# 11 Statistical Learning Methods

## Syllabus

*Statistical Learning Methods - Statistical Learning, Learning with Complete Data, Learning with Hidden Variables ; EM Algorithm.*

## Contents


11.1 Introduction

11.2 Neural Networks ..... **Summer-18,19,20,**

..... **Winter-18,19** ..... Marks 7

11.3 Kernel Machine

11.4 University Questions with Answers

## 11.1 Introduction

Uncertain reasoning methods can be used for learning from observations. Agents can handle uncertainity by using the methods of probability and decision theory. For this, first they should learn their probabilistic theories of the world from experience.

### 11.1.1 Statistical Learning

In statistical learning, data are evidence, that is instantiation of some or all of the random variables describing the domain.

The hypothesis are probabilistic theories of how the domain works, including logical theories as a special case.

For example -

Suppose there are five kinds of bags of candies :

10 % are $h_1$ : 100 % cherry candies

20 % are $h_2$ : 75 % cherry candies + 25 % lime candies

40 % are $h_3$ : 50 % cherry candies + 50 % lime candies

20 % are $h_4$ : 25 % cherry candies + 75 % lime candies

10 % are $h_5$ : 100 % lime candies.

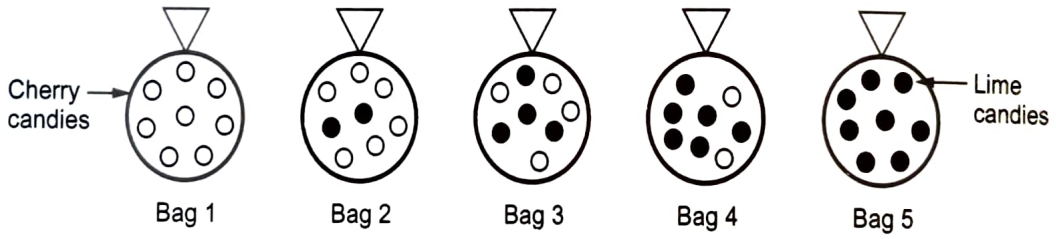



**Fig. 11.1.1 The candy bags example**

Then we observe candies drawn from some bag :

**Fig. 11.1.2 Lime candies drawn from some bag**

What kind of bag is it ? What flavour will the next candy be ?

Despite of its apparent triviality, this scenario serves to introduce many of the major issues. The agent really does need to infer a theory of its world, albeit a very simple one.

## 11.1.2 Bayesian Learning

1) It calculates the probability of each hypotheses, given the data and makes the predictions on that basis.

2) Predictions are made by using all the hypotheses, weighted by their probabilities, rather than a single "best" hypothesis. This way learning is reduced to probabilistic inference.

3) Let D represent all the data with observed value d, then the probability of each hypothesis obtained by Bayes' Rule -

$$P(h_i|d) = \alpha\ P(d\,|h_i)\ P(h_i) \qquad \ldots (11.1.1)$$

If we want to make predicition about an unknown quantity X, then we have,

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

$$= \sum_i P(X|h_i)\ P(h_i|d) \qquad \ldots (11.1.2)$$

Where it is assumed that each hypothesis determines a probability distribution over X.

4) Equation (11.1.2) shows that predictions are weighted averages over the predictions of the individual hypotheses.

5) The important quantities in Bayesian learning are the hypothesis prior - $P(h_i)$ and the likelihood of the data under each hypothesis - $P(d|h_i)$.

6) The basic characteristic of Baysian learning is that "True hypothesis dominates the Bayesian prediction".

7) For any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will eventually vanish, simply because the probability of generating "uncharacteristic" data indefinitely is vanishingly small.

8) The Bayesian prediction is optimal whether the data set be small or large.

9) For real learning problems, the hypothesis space is usually very large or infinite.

10) Approximation in Bayesian learning : -

   i) A prediction can be made on the basis of single most probable hypothesis, $h_i$, that maximizes $P(h_i|d)$. This is called as maximum a posteriori or MAP hypothesis.

   ii) Predictions made according to an MAP hypothesis $h_{map}$ are approximately Bayesian to the extent that $P(X|d) \approx P(X|h_{map})$.

   iii) Finding MAP hypothesis is often much easier than Bayesian learning, because it requires solving an optimization problem instead of a large summation.

iv) Overfitting Trade-offs :

a) Overfitting can occur when the hypothesis space is too expressive, so that it contains many hypotheses that fit the data set well.

b) Rather than placing an arbitrary limit on the hypotheses to be considered, Bayesian and MAP learning methods use the prior to penalize complexity.

c) Typically, more complex hypothesis have a lower prior probability-in part because there are usually many more complex hypotheses than simple hypotheses.

d) On the other hand, more complex hypotheses have a greater capacity to fit the data.

v) Hence, the hypothesis prior embodies a trade-off between the complexity of a hypothesis and its degree of fit to the data.

vi) If H contains only deterministic hypothesis, then in that case, $P(d|h_i)$ is 1 if $h_i$ is consistent and 0 otherwise. Looking at equation (11.1.1) we see that $h_{MAP}$ will then be the simplest logical theory that is consistent with the data. Therefore, maximum a posteriori learning provides a natural embodiment of Ockham's razor.

vii) a) Another trade-off between complexity and degree of fit is obtained by taking the logarithm of equation (11.1.1).

b) Choosing $h_{MAP}$ to maximize $P(d|h_i)\ P(h_i)$ is equivalent to minimizing

$$-\log_2 P(d|h_i) - \log_2 P(h_i)$$

c) Using the connection between information encoding and probability we see that the $-\log_2 P(h_i)$ term equals the number of bits required to specify the hypothesis $h_i$.

d) $\log_2 P(d/h_i)$ is the additional number of bits required to specify the data given the hypothesis.

e) To see this, consider that no bits are required if the hypothesis predicts the data exactly as with $h_5$ and the string of lime candies and $\log_2 1 = 0$.

f) MAP learning is choosing the hypothesis that provides maximum compression of the data.

g) The same task is addressed more directly by the minimum description length or MDL, learning method, which attempts to minimize the size of hypothesis and data encodings rather than work with probabilities.

viii) a) Another approximation is provided by assuming a uniform prior over the space of hypotheses. In that case, MAP learning reduces to choosing an $h_i$ that maximizes $P(d|H_i)$. This is called a **maximum-likelihood (ML)** hypothesis, $h_{ML}$.

b) Maximum-likelihood learning is very common in statistics, a discipline in which many researchers destruct the subjective nature of hypothesis priors. It is reasonable approach when there is no reason to prefer one hypothesis over another a priori. For example, when all hypotheses are equally complex.

c) It provides a good approximation to Bayesian and MAP learning when the data set is large, because the data swamps the prior distribution over hypotheses, but it has problems (all we shall see) with small data sets.

## 11.1.3 Learning with Complete Data

### 11.1.3.1 Maximum-Likelihood Parameter Learning : (Discrete Models)

Statistical learning methods have important task which is parameter learning with complete data. Parameter learning task involves finding the numerical parameters for a probability model whose structure is fixed.

Data is said to be complete when each data points contains values for every variable in the mode.

Consider candy-bag example : -

New manufacturer : Then the lime/Cherry proportions is completely unknown.

Parameter : $\theta \in [0, 1]$ (proportion of cherry)

Hypothesis : $h_\theta$

Assumption : All proportions equally likely a priori.

BN's variables : Flavour $\in$ {Cherry, Lime}

N unwrapped candies : C cherries and $l$ = N – C limes.

Likelihood of this particular data set :

$$P(d|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c (1-\theta)^l$$

- Finding the maximum-likelihood hypothesis $h_{ML}$ is then equivalent to maximising the log-likelihood :

$$L(d|h_\theta) = \log P(d|h_\theta)$$

$$= \sum_{j=1}^{N} \log P(d_j|h_\theta) = c \log \theta + l \log (1-\theta)$$

To that end : 1) differentiate L with respect to $\theta$ and

2) set the resulting expression to 0.

$$\frac{dL(d|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \Rightarrow \theta = \frac{c}{c+l} = \frac{c}{N}$$

- Previous result is obvious, but the process is important : -
  1) Write down the expression for the likelihood of the data as a function of the parameter(s).
  2) Write down the derivative of the log-likelihood with respect to each parameter.
  3) Find the parameter values such that the derivatives are zero.

The last step can be tricky, using iterative solution algorithms or numerical optimisation.

- **Problem with ML learning :**

If some events have never been observed, $h_{ML}$ assigns them 0 probability.

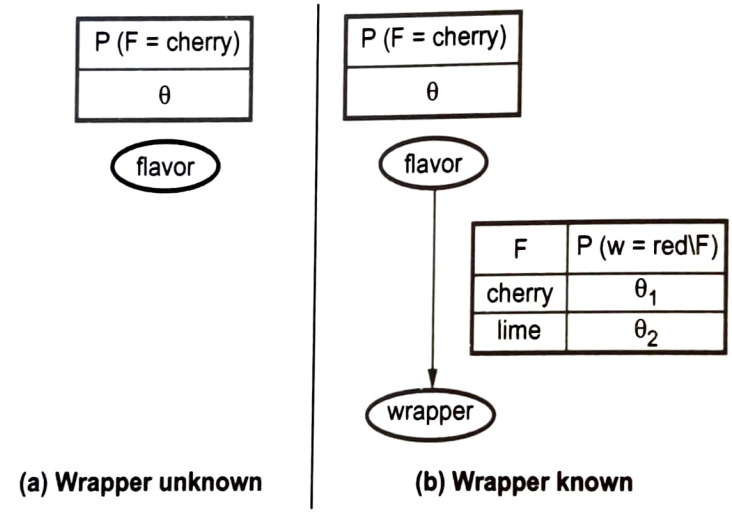



**Fig. 11.1.3 Problem in ML Learning**

New pb(b) : Wrappers red or blue. Colors selected probabilistically (see new BN)

$P(\text{Flavor} = \text{Cherry}, \text{Wrapper} = \text{green} \mid h_{\theta,\theta_1,\theta_2})$

$= P(\text{Flavor} = \text{cherry} \mid h_{\theta,\theta_1,\theta_2})\ P(\text{Wrapper} = \text{green} \mid \text{Flavor} = \text{Cherry}, h_{\theta,\theta_1,\theta_2})$

$= \theta \cdot (1 - \theta_1)$

**Experiment :**

$$N = c + l = (r_c + g_c) + (r_l + g_l)$$

**Likelihood :**

$$P(d|h_{\theta,\theta_1,\theta_2}) = \theta^c (1-\theta)^l \cdot \theta_1^{r_c} (1-\theta_1)^{g_c} \cdot \theta_2^{r_l} (1-\theta_2)^{g_l}$$

**log-likelihood :**

$$L = [c\log\theta + l\log(1-\theta)] + [r_c\log\theta_1 + g_c\log(1-\theta_1)] + [r_l\log\theta_2 + g_l\log(1-\theta_2)]$$

**Derivatives :**

$$\frac{dL}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \qquad \Rightarrow \theta = \frac{c}{c+l}$$

$$\frac{dL}{d\theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \qquad \Rightarrow \theta = \frac{r_c}{r_c + g_c}$$

$$\frac{dL}{d\theta_2} = \frac{r_l}{\theta_2} - \frac{g_l}{1-\theta_2} = 0 \qquad \Rightarrow \theta = \frac{r_l}{r_l + g_l}$$

With complete data, ML parameter learning for a BN decomposes into separate learning problems (one per parameter).

