further a field.
- The use of U^+ means that benefits of exploration are propagated back from the edges of unexplored regions, so that actions that lead toward unexplored regions are weighted more highly, rather than just actions that are themselves unfamiliar.

Learning an action value function:-

- To construct an active temporal difference learning agent, it needs a change in the passive TD approach.
- The most obvious change that can be made in the passive case is that the agent is no longer equipped with a fixed policy, so if it learns a utility function U, it will need to learn a model in order to be able to choose an action based on U via one step look ahead.
- The update rule of passive TD remains unchanged. This might seem old.
- **Reason:-**
  - Suppose the agent takes a step that normally leads to a good destination, but because of non determinism in the environment the agent ends up in a disastrous state.
  - The TD update rule will take this as seriously as if the outcome had been the normal result of the action, where the agent should not worry about it too much since the outcome was a fluke.
  - It can be shown that the TD algorithm will converge to the same values as ADP as the number of training sequences tends to infinity.

Q-Learning:-

- An alternative TD method called Q-Learning.
- It can be used that learns an action value representation instead of learning utilities.
- The notation Q(a, s) can be used to denote the value of doing action "a" in state "s".
- Q values are directly related to utility values as follows,
  $U(s) = \max_a Q(a, s)$
- Q Learning is called a model free method.
- **Reason:-**
  - It has a very important property: a TD that learns a Q-function does not need a model for either learning or action selection.
  - As with utilities, a constraint equation can be written that must hold at equilibrium when the Q-Values are correct,
    $Q(a, s) = R(s) + \gamma \sum_{s`} T(s, a, s`) \max_{a`} Q(a`, s`)$
  - As in the ADP learning agent, this equation can be used directly as an update equation for an iteration process that calculates exact Q-values, given an estimated model.
  - This does, however, require that a model also be learned because the equation uses T(s, a, Sf).
  - The temporal difference approach, on the other hand, requires no model.
  - The update equation for TD Q-Learning is

$$Q(a, s) \leftarrow Q(a, s) + \alpha[R(s) + \gamma \max_{a`} Q(a`, s`) - Q(a, s)]$$

- Which is calculated whenever action a is executed in state s leading to state Sf.

- The following algorithm shows the Q-Learning agent program

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←null
Else $s,a,r \leftarrow s', \text{argmax}_{a'} f(Q[a',s'],N_{sa}[a',s']),r'$
return a

- Some researchers have claimed that the availability of model free methods such as Q-Learning means that the knowledge based approach is unnecessary.
- But there is some suspicion i.e. as the environment becomes more complex.

Generalization in Reinforcement Learning:-

- The utility function and Q-functions learned by the agents are represented in tabular form with one output value for each input tuple.
- This approach works well for small set spaces.
- **Example:-** The game of chess where the state spaces are of the order 10^{50} states. Visiting all the states to learn the game is tedious.
- One way to handle such problems is to use **FUNCTION APPROXIMATION.**
- **Function approximation** is nothing but using any sort of representation for the function other than the table.
- **For Example:-** The evaluation function for chess is represented as a weighted linear function of set of **features or basic functions f1,….fn**

$$U_\theta(S) = \theta_1 f_1(S) + \theta_2 f_2(S) + \ldots\ldots\ldots + \theta n f_n(S)$$

- The reinforcement learning can learn value for the parameters $\theta = \theta_1 \ldots\ldots\ldots \theta_n$.
- Such that the evaluation function U_θ approximates the true utility function.
- As in all inductive learning, there is a tradeoff between the size of the hypothesis space and the time it takes to learn the function.
- For reinforcement learning, it makes more sense to use an online learning algorithm that updates the parameter after each trial.
- Suppose we run a trial and the total reward obtained starting at (1, 1) is 0.4.
- This suggests that $U_\theta(1,1)$, currently 0.8 is too large and must be reduced.
- The parameter should be adjusted to achieve this. This is done similar to neural network learning where we have an error function which computes the gradient with respect to the parameters.
- If $U_j(S)$ is the observed total reward for state S onward in the jth trial then the error is defined as half the squared difference of the predicted total and the actual total. $E^j(S) = (U(S) - U_j(S))^2 / 2$

- The rate of change of error with respect to each parameter θ_i is $\delta E_j / \delta \theta_j$, so to move the parameter in the direction of the decreasing error.

  $\theta_i \leftarrow \theta_i - \theta(\delta E_j(S) / C\theta_j) = \theta_i + \alpha(U_j(S) - U_\theta(S))(\delta U_\theta(S) / \delta \theta_i)$
- This is called **Widrow-Hoff Rule or Delta Rule.**
- **Advantages:-**
  - It requires less space.
  - Function approximation can also be very helpful for learning a model of the environment.
  - It allows for inductive generalization over input states.
- **Disadvantages:-**
  - The convergence is likely to be displayed.
  - It could fail to be any function in the chosen hypothesis space that approximates the true utility function sufficiently well.
  - Consider the simplest case, which is direct utility estimation. With function approximation, this is an instance of supervised learning.