ε.δε.μ²
ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ & ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Binary Classification - Kernel Methods

1. Separability of Patterns, Cover's Theorem
2. Radial-Basis Function (RBF) Networks
3. RBF Hybrid Learning
4. Support Vector Machines (SVM)

Prof. Vasilis Maglaris

maglaris@netmode.ntua.gr

www.netmode.ntua.gr

Room 002, New ECE Building

Tuesday May 6, 2025

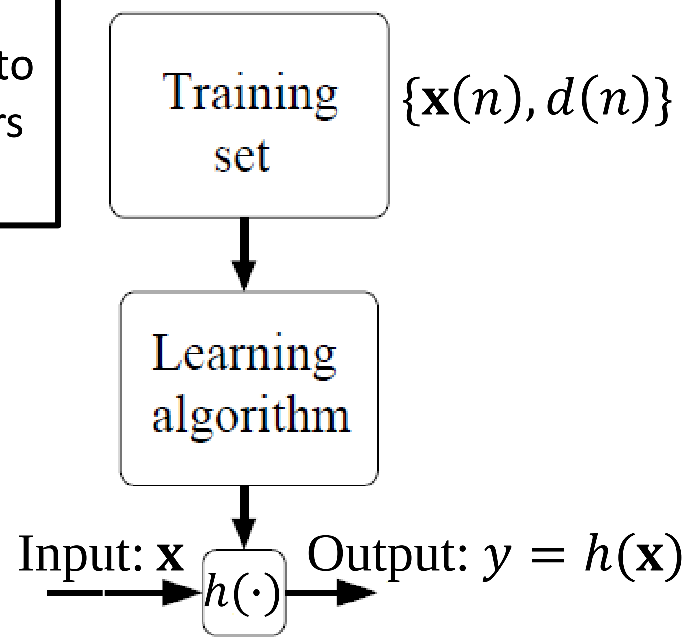
STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Generic Model of Supervised Learning (*repetition*)

Based on Andrew Ng, "CS229 Lecture Notes", Stanford University, Fall 2018

- The system goal is to assign input vectors (***input sample points***, ***examples***, ***instances***) $\mathbf{x} = [x_1\ x_2 \dots x_m]^{\mathrm{T}}$ to output values y (***targets***, ***response values***). The coordinates x_i encode m characteristics (***features***) of the input vector $\mathbf{x}$

We seek the input-output function $y = h(\mathbf{x}) \cong d$ that minimizes deviations (errors) between the ***label*** d (known to an external ***supervisor***) and the response y for input vectors in the ***Training Set*** of N pairs $\{\mathbf{x}(n), d(n)\}$, $n = 1,2,\dots,N$



- The form and parameters of $h(\cdot)$ result from the learning algorithm that converges to the system goal for the N elements of the training sample

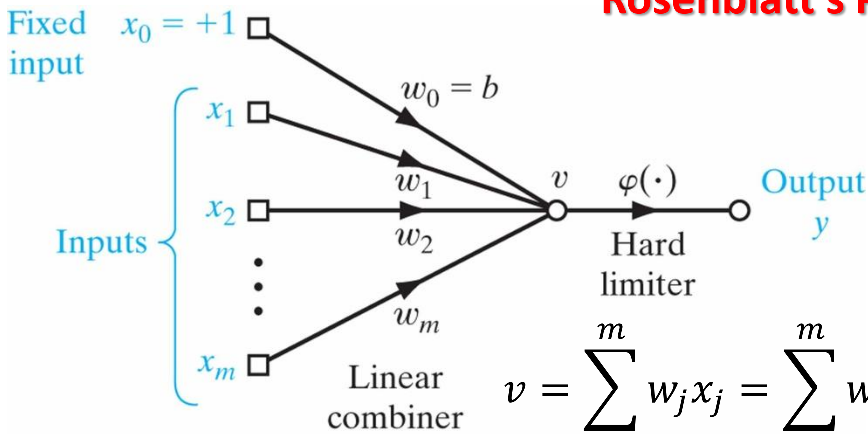
$$d(n) \cong y(n) = h(\mathbf{x}(n))$$

- If y is a finite integer we have a ***Classification*** problem (for 2 classes we have binary classification)

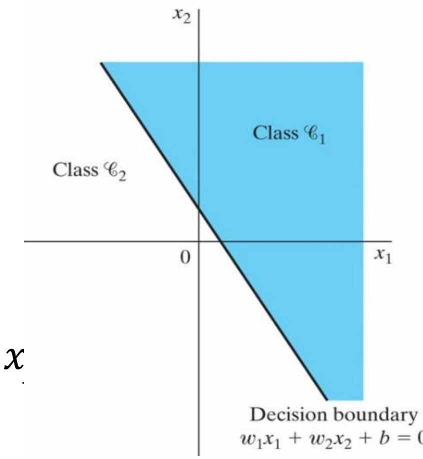
- If y assumes continuous real values we have a ***Regression*** problem

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

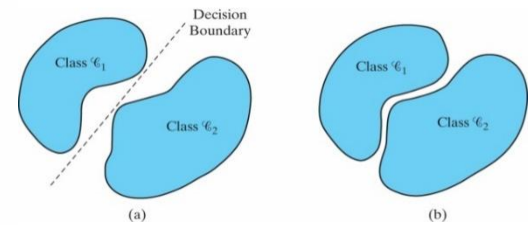
Rosenblatt's Perceptron (*repetition*)



$$v = \sum_{j=0}^{m} w_j x_j = \sum_{j=1}^{m} w_j x$$



Non-linearly separable classes



Linearly separable classes

Overview:

Rosenblatt introduced the ***Single-Layer Perceptron*** as a neuron of ***linear*** **Induced Local Field** v and ***non-linear*** **Activation Function** $\varphi(v)$ (***Threshold Function***, ***Hard Limiter*** or ***Signum Function***) for binary classification of sample elements $\mathbf{x} = [x_0\ x_1 \dots x_m]^{\mathrm{T}}$ into two ***linearly separable*** classes:

$$\mathcal{C}_1 \text{ if } y = \varphi(v) = 1, \qquad \mathcal{C}_2 \text{ if } y = \varphi(v) = 0 \text{ or if } y = \varphi(v) = -1$$

