GNR602
Advanced Methods in Satellite Image Processing

Instructor: Prof. B. Krishna Mohan
CSRE, IIT Bombay
bkmohan@csre.iitb.ac.in

Slot 13

Lecture 02-06 Image Classification Systems

Spring 2020-2021

Contents of the Lecture

- Linear Discriminant Functions
- Perceptron Classifier
- Limitations of Simple Perceptron Classifier
- Multilayer perceptron
- Unsupervised Classification by Kohonen Network
- SVM

Terminology

- Classification – the task of assigning labels to the elements of a given data set so that we can relate the data elements to semantically meaningful entities e.g. a pixel in an image assigned the class water or beach …
- Image is a matrix of M rows x N columns x L features. Each element of the image is a pixel, having a position given by a row number, a column number, and a vector or L values
- Feature is a sensor-recorded brightness level or a derived quantity from the sensor-recorded image through some image processing technique
- Feature space is an L-dimensional space in which each feature vector is a point. For example, a color image is a collection of feature vectors, which are points in a 3-dimensional space (R,G, and B). We can add the row and column positions also as features and make it a 5-dimensional feature space

Terminology

- Discriminant function – a function that produces a value when a pixel feature vector is input to it. Each class k is associated with a discriminant function $g_k(x)$. $g_k(x)$ is a linear discriminant function if it is a linear function of variable x.
- x is assigned to class j when the response of $g_j(x)$ is highest among all $g_k(x)$, i.e., $g_j(x) \geq g_k(x)$, $k=1,2, \ldots, K$
- For K classes, there will be K discriminant functions, when $K>2$. The exception is $K=2$, in which a single discriminant function can be designed such that $g(x) > 0$ if x belongs to class 1 and $g(x) < 0$ if x belongs to class 2. If $g(x) = 0$, then it implies that with the available features, it is not possible to uniquely determine the class to which x should be assigned.
- A training sample is a feature vector whose true class label is known. A small number of pixels in an image can be identified belonging to each class which are used to *train* the classifiers and then the rest of the pixels can be classified by the trained classifier

Terminology

- **Due to the uncertainty in measuring and assigning x to each class, it is considered to be a random variable. A random variable is a function that maps from the feature space to a probability value on the real line in the closed interval [0 1].**
- **For each class ω, the mean of all x is represented by vector μ. $\mu = 1/(N_j).\Sigma$ x, where x $\in \omega_j$**
- **Given x is a vector, each element x_i has a mean and a variance. Along with variance σ_i for each x_i, between each pair of components** m **and** n **of x have a covariance σ_{ij}.**
- σ_{mn} is defined as $E[(x_m - \mu_m)(x_n - \mu_n)]$. If m=n, covariance reduces to variance of x_m
- Considering all elements of vector **x**, all covariances together form a ***covariance matrix*** denoted by Σ

Maximum Likelihood Classification

$$g_i(x) = P(\omega_i \mid x)$$

(max. discrimination corresponds to max. posterior probability!)

This can also be written equivalently as

$$g_i(x) \equiv P(x \mid \omega_i)\, P(\omega_i)$$

$$g_i(x) = \ln P(x \mid \omega_i) + \ln P(\omega_i)$$

Since relative ranking of $g_i(x)$ is important, use of logarithm does not alter it and moreover, it simplifies calculations

Discriminant Functions

Feature space divided into K decision regions

if $g_i(x) > g_j(x)$ $\forall j \neq i$ *then x is in* $\mathcal{R}_i$

($\mathcal{R}_i$ means assign *x* to ω_i)

The two-category case

A classifier is a "dichotomizer" that has two discriminant functions g_1 and g_2

Let $g(x) \equiv g_1(x) - g_2(x)$

Decide ω_1 *if* $g(x) > 0$; Otherwise decide ω_2

Discriminant Functions

– The computation of $g(x) = g_1(x) - g_2(x)$

– Given $g_i(x) = \ln\{P(\omega_i \mid x)\} = \ln\{P(x \mid \omega_i)\} + \ln\{P(\omega_i)\}$

$$g(x) = \ln\{P(\omega_1 \mid x)\} - \ln\{P(\omega_2 \mid x)\}$$

$$= \ln\frac{P(x \mid \omega_1)}{P(x \mid \omega_2)} + \ln\frac{P(\omega_1)}{P(\omega_2)}$$

ln(A) – ln(B) = ln(A/B)

The Normal Density

- Density which is analytically tractable
- Continuous density
- A lot of processes are asymptotically Gaussian

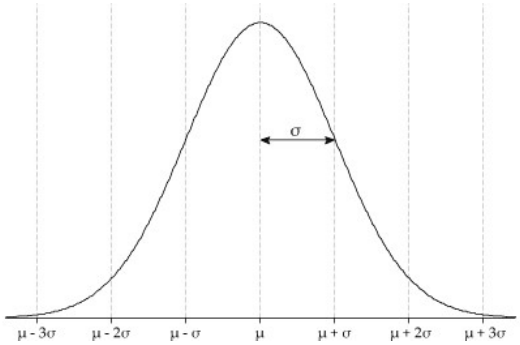
$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right],$$

Where:

μ = mean (or expected value) of x

σ^2 = expected squared deviation or variance

The Normal Density



Source: http://amsi.org.au/ESA_Senior_Years/SeniorTopic4/4f/4f_2content_3.html

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right],$$

The Normal Density

Multivariate density

Multivariate normal density in d dimensions is:

$$p(x \mid \omega) = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)\right]$$

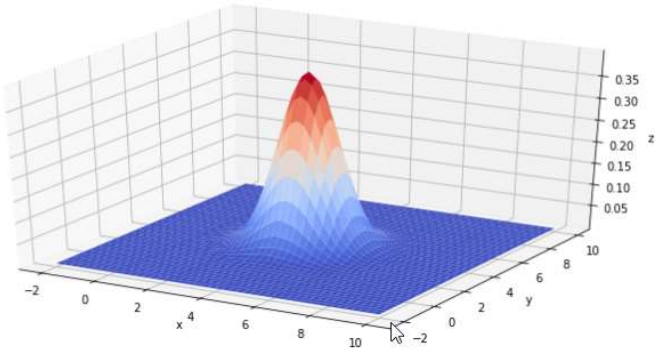
where: $x = (x_1, x_2, \ldots, x_d)^t$

$\mu = (\mu_1, \mu_2, \ldots, \mu_d)^t$ mean vector

Σ = *dxd* covariance matrix

$|\Sigma|$ and Σ^{-1} are determinant and inverse respectively

The Multivariate Normal Density



Source: https://towardsdatascience.com/a-python-tutorial-on-generating-and-plotting-a-3d-guassian-distribution-8c6ec6c41d03

$$p(x \mid \omega) = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)\right]$$

Discriminant Functions for the Normal Density

- The discriminant function for clas i

  $g_i(x) = \ln P(x \mid \omega_i) + \ln P(\omega_i)$

- Case of multivariate normal

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^t \Sigma_i^{-1}(x-\mu_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_i| + \ln P(\omega_i)$$

Discriminant Functions for the Normal Density

Case $\Sigma_i = \sigma^2.I$ (I stands for the identity matrix)

Means alone matter; intra-class variance-covariance not considered

$g_i(x) = w_i^t x + w_{i0}$ (linear discriminant function)

where:

$$w_i = \frac{\mu_i}{\sigma^2}; \quad w_{i0} = -\frac{1}{2\sigma^2}\mu_i^t\mu_i + \ln P(\omega_i)$$

(ω_{i0} is called the threshold for the ***i*th** category!)

The above is by considering only the class dependent terms from previous slide

Discriminant Functions for the Normal Density

– The hyperplane separating classes i and j

$$x_0 = \frac{1}{2}(\mu_i + \mu_j) - \frac{\sigma^2}{\|\mu_i - \mu_j\|^2} \ln \frac{P(\omega_i)}{P(\omega_j)} (\mu_i - \mu_j)$$

$$if \quad P(\omega_i) = P(\omega_j) \quad then \quad x_0 = \frac{1}{2}(\mu_i + \mu_j)$$

Discriminant Functions for the Normal Density

- **Case $\Sigma_i = \Sigma$ (covariances of all classes are identical but arbitrary!)**
  Hyperplane separating $\mathcal{R}_i$ and $\mathcal{R}_j$

$$x_0 = \frac{1}{2}(\mu_i + \mu_j) - \frac{\ln[P(\omega_i)/P(\omega_j)]}{(\mu_i - \mu_j)^t \Sigma^{-1} (\mu_i - \mu_j)} \cdot (\mu_i - \mu_j)$$

(the hyperplane separating $\mathcal{R}_i$ and $\mathcal{R}_j$ is generally not orthogonal to the line between the means!)

Discriminant Functions for the Normal Density

Case Σ_i = arbitrary

The covariance matrices are different for each category

$$g_i(x) = x^t W_i x + w_i^t x = w_{i0}$$

where :

$$W_i = -\frac{1}{2}\Sigma_i^{-1}$$

$$w_i = \Sigma_i^{-1}\mu_i$$

$$w_{i0} = -\frac{1}{2}\mu_i^t\Sigma_i^{-1}\mu_i - \frac{1}{2}ln|\Sigma_i| + ln\,P(\omega_i)$$

Programmed v/s Learned Classifiers

- In programmed classifiers, given the training samples the class discriminant functions are given by fixed formulae, as seen in previous slides.
- In learning based classifiers, given the training samples the class boundaries are *learnt* based on an error minimization logic, using the training algorithm.
- The classes may be separated by linear or non-linear boundaries

Linear Discriminant Function

The linear discriminant function is represented by a line in the 2-D feature space, by a plane in the 3-D feature space and by an (n-1)D hyperplane in n-dimensional feature space
When data are separable by linear discriminants, the analysis of the performance of the classifier is simpler

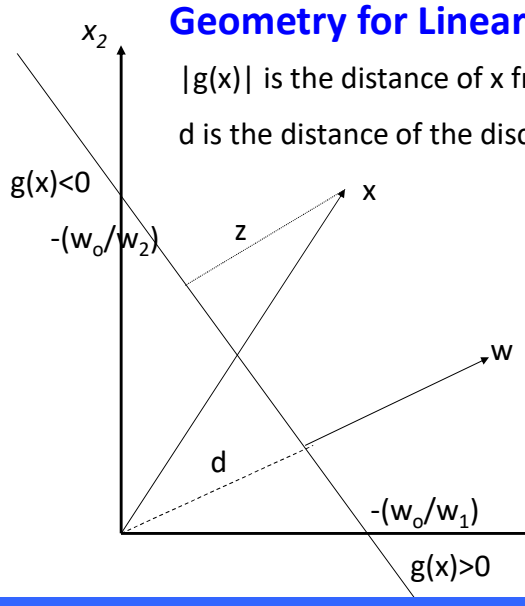
Linear Discriminant Function

The discriminant in the 2-class case can be written as

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_o = 0$$

The weight vector $\mathbf{w} = [w_1 \ldots w_L]^t$ is an L-dimensional vector, where L is the number of bands
If two samples $\mathbf{x_1}$ and $\mathbf{x_2}$ are on the plane, we can write **$\mathbf{w}^t(\mathbf{x_1}-\mathbf{x_2}) = 0$ i.e., w is perpendicular to the discriminant function**

Geometry for Linear Discriminant



$$d = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}$$

and

$$z = \frac{|g(\mathbf{x})|}{\sqrt{w_1^2 + w_2^2}}$$

Algorithm for Linear Discriminant Classifier

- Given two classes c_1 and c_2, we have
- $\mathbf{w}^t\mathbf{x} > 0$ when $\mathbf{x}$ is in class c_1
- $\mathbf{w}^t\mathbf{x} < 0$ when $\mathbf{x}$ is in class c_2
- Training this classifier involves determining the weights $\mathbf{w}$
- They may be determined using training data using optimization methods

Perceptron Training Algorithm

Since **w** is the unknown variable, form the cost function in terms of **w** and minimize it and adopt the $\mathbf{w}^*$ corresponding to the minimum

The Perceptron Algorithm

- Assume linearly separable classes, i.e.,

$$\exists\ \underline{w}^*:\ w^{*T}\ \underline{x} > 0\ \ \forall \underline{x} \in \omega_1$$

$$\underline{w}^{*T}\ \underline{x} < 0\ \ \forall \underline{x} \in \omega_2$$

The discriminant is $\underline{w}^{*T}\ \underline{x} + w_0^*$

Let

$$\underline{w}' \equiv \begin{bmatrix} \underline{w}^* \\ w_0^* \end{bmatrix},\ \underline{x}' = \begin{bmatrix} \underline{x} \\ 1 \end{bmatrix}$$

$$\underline{w}^{*T}\ \underline{x} + w_0^* = \underline{w}'^T\ \underline{x}' = 0$$

The Perceptron Algorithm

Our goal: Compute a solution, i.e., a hyperplane $\underline{w}$, so that

$$\underline{w}^T \underline{x}(><)0 \quad \underline{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases}$$

The steps

- Define a cost function to be minimized
- Choose an algorithm to minimize the cost function
- The minimum corresponds to a solution

The Perceptron Algorithm

The Cost Function

$$J(\underline{w}) = \sum_{\underline{x} \in Y} (\delta_x \underline{w}^T \underline{x})$$

$$J(\underline{w}) \geq 0$$

Where Y is the subset of the vectors **wrongly** classified by $\underline{w}$.
When Y=(empty set) a solution is achieved and J($\underline{w}$)=0

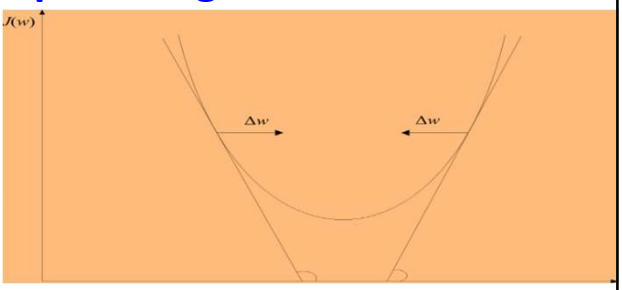
$\delta_x = 1$ if $\underline{x} \in Y$ and $\underline{x} \in \omega_1$ i.e., $\underline{x}$ actually belongs to ω_2

$\delta_x = -1$ if $\underline{x} \in Y$ and $\underline{x} \in \omega_2$ i.e., $\underline{x}$ actually belongs to ω_1

Perceptron Algorithm

$$\underline{w}(\text{new}) = \underline{w}(\text{old}) + \Delta\underline{w}$$

$$\Delta\underline{w} = -\mu \frac{\partial J(\underline{w})}{\partial \underline{w}} \Big| \underline{w} = \underline{w}(\text{old})$$



$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} \left(\sum_{\underline{x} \in Y} \delta_x \underline{w}^T \underline{x} \right) = \sum_{\underline{x} \in Y} \delta_x \underline{x}$$

$$\underline{w}(t+1) = \underline{w}(t) - \rho_t \sum_{\underline{x} \in Y} \delta_x \underline{x}$$