### 11.1.3.2 Naive Bayes Models

1) This is the most common Bayesian network model used in machine learning.
2) In this model, the "class" variable C (which is to be predicted) is the root and the "attribute" variables $X_i$ are the leaves.
3) The model is "naive" because it assumes that the attributes are conditionally independent of each other, given the class.
4) Assuming Boolean variables the parameters are,

$$\theta = P(C = \text{true}), \theta_{i1} = P(X_i = \text{true} \mid C = \text{true}),$$

$$\theta_{i2} = P(X_i = \text{true} \mid C = \text{false})$$

The maximum - likelihood parameter values are found in exactly the same way as shown in Fig. 11.1.3 (b).

5) Once the model has been trained in this way, it can be used to classify new examples for which the class variable C is unobserved. With observed attribute values, $x_1, x_2, \ldots x_n$, the probability of each class is given by,

$$\mathbf{P}(C \mid x_1, x_2, \ldots x_n) = \alpha \; \mathbf{P}(C) \prod_i \mathbf{P}(x_i \mid C)$$

6) A deterministic prediction can be obtained by choosing the most likely class.
7) The method learns fairly well but not as well as decision-tree learning; this is presumably because the true hypothesis - which is a decision tree - is not representable exactly using a naive Bayes model.
8) Naive Bayes learning do well in a wide range of applications; the boosted version is one of the most effective general-purpose learning algorithm.
9) Naive Bayes learning scales well to very large problems : with n Boolean attributes, there are just 2n+1 parameters, and no search is required to find $h_{ML}$, the maximum - likelihood naive Bayes hypothesis.

10) Naive Bayes learning has no difficulty with noisy data and can give probabilistic predictions when appropriate.

### 11.1.3.3 Maximum-Likelihood Parameter Learning : (Continuous Models)

1) Continuous probability model such as the linear-Gaussian model is used for maximum - likelihood parameter learning.

2) Because continuous variables are ubiquitous in real-world applications, it is important to know how to learn continuous models from data.

3) The principles for maximum likelihood learning are identical to those of the discrete case.

4) a) Let us begin with a very simple case : Learning the parametes of a Gaussian density function on a single variable.

b) That is, the data are generated as follows : -

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

The parameters of this model are the mean $\mu$ and the standard deviation $\sigma$. (Notice that the normalizing "constant" depends on $\sigma$, so we cannot ignore it.)

c) Let the observed values be $x_1, \ldots, x_N$. Then the log likelihood is,

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

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

d) Setting the derivatives to zero as usual, we obtain,

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

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

5) The maximum-likelihood value of the mean is the sample average and the maximum-likelihood value of the standard deviation is the square root of the sample variance. Again, these are comforting results that confirm "commonsense" practice.

6) a) Now consider a linear Gaussian model with one continuous parent X and a continuous child Y.

b) Y has a Gaussian distribution whose mean depends linearly on the value of X and whose standard deviation is fixed.

c) To learn the conditional distribution $P(Y|X)$, we can maximize the conditional likelihood.

$$P(y|x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}}$$

d) Here, the parameters are $\theta_1$, $\theta_2$ and $\sigma$. The data are a collection of $(x_j, y_j)$ pairs, as shown in Fig. 11.1.4.

e) Using the usual methods, we can find the maximum-likelihood values of the parameters.

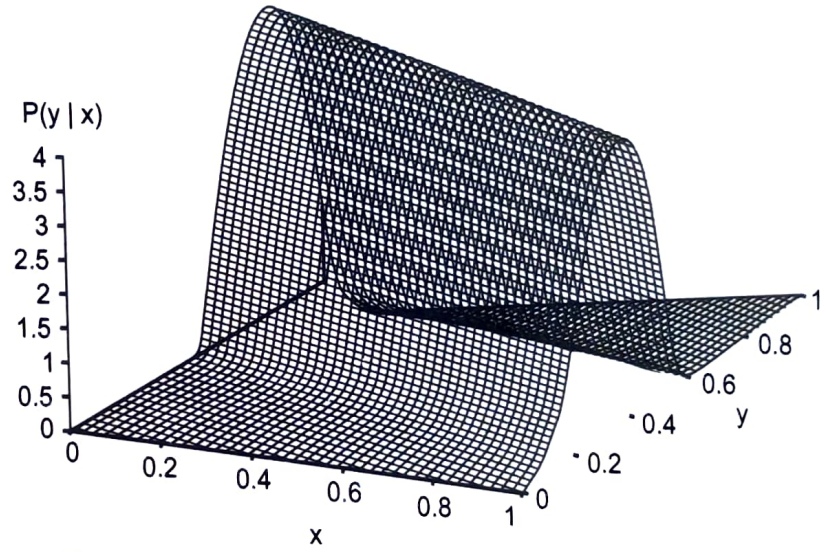



**Fig. 11.1.4 (a) A linear Gaussian model described as $y = \theta_1 x + \theta_2$ plus Gaussian noise with fixed variance**

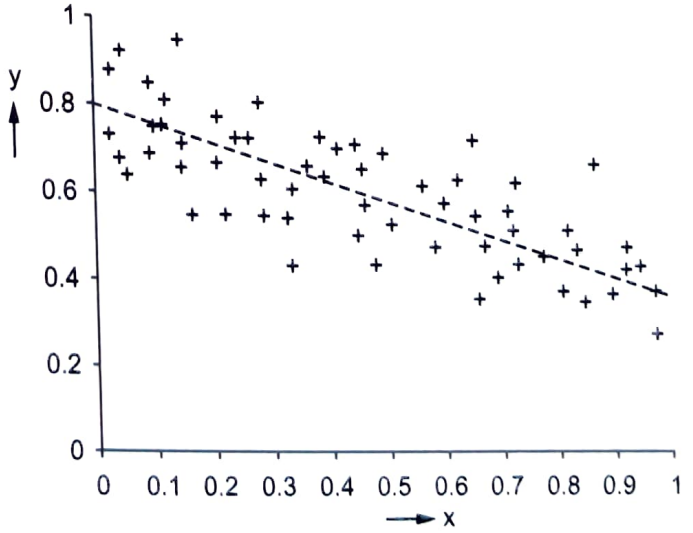



**Fig. 11.1.4 (b) A set of 50 data points generated from model in Fig. 11.1.4 (a)**

f) Here we want to make a different point. If we consider just the parameters $\theta_1$ and $\theta_2$ that define the linear relationship between x and y, it becomes clear that maximizing the log-likelihood with respect to these parameters is the same as minimizing the numerator in the exponent of above equation.

$$E = \sum_{j=1}^{N} (y_j - (\theta_1 x_j + \theta_2))^2$$

g) The quantity $(y_j - (\theta_1 x_j + \theta_2))$ is the error for $(x_j, y_j)$ that is, the difference between the actual value $y_i$ and the predicted value $(\theta_1 x_j + \theta_2)$. So E is the well-known sum of squared errors.

h) This is the quantity that is minimized by the standard linear regression procedure.

i) Minimizing the sum of squared errors gives the maximum - likelihood straight line model, provided that the data are generated with Gaussian noise of fixed variance.

### 11.1.3.4 Bayesian Parameter Learning

ML learning is simple, but not appropriate for small data sets.

Example -

If only cherries have been observed, $h_{ML} \rightarrow \theta = 1.0$

- Bayesian Approach :
  - It uses hypothesis prior over possible values of the parameters.
  - Update of this distribution is used as the data arrives.
- Candy example with Bayesian view : -
  - $\theta$ : unknown value of a variable $\Theta$.
  - Hypothesis prior : P ($\Theta$). (Continuous over [0, 1] and integrating to 1).

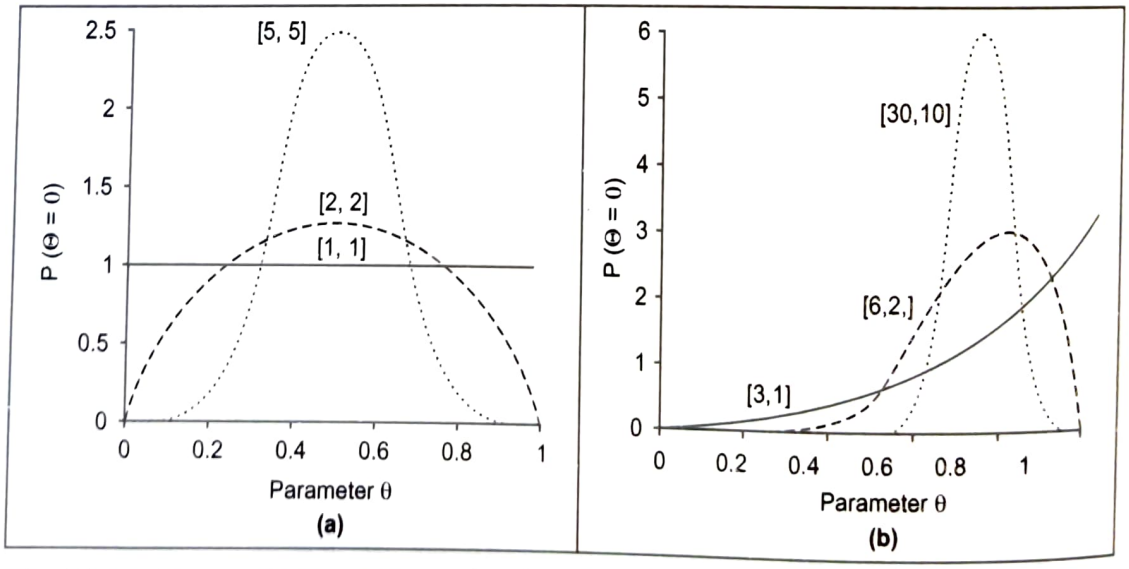



**Fig. 11.1.5 (a) and (b) Examples of the beta [a, b] distribution for different values of [a, b]**

- Candidates :
  beta distributions, defined by 2 hyperparameters a and b
  such that :
  beta [a, b] $(\theta) = \alpha\, \theta^{a-1}\, (1-\theta)^{b-1}$

- Nice property of the beta family :

If $\Theta$ has prior beta [a, b] and a data point is observed, then the posterior for $\Theta$ is also a beta distribution.