Synaptic weights $\mathbf{w} = [w_0\ w_1 \dots w_m]^{\mathrm{T}}$ are tuned on-line (***stochastic iterative method***) via an error-correction algorithm on labeled training sample elements $\{\mathbf{x}(n), d(n)\}$, $n = 1,2,\dots,N$ via ***supervised learning*** to minimize an error function (e.g. ***MSE***) of ***deviations*** $[d(n) - y(n)]$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

If the ***learning-rate hyperparameter*** $\eta, 0 < \eta \leq 1$ is small it usually leads to (slow) convergence. If it is large it may lead to fast convergence (e.g. for environments of significant element deviations) but may skip optimality dua to oscillations)

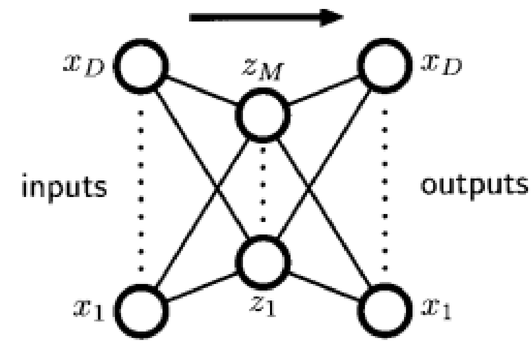
Note: With Gaussian elements $\mathbf{x}(n)$ ***Bayes Classifiers*** into two classes $\mathcal{C}_1, \mathcal{C}_2$ (based on minimization of error probability with a-known a-priori probabilities p_1, p_2) is identical to the ***Rosenblatt Perceptron***

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Pattern Association (S. Haykin: *Introduction, Section 9*)

Associative learning process for associating ***input vectors*** $\mathbf{x}_k$ to q ***memorized patterns*** $\mathbf{y}_k$

Key Patterns → Memorixed Patterns



Methods of Associative Learning:

- **Autoassociation**: $\mathbf{x}_k = \mathbf{y}_k$
  Vectors $\mathbf{x}_k$ are (distorted) examples to be paired (associated, classified) with pre-stored patterns $\mathbf{y}_k$ of the same dimensionality D. Using a ***Multilayer Perceptron*** (***MLP***) we may ***encode*** $\mathbf{x}_k$ to hidden (***latent***) vectors $\mathbf{z}_k$ of lower dimension $M < D$. These are ***decoded*** in a next ***layer*** as in ***autoencoders***. The ***MLP*** parameters are tuned via ***Unsupervised Learning*** using the q ***training key patterns*** $\mathbf{x}_k = \mathbf{y}_k, k = 1,2, \dots, q$

- **Heteroassociation**: $\mathbf{x}_k \neq \mathbf{y}_k$
  Pairing of arbitrary input and output vectors (patterns) via ***Supervised Learning***

Phases of Associative Learning:

- **Storage** of *key patterns*: Training the system by using several input vectors $\mathbf{x}_k$
- **Recall** involving association (classification) of a new example $\mathbf{x}_k$ (*stimulus*, *input vector*, e.g. hand-written decimal numbers or distorted images) to a pre-stored pattern $\mathbf{y}_k$

https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

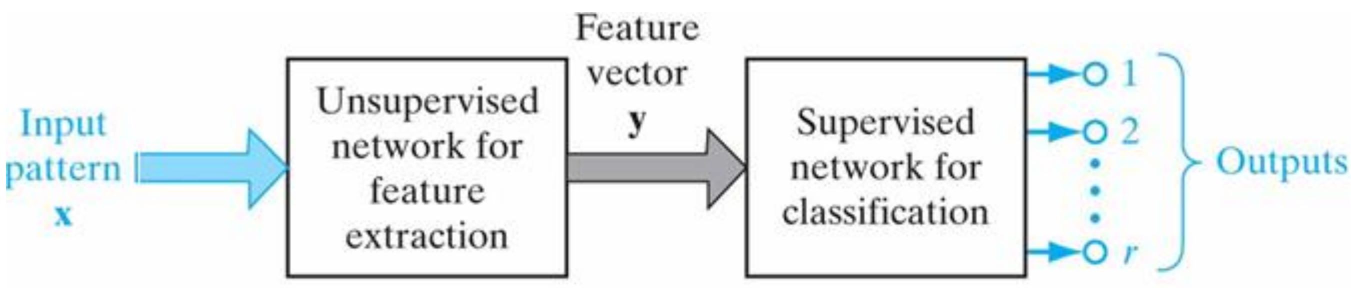
STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Pattern Recognition (S. Haykin: *Introduction, Section 9*)

Recognition of a new input *pattern* by extracting its basic features and classifying it to a *class*, statistically consistent with pre-stored patterns during system training

The process may comprise 2 steps:

- **Feature Extraction**: Transformation of input $\mathbf{x}$ (vector of m dimensions) to an intermediate vector $\mathbf{y}$ of dimension $q \leq m$ via *unsupervised learning*. With $q < m$ we may have *data compression* or extraction of *important features* to simplify the classification process
- **Classification**: Association of $\mathbf{y}$ into r discrete classes via *supervised learning* (involving the hidden layers of the feature extraction module). If $r = 2$ we have **binary classification**



A Labeled Training Sample for Classification of Hand-written Numbers:
MNIST Database to classify had-written numbers in $r = 10$ classes $(0, \ldots, 9)$
https://en.wikipedia.org/wiki/MNIST_database

Separability of Patterns

Classification via Separable Patterns

Involves pairing of an input vector $\mathbf{x}$ (example, instance) to N pre-stored separable patterns $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ of dimension m_0. Binary classification assumes 2 classes C_1 & C_2

Cover's Theorem (1965)

- *A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated*
- For easier pattern classification, ***linear separability*** may be enforced by performing a ***non-linear transformation*** of the vector-space, even if it involves a higher dimensional space

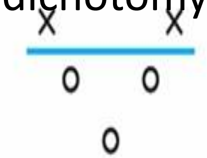
Hidden Functions

Vectors $\mathbf{x}$ of dimension m_0 are mapped via a ***non-linear transformation*** to vectors $\boldsymbol{\varphi}(\mathbf{x})$ of dimension $m_1 \geq m_0$