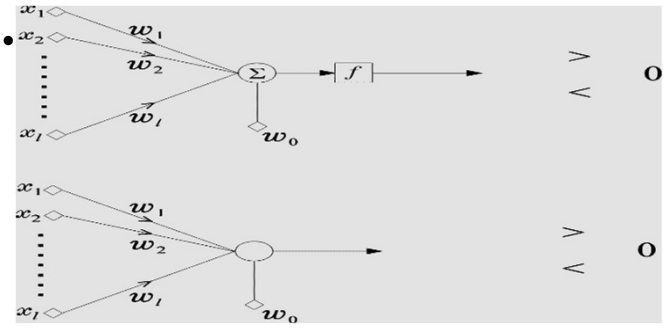
Reward and Punishment Learning

$$\underline{w}(t+1) = \underline{w}(t) + \rho \underline{x}_{(t)}, \quad \underline{w}^T(t)\underline{x}_{(t)} \le 0, \; \underline{x}_{(t)} \in \omega_1$$

$$\underline{w}(t+1) = \underline{w}(t) - \rho \underline{x}_{(t)}, \quad \underline{w}^T(t)\underline{x}_{(t)} \ge 0, \; \underline{x}_{(t)} \in \omega_2$$

$$\underline{w}(t+1) = \underline{w}(t) \quad \text{otherwise}$$

Perceptron as a Learning Machine



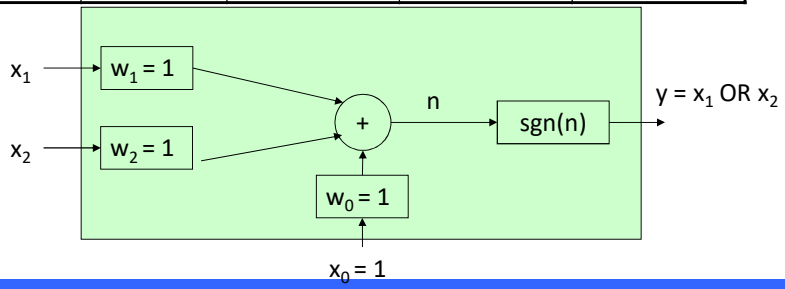
A learning machine that learns from the training vectors via the perceptron algorithm

$w_i's$ synapses or synaptic weights

w_0 threshold

Linearly Separable Function

x_1	x_2	x_1 AND x_2	x_1 OR x_2	NOT x_1
-1	-1	-1	-1	1
-1	1	-1	1	1
1	-1	-1	1	-1
1	1	1	1	-1



Linearly Separable Function

- Given a binary system that has two inputs we would end up with the following equation:

$$w \bullet x = [\, w_0\ w_1\ w_2 \,] \bullet [\, x_0\ x_1\ x_2 \,]^T$$

if we assume that $x_0 = 1$,

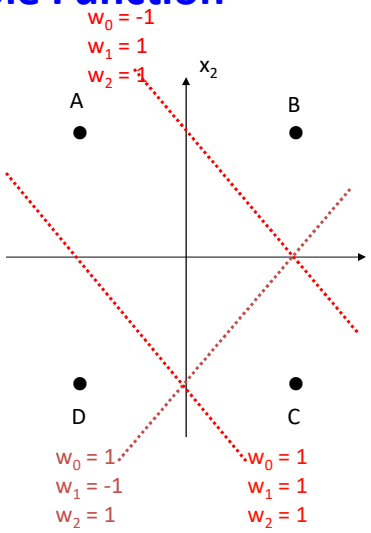
$$n = w_0 + w_1 x_1 + w_2 x_2$$

- We can graphically display all the possible inputs / outputs as follows.

Linearly Separable Function

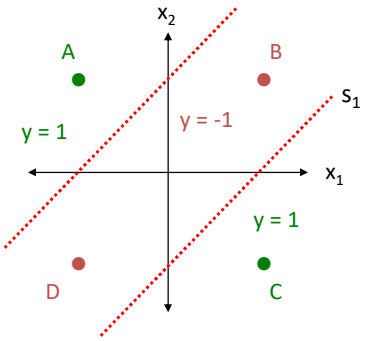
- The equation for n, can be viewed as an equation of a line. Depending on the values of the weights, this line will separate the four possible inputs into two categories.
- Equation of the line becomes:

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{w_0}{w_2}$$



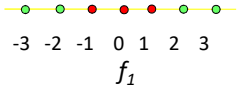
Linearly Non-Separable Function

- Some functions are not linearly separable (such as the XOR function). These functions can not be graphically separated by a single straight line as in the previous example.
- Functions such as XOR, require two lines to separate the points into the appropriate classes.
-



Simple Quadric Example

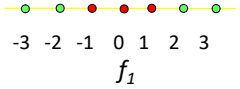
Source: Lecture slides of Prof. Tony Martinez
http://axon.cs.byu.edu/~martinez/



- Perceptron with just feature f_1 cannot separate the data
- Could we add a transformed feature to our perceptron?

Simple Quadric Example

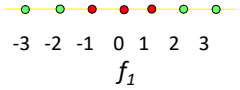
Source: Lecture slides of Prof. Tony Martinez
http://axon.cs.byu.edu/~martinez/

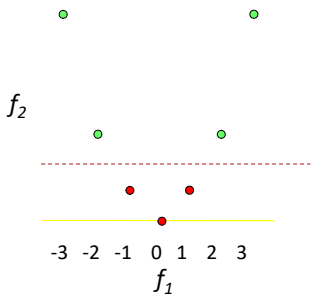


- Perceptron with just feature f_1 cannot separate the data
- Could we add a transformed feature to our perceptron?
- $f_2 = f_1^2$

Simple Quadric Example

Source: Lecture slides of Prof. Tony Martinez
http://axon.cs.byu.edu/~martinez/





- Perceptron with just feature f_1 cannot separate the data
- Could we add another feature to our perceptron $f_2 = f_1^2$
- Note could also think of this as just using feature f_1 but now allowing a quadric surface to divide the data

Multilayer Perceptron Neural Networks

What Are Artificial Neural Networks?

- Parallel distributed computing mechanisms inspired by the functioning of human brain
- computation is highly parallel
- each processor performs an extremely simple function
- operating in continuous domains is often preferred
- system behavior is often ***learned*** rather than *programmed*

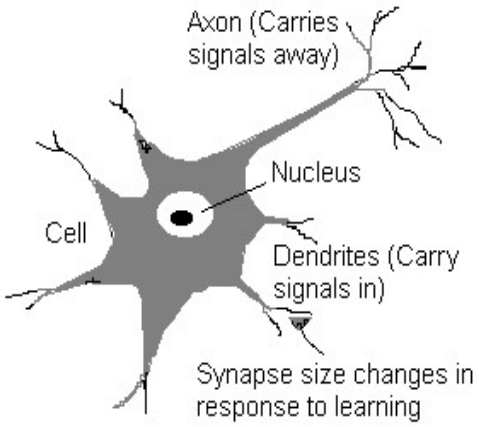
Why Study Artificial Neural Networks?

- Model Brain & Behavior
- **Solve Difficult Engineering Problems such as classification and regression**

Artificial Neural Networks for Engineering Problems

- face recognition
- detect fraudulent use of credit cards
- control blast furnaces
- rate bond investment risk
- automatically diagnose the cause of aircraft engine failures
- drive robotic automobiles down busy highways
- **Satellite image classification**

Inspiration from Neurobiology



Inspiration from Neurobiology

- A neuron: many-inputs / one-output unit
- Output can be *excited* or *not excited*
- Incoming signals from other neurons determine if the neuron shall *excite* ("fire")
- Output subject to attenuation in the *synapses,* which are junction parts of the neuron
- The contribution of the signals depends on the strength of the *synaptic* connection

Biological Neural Systems

- Neuron switching time : > 10^{-3} secs
- Number of neurons in the human brain: ~10^{10}
- Connections (synapses) per neuron : ~10^{4}–10^{5}
- Face recognition : 0.1 secs
- High degree of parallel computation
- Distributed representations

Inspiration from Neurobiology

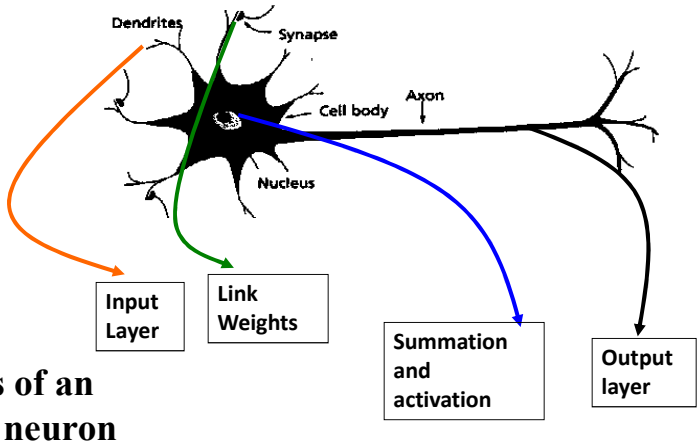
- What are animals so good at?
- What are computer not so good at?

So good at
Dealing with noisy data
Dealing with unknown data
Massive parallelism
Fault tolerance
Adapting to circumstances
Knowledge storage

Not so good at
Dealing with noisy data
Dealing with unknown data
Massive parallelism
Fault tolerance
Adapting to circumstances
Knowledge storage

Artificial Neuron

A physical neuron



Elements of an artificial neuron

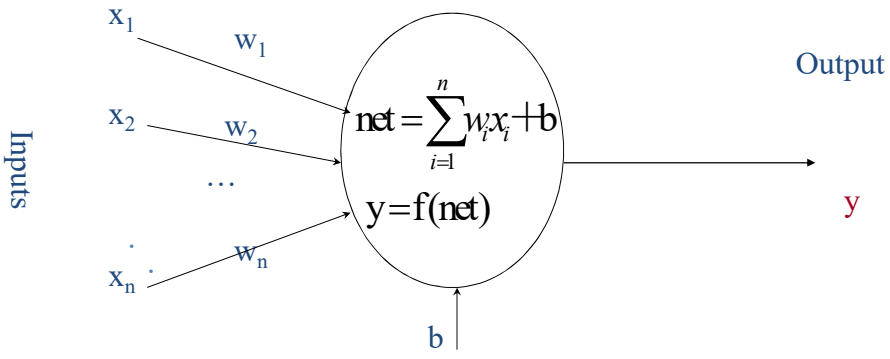
Artificial Neuron

- Strength of connection between the neurons is stored as a weight-value for the specific connection.
- Learning the solution to a problem = finding the correct connection weights

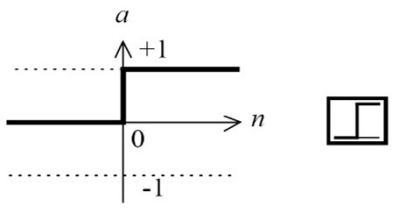
Artificial Neuron

- The internal components of the Artificial Neuron consist of:
  - **Weights.**
  - **Threshold/Bias.**
  - **Summing Unit.**
  - **Activation Function.**

Mathematical Representation

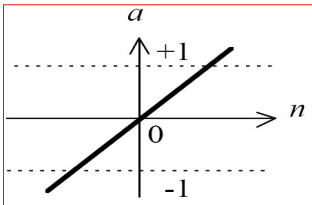


Mathematical Representation of the Activation Function



$a = hardlim(n)$

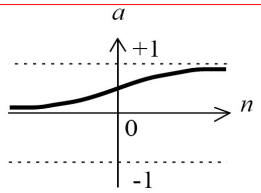
Hard-Limit Transfer Function



$a = purelin(n)$

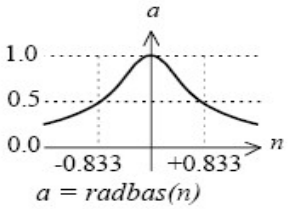
Linear Transfer Function

Mathematical Representation of the Activation Function



$a = logsig(n)$

Log-Sigmoid Transfer Function



$a = radbas(n)$

Radial Basis Function

Examples

Nonlinear form of the neuron:

$$y = f(x, w)$$

$$y = \frac{1}{1 + e^{-w^T x}}$$ ← Sigmoidal Function

$$y = e^{-\frac{||x-w||^2}{2a^2}}$$ ← Gaussian Function

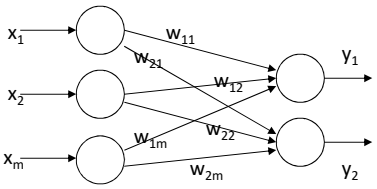
A Simple Perceptron Training Algorithm

- It's a single-layer network
- Change the weight by an amount proportional to the difference between the target output and the actual output.

$$\Delta W = \eta * (T-Y)X$$

$$W^{new} = W^{old} + \Delta W$$

Perceptron Learning Rule



Y=hardlim(WX)

Delta Rule

The delta rule requires that the activation function be differentiable. The learning takes place using a set of pairs of inputs and corresponding known outputs and weights are adjusted accordingly.

- If $y_k^n = t_k^n$ then change no weights.
- If $y_k^n > t_k^n$ then $\Delta w_{kj} = -\eta\, x_j^n$
- If $y_k^n < t_k^n$ then $\Delta w_{kj} = +\eta\, x_j^n$

$$\Delta w_{kj} = \eta\, x_j^n\, t_k^n$$

If a perceptron can solve the problem, the perceptron learning procedure will converge in a finite number of steps to a solution.

The Knowledge Is In The Weights

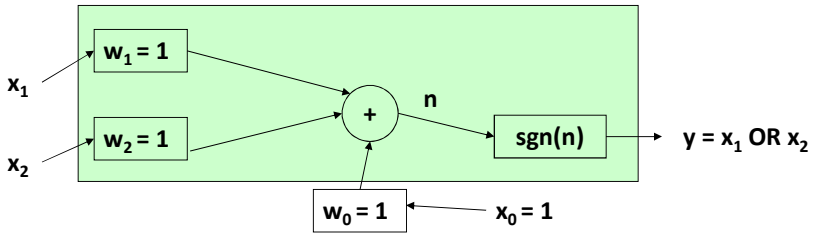
In the brain, the activation function of a given neuron is determined during neural development. Thus, learning is thought to involve changes in synaptic strengths or connection weights.

Artificial neural network systems use a fixed activation function and often a fixed pattern of connectivity between processing elements. Learning involves re-computing the connection weights.

These systems are sometimes called **connectionist** systems.

Linearly Separable Functions

x_1	x_2	x_1 AND x_2	x_1 OR x_2	NOT x_1
-1	-1	-1	-1	1
-1	1	-1	1	1
1	-1	-1	1	-1
1	1	1	1	-1



Linearly Separable Functions

- Given a binary system that has two inputs we would end up with the following equation:

$$\mathbf{w.x} = [\, w_0, w_1, w_2 \,] \bullet [\, x_0, x_1, x_2 \,]^T$$

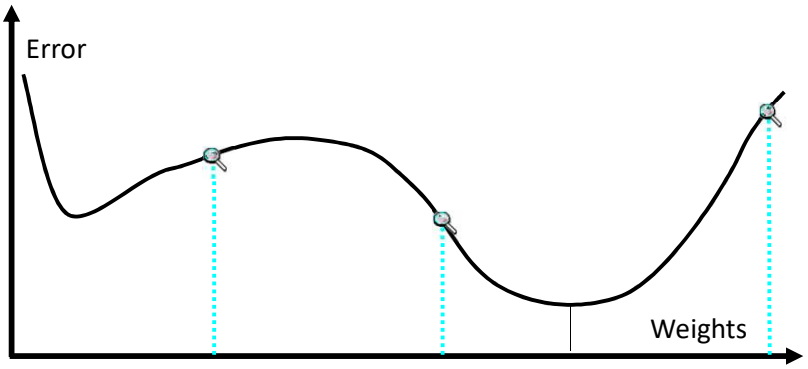
if we assume that $x_0 = 1$,

$$n = w_0 + w_1 x_1 + w_2 x_2$$

- We can graphically display all the possible inputs / outputs as follows.