**Beta family :**

A beta family is called as the **conjugate prior** for the family of distributions for a Boolean variable.

$$P(\theta \mid D_1 = \text{Cherry}) = \alpha\, P(D_1 = \text{Cherry} \mid \theta)\, P(\theta)$$

$$= \alpha' \theta \cdot \text{beta}\,[a, b]\,(\theta) = \alpha'\, \theta \cdot \theta^{a-1} (1-\theta)^{b-1}$$

$$= \alpha'\, \theta^{a}\, (1-\theta)^{b-1} = \text{beta}\,[a+1, b]\,(\theta)$$

**Note :** a and b are virtual counts (starting with beta [1, 1]).

With wrappers : 3 parameters. It need to specify $P(\theta, \theta_1, \theta_2)$.

Assuming parameter independence : $P(\theta, \theta_1, \theta_2) = P(\theta)\, P(\theta_1)\, P(\theta_2)$

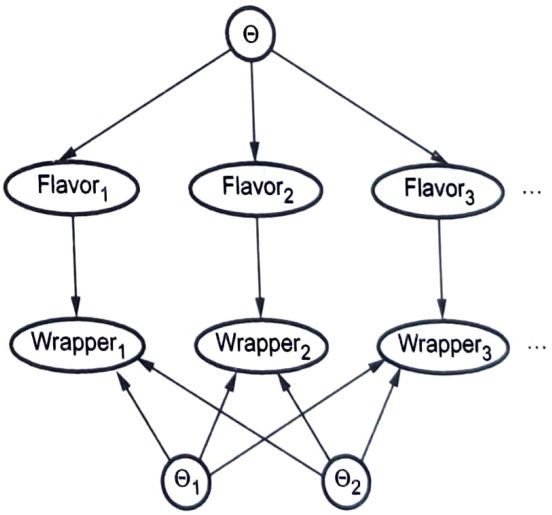



**Fig. 11.1.6 A Bayesian network that corresponds to a Bayesian learning process. Posterior distributions for the parameter variables $\Theta$, $\Theta_1$, $\Theta_2$ can be inferred from their prior distributions and the evidence in the flavour$_i$, wrapper$_i$ variables**

- **View of Bayesian learning as inference in a BN.**

**Variables :** Unknown parameters + variables describing each instance.

$$P(\text{Flavor}_i = \text{Cherry} \mid \Theta = \theta) = \theta$$

$$P(\text{Wrapper}_i = \text{red} \mid \text{Flavor}_i = \text{Cherry}, \Theta_1 = \theta_1) = \theta_1$$

#### 11.1.3.5 Learning Bayes Net Structures

Often the Bayes net structures are easy to get from expert knowledge. But sometimes the causality relationships are debatable.

For example :

Smoking ⇒ Cancer ?

too Much TV ⇒ Bad at school ?

- To search for a good model :
  - Start with a linkless model and add parents to each node, then learn parameters and measure accuracy of the resulting model.
  - Start with an initial guess of the structure, and use hill-climbing or simulated annealing to make modifications (reversing, adding, or deleting arcs).

**Note :** To avoid cycles, many algorithms use an ordering over nodes to constrain arcs' orientations.

- Two ways for deciding when a good structure has been found : -
  1) Test whether the conditional independence assertions implicit in the structure, are satisfied in the data.

     $$P(Fri \mid Sat, Bar \mid WillWait) = P(Fri \mid Sat \mid WillWait)\ P(Bar \mid WillWait)$$

     It requires an appropriate statistical test (with appropriate threshold).
  2) Measure the degree to which the proposed model explains the data. The ML hypothesis would give a fully connected network.
- Here we need to penalize complexity : -
  a) MAP (MDL) approach : Substracts a penalty from the likelihood of each structure.
  b) Bayesian approach : places a joint prior over structures and parameters.

     It too many structures are there then use sampling rather than exact methods.

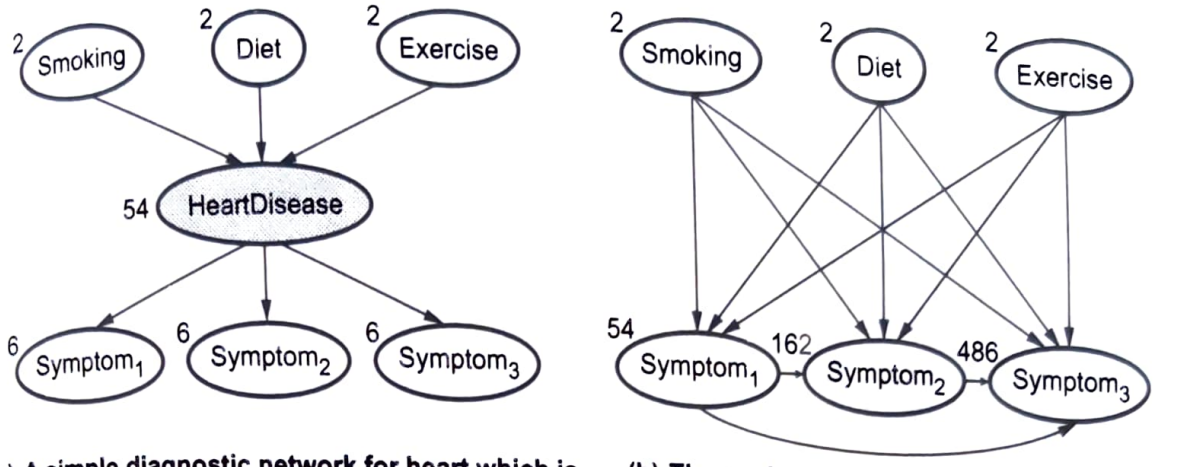
## 11.1.4 Learning with Hidden Variables

### 11.1.4.1 Introduction

Many real world problems have hidden (or latent) variables. There is no training data variable. The model cannot be built without training data and inference can not be done without model. So we have a problem in hand that-How to learn models with hidden variables ?

For example :

**Example :** Medical records include observed symptoms, treatment applied and outcome of the treatment, but rarely the disease.




(a) A simple diagnostic network for heart which is assumed to be a hidden variable. Each variable has three possible values. Each veriable is labeled with the number of independent parameters in its conditional distribution; the total number is 78

(b) The equivalent network with heart disease removed. Notice that symptom variables are no longer conditionally independent given their parents. This network requires 708 parameters

**Fig. 11.1.7**

**Question :** Why include the disease in the model ?

- Each node has 3 values : None, moderate, severe.
- Model with hidden variable has 78 parameters.
- Model without hidden variable has 708 parameters.
- Hidden variables allow simpler models.

### 11.1.4.2 Unsupervised Clusering : Learning Mixtures of Gaussians

Unsupervised clustering is the problem of discerning multiple categories in a collection of objects.

It is "unsupervised" because no "category" label is given.

Example :

- "Red giant" or "white dwarfs" are categories created according to characteristics of stars (spectrum, size ... ).
- **Linnaean** taxonomy of organisms : species, genera, orders ...
- **Probabilistic mixture models :**
  - Consider a mixture model of the form

$$P(X|\pi,\theta) = \sum_{h=1}^{k} \Pi_h P(X \mid \theta_h)$$

a) $P(X \mid \theta_h)$ is the $h^{th}$ mixing component.

b) $\Pi_h$ is the mixing weight.

**– Mixing component :**

a) $P(X \mid \theta_h)$ is a density function with parameter $\theta_h$.

b) Example : Gaussians, multinomials, Bernoulli.

c) Each component corresponds to one cluster.

**– Mixing weight :**

a) Forms a probability distribution over components, $\sum_h \Pi_h = 1$.

b) Relative proportion of each cluster.

- **The mixture model learning problem : -**

– Given : Data $\chi = \{X_1, \ldots, X_n\}$

– Assume :

a) $\chi$ is generated by a mixture model.

b) k is the number of components in the mixture model.

c) $P(X \mid \theta_h)$ is from a known (exponential) family.

– A point X is generated as follows :

a) Sample component h with probability $\Pi_h$.

b) Sample X with probability $P(X \mid \theta_h)$

– Problem : Find $(\pi, \theta)$ that maximizes

$$P(\chi \mid \pi, \theta) = \prod_{i=1}^{n} P(X_i \mid \pi, \theta) = \sum_{i=1}^{n} \log P(X_i \mid \pi, \theta)$$

– $(\pi^*, \theta^*)$ best explains the observed data.

This can be used as a model for the data domain.

- **EM algorithm for mixture models :**
  - Assume $\chi$ is generated from a mixture of Gaussians.
  - Each point X generated from one component.
    a) It is unknown that which component generated X.
    b) It is unknown what the $(\mu_h, \Sigma_h)$ of each component is
  - If component assignments were known,
    a) It can estimate $(\mu_h, \Sigma_h)$ using ML estimates.
    b) It can estimate $\Pi_h$ using ML estimate.
  - If parameters $\{\Pi_h, (\mu_h, \Sigma_h)\}$ were known

a) It can estimate (probabilistic) component assignments $P(h|x)$.

- Learning mixture models

a) Pretend component parameters $\{\Pi_h, (\mu_h, \Sigma_h)\}$ are known.

b) Estimate component assignments $P(h|x)$ for every point.

c) Estimate parameters $\{\Pi_h, (\mu_h, \Sigma_h)\}$ based on current assignments.

d) Repeat till convergence.