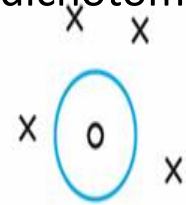
$$\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x}) = \left[\varphi_1(\mathbf{x}) \;\; \varphi_2(\mathbf{x}) \dots \varphi_{m_1}(\mathbf{x})\right]^{\mathrm{T}}$$

$\varphi_j(\mathbf{x}) \in \mathbb{R}, j = 1,2,\dots,m_1$ are ***Hidden Functions***, mostly ***Radial-Basis Functions*** (**RBF**) that depend on *Euclidean Distance* $(\mathbf{x}, \boldsymbol{\mu}_j)$ from $\mathbf{x}$ to center $\boldsymbol{\mu}_j$

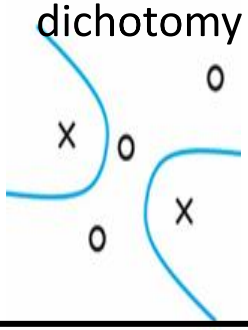
(a) Linearly separable dichotomy



(b) Spherically separable dichotomy



(c) Quadrically separable dichotomy



For the binary case, the new pattern space exhibits $\boldsymbol{\varphi}$-separable dichotomy if there exists a vector $\mathbf{w}$ with $m_1 \geq m_0$ coordinates that defines two linearly-separable regions C_1 & C_2 of $\mathbf{x}$:

$$\mathbf{w}^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}) > 0 \Rightarrow \mathbf{x} \in C_1 \text{ and } \mathbf{w}^{T}\boldsymbol{\varphi}(\mathbf{x}) < 0 \Rightarrow \mathbf{x} \in C_2$$

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Pattern Separablity – The XOR Problem

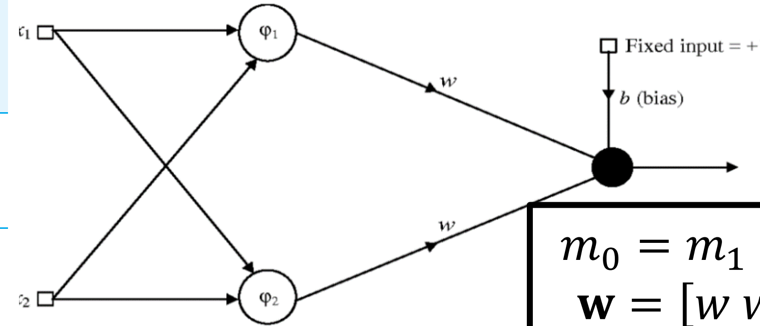
Gaussian Radial-Basis Function **(RBF)**

A usual choice: $\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x})\ \ \varphi_2(\mathbf{x}) \dots \varphi_{m_1}(\mathbf{x})]^{\mathrm{T}} \in \mathbb{R}^{m_1}$, $\varphi_j(\mathbf{x}) \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^{m_0}$, $m_0 \leq m_1$

$\varphi_j(\mathbf{x}) = \exp\left(-\|\mathbf{x} - \boldsymbol{\mu}_j\|^2\right)$, $\mathbf{x} \in \mathbb{R}^{m_0}, m_0 \leq m_1$, $\boldsymbol{\mu}_j$ *center* of $\varphi_j(\mathbf{x})$, $\|\mathbf{x} - \boldsymbol{\mu}_j\|$ *distance* $(\mathbf{x}, \boldsymbol{\mu}_j)$

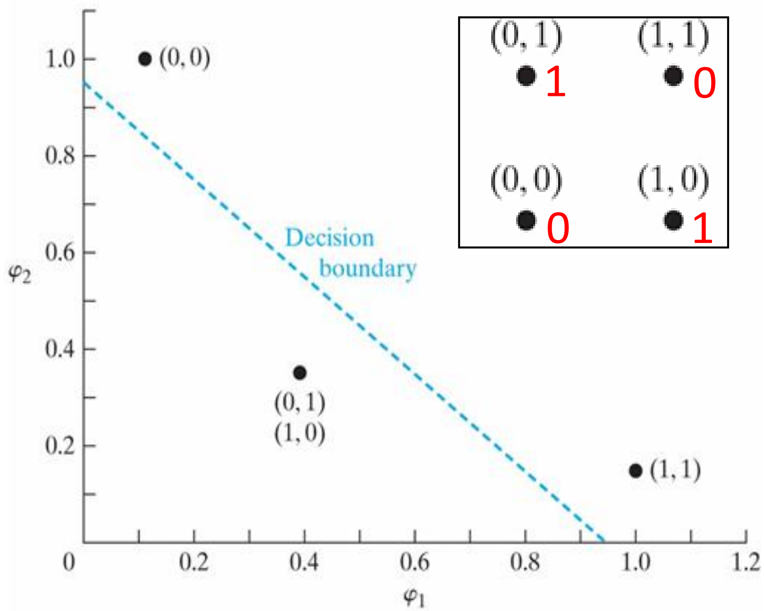
TABLE 5.1 Specification of the Hidden Functions for the XOR Problem of Example 1

Input Pattern $\mathbf{x}$	First Hidden Function $\varphi_1(\mathbf{x})$	Second Hidden Function $\varphi_2(\mathbf{x})$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(0,0)	0.1353	1
(1,0)	0.3678	0.3678



$m_0 = m_1 = 2$
$\mathbf{w} = [w\ w]^{\mathrm{T}}$

$\mathbf{x} = [x_1\ x_2]^{\mathrm{T}} \rightarrow \boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x})\ \varphi_2(\mathbf{x})]^{\mathrm{T}}$
$\varphi_1(\mathbf{x}) = \exp(-\|\mathbf{x} - \boldsymbol{\mu}_1\|^2)$, $\boldsymbol{\mu}_1 = [1{,}1]^{\mathrm{T}}$
$\varphi_2(\mathbf{x}) = \exp(-\|\mathbf{x} - \boldsymbol{\mu}_2\|^2)$, $\boldsymbol{\mu}_2 = [0{,}0]^{\mathrm{T}}$