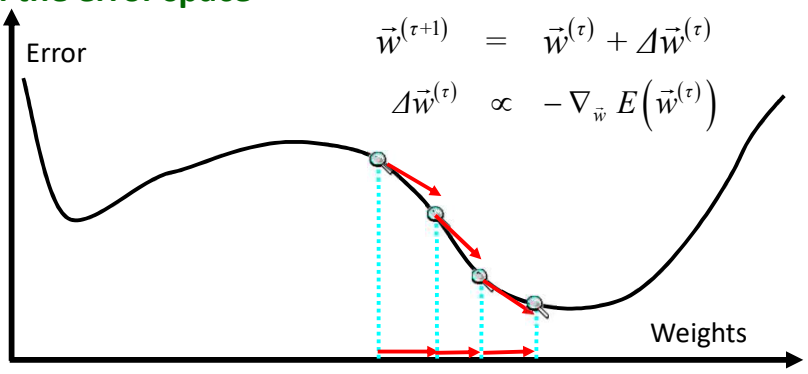
Learning As Error Minimization

Search problem to locate link weights with minimum error



Gradient Descent Technique

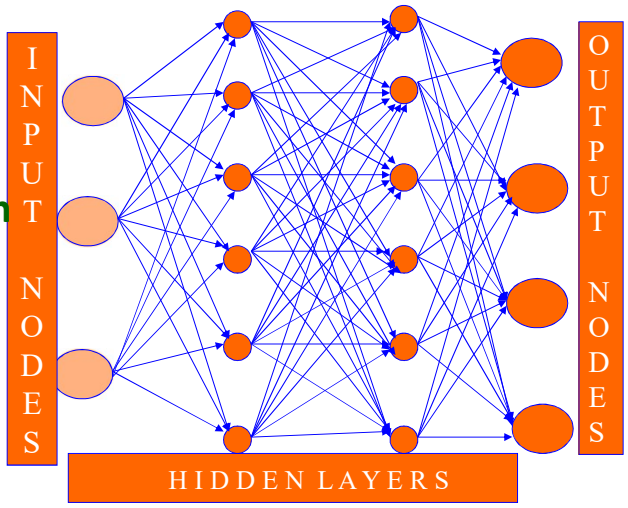
Move in a direction along the negative of the gradient in the error space



$$\vec{w}^{(\tau+1)} = \vec{w}^{(\tau)} + \Delta\vec{w}^{(\tau)}$$

$$\Delta\vec{w}^{(\tau)} \propto -\nabla_{\vec{w}} E\left(\vec{w}^{(\tau)}\right)$$

Multilayer Perceptron Network



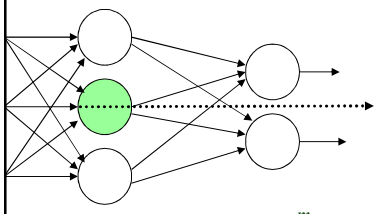
Multilayered Perceptron

- ANN research was in the limbo for nearly two decades when the single stage perceptron network was found unable to deal with many commonly encountered problems
- The multistage perceptron network was felt to be the answer to this problem. However, the problem is

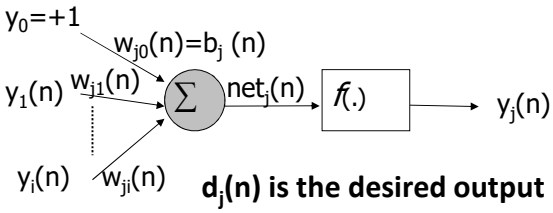
How is the multilayered perceptron network trained?

Training the Multilayer Perceptron Network

- Notation



Any given neuron



$d_j(n)$ is the desired output and $y_j(n)$ is the computed output

$$net_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

$$y_j(n) = f_j(net_j(n))$$

$$e_j(n) = d_j(n) - y_j(n)$$

Training the Multilayer Perceptron Network

- Total Error computed over all neurons in the output layer

$$\xi(n) = \frac{1}{2} \sum_{j \in C} e^2(n)$$

- If there are N training samples, then the average error can be computed as

$$\xi_{av} = \frac{1}{N} \sum_{n=1}^{N} \xi(n)$$

Training the Multilayer Perceptron Network

- Given that the weights are the unknowns, find the derivative of the error function with respective to the weights and move in the direction of the negative gradient.

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial net_j(n)} \frac{\partial net_j(n)}{\partial w_{ji}(n)}$$

We have to compute each of the terms on the RHS to obtain the derivative on the LHS

Training the Multilayer Perceptron Network

- Given that $\xi(n) = \frac{1}{2} \sum_{j \in C} e^2(n)$
- we have $\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n)$
- Given that $e_j(n) = d_j(n) - y_j(n)$
- we have $\frac{\partial e_j(n)}{\partial y_j(n)} = -1$

Training the Multilayer Perceptron Network

- Given that $y_j(n) = f_j(net_j(n))$
- we have $\frac{\partial y_j(n)}{\partial net_j(n)} = f'_j(net_j(n))$
- Given that $net_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$
- we have $\frac{\partial net_j(n)}{\partial w_{ji}(n)} = y_i(n)$

Training the Multilayer Perceptron Network

- Given that

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial net_j(n)} \frac{\partial net_j(n)}{\partial w_{ji}(n)}$$

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n) \qquad \frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

$$\frac{\partial y_j(n)}{\partial net_j(n)} = f'_j(net_j(n)) \qquad \frac{\partial net_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n) f'_j(net_j(n)) y_i(n)$$

Training the Multilayer Perceptron Network

- By gradient descent rule,

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

$$\delta_j(n) = e_j(n) f'(net_j(n))$$

Error term

If we use the sigmoidal activation function, then

$$f_j(net_j(n)) = \frac{1}{1+\exp(-net_j(n))}$$

$$f'(net_j(n)) = y_j(n)[1-y_j(n)] \qquad y_j(n) = f_j(net_j(n))$$

Training the Multilayer Perceptron Network

- Case 1: Neuron k is an output node

$$\delta_k(n) = e_k(n) f'(net_k(n)) = [d_k(n) - y_k(n)] y_k(n)[1-y_k(n)]$$

- Case 2: Neuron j is a hidden node; the error in the output of the hidden node is not straight-forward because there is no defined output at a hidden node

$$\delta_j(n) = f'(net_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

$$= y_j(n)[1-y_j(n)] \sum_k \delta_k(n) w_{kj}(n)$$

- The way out? Differentiate the output error w.r.t. hidden node link weight to obtain the correction for hidden node weight

Training the Multilayer Perceptron Network

$$\delta_j(n) = f'(net_j(n))\sum_k \delta_k(n) w_{kj}(n)$$

$$= y_j(n)[1 - y_j(n)]\sum_k \delta_k(n) w_{kj}(n)$$

This can be shown in a sequence of steps that are similar to the chain rule of partial differentiation shown before when dealing with the weights linking to the output node.

Let j denote a hidden node and k one of the output nodes. The idea is to express the derivative of the error with respective to the unknown (hidden node weights) in terms of the computable partial derivatives

An excellent discussion of the multilayer neural networks can be found in the book ***Neural Networks and Learning Machines*** by Simon Haykin

Training the Multilayer Perceptron Network

$$\delta_j(n) = \frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial net_j(n)}$$

$$= \frac{\partial \xi(n)}{\partial y_j(n)} \phi_j'(net_j(n))$$

Given that $\xi(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n)$, where k is an o/p node

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_{k \in C} e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)}$$

Again,

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_{k \in C} e_k(n) \frac{\partial e_k(n)}{\partial net_j(n)} \frac{\partial net_j(n)}{\partial y_j(n)}$$

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \phi_k(net_k(n))$$

Therefore $\frac{\partial e_k(n)}{\partial net_j(n)} = -\phi_k'(net_k(n))$

Training the Multilayer Perceptron Network

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \phi_k(net_k(n))$$

Therefore $\frac{\partial e_k(n)}{\partial net_j(n)} = -\phi_k'(net_k(n))$

$$net_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n)$$

m is the total number of inputs applied to output neuron k

From the previous equation, it is clear that

$$\frac{\partial net_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

Thus, all the previous relations can be combined into

$$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_{k \in C} e_k(n) \phi_k'(net_k(n)) w_{kj}(n)$$

$$= -\sum_{k} \delta_k(n) w_{kj}(n)$$

Therefore the local gradient $\delta_j(n) = \phi_j'(net_k(n)) \sum_k \delta_k(n) w_{kj}(n)$

Training the Multilayer Perceptron Network

- Weight updating rule:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

Increment — Scale factor — Error term — Input

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n)$$

Weight Updation at Hidden Node

$$\frac{\partial z_k}{\partial net_k}\cdot\frac{\partial net_k}{\partial y_j} = f'(net_k)\cdot w_{kj}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{ji}} = \frac{\partial J(\mathbf{w})}{\partial y_j}\cdot\overbrace{\frac{\partial y_j}{\partial net_j}}^{f'(net_j)}\cdot\overbrace{\frac{\partial net_j}{\partial w_{ji}}}^{x_i}$$

$$\frac{\partial J(\mathbf{w})}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\cdot\overbrace{\frac{\partial z_k}{\partial y_j}}$$

$$= -\sum_{k=1}^{c}\underbrace{(t_k - z_k)f'(net_k)}_{\delta_k}\cdot w_{kj} = -\sum_{k=1}^{c}\delta_k w_{kj}$$

$$\Rightarrow \Delta w_{ji} = -\eta\frac{\partial J}{\partial w_{ji}} = \eta\underbrace{\left[\sum_{k=1}^{c}\delta_k w_{kj}\right]f'(net_j)}_{=\delta_j}x_i = \eta\cdot\delta_j\cdot x_i$$

Error Backpropagation

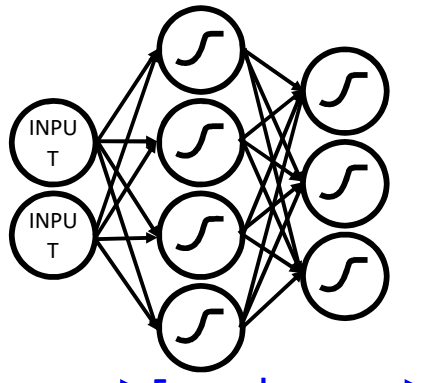
- The algorithm takes its name – backpropagation – from the fact that during training, the error is propagated back, from the output to the hidden layer!

Computational Complexity

The learning algorithm based on backpropagation of error is relatively efficient ...

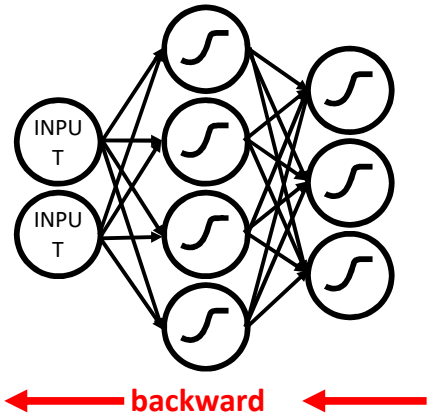
... at least with regard to a single pass through the training set.

Multilayer Feed-forward Network



Signals flow from the input layer towards the output layer

Error is backpropagated!



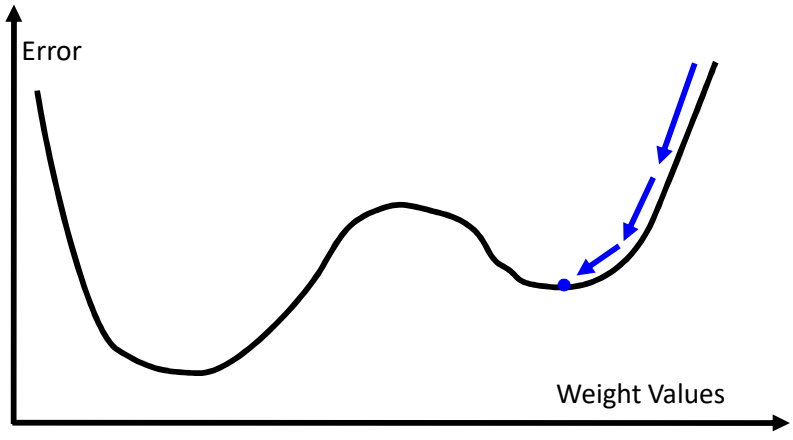
Error signal flows from output layer backwards towards the input layer, updating the weights along the way

Moving Towards Error Minimum

- The error does not decrease monotonously towards the minimum value
- Oscillations, and stagnation are common during the gradient descent procedure

Learning Curves

Local Minima Of The Error Function



Network Configuration issues

- The network size cannot be too small. It cannot learn the relationship between the input and the output
- The network size cannot be too large. It takes too long to train. Then it generalizes poorly.

Weight Initialization

- Before training begins, what values should weights have?
- Too large weights or all zero weights are not desirable.
- Thus, weights are typically initialized to small random values.
- Optimization techniques can be used for *smart* network weight initialization. **Genetic algorithms** are one such approach

Momentum

Oscillations can be reduced:

$$w_{ij}^{(\ell)}(t+1) = w_{ij}^{(\ell)}(t) - \eta \frac{\partial E}{\partial w_{ij}^{(\ell)}(t)} + \mu\left(w_{ij}^{(\ell)}(t) - w_{ij}^{(\ell)}(t-1)\right)$$

Weights at iteration t+1 depend on error derivative and difference between weights at iterations t and t-1.

μ is the momentum term. Usually smaller than the gain term η. (ℓ) is a particular layer; w_{ij} is the weight connecting j^{th} node in that layer to i^{th} node in the previous layer

Problems with BP Algorithm

- **Inability to learn**

  Error on training patterns never reaches a low level. This typically means that either the network architecture is inappropriate or the learning process has pushed the network into a bad part of weight space.

- **Inability to generalize**

  The network may master the training patterns but fail to generalize to novel situations. This typically means that either the training environment is impoverished or the network has over-learned from the inputs

Thumb rules for failure to train

- Longer training
- Different initial weights
- If oscillating too much, start with lower initial weights or decrease gain factor or increase momentum
- If too slow, increase gain term.
- Try more hidden units
- Try more hidden layers.
- Re-select training patterns
- Try a different encoding of inputs and/or targets.

Thumb rules for improving generalization

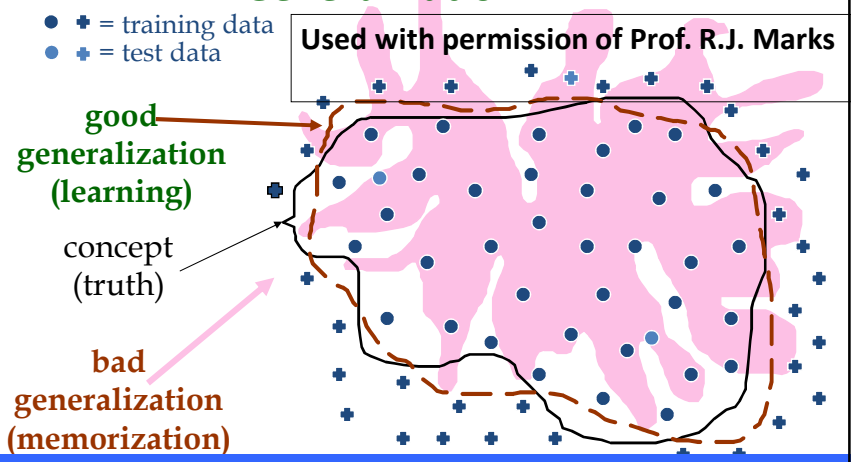
1. Reduce hidden layers/hidden nodes
2. Let the network learn slowly
3. Increase momentum
4. Prevent over-training
5. Reselect the training/testing patterns
6. Analyze what is being successfully generalized and where it is failing

IIT Bombay Slide 73

Illustration of Good Training and Poor Generalization

= training data
= test data

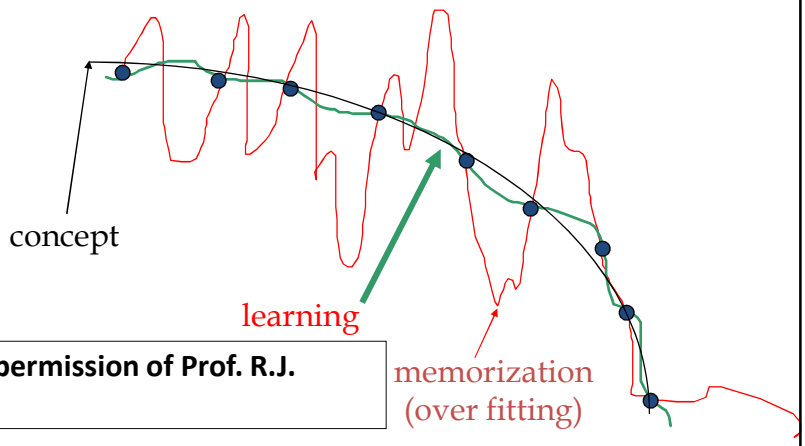
Used with permission of Prof. R.J. Marks



GNR602 Lecture 02-06 B. Krishna Mohan

IIT Bombay Slide 74

Alternate View



Used with permission of Prof. R.J. Marks

GNR602 Lecture 02-06 B. Krishna Mohan

How Can I Find Out What Was Learned?