- **EM Algorithm :**
  - **Initialize :** component parameters $\{(\Pi_h, (\mu_h, \Sigma_h))\}$
  - **E-step :** compute $P(h|x_i)$

    Bayes Rule, $P(h|x_i) = P_{hi} = \propto \Pi_h P(x_i/\mu_h, \Sigma_h)$
  - **M-step :** compute component parameters

$$P_h = \sum_i P_{hi}$$

$$\Pi_h = \frac{P_h}{\sum_{h'} P_{h'}}$$

$$\mu_h = \sum_i P_{hi} X_i / P_h$$

$$\sum_h = \sum_i P_{hi} (X_i - \mu_h) (X_i - \mu_h)^T / P_h$$

  - **Increase log-likelihood** at every iteration.
  - **It converges to local minimum** (mostly).
  - **Issues :**

    a) Degenerate solutions can give high likelihood.

    b) Local minima (or saddle point) may be very bad.

- **Learning mixtures of Gaussians : -**

The question is that what probability distribution has generated the data ?

The answer is,

Clustering → mixture distribution P : k components, each being a distribution.

$$P(X) = \sum_{i=1}^{k} P(C = i) P(X | C = i)$$

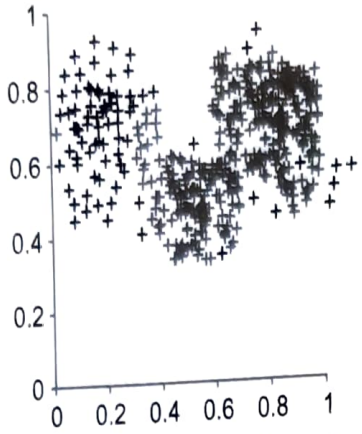
Data point generation : -

1) Choose a component, then

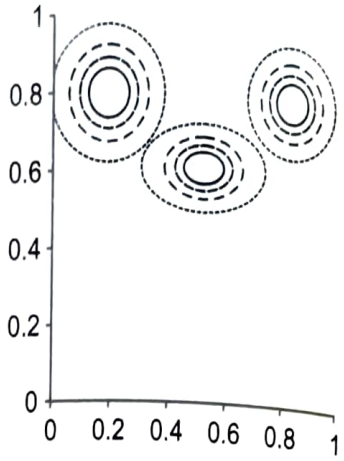
2) Sample from that component.

For continuous data : a multivariate Gaussian for each component is used (mixture of Gaussians family).

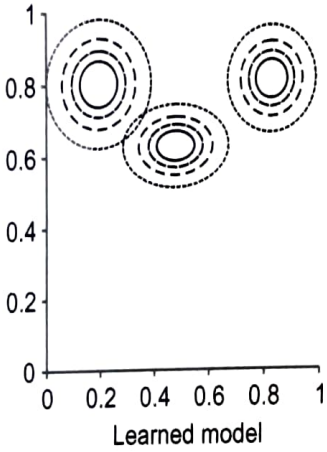
- (Original mixture of Gaussians used for generating the 500 points).




(a) 500 datapoints in two dimensions, suggesting the presence of three clusters




(b) A Gaussian mixture model with three components. The weigths (left to right) are 0.2, 0.3 and 0.5. The data in 11.1.8 (a) were generated from this




(c) The model reconstructed by EM from the data in 11.1.8 (b)

**Fig. 11.1.8**

Parameters (for each component) :

$$w_i = P(C = i), \mu_i, \Sigma_i$$

**Problems in learning :**

- We do not know from which component each data point comes.
- With the parameters, we could compute $P(C \mid X)$.

Basic idea of EM, is iterate until convergence :

- Pretend we know the parameters.
- Infer $P(C \mid X)$ for each data point.
- Refit each component i to data. Points (weighted by $P(C = i \mid X)$)

- **For mixtures of Gaussians :**

**1) E-step : Compute**

$$P_{ij} = P(C = i \mid X_j)$$

$$= \alpha \, P(X_j \mid C = i) \; P(C = i)$$

Define, $P_i = \Sigma_i \, P_{ij}$

**2) M-step :**

$$\vec{\mu}_i \leftarrow \frac{1}{P_i} \sum_j P_{ij} \, X_j$$

$$\Sigma_i \leftarrow \frac{1}{P_i} \sum_j P_{ij} \, (X_j - \vec{\mu}_i)(X_j - \vec{\mu}_i)^T$$

$$w_i \leftarrow P_i$$

- E-step computes the expected values $P_{ij}$ of hidden indicator variables.
- M-step finds parameter values maximizing the log-like lihood of the data.

$Z_{ij}$ (= 1 if datum $X_i$ generated by $i^{th}$ component)

- Comments :
  - Log-likelihood of learned model exceeds that of original model !
  - EM increases the log-likelihood at each iteration. (This fact can be proved in general). In certain conditions, EM can be proven to reach a local maximum.
  - EM resembles a gradient-based hill-climbing (with no step size).
- Bad Behaviours :
  - A Gaussian component covers only one data point.

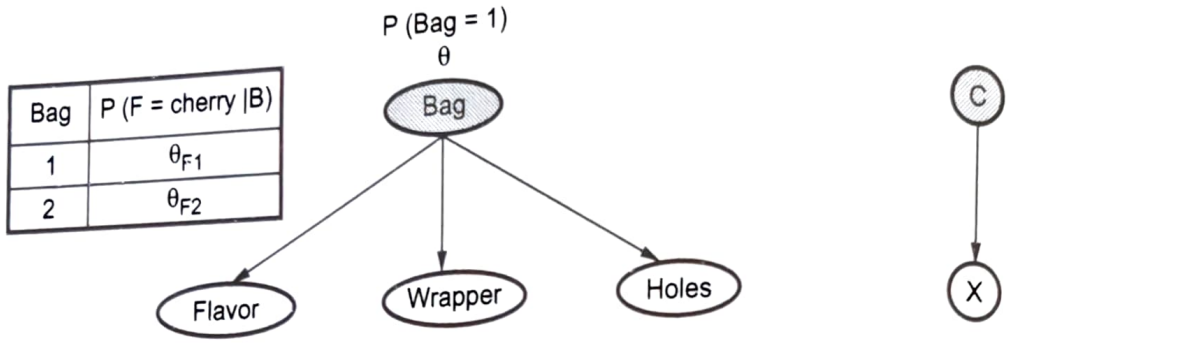
    (Varience $\rightarrow$ 0, likelihood $\rightarrow \infty$)
  - 2 components are merged.

    (same data points, same mean and varience).
- Solutions :
  - MAP version of EM (with priors on parameters), or
  - restart a component with random parameters.
- About initialisation :
  - Reasonable values, and
  - Random ones components have to be distinct !!!).

### 11.1.4.3 Learning BN with Hidden Variables

**Problem :**

- 2 bags of candies mixed together.
- Candies have 3 features : Flavour, Wrapper, Hole (see below Fig. 11.1.9).
- Distribution in each bag : Naive Bayes model.




| Bag | P (F = cherry \|B) |
|---|---|
| 1 | $\theta_{F1}$ |
| 2 | $\theta_{F2}$ |

**(a) A mixture model for candy. The proportions different flavours, wrappers and numbers of holes depend on the bag, which is not observed**

**(b) Bayesian network for a gaussian mixture. The mean and covariance of the observable variables X depend on the component C.**

**Fig. 11.1.9**

Can we recover composition of the 2 bags ?

Let's apply EM.

1000 data samples generated with parameters : -

$$\theta = 0.5, \theta_{F1} = \theta_{W1} = \theta_{H1} = 0.8, \theta_{F2} = \theta_{W2} = \theta_{H2} = 0.3$$

| | W = Red | | W = Green | |
|---|---|---|---|---|
| | H = 1 | H = 0 | H = 1 | H = 0 |
| F = Cherry | 273 | 93 | 104 | 90 |
| F = Lime | 79 | 100 | 94 | 167 |

Initially : $\theta^{(0)} = 0.6$,

$$\theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6,$$

$$\theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4$$

For $\theta$ : Cannot count candies from bags 1 and 2 $\rightarrow$ expected counts :

$$\theta^{(1)} = \frac{\hat{N}\,(\text{Bag} = 1)}{N}$$

$$= \frac{1}{N} \sum_{j=1}^{N} P(\text{Bag} = 1 \mid \text{Flavor}_j, \text{Wrapper}_j, \text{holes}_j)$$

$$= \frac{1}{N} \sum_{j=1}^{N} \frac{P(f_j \mid \text{Bag}=1)\, P(W_j \mid \text{Bag}=1)\, P(h_j \mid \text{Bag}=1)\, P(\text{Bag}=1)}{\sum_i P(f_j \mid \text{Bag}=i)\, P(W_j \mid \text{Bag}=i)\, P(h_j \mid \text{Bag}=i)\, P(\text{Bag}=i)}$$

With the 273 red-wrapped cherry candies :

$$\frac{273}{1000} \cdot \frac{\theta^{(0)}_{F1}\, \theta^{(0)}_{W1}\, \theta^{(0)}_{H1}\, \theta^{(0)}}{\theta^{(0)}_{F1}\, \theta^{(0)}_{W1}\, \theta^{(0)}_{H1}\, \theta^{(0)} + \theta^{(0)}_{F2}\, \theta^{(0)}_{W2}\, \theta^{(0)}_{H2}\, (1 - \theta^{(0)})} \simeq 0.22797$$

$$\theta^{(1)} = 0.6124$$

For other parameters, as $\theta_{F1}$, the expected count of cherry candies from bag 1 is :

$$\sum_{j:\, \text{Flavor}_j = \text{Cherry}} P(\text{Bag} = 1 \mid \text{Flavor}_j = \text{Cherry}, \text{wrapper}_j, \text{holes}_j)$$

Again, with a Bayes Net algorithm, we get :

$$\theta^{(1)} = 0.6124$$

$$\theta^{(1)}_{F1} = 0.6648 \qquad \theta^{(1)}_{F2} = 0.3886$$

$$\theta^{(1)}_{W1} = 0.6483 \qquad \theta^{(1)}_{W2} = 0.3817$$

$$\theta^{(1)}_{H1} = 0.6558 \qquad \theta^{(1)}_{H2} = 0.3827$$

Log-likelihood increases from – 2044 to – 2021 (likelihood $\times 10^{10}$)

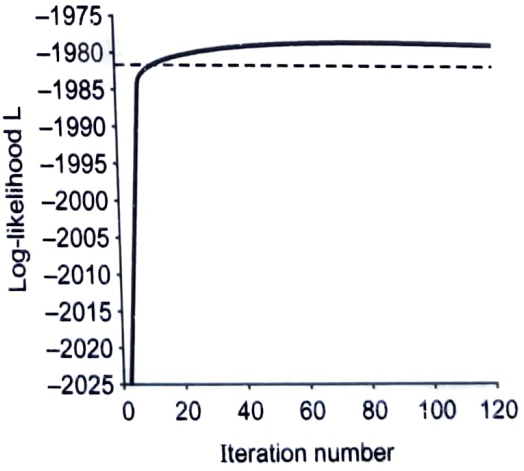



**Fig. 11.1.10 Graph showing the log-likelihood of the data L, as a function of them iteration. Graph for the Bayesian network in Fig. 11.1.9 (a)**

- After 10 iterations :
  - Better likelihood than original model.
  - Very slow progress.