Example of Evaluation of $\boldsymbol{\varphi}(\mathbf{x})$, $\mathbf{x} = [1\ 1]^{\mathrm{T}}$

$$\varphi_1(1{,}1) = \exp\left(-\left\|[1\ 1]^{\mathrm{T}} - [1\ 1]^{\mathrm{T}}\right\|^2\right) = 1$$

$$\varphi_2(1{,}1) = \exp\left(-\left\|[1\ 1]^{\mathrm{T}} - [0\ 0]^{\mathrm{T}}\right\|^2\right) = 0.1353$$

Output: $y = w\varphi_1(\mathbf{x}) + w\varphi_2(\mathbf{x}) + b$

$(1{,}1)$: $w + w \times 0.1353 + b = 0$

$(0{,}1)$: $w \times 0.3678 + w \times 0.3678 + b = 1$

$(0{,}0)$: $w \times 0.1353 + w + b = 0$

$(1{,}0)$: $w \times 0.3678 + w \times 0.3678 + b = 1$

Solution

$w = -2.502$
$b = 2.841$

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Definitions of Radial-Basis Function (RBF), Kernels & Hybrid Learning

(based on ***C. M. Bishop***, **Ch.6: Kernel Methods** https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf)

Radial-BasIs Function (RBF): $\mathbf{x} \in \mathbb{R}^{m_0} \to \varphi_j(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{x}_j\|) = \varphi(r) \in \mathbb{R}, j = 1,2,\dots,m_1$

where $r = \|\mathbf{x} - \mathbf{x}_j\|$ is the non-negative *Euclidean* radial distance $(\mathbf{x}, \mathbf{x}_j)$

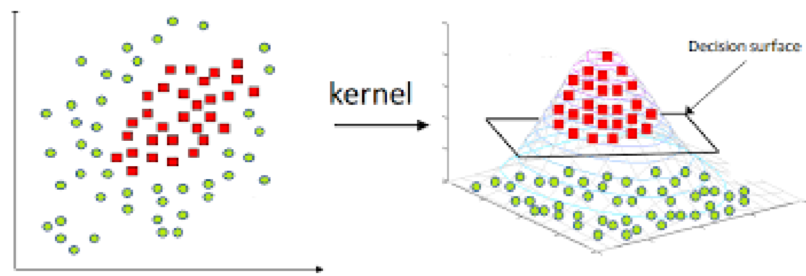
- **Transformation**: $\mathbf{x} \to \boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x})\ \varphi_2(\mathbf{x}) \dots \varphi_{m_1}(\mathbf{x})]^{\mathrm{T}}, m_1 \geq m_0$ leading as per *Cover's Theorem* to *linearly separable* classification (linear decision surface)
- **Example**: A *Gaussian RBF* $\varphi_j(\mathbf{x}) = \exp\left(-\|\mathbf{x} - \mathbf{x}_j\|^2\right)$ is a function of the *Euclidean* distance from an *input vector* $\mathbf{x}$ with m_0 coordinates (*features*) from *pattern* $\mathbf{x}_j$
- **Hidden Functions**: The $\varphi_j(\mathbf{x})$ represent m_1 *hidden features* of $\mathbf{x}$ as distances from the $\mathbf{x}_j$ centroid *patterns* determined in the **1st Phase of Hybrid Learning** that performs clustering via *unsupervised learning* (e.g. via the *K-Means* algorithm)
- **Pattern Classification**: The **2nd Phase of Hybrid Learning** involves the final *pattern choice* for a vector $\mathbf{x} \in \mathbb{R}^{m_0}$ via a *feed-forward* network and *supervised learning*

Kernel: $k(\mathbf{x}, \mathbf{x}_j) \in \mathbb{R}$ is a similarity metric ($\sim$*inner product*) between $\mathbf{x}$ and centers $\mathbf{x}_j \in \mathbb{R}^{m_0}$

- **Relationship to RBF**: $k(\mathbf{x}, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}_j), j = 1,2,\dots,m_1$ (*inner product*)
- $k(\mathbf{x}, \mathbf{x}_j)$ is *symmetric* about $\mathbf{x}_j$, has *constant volume* under its surface and attains a *maximum* at $\mathbf{x} = \mathbf{x}_j$

With volume normalized to 1, kernels $\sim$ probability densities

Kernel Trick: Enforce linearity of ***decision surface*** by increasing dimensionality and proper selection of ***kernel*** (*Cover's Theorem*)



STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

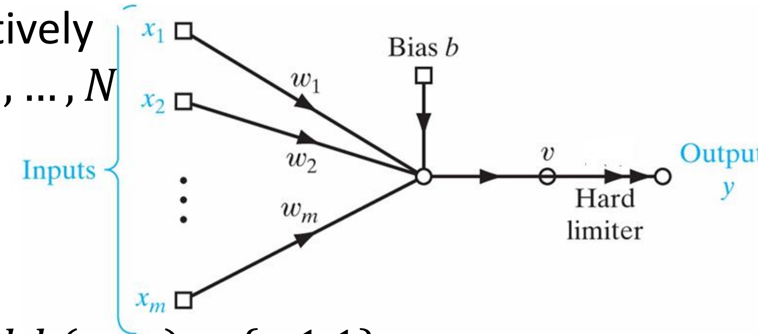
Binary Classification - Kernel Perceptron

https://en.wikipedia.org/wiki/Kernel_perceptron,

1. Linear Perceptron Classifier: $y = \mathrm{sgn}(\mathbf{w}^{\mathrm{T}}\mathbf{x}) \in \{-1,1\}$

Training Algorithm: Initialize to $\mathbf{w} = [0\ 0 \dots 0]^{\mathrm{T}}$ and iteratively apply *labeled* input patterns $\{\mathbf{x}_i, d_i\}$, $d_i \in \{-1,1\}, i = 1,2,\dots,N$ to evaluate $y = \mathrm{sgn}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)$:

$$\mathbf{w} \leftarrow \begin{cases} \mathbf{w} & \text{if } \ y = d_i \quad (\textit{right} \text{ decision}) \\ \mathbf{w} + d_i\mathbf{x}_i & \text{if } \ y \neq d_i \quad (\textit{wrong} \text{ decision}) \end{cases}$$