- Look at connection weights.
- Try to understand the conditions under which each hidden unit becomes strongly active or inactive.
  - Check for correlations between hidden unit activity and input features.
  - Examine the "receptive field" of each hidden unit.

Selected Applications of MLP Classifier

- Landuse/Landcover classification
- Edge and line detection

Supervised Image Classification

- Identify the number of classes
- Identify the training data and generate the training patterns
- Define the network
  - Input/Output layers
  - Hidden layers
  - Gain and momentum terms

Supervised Image Classification

- Size of input layer = number of bands in the input data
- Size of output layer = number of classes into which the data is mapped
- Hidden layer(s): = in practice, one or two hidden layers are used
- Usually first hidden layer has more nodes
- And second hidden layer has fewer nodes

Supervised Image Classification

- Gain term = 0.1 to 0.5 in practice
- Higher the gain term value, faster the change in weights, but can lead to instability
- Momentum term = 0.01 to 0.1 in practice
- Gain and momentum values are modified if convergence is slow

Practical Issues

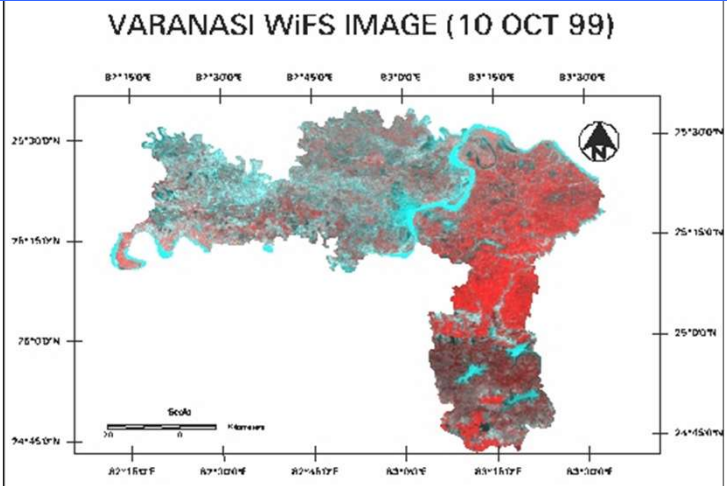
- Training samples are shuffled so that they are fed to the network in a random order
- This is essential so that all samples of any one class are not presented to the network sequentially

Typical coding of training data

- $S_{1,1}$, $S_{2,1}$, …, $S_{n,1}$, C_j
- $S_{i,1}$ refers to the i^{th} band of the 1^{st} sample
- C_j refers to the j^{th} class
- The training data is usually generated by taking the training windows and reading out pixels within those windows into a file along with the class code

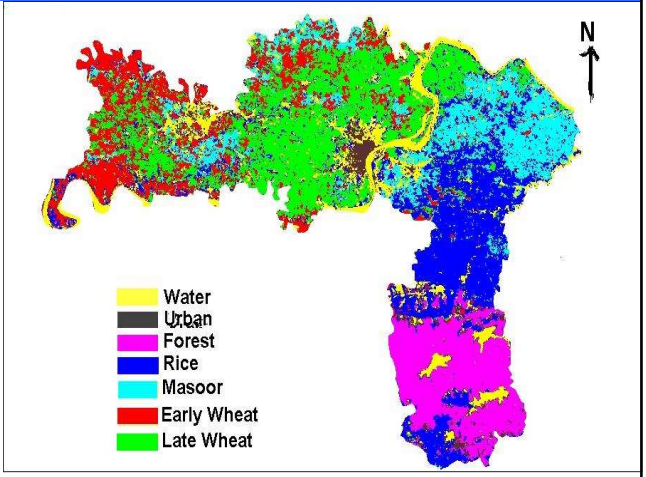
Coding output layer values

- Suppose a sample belongs to class 5
- Then the desired output layer node values will be
- 0 0 0 0 1 0 0 …
- Here the i^{th} node output should be 1 if the sample belongs to class i.



Low resolution image (WiFS)

Seven class classification using 3-layer neural network



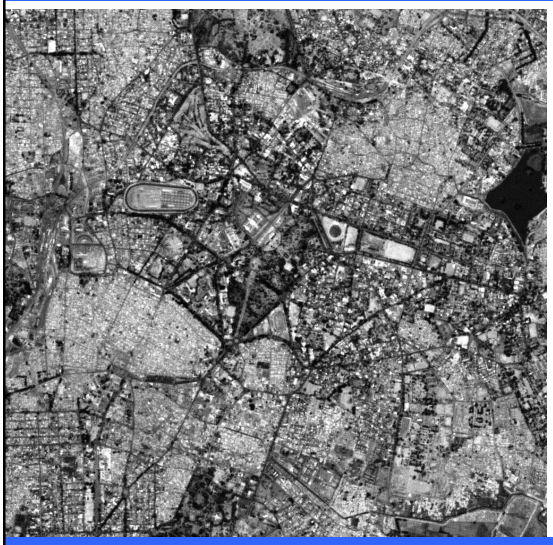
Neural network classification

MUMBAI

Data: IRS-1C, PAN

Consists of 1024x1024 pixels.

Other Examples

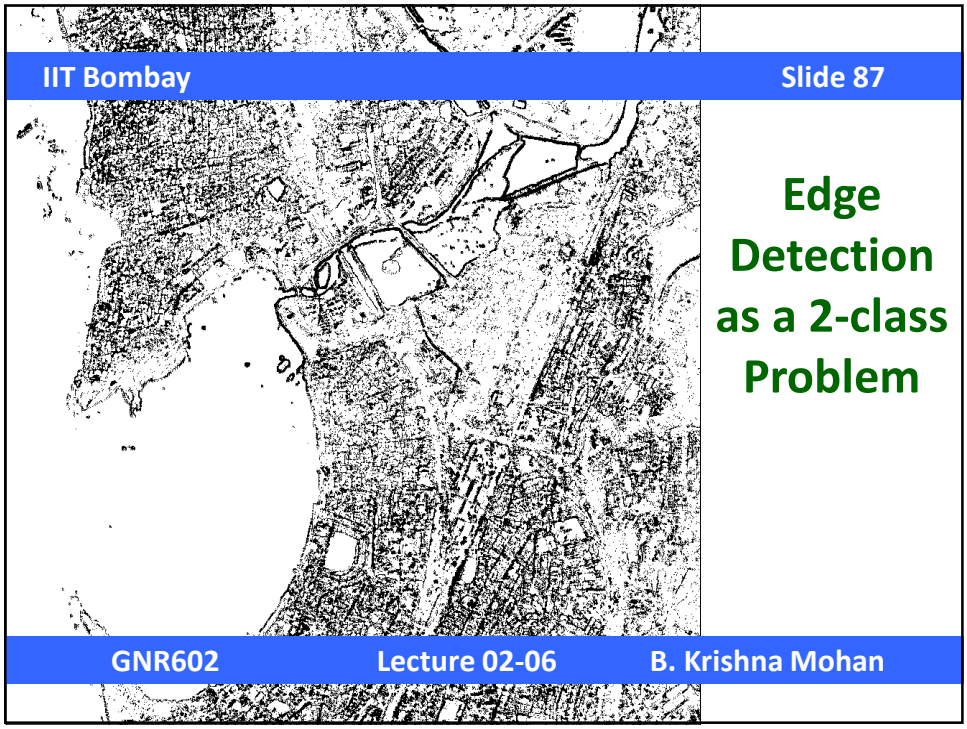


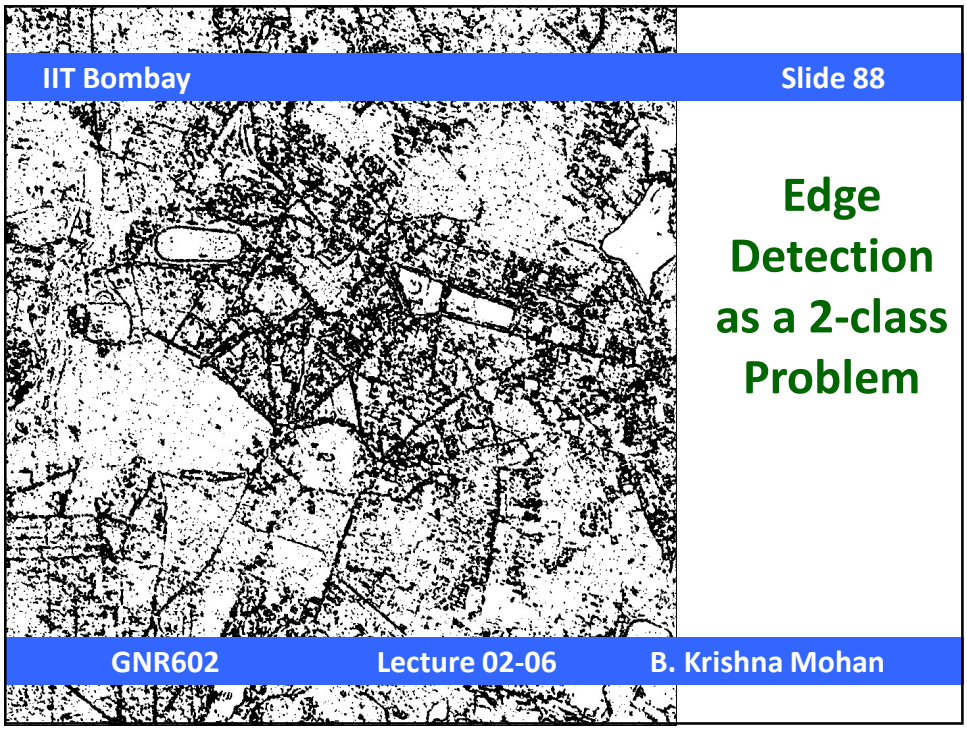
Bangalore

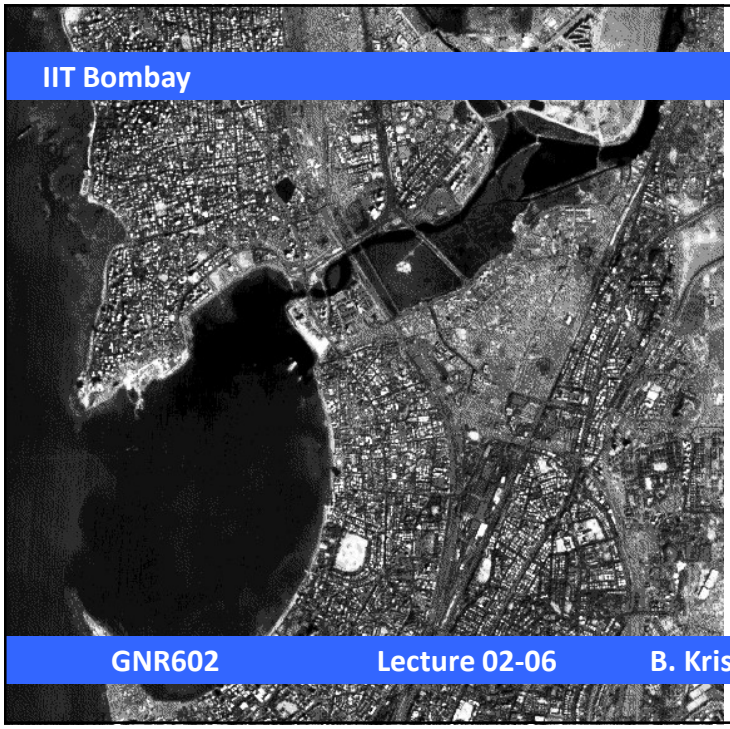
Data: SPOT, PLA

Consists of 1024x1024 pixels.

Other Examples


IIT Bombay
Slide 87
Edge Detection as a 2-class Problem
GNR602
Lecture 02-06
B. Krishna Mohan


IIT Bombay
Slide 88
Edge Detection as a 2-class Problem
GNR602
Lecture 02-06
B. Krishna Mohan



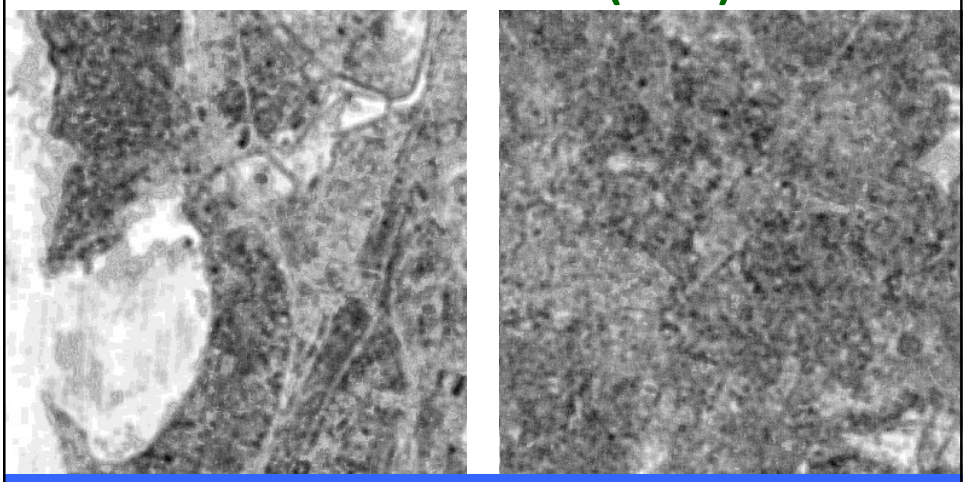
IIT Bombay Slide 89

Texture Analysis

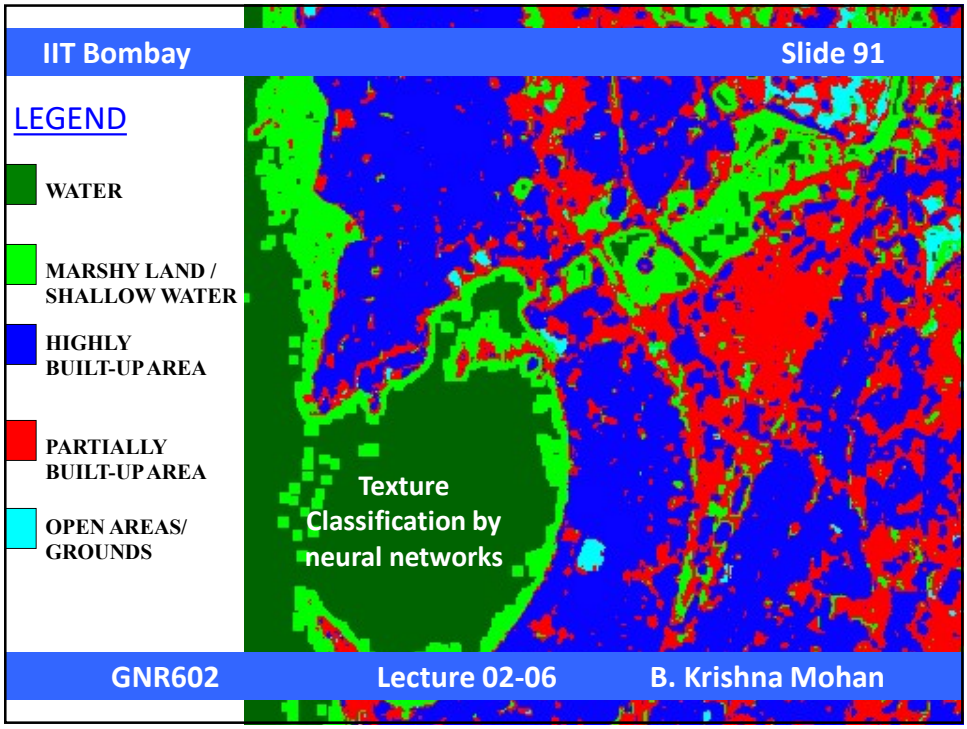
GNR602 Lecture 02-06 B. Krishna Mohan

IIT Bombay Slide 90

Texture Feature (IDM)