    (gradient-based algorithm to speed-up learning).
- Bayesian network learning with hidden variables : -
  - Repeat : do inference, update parameters (based on inference).
  - Only local posterior probabilities are needed for each parameter.

**General case :**

$$\theta_{ijk} = P(X_i = X_{ij} \mid P_{a_i} = Pa_{ik})$$

$$\leftarrow \hat{N}(X_i = X_{ij}, P_{a_i} = pa_{ik}) \mid \hat{N}(P_{a_i} = pa_{ik})$$

- Learning HMMs :
  - Unrolled dynamic Bayesian network.
  - Estimate transition probabilities from observation sequences.
  - The model does not change $\theta_{ijt} = \theta_{ij}$.
  - Initialize transition probabilities $\theta_{ij}$.
  - Estimate the number of times a transition happens.
- E-step, can be solved by inference.
  - Estimate the parameters of the HMM.
- M-step, simple ML estimates.
  - Baum-Welch Algorithm

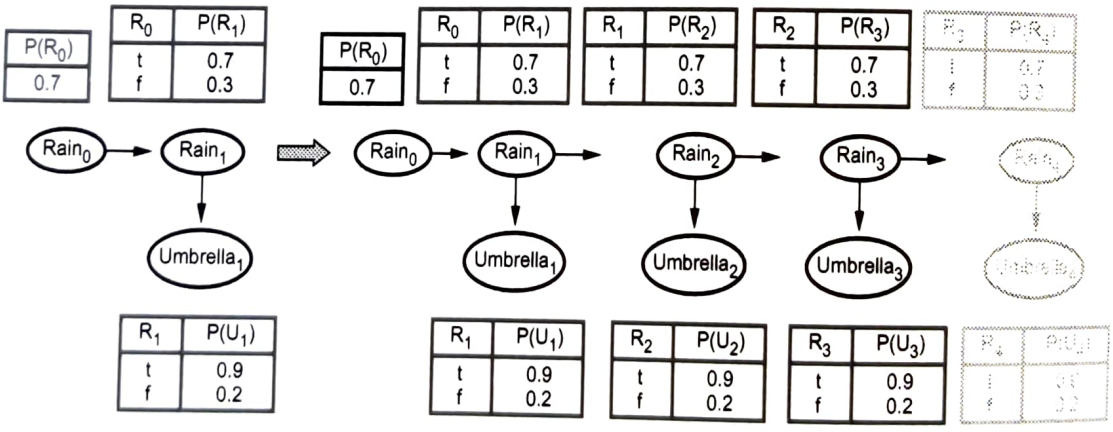



**Fig. 11.1.11 An unrolled dynamic Bayesian network that represents a hidden Markov model**

### 11.1.4.4 General Form of EM Algorithm

- Compute expected values of hidden variables.
- Recompute parameters.
- Model has known and hidden components (X, Z).
- The log-likelihood of the complete model L(X, Z).
  - But Z is a random variable.
  - For a given $\theta^{(t)}, P(Z = z|x, \theta^{(t)})$ is known.

With observed variable x, hidden variables Z and parameters $\vec{\theta}$, EM algorithm computes,

$$\theta^{(t+1)} = \text{argmax}_\theta \sum_z P(Z = z|x,\theta^{(t)}) \; L(x, Z = z|\theta)$$

E-step : Summation (computes the expected log-likelihood of the "completed" data)

M-step : Maximisation (of this log-likelihood).

[**Note :** Important step : Identifying hidden variables.]

Computing the expectations :

- May very straight forward, example - Mixture models.
- May need exact inference algorithms, example - Bayes nets.
- May need approximate inference such as Gibbs sampling.

- Learning BN with hidden variables :

Here there are 2 combined difficulties,

1) Simple case : List of hidden variables known.

   Example - learn the structure knowing that HeartDisease exists.

2) With no or incomplete knowledge of hidden variables : One has to learn structure with possibility to add or remove variables.

**Note :** When inventing variables, the algorithm does not know what it represents (can not call it HeartDisease ...)

But a human expert may be able to interpret the network and identify variables.

As with full observability, a complexity penalty should be used (to avoid fully connected BNs).

Upto now, process with two loops : -

Outer loop : Structural search.

Inner loop : EM (+ gradient ?), including NP-hard problem of computing posteriors in a BN.

A more practical approach is structural EM : -

EM with possibility to update structure as well as parameters.

## 11.2 Neural Networks

GTU : Summer-18,19,20, Winter-18,19

### Introduction

A neuron is a cell in the brain whose principle function is the collection, processing and dissemination of electrical signals.

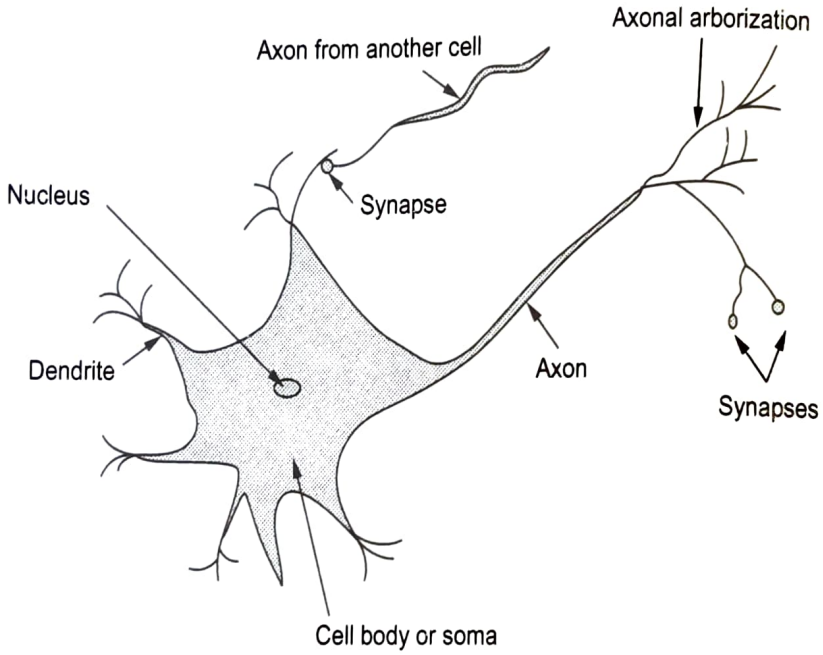



**Fig. 11.2.1 Schematic diagram of a typical neuron**

**Note** *[The brain has $10^{11}$ neurons of greater than 20 types, $10^{14}$ synapses, 1 ms - 10 ms cycle time. The signals are noisy "spike trains" of electrical potential. This method works for learning. The models have been developed based on neurons.]*

The brain's information-processing capacity is thought to emerge primarily from networks of such neurons. For this reason, some of the earliest AI work aimed to create artificial neural networks.

- McCulloch - Pitts (1943)

It is the basis of neural networks which is mathematical model of neuron as shown in following Fig. 11.2.2.

Roughly speaking, it "fires" when a linear combination of its inputs exceeds some threshold.

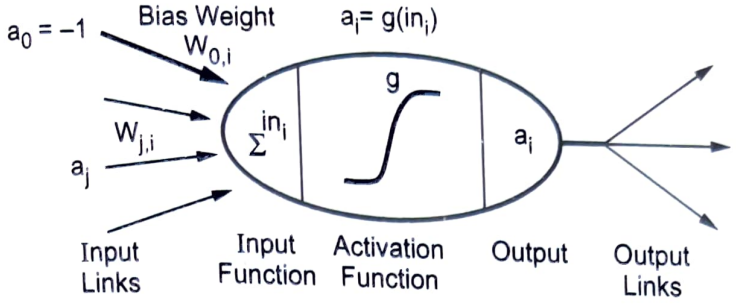



**Fig. 11.2.2 Mathematical model of Neuron**

The output is a "squashed" linear function (mostly weighted sum) of the inputs. A gross oversimplification of real neurons, but its purpose is to develope understanding of what networks of simple units can do.

- **Various parts in neural networks : -**
  - i) Neural networks are composed of nodes or units connected by directed links.
  - ii) A link from unit j to unit i serves to propagate the activation $a_j$ from j to i.
  - iii) Each link also has a numeric weight $W_{j,i}$ associated with it, which determines the strength and sign of the connection.
  - iv) Each unit i first computes a weighted sum of its inputs :

$$in_i = \sum_{j=0}^{n} W_{j,i}\, a_j$$

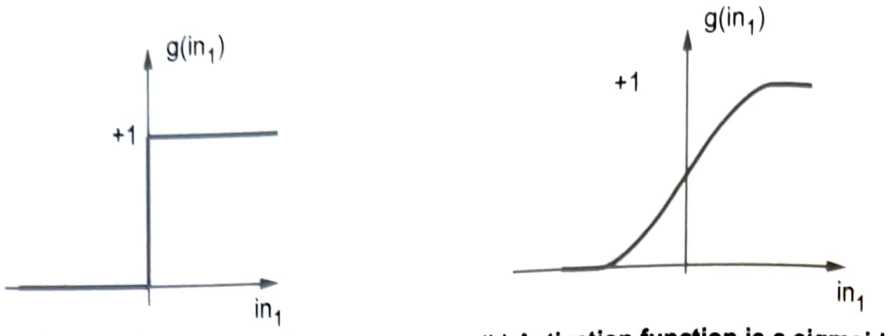
  - v) Then it applies an activation function g to this sum to derive the output :

$$a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i}\, a_j\right)$$

  - vi) Notice that we have included a bias weight $W_{0,i}$ connected to a fixed input $a_0 = -1$.