2. Non-Linear Kernel Method Classifier: $y = \mathrm{sgn}\sum_{i=1}^{N}\alpha_i d_i k(\mathbf{x},\mathbf{x}_i) \in \{-1,1\}$

The *Classifier* stores N training patterns $\mathbf{x}_i \in \mathbb{R}^{m_0}$ of m_0 coordinates along with *labeled* pairs $\{\mathbf{x}_i, d_i\}$, and proceeds to updates counters α_i for classifications $\mathbf{x} \to y \in \{-1,1\}$. The machine processes the output y for an input $\mathbf{x}$ based on **selecting** a *kernel* $k(\mathbf{x},\mathbf{x}_i)$ and using the rule:

$$y = \mathrm{sgn}\sum_{i=1}^{N}\alpha_i d_i k(\mathbf{x},\mathbf{x}_i) \in \{-1,1\}$$

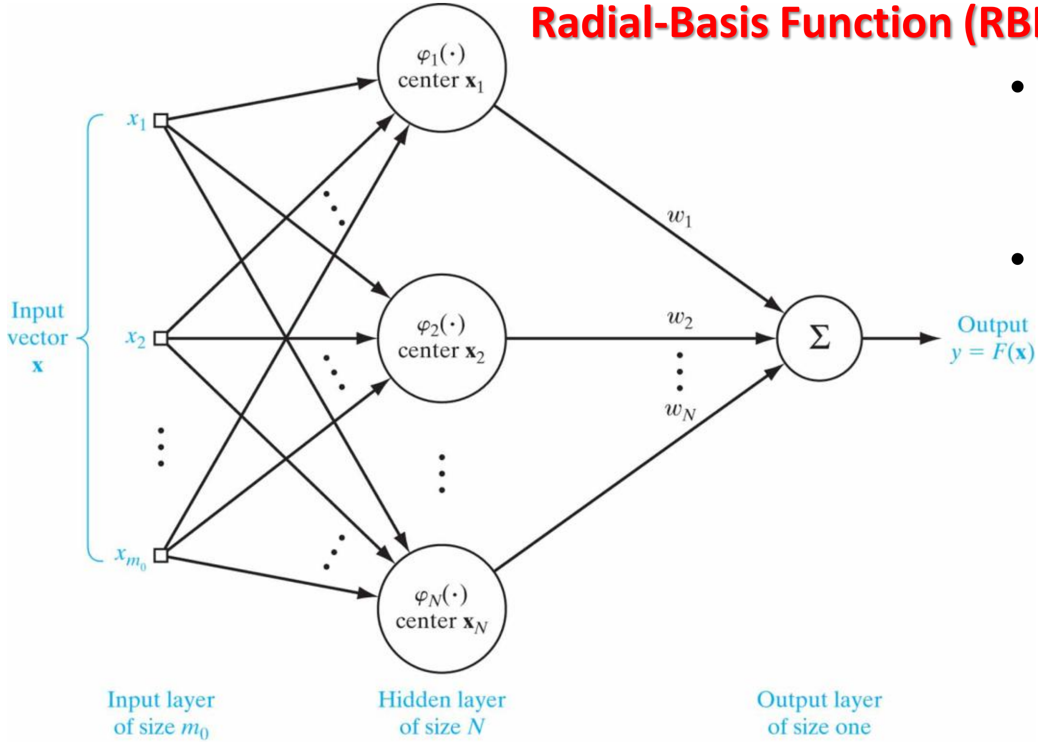
- The *Kernel* $k(\mathbf{x},\mathbf{x}_i) \in \mathbb{R}$ is the *inner product* of non-linear *hidden functions* of dimensionality $m_1 \geq m_0$: $k(\mathbf{x},\mathbf{x}_i) = \boldsymbol{\varphi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}_i) = \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}) = k(\mathbf{x}_i,\mathbf{x})$
- It stresses the impact of similarity between vectors $\boldsymbol{\varphi}(\mathbf{x})$ and all $\boldsymbol{\varphi}(\mathbf{x}_i)$ by considering the non-linear mapping of vectors $\mathbf{x}$ & $\mathbf{x}_i$ into a space of augmented dimensionality

Training Algorithm: Weights are updated as $\mathbf{w} = \sum_{i=1}^{N}\alpha_i\, d_i, \mathbf{x}_i$ with α_i the counter of ***wrong*** decisions $\mathbf{x}_i \to y \neq d_i$

For every training *labeled pattern pair* $\{\mathbf{x}_i, d_i\}$, $i = 1,2,\dots,N$ evaluate $y = \mathrm{sgn}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i) = \mathrm{sgn}\sum_{j=1}^{N}\alpha_j d_j k(\mathbf{x}_i,\mathbf{x}_j)$. If $y \neq d_i$ *increment* the counter $\alpha_i \leftarrow \alpha_i + 1$

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Radial-Basis Function (RBF) Network Model



- **Input Layer**: Input vectors $\mathbf{x}$ with m_0 *features* feed with no modification an intermediate *hidden layer*
- **Hidden Layer**: For the *Training Dataset* of $N \geq m_0$ elements (*patterns*), define N hidden nodes that enable *Gaussian RBF*:

$$\varphi_j(\mathbf{x}) = \varphi(\mathbf{x}, \mathbf{x}_j) = \varphi(\|\mathbf{x} - \mathbf{x}_j\|)$$
$$= \exp(-\|\mathbf{x} - \mathbf{x}_j\|^2)$$

(*Gaussian* functions of *Euclidean* distances $\|\mathbf{x} - \mathbf{x}_j\|^2$ for the N $(\mathbf{x}, \mathbf{x}_j)$ training pairs)

- **Output Layer**:
Sum-of-products of base function $\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x})\ \varphi_2(\mathbf{x}) \ldots \varphi_N(\mathbf{x})]^{\mathrm{T}}$ weighted by $w_1, w_2, \ldots, w_N$

$$y = F(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}) = \sum_{j=1}^{N} w_j \varphi(\|\mathbf{x} - \mathbf{x}_j\|), y \in \{-1,1\}$$