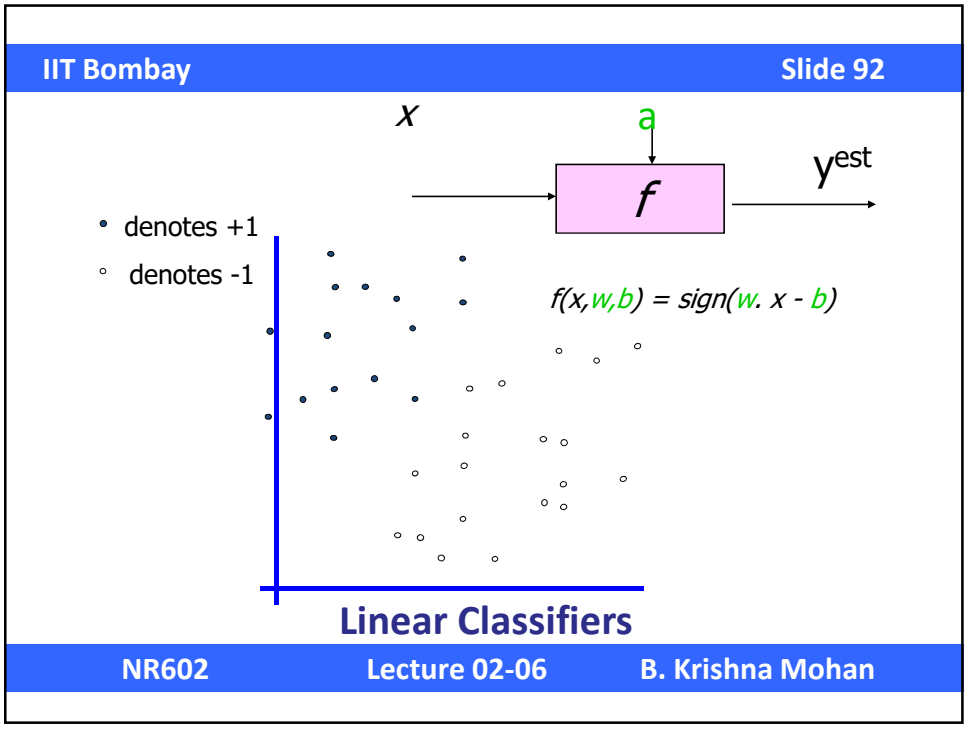
GNR602 Lecture 02-06 B. Krishna Mohan

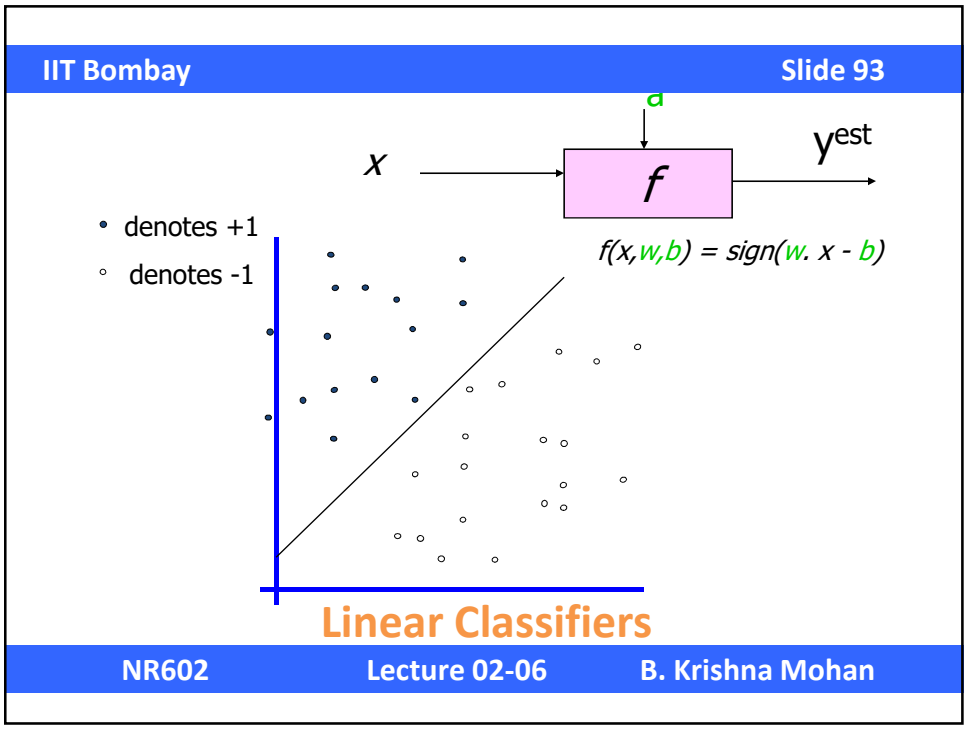


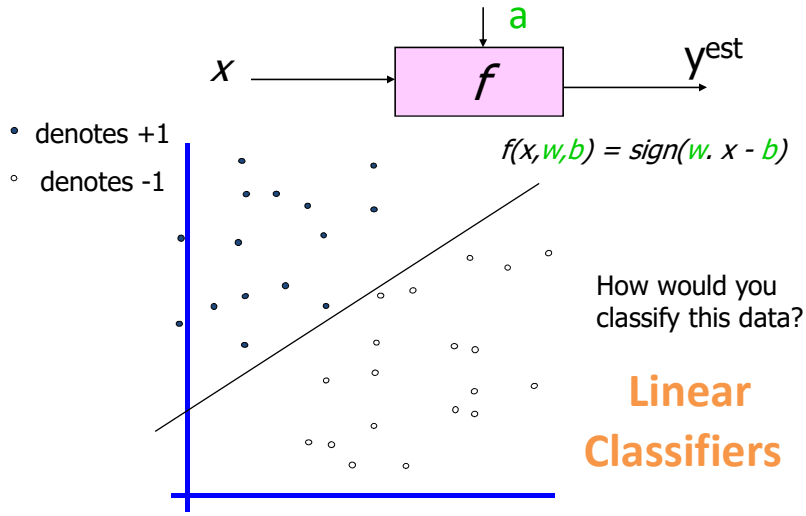
Support Vector Machines

Source: Prof. Andrew Moore's tutorials on SVM

Used with permission granted by Prof. Moore to instructors

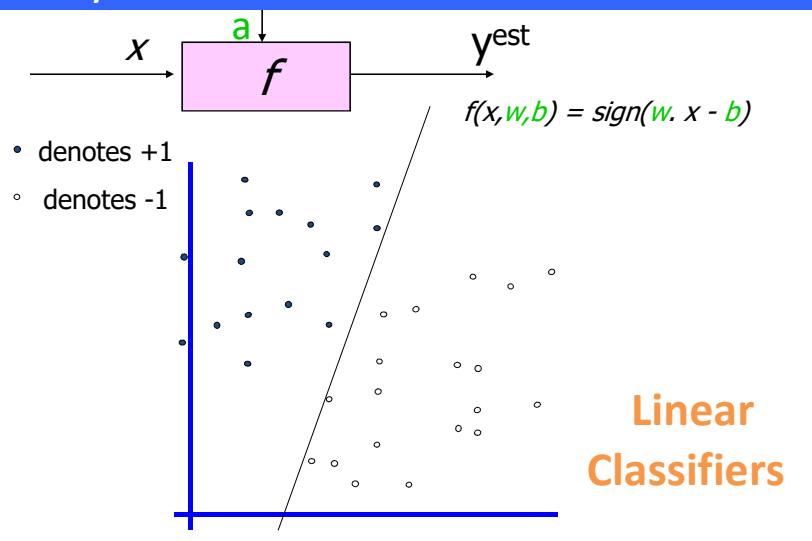

IIT Bombay
Slide 92
x
a
f
yest
denotes +1
denotes -1
f(x,w,b) = sign(w. x - b)
Linear Classifiers
NR602
Lecture 02-06
B. Krishna Mohan


IIT Bombay
Slide 93
a
x
f
yest
denotes +1
denotes -1
f(x,w,b) = sign(w. x - b)
Linear Classifiers
NR602
Lecture 02-06
B. Krishna Mohan

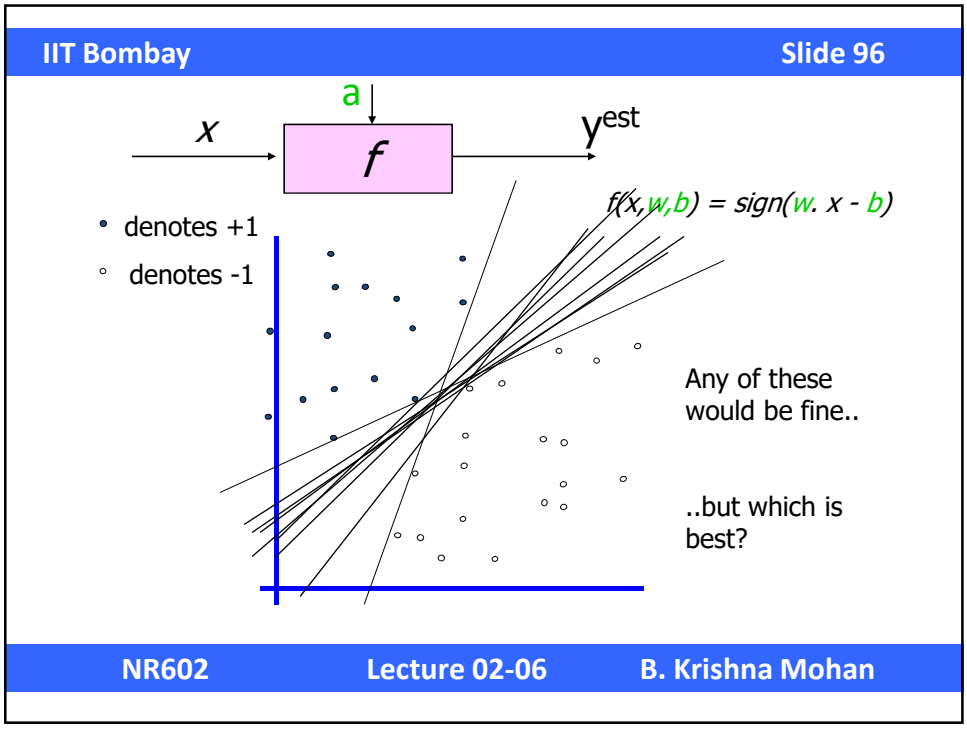


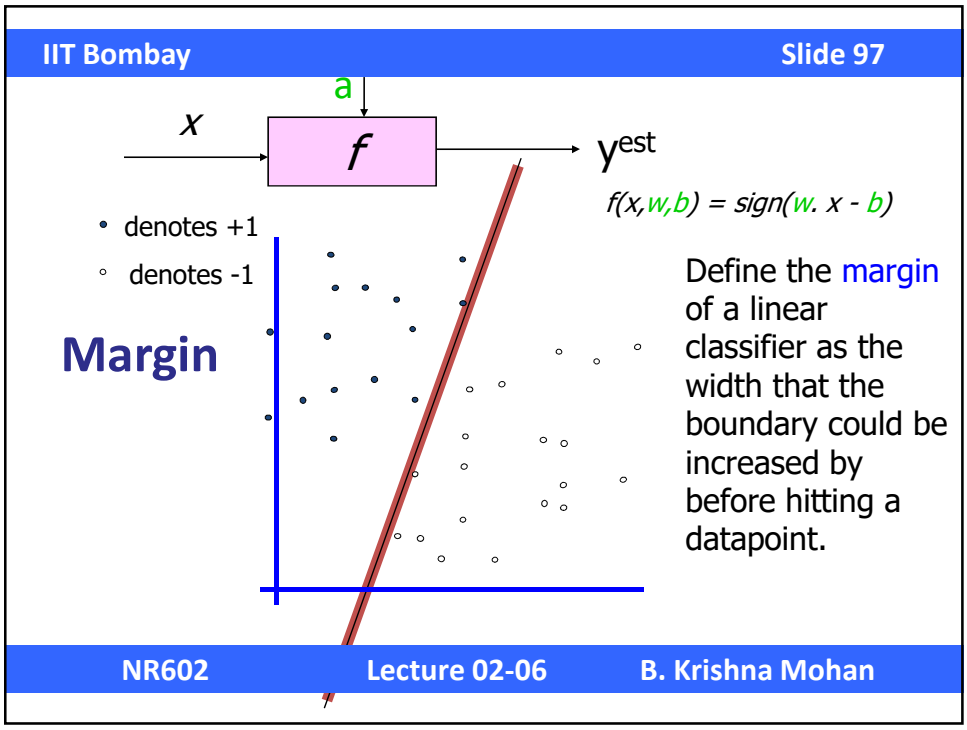
How would you classify this data?

Linear Classifiers

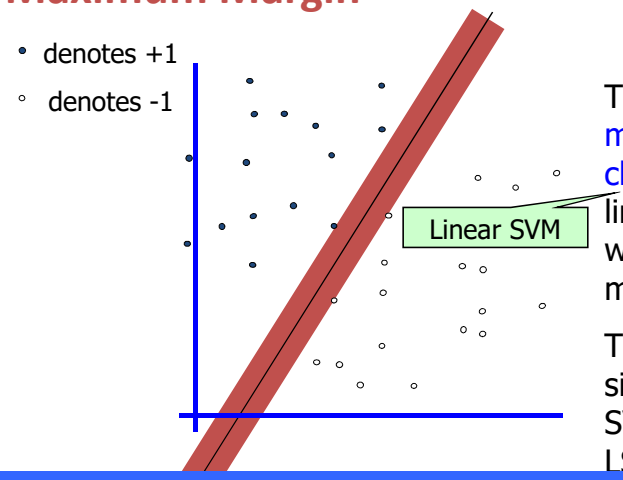


Linear Classifiers


IIT Bombay
Slide 96
a
x
f
y^est
f(x,w,b) = sign(w. x - b)
denotes +1
denotes -1
Any of these would be fine..
..but which is best?
NR602
Lecture 02-06
B. Krishna Mohan


IIT Bombay
Slide 97
a
x
f
y^est
f(x,w,b) = sign(w. x - b)
denotes +1
denotes -1
Margin
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.
NR602
Lecture 02-06
B. Krishna Mohan

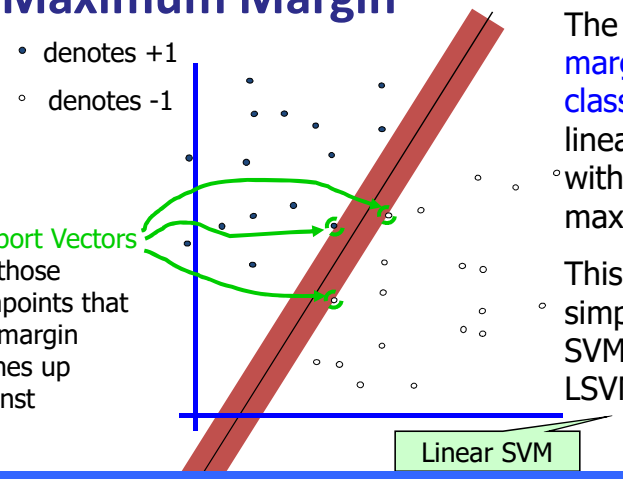
Maximum Margin



The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Maximum Margin

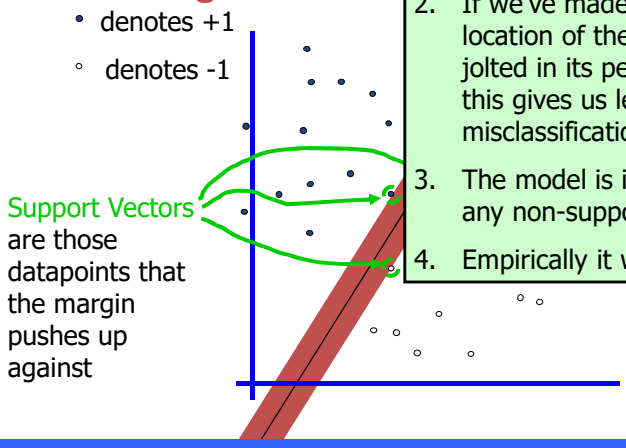


Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

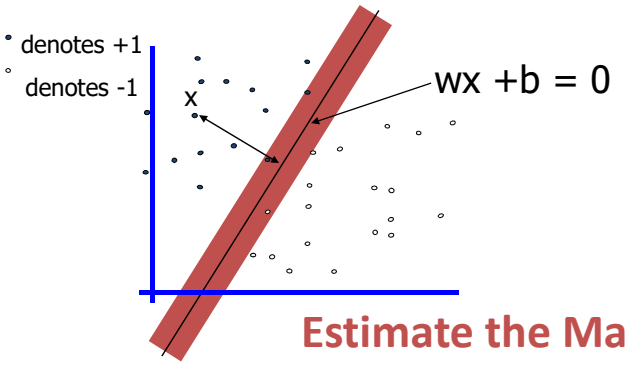
This is the simplest kind of SVM (Called an LSVM)

Why Maximum Margin?