- Activation function :

The activation function g is designed to meet two things. First, we want the unit to be "active" (near + 1) when the "right" inputs are given and "inactive" (near 0) when the "wrong" inputs are given. Second, the activation needs to be nonlinear, otherwise the entire neural network collapse into a simple linear function.




(a) Activation function is a step function or threshold function.

(b) Activation function is a sigmoid function $1/(1 + e^{-x})$.

**Fig. 11.2.3 Activation function example**

Changing the bias weight $W_{0,i}$ moves the threshold location.

- Implementing logical functions :

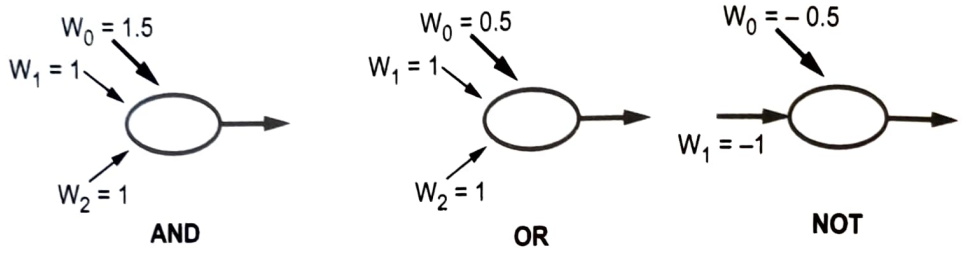



**Fig. 11.2.4 Logical functions implementations**

McCulloch and Pitts original motivation is that, every Boolean function can be implemented.

- **Network structure :**

1) Feed-forward networks :

These Acyclic networks, called as feed-forward networks implement functions and they have no internal state. There can be, single-layer perceptrons or multi-layer perceptrons feed-forward networks.

2) Recurrent networks :

Recurrent neural nets have directed cycles with delays.

This have internal state (like flip-flops), they can oscillate, reach stability, or even expose chaotic behaviour, etc.

Examples :

i) Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$).

ii) $g(x) = \text{sign}(X)$, $a_i = \pm 1$, holographic associative memory.

iii) Boltzmann machines use stochastic activation functions.

- **Feed-forward example :**

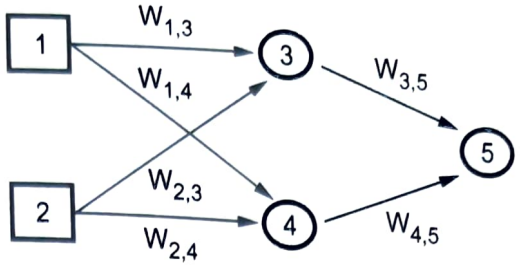



**Fig. 11.2.5 Feed forward network [Note : Bias is ignored in this example]**

- Feed-forward network = A parameterized family of nonlinear functions : -

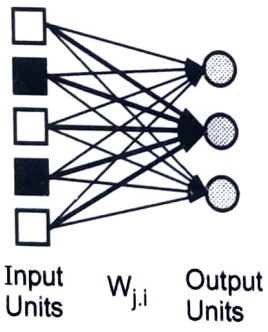
$$a_5 = g(W_{3,5}\, a_3 + W_{4,5}\, a_4)$$

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

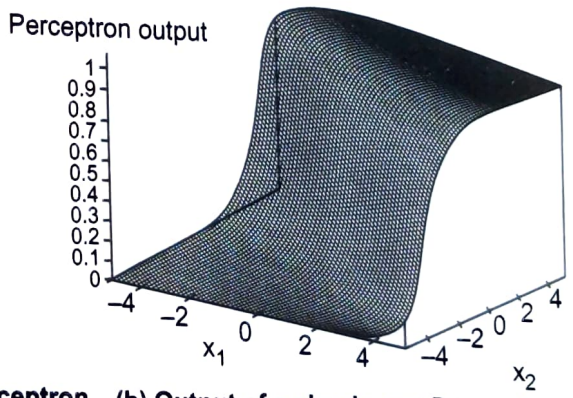
- Adjusting the weights changes the function.
- We can simulate learning by stepwise adjusting the weights.

- **Single-layer perceptrons :**

Feed-forward networks are normally arranged in layers. The simplest one having only one layer, no hidden layers is called as perceptron network.




**(a) A perceptron network consisting of three perceptron output units that share five inputs. Note that, in second output unit, weights on its incoming links have no effects on the other output units.**




**(b) Output of a simple one-Perception unit with two input value**

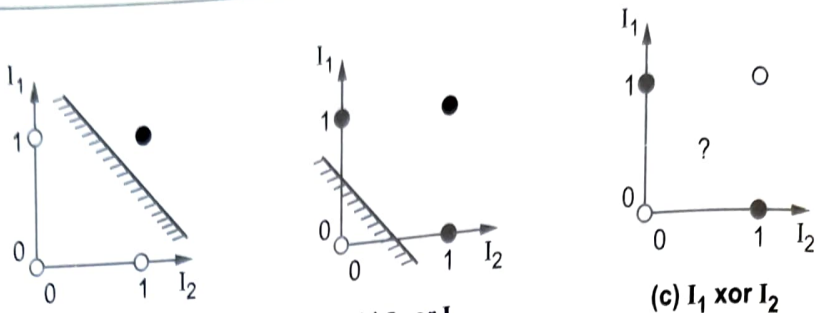
**Fig. 11.2.6 Single layer perceptrons**

The output units all operate separately and there are no shared weights.

Adjusting weights moves the location, orientation and steepness of cliff.

- **Expressivenes of perceptrons :**

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960). It can represent AND, OR, NOT, majority, etc. but not XOR. It represents a linear separator in input space.




(a) $I_1$ and $I_2$ (b) $I_1$ or $I_2$ (c) $I_1$ xor $I_2$

**Fig. 11.2.7 Linear separability in threshold perceptrons. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The perceptron returns 1 on the region on the non-shaded side of the line. In (c) no such line exists that correctly classifies the inputs**

The insufficiency of perceptrons, pointed out by Minsky and Papert (1969).

- **Perceptron learning :**

Learn by adjusting weights to minimize squared error on training set. The squared error for an example with input x and true output y is,

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - hw(x))^2$$

Perform optimization search by gradient descent of E (the gradient of a vector gives the direction of the steepest descent).

For us this means, we have to calculate for each weight,

$$\frac{\delta E}{\delta W_j} = Err \times \frac{\delta Err}{\delta W_j}$$

$$= Err \times \frac{\delta}{\delta W_j} g\left( y - \sum_{j=0}^{n} W_j x_j \right)$$

$$= - Err \times g'(in) \times x_j$$

Here, g′ is the derivation of the activation function, example - for sigmoid,

$g' = g(1 - g)$. This leads to a simple weight update rule.

$$W_j \leftarrow W_j + \propto \times Err \times g'(in) \times x_j$$

Example :

Positive error → Increase network

Output → Increase weights on positive inputs, decrease on negative inputs.

- The perceptrons learning algorithm :

**Function PERCEPTRONS-LEARNING (examples, network) returns a perceptrons hypothesis**

Inputs : Examples, a set of examples, each with input $X = x_1, \ldots, x_n$ and output y.

Network, a perceptrons with weights $W_j$, $j = 0 \ldots n$ and activation function g

Repeat.

For each e in examples do.

$$in \leftarrow \sum_{j=0}^{n} W_j x_j[e]$$

$$Err \leftarrow y[e] - g(in)$$

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$$

until some stopping criterion is satisfied.

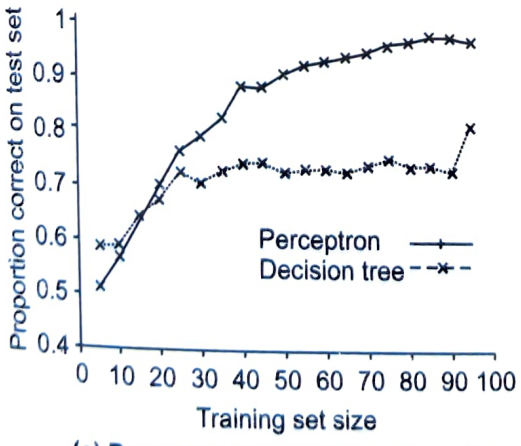
return NEURAL-NET-HYPOTHESIS (Network).

**Remarks on perceptrons learning algorithm :-**

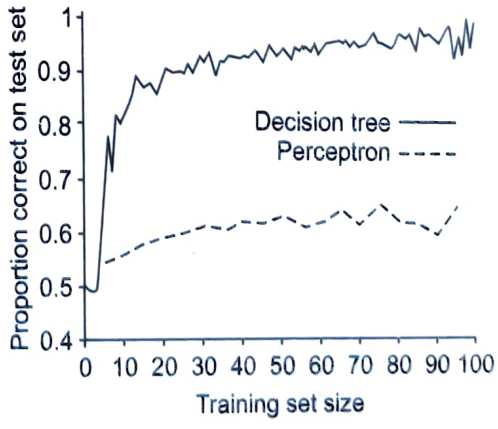
- For threshold functions (where g′ cannot be calculated), g′ is simply committed. Since g′ is the same for all weight, it only changes the magnitude but not the direction of the update for each example.
- Each cycle through the examples is called an epoch.
- The stop criterion is typically that the weight changes become very small.

- **Perceptron learning (diagramatic representation)**

Perceptron learning rule converges to a consistent function for any linearly separable data set.




(a) Perceptrons are better at learning the majority function of 11 inputs




(b) Decision trees are better at learning the will wait predicate in the restaurant example

**Fig. 11.2.8 Comparing the performance of perceptrons and decision trees**

- Perceptron learns majority function easily, DTL is hopeless.
- DTL learns restaurant function easily, perceptron cannot represent it.

(recall : Only linearly separable functions are learnable).

- **How to overcome the limitation of linearly separable functions ?**

**Solution :** Multi-layer perceptron networks !

- **Multi-layer perceptrons :**
  - Layers are usually fully connected.
  - Numbers of hidden units typically chosen by hand.