- **Training via Supervised Learning**:
  - ✓ Solve the linear system of N equations $F(\mathbf{x}_i) = \sum_j w_j\, \varphi(\|\mathbf{x}_i - \mathbf{x}_j\|) = d_i$ resulting from the N *labeled* elements $\{\mathbf{x}_i, d_i\}$ of the training sample to determine the N weights w_j
  - ✓ The hidden neurons $F(\mathbf{x}_i) = d_i$ define a *hyper-surface* for *binary decisions*
  - ✓ The linear system always yields a solution $\mathbf{x}_i$ (*Micchelli's Theorem*)

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Radial-Basis Function (RBF) Network for XOR

(based on "*Hybrid Learning* – RBF" in Lectures 2019-2020 by *A. Stafylopatis*, E.C.E., NTUA)

Training Data: $N = 4$ *labeled* elements (*patterns*) $\mathbf{x}_i = [x_i(1)\ x_i(2)]^{\mathrm{T}} \rightarrow y = F(\mathbf{x}_i)$ where $\{x_i(1), x_i(2), y\}$ binary variables $\in \{0,1\}$ and $i \leq N = 4$

Radial-Basis Functions: Select *Gaussian* $\varphi_j(\mathbf{x}) = \exp(-\|\mathbf{x} - \boldsymbol{\mu}_j\|^2)$, $j = 1,2,3,4$ placed around 4 centers: $\boldsymbol{\mu}_1 = [1\ 1]$, $\boldsymbol{\mu}_2 = [0\ 0]$, $\boldsymbol{\mu}_3 = [0\ 1]$, $\boldsymbol{\mu}_4 = [1\ 0]$

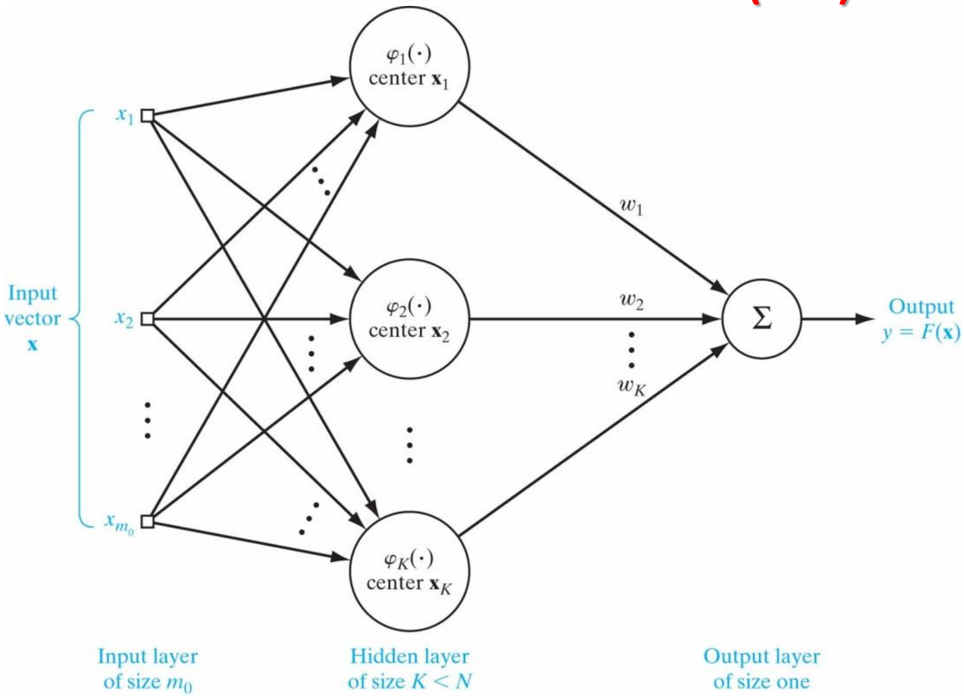
$$y = F(\mathbf{x}) = w_1\varphi_1(\mathbf{x}) + w_2\varphi_2(\mathbf{x}) + w_3\varphi_3(\mathbf{x}) + w_4\varphi_4(\mathbf{x})$$

$\mathbf{x}$	$\varphi_1(\mathbf{x})$	$\varphi_2(\mathbf{x})$	$\varphi_3(\mathbf{x})$	$\varphi_4(\mathbf{x})$	y
(1,1)	1	0.1353	0.3678	0.3678	0
(0,0)	0.1353	1	0.3678	0.3678	0
(0,1)	0.3678	0.3678	1	0.1353	1
(1,0)	0.3678	0.3678	0.1353	1	1

Parameter Tuning of RBF Network

$$w_1 = w_2 = -0.9843$$
$$w_3 = w_4 = 1.5188$$

Radial-Basis Function (RBF) Networks – Practical Implementation



RBF networks are *trained* in *reasonable time* but require *excessive storage* for N hidden nodes (**equal to the number of elements in the training dataset**), *accurate measurements* of labeled elements $\{\mathbf{x}_i, d_i\}$ and significant *computational complexity*

Approximate Implementation:
Deploy a smaller number of hidden nodes $K < N$ that defines a vector space of K dimensions and tune for K wights:

$$y = F(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}) = \sum_{j=1}^{K} w_j \varphi\left(\left\|\mathbf{x} - \boldsymbol{\mu}_j\right\|\right)$$

Hybrid Learning

Training set of N vectors, number of hidden nodes $K < N$, tuning for *fewer* synaptic weights w_j,