1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. The model is immune to removal of any non-support-vector datapoints.
4. Empirically it works very very well.

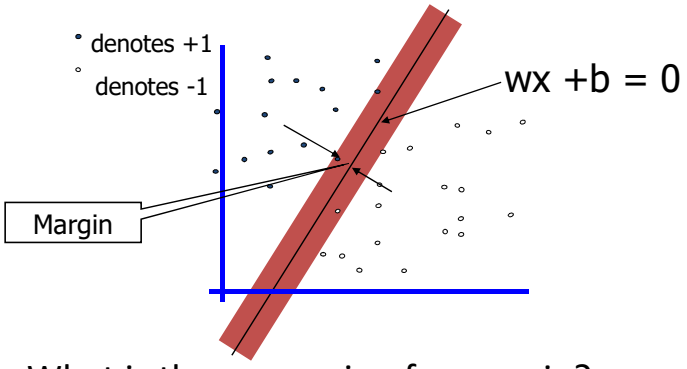
This is the simplest kind of SVM (Called an LSVM)



Estimate the Margin

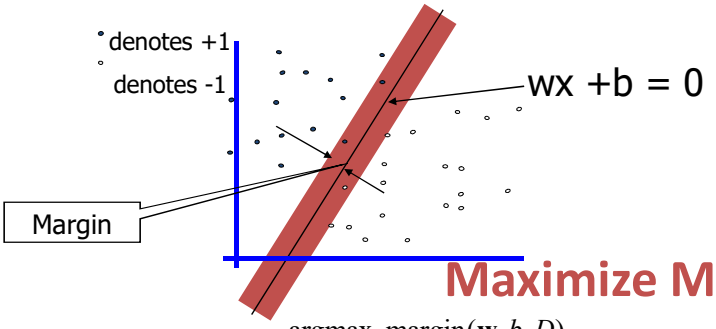
- What is the distance expression for a point **x** to a line **wx**+b= 0?

$$d(\mathbf{x}) = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$



- What is the expression for margin?

$$\text{margin} \equiv \arg\min_{\mathbf{x}\in D} d(\mathbf{x}) = \arg\min_{\mathbf{x}\in D} \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

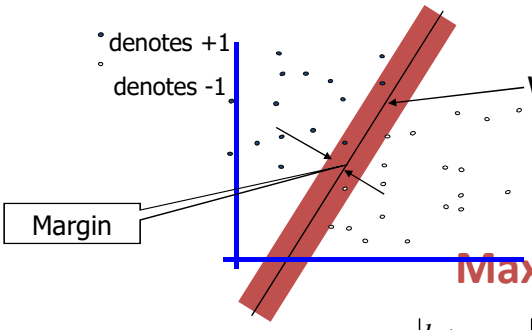


Maximize Margin

$$\arg\max_{\mathbf{w},b} \text{margin}(\mathbf{w},b,D)$$

$$= \arg\max_{\mathbf{w},b} \arg\min_{\mathbf{x}_i\in D} d(\mathbf{x}_i)$$

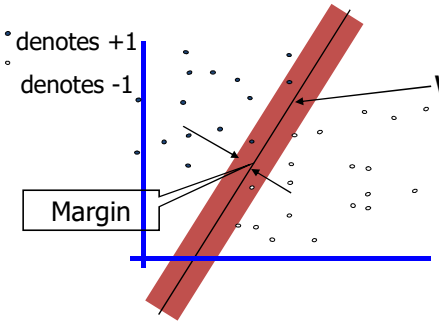
$$= \arg\max_{\mathbf{w},b} \arg\min_{\mathbf{x}_i\in D} \frac{|b+\mathbf{x}_i\cdot\mathbf{w}|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$



Maximize Margin

$$\underset{\mathbf{w},b}{\text{argmax}} \; \underset{\mathbf{x}_i \in D}{\arg\min} \frac{|b + \mathbf{x}_i \cdot \mathbf{w}|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

subject to $\forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) > 0$



$$\underset{\mathbf{w},b}{\text{argmax}} \; \underset{\mathbf{x}_i \in D}{\arg\min} \frac{|b + \mathbf{x}_i \cdot \mathbf{w}|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

subject to $\forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 0$

Maximize Margin

Strategy:

$\forall \mathbf{x}_i \in D : \; |b + \mathbf{x}_i \cdot \mathbf{w}| \geq 1$

$$\underset{\mathbf{w},b}{\text{argmin}} \sum_{i=1}^{d} w_i^2$$

subject to $\forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 1$

$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b}{\text{argmax}} \sum_{k=1}^{d} w_k^2$$

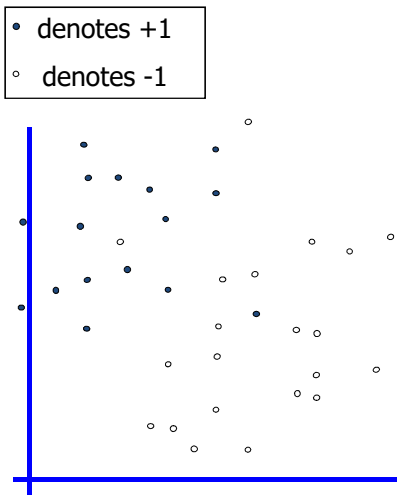
subject to

$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1$$
$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1$$
....
$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1$$

Maximum Margin Linear Classifier

Solution to this constrained optimization is needed

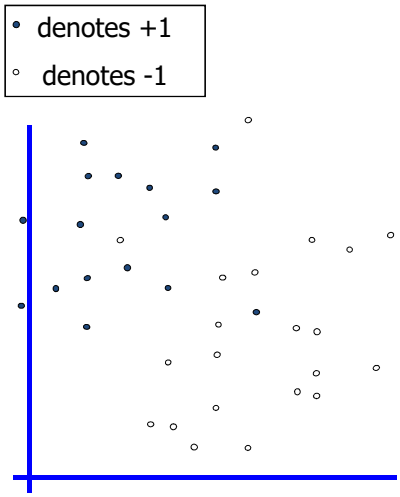
Slide 107



What if the data has noisy samples or outliers?

Perfect boundary in such a case with margin is not possible!

Slide 108



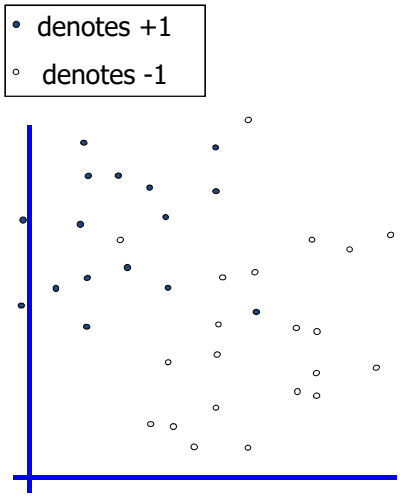
What should we do?

Idea 1:

Find minimum *w.w*, while minimizing number of training set errors.

Problemette: Two things to minimize simultaneously makes for an ill-defined optimization

Slide 109



What should we do?

Idea 2.0:

Minimize
w.w + C (distance of error points to their correct place)

Support Vector Machine (SVM) for Noisy Data

Slide 110

$$\{\vec{w}^*, b^*\} = \min_{\vec{w},b} \sum_{i=1}^{d} w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

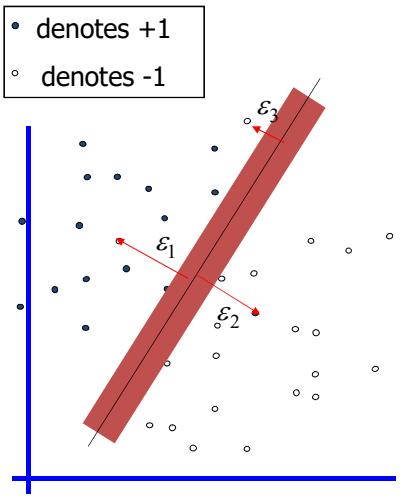
$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1 - \varepsilon_1$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1 - \varepsilon_2$$

...

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1 - \varepsilon_N$$

- Any problem with the above formalism?



Support Vector Machine (SVM) for Noisy Data

Slide 111

$$\{\vec{w}^*, b^*\} = \min_{\vec{w},b} \sum_{i=1}^{d} w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

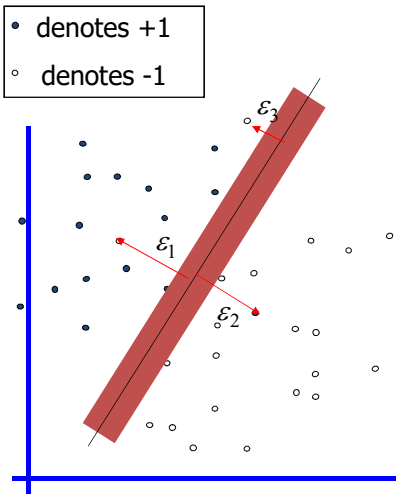
$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1 - \varepsilon_1, \varepsilon_1 \geq 0$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1 - \varepsilon_2, \varepsilon_2 \geq 0$$

...

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1 - \varepsilon_N, \varepsilon_N \geq 0$$

- Balance the trade off between margin and classification errors



Support Vector Machine for Noisy Data

Slide 112

$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b}{\operatorname{argmin}} \sum_i w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

$$\left.\begin{array}{l} y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1-\varepsilon_1, \varepsilon_1 \geq 0 \\ y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1-\varepsilon_2, \varepsilon_2 \geq 0 \\ \ldots. \\ y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1-\varepsilon_N, \varepsilon_N \geq 0 \end{array}\right\} \text{inequality constraints}$$

How do we determine the appropriate value for *c*?

The Dual Form of QP

Slide 112

Maximize $\sum_{k=1}^{R}\alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l(\mathbf{x}_k \cdot \mathbf{x}_l)$

Subject to these constraints: $0 \leq \alpha_k \leq C \quad \forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

Then classify with:

f(x,w,b) = sign(w. x - b)

Slide 113

The Dual Form of QP

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

Slide 114

An Equivalent QP

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k . \mathbf{x}_l)$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

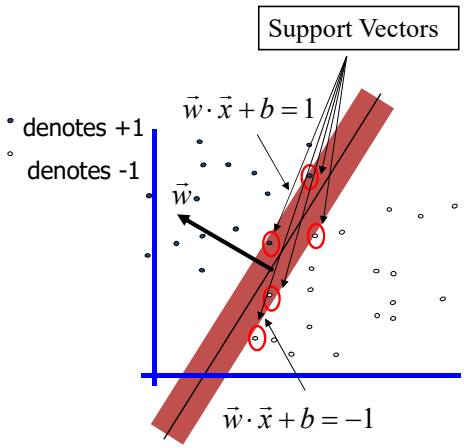
$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

Datapoints with $\alpha_k > 0$ will be the support vectors

..so this sum only needs to be over the support vectors.

Slide 115

Support Vectors



$$\forall i:\alpha_i\left(y_i\left(\vec{w}\cdot\vec{x}_i+b\right)-\left(1-\varepsilon_i\right)\right)=0$$

α_i = 0 for non-support vectors

$\alpha_i \neq 0$ for support vectors

$$\mathbf{w}=\sum_{k=1}^{R}\alpha_k y_k \mathbf{x}_k$$

Decision boundary is determined only by those support vectors !

Slide 116

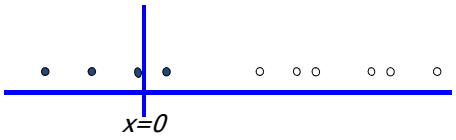
So, what is *soft* here in SVM?

- Simple SVM determines a class boundary such that one side of the boundary is class 1 and on the other side class 2. The boundary is determined such that there is maximum margin available for samples to vary beyond the training data range
- In case of noisy data or with outliers, it will not be possible to compute a boundary with strict conditions. A soft margin is determined such that a small number of samples are allowed to be misclassified for an overall good solution

Slide 117

Suppose we're in 1-dimension

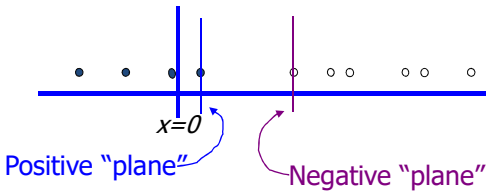
What would SVMs do with this data?



Slide 118

Suppose we're in 1-dimension

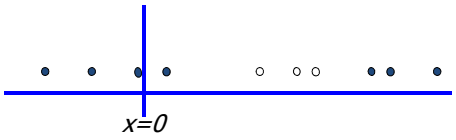
Not a big surprise



Slide 119

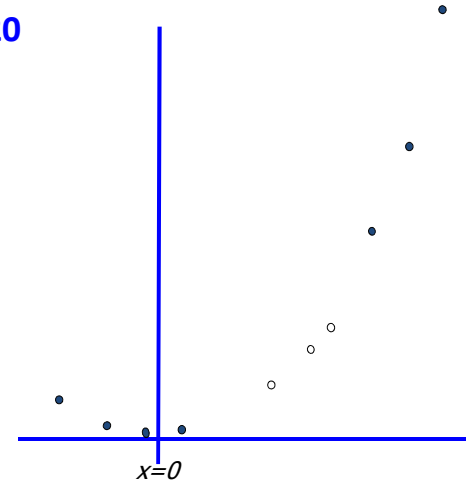
Harder 1-dimensional dataset

What can be done about this?