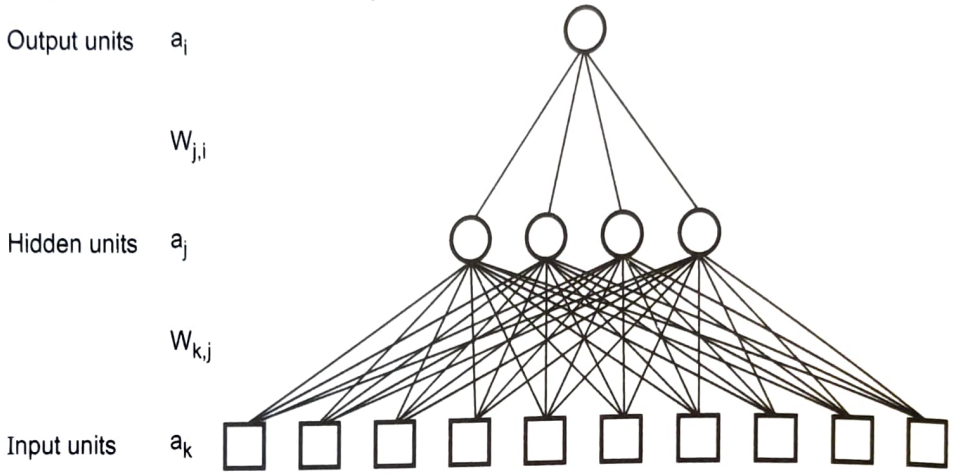
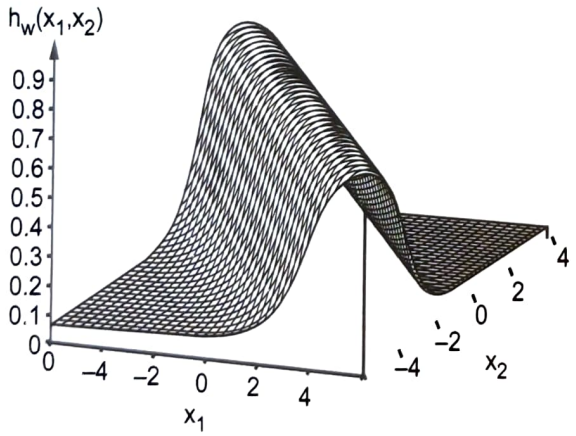


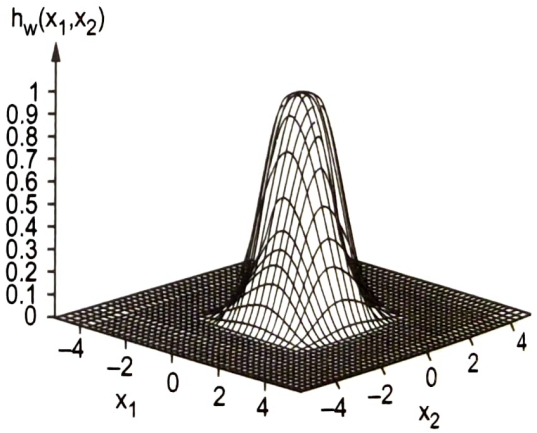

**Fig. 11.2.9 Multilayer percetrons network**

- Expressiveness of MLPs :

All continuous functions with 2 layers, all functions with 3 layers.




**(a) The result of combining two opposite - facing soft threshold functions to produce a ridge.**




**(b) The result of combining two ridges to produce a pump**

**Fig. 11.2.10**

- Combine two opposite-facing threshold functions to make a ridge.
- Combine two perpendicular ridges to make a bump.
- Add bumps of various sizes and locations to fit any surface.

**Problem :** Requires exponentially (over the input) many hidden units. In general, Example for all boolean functions : $2^n/n$ units are needed.

- **Back-propagation learning :**

Here output layer is same as for single-layer perceptron.

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where, $\Delta_i = Err_i \times g'(in_i)$

**Hidden layer : Back-propagate the error from the output layer.**

$$\Delta_j = g'(in\, j) \sum_i W_{j,i}\, \Delta_i$$

**Update rule for weights in hidden layers.**

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

(**Remark :** Most neuroscientists deny that back-propagation occurs in the brain).

- Back-propagation learning algorithm :

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 $a_j \leftarrow \mathbf{x}_j\,[e]$

for $l$ = 2 to M 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 $l$ = M – 1 to 1 do

for each node j in layer $l$ do

$\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$

for each node i in layer $l$ + 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).

- **Back-propagation learning [diagramic representation]**

At each epoch, sum gradient updates for all examples.

Training curve for 100 restaurant examples (with a single hidden-layer network with 4 hidden units) is shown in Fig. 11.2.9. They converge to a perfect fit to the training data.

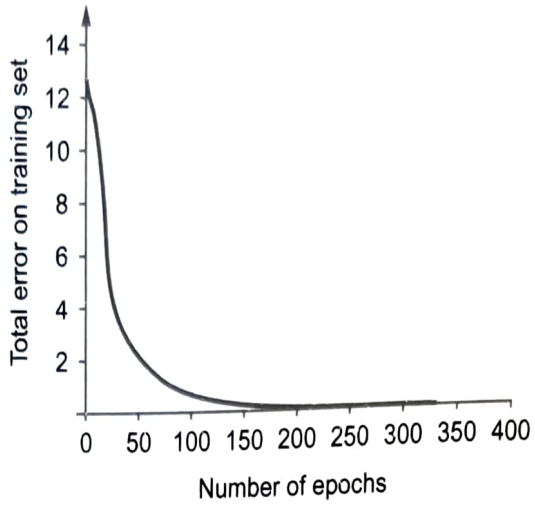



**Fig. 11.2.11 (a) Training curve showing gradual reduction in error as weights are modified over several epochs, for a given set of examples in the restaurant domain**

Typical problems : Slow convergence, local minima.

Learning curve for MLP with 4 hidden units : -

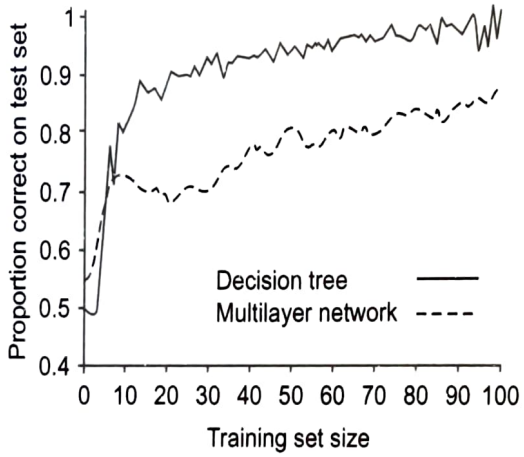



**Fig. 11.2.11 (b) Comparative earning curves showing that decision-tree learning does slightly better than back-propagation in a multilayer network**

MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily.

- **Neural Networks-Summary :**

- Most brains have lots of neurons : Each neuron ≈ linear - threshold unit (?)
- Perceptrons (one-layer networks) is insufficient in expressiveness.
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e. error back propagation.
- Many applications : Speech recognition, handwriting, fraud detection, economics, etc.
- Engineering, cognitive modelling, and neural system modelling subfields have largely diverged.

## 11.3 Kernel Machine

It is relatively new family of learning methods that use an efficient training algorithm and can represent complex, non-linear function. Kernel machines are generally called as Support Vector Machines (SVM).

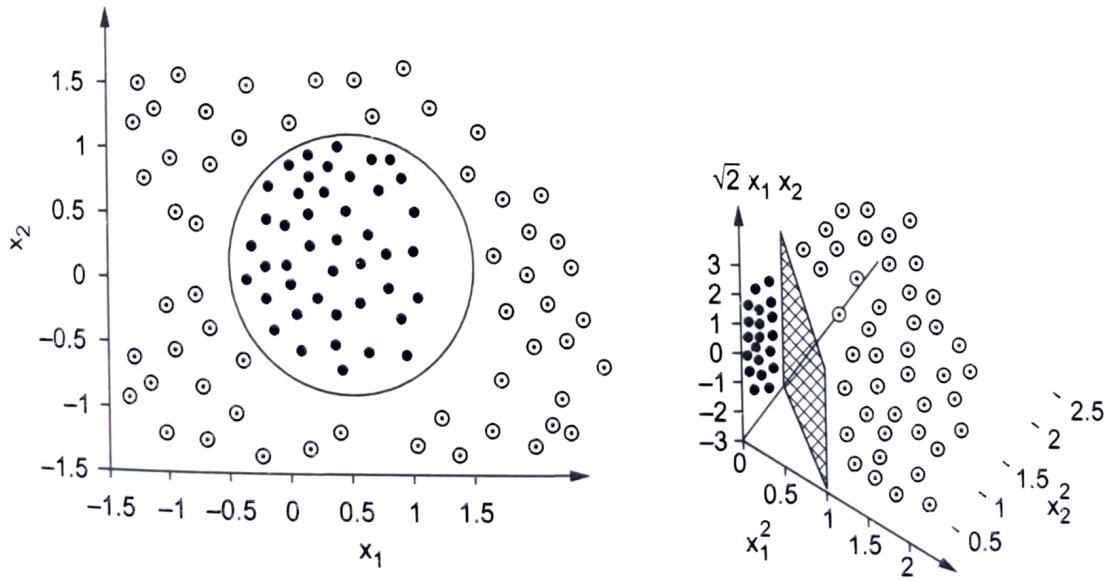
A classifier derived from statistical learning theory by V. Vapnik, et.al.in 1992.

SVM became famous when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwritting recognition task.

Currently, SVM is widely used in object detection and recognition, content-based image retrieval, text recognition, biometrics, speech recognition, etc. They are also used for regression.

- **Characteristics of SVM :**
  - It is supervised learning algorithm (Vapnik 98).
  - It is efficient, it can represent complex, non-linear functions.
  - It is suitable for training data that is not linearly separable (input space), the data is mapped to a higher-dimensional space (feature space).
  - The data are linearly-separable in the new space by maping data into a higher dimension space, they will always be linearly separable (example - N-data points in a space of N–1 dimensions).
- Mapping a bidimensional data to 3-D :

The example :




**(a) A two dimensional training with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown.**

**(b) The same data after mapping into three dimensional input space $x_1^2$, $x_2^2$, $(\sqrt{2}\, x_1, x_2)$. The circular decision boundry in (a) becomes a linear decision boundry in three dimensions.**

**Fig. 11.3.1**

Ideal : We express all inputs through a vector of features :

$$f_1 = x_1^2$$

$$f_2 = x_2^2$$

$$f_3 = \sqrt{2x_1x_2}$$

- In the new 3-dimensional space the data becomes linearly separable !
- The observation in a space with sufficiently many dimensions, makes all data to become linearly separable !