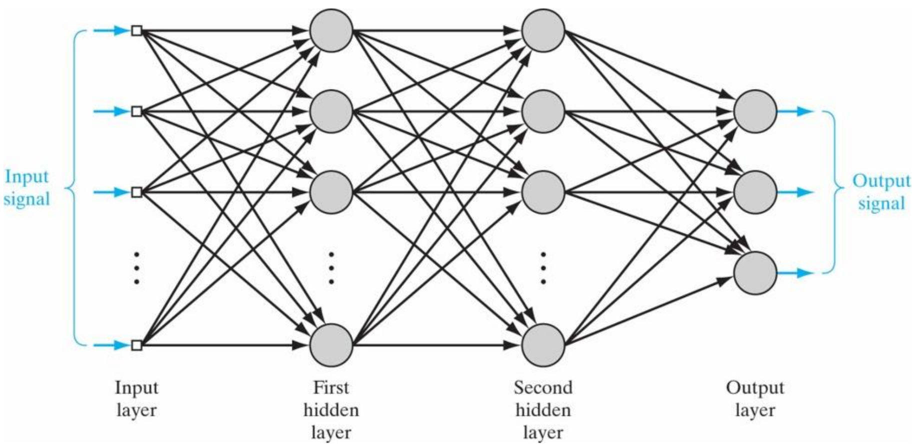
- **Input Layer**: Vector elements $\mathbf{x}$ with m_0 coordinates (*features*) $m_0 \leq K < N$
- **Hidden Layer**: K *hidden* nodes implementing $\varphi\left(\left\|\mathbf{x} - \boldsymbol{\mu}_j\right\|\right)$, with $\boldsymbol{\mu}_j$ *the centroids* (*cluster heads*) of $\mathbf{x} \in \mathbb{R}^{m_0}$ as per *K-Means Clustering* with *Euclidean* distance $\left\|\mathbf{x} - \boldsymbol{\mu}_j\right\|^2$
- **Output Layer**: Linear combination of K **RBF**'s $\varphi_j(\mathbf{x})$:

$$y = F(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}) = \sum_{j=1}^{K} w_j \varphi\left(\left\|\mathbf{x} - \boldsymbol{\mu}_j\right\|\right)$$

Tuning for K weights w_j from the N training pairs $\{\boldsymbol{\varphi}(\mathbf{x}_i), d_i\}$ with *Supervised Learning* and *MSE* approximations: $d_i \cong F(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}_i) = \sum_{j=1}^{K} w_j \varphi\left(\left\|\mathbf{x}_i - \boldsymbol{\mu}_j\right\|\right)$

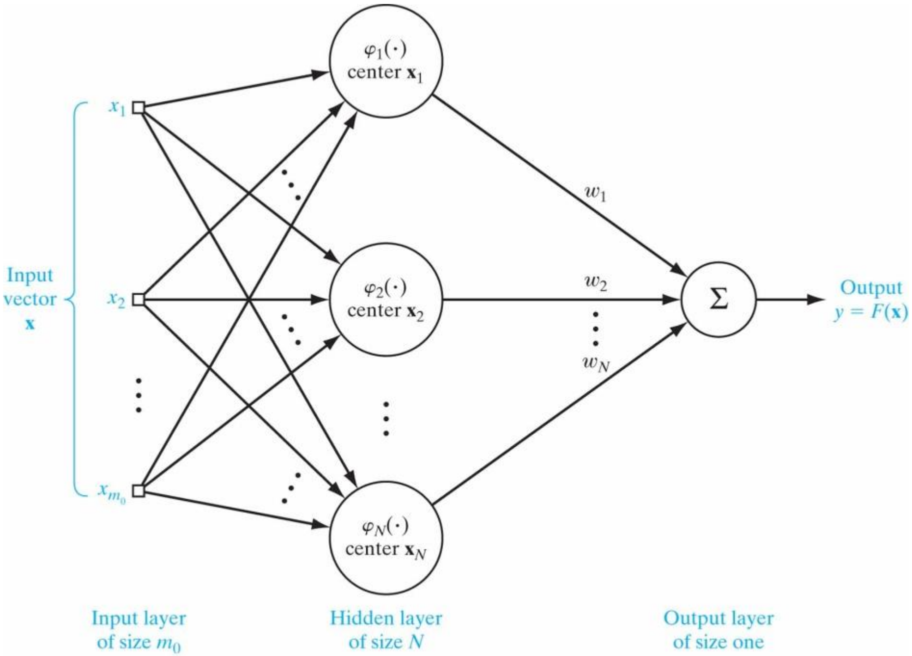
STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Multi-layer Perceptron (MLP) vs. RBF



MLP

- Many Hidden Layers
- Supervised Learning
- Batch or On-line (Stochastic) Learning
- Back-propagation Algorithm
- Non-linear Activation Function
- Slow Training Convergence
- Tolerant to Input Inaccuracies & Noisy Vectors

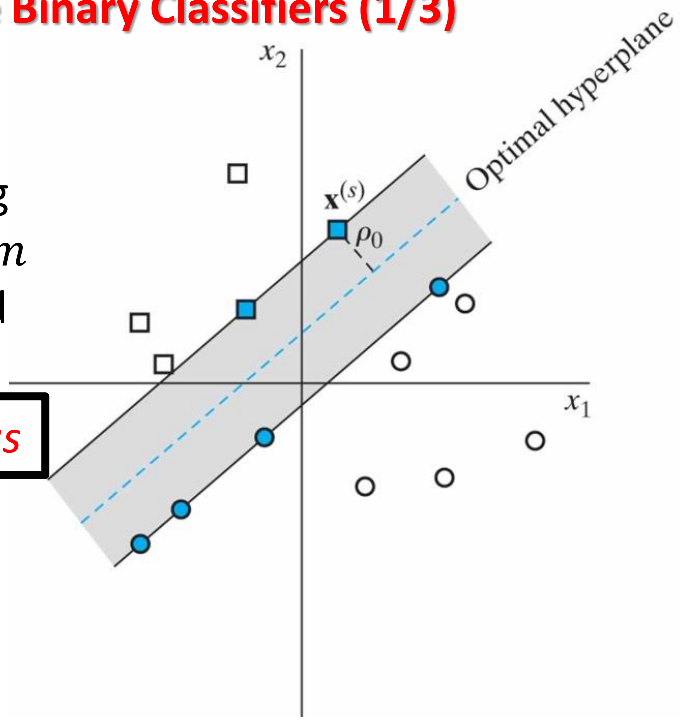


RBF

- A Single Hidden Layer
- Hybrid Learning
- Non-linear Transformation of Input Vectors via Radial-Basis Functions (Gaussian)
- Flexibility in Region Separation for Classification of Input (pattern vectors)
- Fast Training Convergence
- Sensitive to Measurement Accuracy of Sample Vectors
- The Hyper-surface for Binary Separability may be Generalized to Classify Noisy Vectors via Interpolation