Harder 1-dimensional dataset

Slide 120



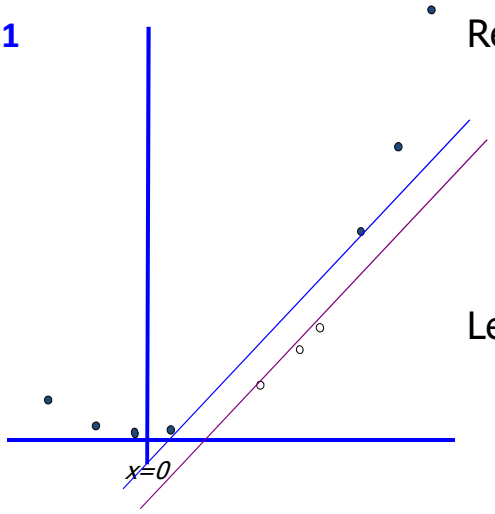
Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Harder 1-dimensional dataset

Slide 121



Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Common SVM basis functions

Slide 122

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 1 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \exp\left(-\frac{|\mathbf{x}_k - \mathbf{c}_j|^2}{\sigma^2}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

Slide 123

SVM Kernel Functions

- $K(\mathbf{a},\mathbf{b})=(\mathbf{a} \cdot \mathbf{b} +1)^d$ is an example of an SVM Kernel Function
- Beyond polynomials there are other very high dimensional basis functions that can be made practical by finding the right Kernel Function
  - Radial-Basis-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^2}{2\sigma^2}\right)$$

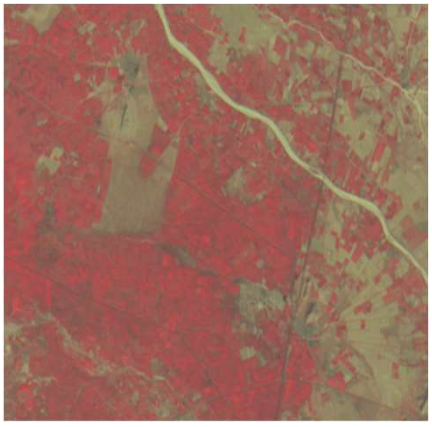
  - Neural-net-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \tanh(\kappa \mathbf{a}.\mathbf{b} - \delta)$$

Slide 124

Classification Results

- This is raw image of Khimara region
- IRS – 1B LISS II data having spatial resolution 36.25 m
- Four band data
- Image size is 300 x 400 pixels



Slide 125

Classification Results Contd..

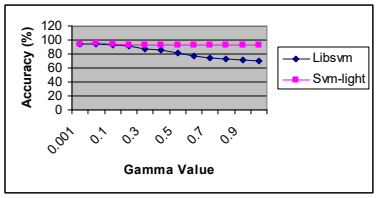
- Classification results of Khimara region using RBF kernel function for different values of γ
- Both software give optimum accuracy at γ=0.01

Gamma value (γ)	%Classification accuracy (using Libsvm)	%Classification accuracy (using SVM^{light})
0.001	93.68	94.21
0.01	94.21	95.26
0.1	93.16	93.68
0.2	91.05	93.16
0.3	87.37	93.16
0.4	85.26	93.16
0.5	81.58	93.16
0.6	77.37	93.16
0.7	74.74	93.16
0.8	73.16	93.16
0.9	71.05	93.16
1.0	70.00	93.16

Slide 126

Classification Results Contd..

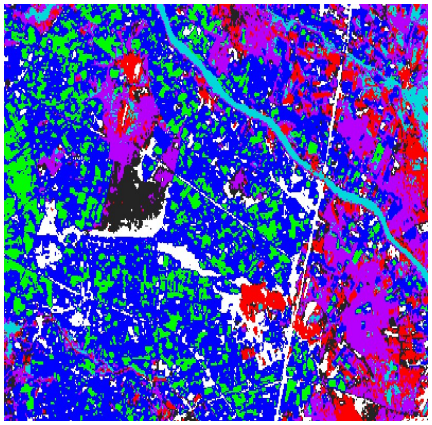
- Comparison of results of Khimara region with various Gamma values
- Slope of the curve in case of libsvm is more than that of SVMl^{light}



Slide 127

Classification Results Contd..

- Classified image of Khimara region (Libsvm)
- Classified into 7 classes



Slide 128

Classification Results Contd..

- Classification results of Khimara region
- In case of linear and polynomial kernel the performance of Libsvm is better than SVM^{light}
- In RBF kernel the performance of Libsvm and SVM^{light} are comparable and better than ANN

Kernel function	% Classification accuracy (using Libsvm)	% Classification accuracy (using SVM^{light})	% Classification accuracy (using ANN)
Linear	93.16	86.84	-
Polynomial (d = 2)	93.16	87.89	-
Radial basis	94.21	95.26	90.72

Slide 129

Classification Results Contd..

- This is raw image of Thane region
- Landsat TM data having spatial resolution 30 m
- Four band data
- Image size is 440 x 420 pixels

Slide 130

Classification Results Contd..

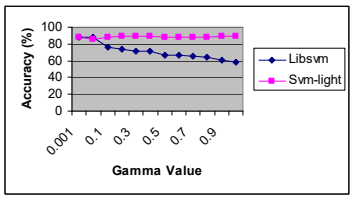
- Classification results of Thane region using RBF kernel function for different values of γ
- Libsvm gives optimum accuracy at γ=0.001 whereas SVM^{light} perform consistently

Gamma value (γ)	%Classification accuracy (using Libsvm)	%Classification accuracy (using SVM^{light})
0.001	88.44	88.33
0.01	87.78	86.11
0.1	76.11	88.33
0.2	73.33	88.89
0.3	71.67	88.89
0.4	71.11	88.89
0.5	67.22	88.33
0.6	66.11	88.33
0.7	65.00	87.78
0.8	64.44	88.33
0.9	61.11	88.89
1.0	58.89	88.89

Slide 131

Classification Results Contd..

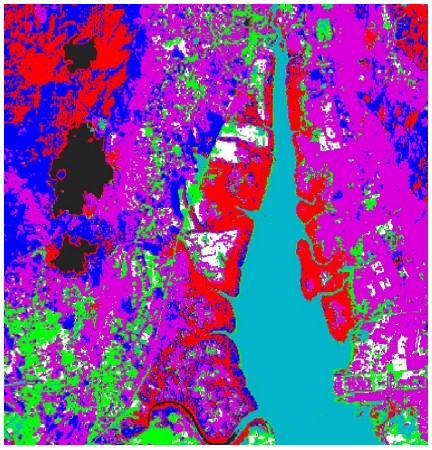
- Comparison of results of Thane region with various Gamma values
- Slope of the curve in case of libsvm is more than that of SVMl[light]



Slide 132

Classification Results Contd..

- Classified image of Thane region (Libsvm)
- Classified into 7 classes



Slide 133

Classification Results Contd..

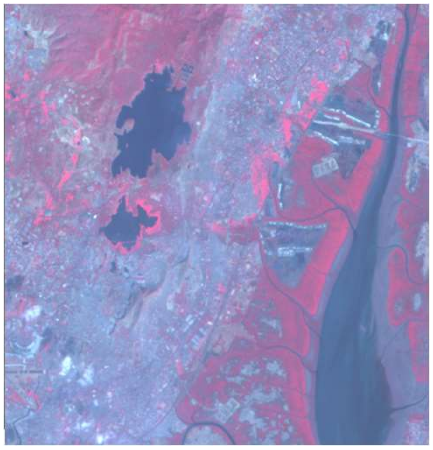
- Classification results of Thane region
- In case of linear and polynomial kernel the performance of Libsvm is better than SVM^{light}
- In RBF kernel the performance of Libsvm and SVM^{light} are comparable and better than ANN

Kernel function	% Classification accuracy (using Libsvm)	% Classification accuracy (using SVM^{light})	% Classification accuracy (using ANN)
Linear	89.44	72.22	-
Polynomial (d = 2)	90.00	68.33	-
Radial basis	88.44	88.89	80.56

Slide 134

Classification Results

- This is raw image of Powai region
- IRS – 1C LISS III data having Spatial resolution 23.5 m
- Four band data
- Image size is 1024 x 1024 pixels



Slide 135

Classification Results Contd..

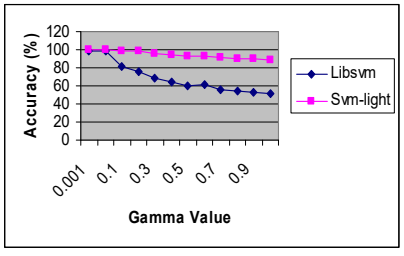
- Classification results of RBF kernel to assess optimum γ value
- Both software give optimum accuracy at γ= 0.01

Gamma value (γ)	%Classification accuracy (using Libsvm)	%Classification accuracy (using SVM^{light})
0.001	98.09	99.52
0.01	98.09	99.52
0.1	81.90	99.05
0.2	75.71	98.09
0.3	68.10	96.19
0.4	63.81	94.76
0.5	60.47	93.33
0.6	61.90	92.38
0.7	55.71	91.90
0.8	54.28	90.00
0.9	53.33	89.52
1.0	51.43	88.57

Slide 136

Classification Results Contd..

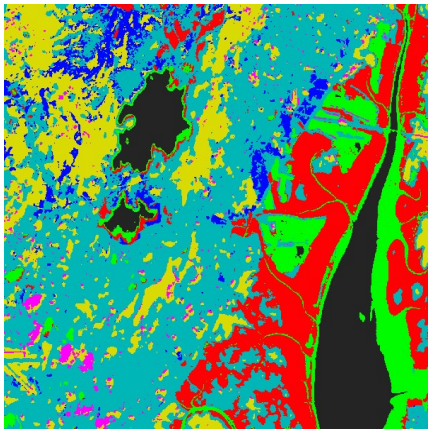
- Comparison of results of Powai region with various Gamma values
- Large variation is seen in case of Libsvm



Slide 137

Classification Results Contd..

- Classified image of Powai region (Libsvm)
- Classified into 7 classes



Slide 138

Classification Results Contd..

- Classification results of Powai region
- Linear and polynomial kernel of Libsvm is better than SVM^{light}
- Performance in RBF kernel is comparable

Kernel function	% Classification accuracy (using Libsvm)	% Classification accuracy (using SVM^{light})	% Classification accuracy (using ANN)
Linear	99.52	91.90	-
Polynomial (d = 2)	99.52	91.42	-
Radial basis	98.09	99.05	100.00

Slide 139

Classification Results Contd..

- This is raw image of Sally region
- IRS – 1C LISS III data having spatial resolution 23.5 m
- Four band data
- Image size is 600 x 400 pixels

Slide 140

Classification Results Contd..

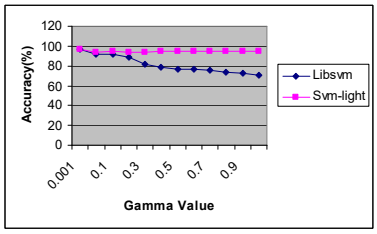
- Classification results of Sally region using RBF kernel function for different values of γ
- Both the software give optimum accuracy at γ=0.001

Gamma value (γ)	%Classification accuracy (using Libsvm)	%Classification accuracy (using SVM^{light})
0.001	96.97	96.97
0.01	92.12	93.94
0.1	92.12	95.15
0.2	88.48	93.94
0.3	81.82	93.94
0.4	78.18	94.55
0.5	76.36	94.55
0.6	76.36	94.55
0.7	75.15	94.55
0.8	73.94	95.15
0.9	72.73	95.15
1.0	70.30	95.15

Slide 141

Classification Results Contd..

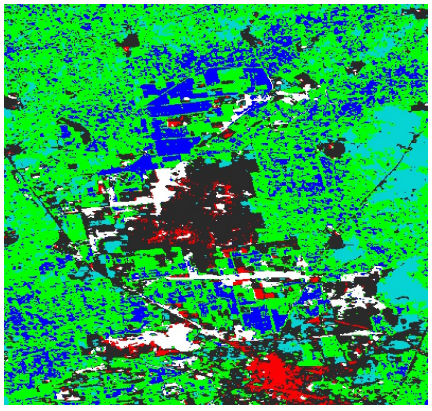
- Comparison of results of Sally region with various Gamma values
- Slope of the curve in case of libsvm is more than that of SVMI[light]



Slide 142

Classification Results Contd..

- Classified image of Sally region (Libsvm)
- Classified into 6 classes



Slide 143

Classification Results Contd..

- Classification results of Sally region
- In case of linear and polynomial kernel the performance of Libsvm is better than SVM^{light}
- In RBF kernel the performance of Libsvm and SVM^{light} are comparable and better than ANN

Kernel function	% Classification accuracy (using Libsvm)	% Classification accuracy (using SVM^{light})	% Classification accuracy (using ANN)
Linear	94.54	85.45	-
Polynomial (d = 2)	96.36	38.18	-
Radial basis	96.97	96.97	91.56

Multiclass Classification

- 1-class at a time
- Separate every class from every other class
- 1-class at a time ➔ One v/s Rest
- N-1 SVMs will yield N classes in case of One v/s Rest
- Every class separated from every other class ➔ One v/s One approach
- N_{C_2} SVMs will be needed for One V/s One approach
- Majority rule is used in One V/s One SVM classification since each sample is classified by multiple SVMs

Slide 143a

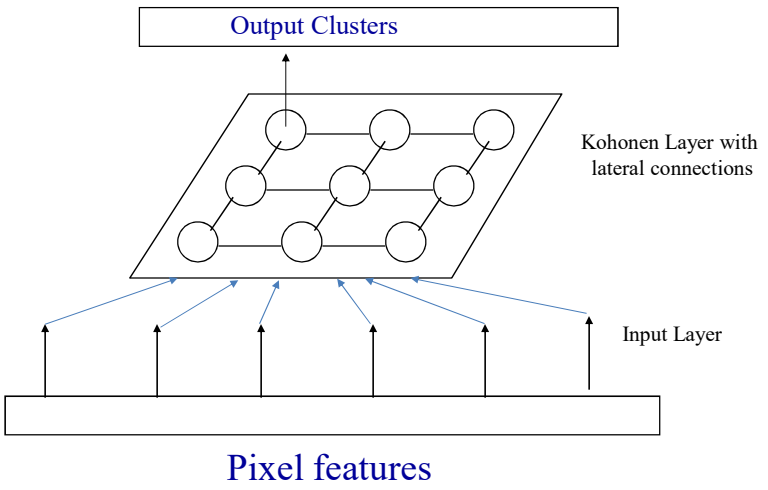
Unsupervised Learning by Kohonen Network

Principle of Self-Organization

- "The principal goal of the self-organizing map (SOM) is to transform an incoming signal pattern of arbitrary dimension into a one- or two-dimensional discrete map and to perform this transformation adaptively in a topologically ordered fashion" – From Haykin's book, p. 428
- Multi-dimensional data can be mapped onto a two-dimensional surface
  - Facilitates representation of clusters in the data

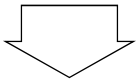
Unsupervised Learning

- Unsupervised Networks
- Closely related to clustering
- Do not require target outputs for each input vector in the training data
- Inputs are connected to a two-dimensional grid of neurons
  - Neighbourhood relations can be explicitly maintained, or
  - each neuron can have lateral connections to its 'neighbours'
- Multi-dimensional data can be mapped onto a two-dimensional grid
  - Facilitates representation of clusters in the data

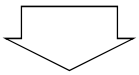


Kohonen Network

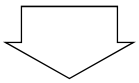
Identify winner through competitive learning



Identify neighbors for cooperative learning

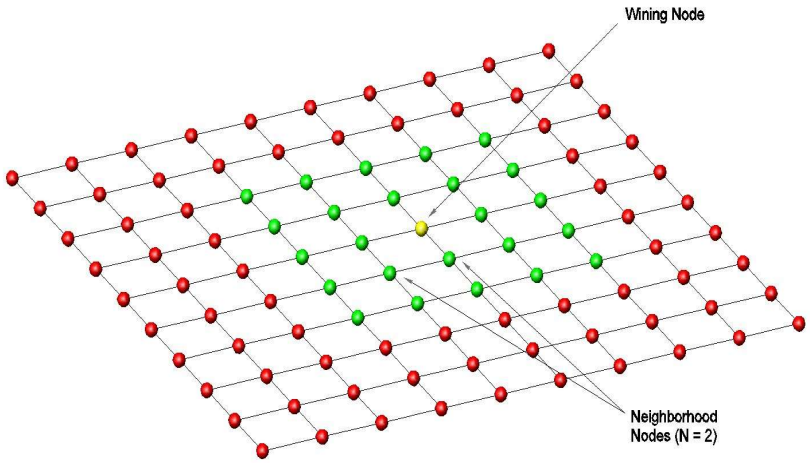


Update all the neurons through weight updation rule



Iterate the process till convergence ➔ stabilized weights

Kohonen Update Process



Winning Neuron and Neighborhood

Winning Neuron

Output unit with the lowest Euclidean distance to the input pixel feature vector is chosen:

$$\| X - W_k \| = \min_i \{\| X - W_i \|\}$$

k denotes the 'winning' neuron in the Kohonen layer

"Competitive Learning"

There are two types of competitive learning: hard and soft.