**Problem : Danger of overfitting if D ~ N !**

- **Optimal linear separator :**
  - If d ≈ N, d - dimension, N - number of examples overfitting.
  - Optimal linear separator : The largest margin between it and the positive examples on one side and the negative examples on the other side.
  - Quadratic programming optimization problem :

Examples $x_1$ classifications : $y_1 = \pm 1$.

- Maximizing the expression :

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j, y_i y_j (x_i \cdot x_j)$$

This function has a single global maximum !

The equation of the separator : -

$$h(X) = \text{sign}\left( \sum_i \alpha_i y_i (X \cdot x_i) \right)$$

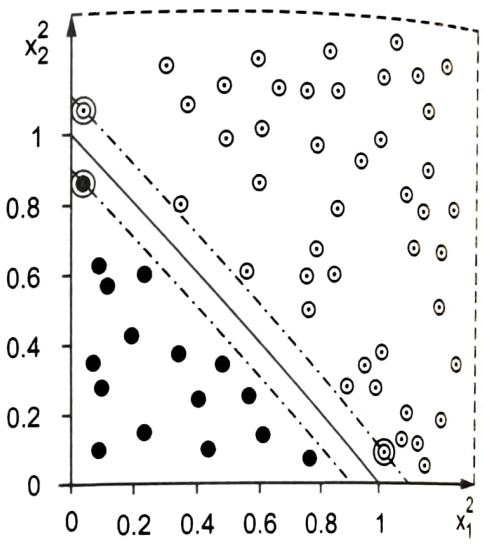



**Fig. 11.3.2 A close-up projected onto the first two dimensions, of the optimal seperator shown in Fig. 11.3.1 (b). The separator is shown as a heavy line, with the closest points-the support vectors - marked with the circles. The margin is the seperation between the positive and negative examples**

- **Support vectors : -**

Are the points closest to the separator (the only ones for which the weights are not 0).

The optimal number of support vectors is usually much smaller than N!

- Non-linear SVMs :

Datasets that are linearly separable with noise, work out great : -

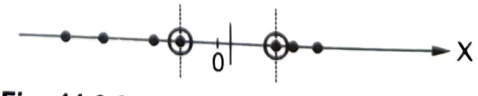



**Fig. 11.3.3 (a) Linearly seperable data**

But what are we going to do if the dataset is just too hard ?


**Fig. 11.3.3 (b) Hard dataset**

How about mapping data to a higher-dimensional space as shown in below figure.

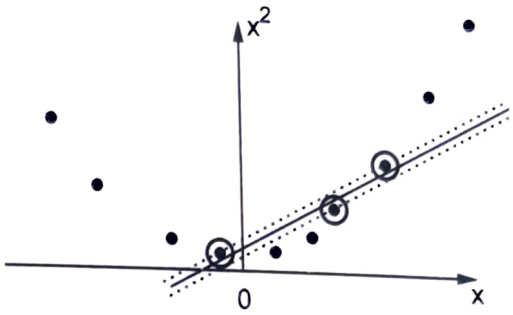



**Fig. 11.3.3 (c) Data mapped to higher dimension**

- **Non-linear SVMs : Feature space**
  - General idea : The original input space can be mapped to some higher-dimensional feature space where the training set is separable.
- **Non-linear SVM : The Kernel Trick**
  - With this mapping, our discriminant function is now : -

$$g(x) = W^T \phi(x) + b = i \in S V \sum_{i \in sv} \alpha_i \boxed{\phi(x_i)^T \phi(x)} + b$$

  - No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.
  - A kernel function is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space.

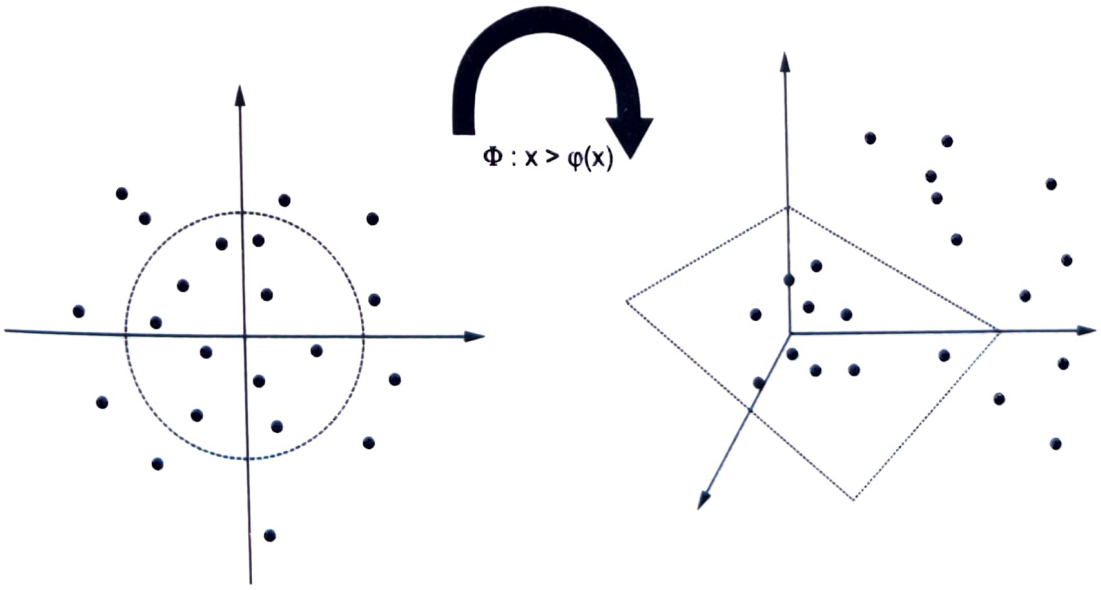



**Fig. 11.3.3 (d) Input mapping to higher dimension**

$$K(X_i, X_j) \equiv \phi(X_i)^T \phi(X_j)$$

– An example : 2-dimensional vectors $X = [x_1 \ x_2]$ ;

Let $K(x_i, X_j) = (1 + X_i^T X_j)^2$

Need to show that $K(X_i, X_j) = \phi(X_i)^T \phi(X_j)$ :

$$K(X_i, X_j) = (1 + X_i^T X_j)^2$$

$$= 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1} x_{j1} + 2x_{i2} x_{j2}$$

$$= [1 \ x_{i1}^2 \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \sqrt{2} x_{i1} \sqrt{2} x_{i2}]^T \ [1 \ x_{j2} \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \sqrt{2} x_{j1} \sqrt{2} x_{j2}]$$

$$= \phi(X_i)^T \phi(X_j), \text{ where } \phi(X) = [1 \ x_1^2 \sqrt{2} x_1 x_2 \ x_2^2 \sqrt{2} x_1 \sqrt{2} x_2]$$

- **Examples of commonly-used kernel functions : -**

i) Linear kernel :

$$K(X_i, X_j) = X_i^T X_j$$

ii) Polynomial kernel :

$$K(X_i, X_j) = (1 + X_i^T X_j)^P$$

iii) Gaussian (Radial-Basis Function (RBF)) kernel :

$$K(X_i, X_j) = \exp\left(-\frac{||X_i - X_j||^2}{2\sigma^2}\right)$$

iv) Sigmoid :

$$K(X_i, X_j) = \tan h(\beta_0 X_i^T X_j + \beta_1)$$

– In general, functions that satisfy Mercer's condition can be kernel functions.

- Support vector machine : Algorithm
  1) Choose a kernel function.
  2) Choose a value for C.
  3) Solve the quadratic programming problem (many software packages are available).
  4) Construct the discriminant function from the support vectors.

- **Some issues of SVM :**

⇒ Choice of kernel :

- Gaussian or polynomial kernel is default.
- If ineffective, more elaborate kernels are needed.
- Domain experts can give assistance in formulating appropriate similarity measures.

⇒ Choice of kernel parameters :

- Example : $\sigma$ in Gaussian kernel.
- $\sigma$ is the distance between closest points with different classifications.
- In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

⇒ Optimization criterion :

- Hard margin V.S. soft margin
- A lengthy series of experiments in which various parameters are tested.

## Answer in Brief

1. *Write a note on statistical learning.* **(Refer section 11.1.1)**
2. *How is learning done with hidden variables ? Explain unsupervised clustering with hidden variables.* **(Refer section 11.1.4)**
3. *Explain EM algorithm.* **(Refer section 11.1.4)**
4. *Explain the concept on learning using decision trees and neural networks approach.* **(Refer section 11.2)**
5. *Distinguish between feed-forward and recurrent neural network strucuture.* **(Refer section 11.2)**
6. *Explain back propagation process with its algorithm.* **(Refer section 11.2)**
7. *How statistical learning method differs from reinforcement learning method ?*

**Ans. :** Reinforcement learning is concerned with how an agent decides to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent thought to take in those states.

Statistical learning is inferring a function from supervised training data. The training data consist of a set of training examples. In this learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

Reinforcement learning differs from statistical learning because correct input/output pairs are never presented nor sub optimal actions explicitly corrected. In reinforcement

learning there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

## 11.4 University Questions with Answers

**Summer - 18**

**Q.1** *Explain Hopfield Network.* **(Refer section 11.2)** [4]

**Q.2** *What is meant by perceptron ? Give one example.* **(Refer section 11.2)** [3]

**Winter - 18**

**Q.3** *List and explain the application of neural network.* **(Refer section 11.2)** [7]

**Summer - 19**

**Q.4** *Define epoch with respect to ANN.* **(Refer section 11.2)** [3]

**Q.5** *Discuss perceptron.* **(Refer section 11.2)** [3]

**Q.6** *Explain hopfield network.* **(Refer section 11.2)** [4]

**Winter - 19**

**Q.7** *Enlist some applications of neural networks.* **(Refer section 11.2)** [3]

**Summer - 20**

**Q.8** *How do you define artificial neural network ? How does it learn ?* **(Refer section 11.2)** [4]

**Q.9** *What do you understand by classification in neural network ? Briefly explain perceptron algorithm and also narrate its limitation.* **(Refer section 11.2)** [7]