Hard competitive learning is essentially a "winner-take-all" scenario. The winning neuron is the only one which receives any training response.

Soft cooperative learning / cooperative learning involves sharing with neighbors

Simple Weight Increment

Update weights of the winning neuron to reduce difference With input neuron

$$\Delta w_{ij} = \alpha_t (x_j - w_{ij})$$

α_t - is a learning rate parameter

This is purely competitive learning, where just the winning neuron is updated, without affecting neighboring neurons around the winner

Simple Weight Updation

Update weights of winning neuron and its neighbors

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$

Reduce learning rate and neighbourhood size with iterations

This is purely competitive learning, where just the winning neuron is updated, without affecting neighboring neurons around the winner

Learning Rate

Initial learning rate is usually set high
e.g., 0.6 and decreased gradually

$$\alpha_t = \alpha_0 (1 - (t/T))$$

T - total number of training iterations

t - is the current training iteration

α_t - is the learning rate for the current training iteration

α_0 - is the initial value of the learning rate

Sharing with Neighborhood

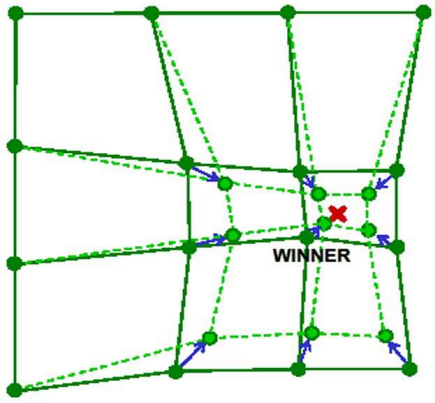
The weights of neighboring neurons of the winner are also updated

Highest update to the winner's weights, less to the neighbors

Farther a neighbour, smaller the update of the weights

The size of the neighbourhood is reduced linearly with each iteration; the update of the neighbour's weight vector is a scale factor (< 1) times the winner's update

Shrinking size of neighborhood with iteration



Find the closest node to the presented feature vector and some in the neighborhood, and stimulate them by making them a little more like the presented feature vector. The size of the neighborhood shrinks iteratively

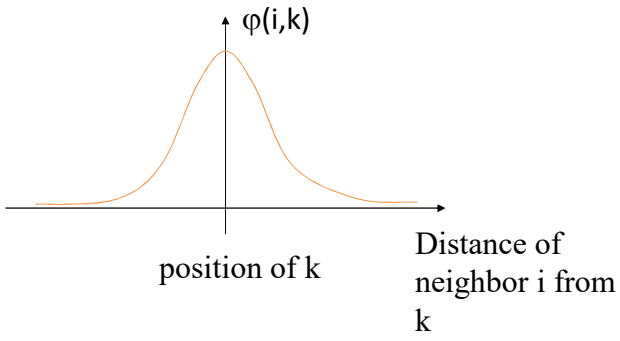
Effect on Neighborhood

An interpolation function is used so that the neuron with the lowest difference with the input feature vector fires with the highest value, and a pre-determined number of other neurons which are the next closest to the winner fire with lower values

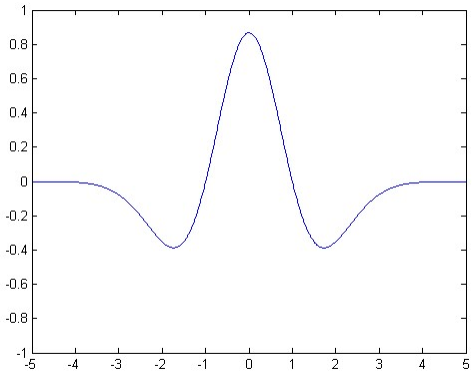
The size of the neighbourhood decreases with learning, largest at the first iteration to minimum (3x3) at the time of convergence

Neighbor scaling function

A neighborhood function φ(i, k) indicates the influence of the winning neuron k on neighboring neuron i as a function of physical separation (distance) between the two on the SOM grid



Another Sample Neighborhood Function

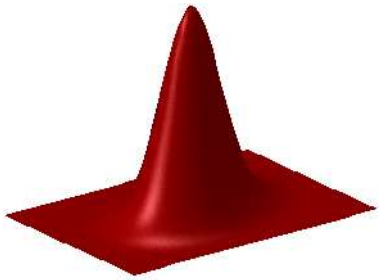


$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1-x^2)e^{-x^2/2}$$

x is the distance between winning neuron and neighbor

Reducing size of the neighborhood

$$\phi_t(r) = e^{-\frac{1}{2}(\frac{r^2}{\sigma_t^2})}$$



$$\sigma_t = \sigma_0(1-(t/T))$$

Width of the function decreases with iterations, by decreasing value of σ

T is total iterations considered, t is current iteration

Overall Algorithm

(1) Choose random weight vector $\mathbf{w}_i$ for neuron i, i = 1, ..., m

(2) Input next feature vector $\mathbf{x}$

(3) Determine winner neuron k:

$||\mathbf{w}_k - \mathbf{x}|| = \min_i ||\mathbf{w}_i - \mathbf{x}||$ (Euclidean distance)

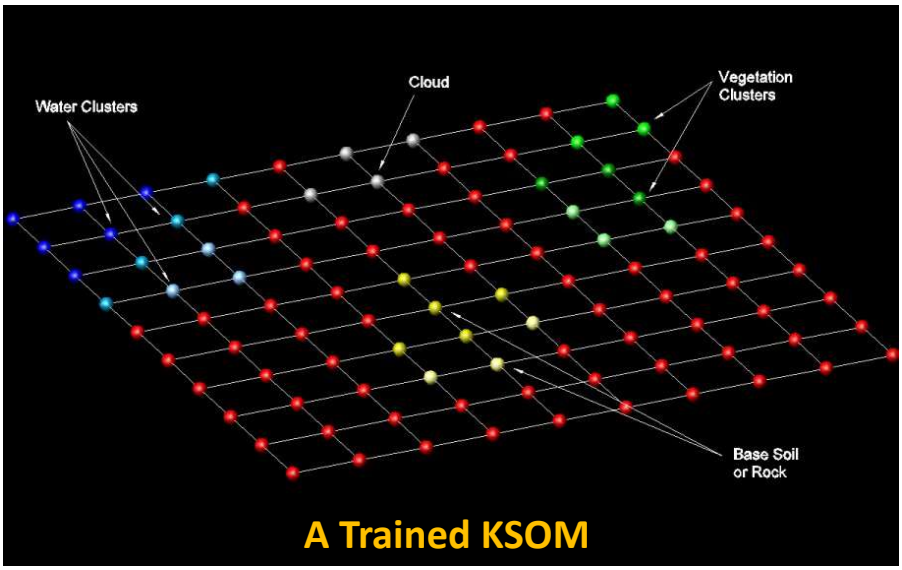
(4) Update all weight vectors of all neurons i in the neighborhood of neuron k:

$\mathbf{w}_i(\text{new}) := \mathbf{w}_i(\text{old}) + \alpha_t \cdot \varphi_t(i, k) \cdot (\mathbf{x} - \mathbf{w}_i(\text{old}))$

(w_i is shifted towards x)

(5) If all feature vectors are input, current iteration is completed.

(6) If convergence criterion met, STOP. Otherwise, reduce the neighborhood size φ_t, learning rate α_t and go to (2).



A Trained KSOM

Kohonen-style network training is also called "Competitive Learning"

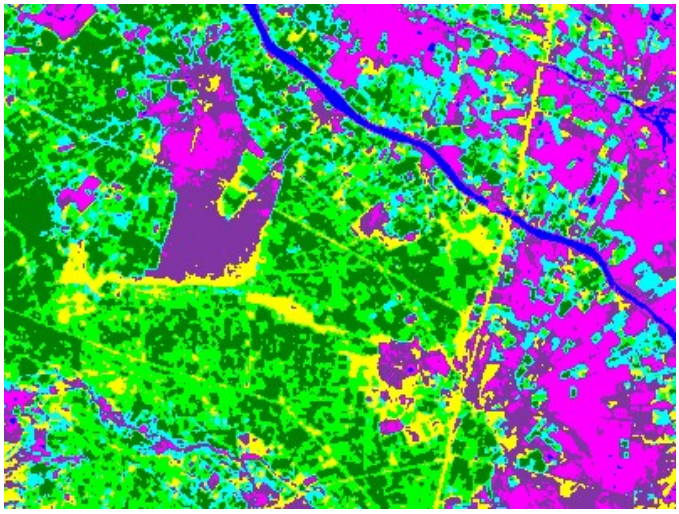
This is due to the fact that during a single training epoch, the cell with closest distance to the presented training vector is can be considered as the 'winner', and is thought to have successfully competed for the honor, hence the concept of "competitive learning".

When Training is Finished, Classify

- Take all of the test data from the original data set.
- Find the closest distance for each, and record the position of the neuron on the map that was closest.
- Create a cluster map, and analyze the groupings.
- Assign classes to the groups.
- Create a visually "classed" map.
- Analyze the errors.

IIT Bombay Slide 168

7-class classifi-cation using Kohonen network



GNR602 Lecture 02-06 B. Krishna Mohan

IIT Bombay Slide 169

7-class classifi-cation using Kohonen network

Dry Channel
Crop (greenary)
Habitat/Builtup
Other Vegetation
Open/ wet land
Ag. Field after Harvesting
Fodder

GNR602 Lecture 02-06 B. Krishna Mohan

Kohonen Network as a Compressor

- The principle is to use the final SOM grid as an encoded version of the input data
- As a compressor, used as vector quantizer of input feature vectors
- Each element of the self-organizing map acts as a code-word for a bunch of feature vectors
- The entire map is treated as a dictionary or codebook

Vector Quantization using SOM

In place of input feature vector, that may be of any dimension, or data type (integer, real, byte…), store the positive of the matching neuron. For example, if the 3rd row, 5th column neuron is the winner, store (3xcols+5) corresponding to the input feature vector

- To reconstruct, the weight vector located at the position of the winning neuron is output corresponding to the original feature vector.
- This is a lossy compression system. Bigger the size of the SOM grid, less is the distortion introduced by the vector quantization based compressor

Learning Vector Quantization

- LVQ slides borrowed from lecture materials of
- CS 476: Networks of Neural Computation
- Dr. Stathis Kasderidis
- Dept. of Computer Science
- University of Crete, Greece

This is additional material, included for general interest only, not part of the course content.

LVQ

- *Vector Quantisation* is a technique that exploits the underlying structure of input vectors for the purpose of data compression.
- An input space is divided in a number of distinct regions and for each region a reconstruction (representative) is defined.
- When the quantizer is presented with a new input vector, the region in which the vector lies is first determined, and is then represented by the reproduction vector for this region.
- The collection of all possible reproduction vectors is called the *code book* of the quantizer and its members are called *code words*.

LVQ

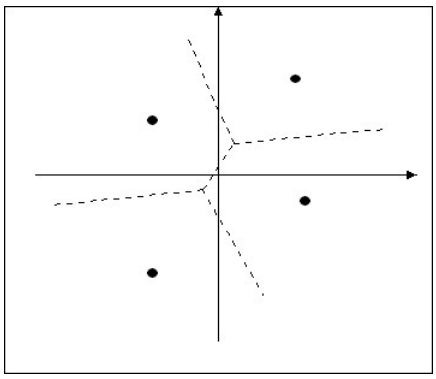
LVQ-1

LVQ

- A vector quantizer with minimum encoding *distortion* is called *Voronoi* or *nearest-neighbour quantizer*, since the Voronoi cells about a set of points in an input space correspond to a partition of that space according to the nearest-neighbour rule based on the Euclidean metric.
- An example with an input space divided to four cells and their associated Voronoi vectors is shown below:

LVQ-2

LVQ



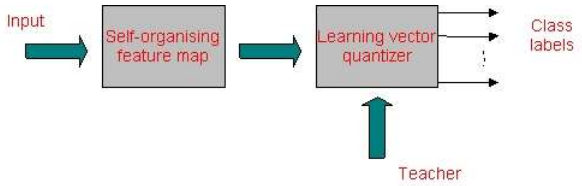
- The SOM algorithm provides an approximate method for computing the Voronoi vectors in an unsupervised manner, with the approximation being specified by the

LVQ-3

weight vectors of the neurons in the feature map.

•Computation of the feature map can be viewed as the first of two stages for adaptively solving a pattern classification problem as shown below. The second stage is provided by the learning vector quantization, which provides a method for fine tuning of a feature map.

LVQ



LVQ-4

•*Learning vector quantization* (LVQ) is a supervised learning technique that uses class information to move the Voronoi vectors slightly, so as to improve the quality of the classifier decision regions.

•An input vector **x** is picked at random from the input space. If the class labels of the input vector and a Voronoi vector **w** agree, the Voronoi vector is moved in the direction of the input vector **x**. If, on the other hand, the class labels of the input vector and the Voronoi vector disagree, the Voronoi vector **w** is moved away from the input vector **x**.

LVQ

•Let us denote $\{\mathbf{w}_j\}_{j=1}^{l}$ the set of Voronoi vectors, and let $\{\mathbf{x}_i\}_{i=1}^{N}$ be the set of input vectors. We assume that

LVQ-5

LVQ

N >> l.

- The LVQ algorithm proceeds as follows:

i. Suppose that the Voronoi vector $\mathbf{w}_c$ is the closest to the input vector $\mathbf{x}_i$. Let C_{wc} and C_{xi} denote the class labels associated with $\mathbf{w}_c$ and $\mathbf{x}_i$ respectively. Then the Voronoi vector $\mathbf{w}_c$ is adjusted as follows:

- If $C_{wc} = C_{xi}$ then

$$\mathbf{W}_c(n+1)= \mathbf{w}_c(n)+a_n[\mathbf{x}_i- \mathbf{w}_c(n)]$$

Where $0< a_n <1$

LVQ-6

LVQ

- If $C_{wc} \neq C_{xi}$ then

$$\mathbf{W}_c(n+1)= \mathbf{w}_c(n)-a_n[\mathbf{x}_i- \mathbf{w}_c(n)]$$

ii. The other Voronoi vectors are not modified.

- It is desirable for the learning constant a_n to decrease monotonically with time n. For example a_n could be initially 0.1 and decrease linearly with n.
- After several passes through the input data the Voronoi vectors typically converge at which point the training is complete.

Conclusions

- The SOM model is neurobiologically motivated and it captures the important features contained in an input space of interest.
- The SOM is also a vector quantizer.
- It supports the form of learning which is called *unsupervised* in the sense that no target information is given with the presentation of the input.
- It can be combined with the method of Learning Vector Quantization in order to provide a combined *supervised* learning technique for fine-tuning the Voronoi vectors of a suitable partition of the input space.

Conclusions

Contd…