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Support Vector Machines (SVM) - Linearly Separable Binary Classifiers (1/3)



- For *labeled* training sample vectors $\{\mathbf{x}_i, d_i\}$, $d_i \in \{-1, +1\}$, $i = 1,2, \ldots, N$ the **SVM** defines binary classification regions with the largest *margin of separation* via *supervised* learning
- For linearly separable regions and a vector $\mathbf{x}$ (*pattern*) with m dimensions (*features*), the separation hyper-plane is defined by the equation:

$\boxed{\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0 \text{ with } \mathbf{w} \text{ denoting synaptic weights \& } b \text{ is a } bias}$

Classification of training sample vectors follows the rule:

$$\boxed{\begin{array}{l} \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \geq 0 \text{ if } d_i = +1 \\ \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b < 0 \text{ if } d_i = -1 \end{array}}$$

- The distance ρ_0 of the closest vector $\mathbf{x}_0$ from the separation hyper-plane identifies the *margin* that should by maximized to yield *optimal separation of regions*: $\mathbf{w}_{\mathrm{o}}^{\mathrm{T}}\mathbf{x} + b_{\mathrm{o}} = 0$
- We have from geometry that $\rho = \frac{2}{\|\mathbf{w}_{\mathrm{o}}\|}$ with $\|\mathbf{w}_{\mathrm{o}}\|$ the *Euclidean* norm of weight vector $\mathbf{w}_{\mathrm{o}}$
- For the training vectors $\{\mathbf{x}_i, d_i\}$ we have:

$$\mathbf{w}_{\mathrm{o}}^{\mathrm{T}}\mathbf{x}_i + b_{\mathrm{o}} \geq 1 \text{ if } d_i = +1$$
$$\mathbf{w}_{\mathrm{o}}^{\mathrm{T}}\mathbf{x}_i + b_{\mathrm{o}} \leq 1 \text{ if } d_i = -1$$

- Training vectors $\mathbf{x}_i$ for which equality holds in the relations above are identified as the ***Support Vectors*** $\mathbf{x}_i^S$ that define the *separation zone*
- The two relations can be expressed as *unified constraints* for the training set:

$$d_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1, i = 1,2, \ldots, N$$

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Support Vector Machines (SVM) - Linearly Separable Binary Classifiers (2/3)

(based on "***Support Vector Machines***" in Lectures 2019-2020 by ***G. Stamou***, E.C.E., NTUA)

Non-Linear Programming Formulation

Maximization of Separation Margin $\rho = \frac{2}{\|\mathbf{w}_o\|} \Leftrightarrow$ Minimization of $\|\mathbf{w}_o\|^2 = \mathbf{w}_o^{\mathrm{T}}\mathbf{w}_o$

Constrained Optimization to determine the **SVM** parameters (synaptic weights $\mathbf{w}$ and bias b):

$$\min_{\mathbf{w}} \Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \quad \text{given} \quad d_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1, i = 1,2, \dots, N$$

The ***objective function*** is ***non-linear*** (sum of squares) under ***linear*** constraints. The weight vector $\mathbf{w}$ can be determined by using classical **Mon-Linear Programming** methods, e.g. by introducing **Lagrange Multipliers** λ_i assigned to constraints $d_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1$:

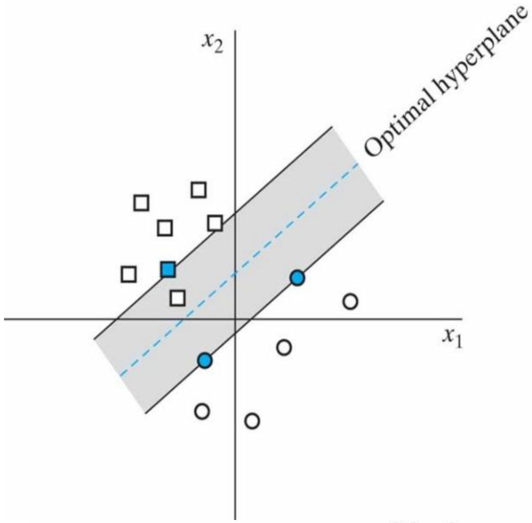
- Define the ***Lagrangian*** $J(\mathbf{w}, b, \lambda_1, \lambda_2, \dots, \lambda_N) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \sum_{i=1}^{N} \lambda_i [\, d_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b)]$
- The ***optimal*** point for the N training vectors $\mathbf{x}_i$ must satisfy the ***Kuhn-Tucker*** conditions:

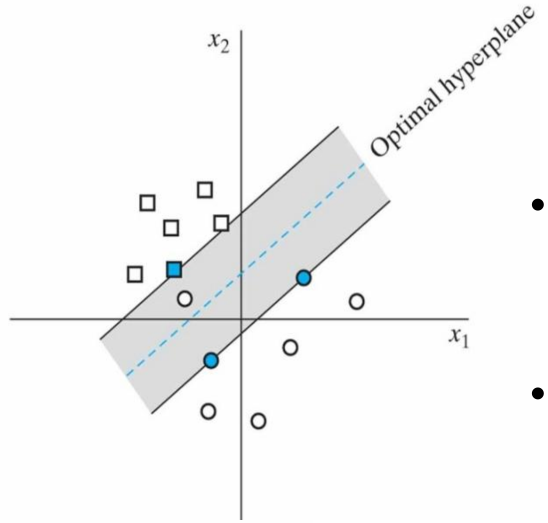
$$\frac{\partial J}{\partial \mathbf{w}} = 0 \;\rightarrow\; \mathbf{w} = \sum_{i=1}^{N} \lambda_i d_i \mathbf{x}_i$$

$$\frac{\partial J}{\partial b} = 0 \;\rightarrow\; \sum_{i=1}^{N} \lambda_i d_i = 0$$

- Variables $\mathbf{w}, b$ identify the ***optimal*** separation hyper-plane $\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$
- The ***Support Vectors*** $\mathbf{x}_i^S$ correspond to $\lambda_i > 0$. The remaining $\mathbf{x}_i$'s to $\lambda_i = 0$

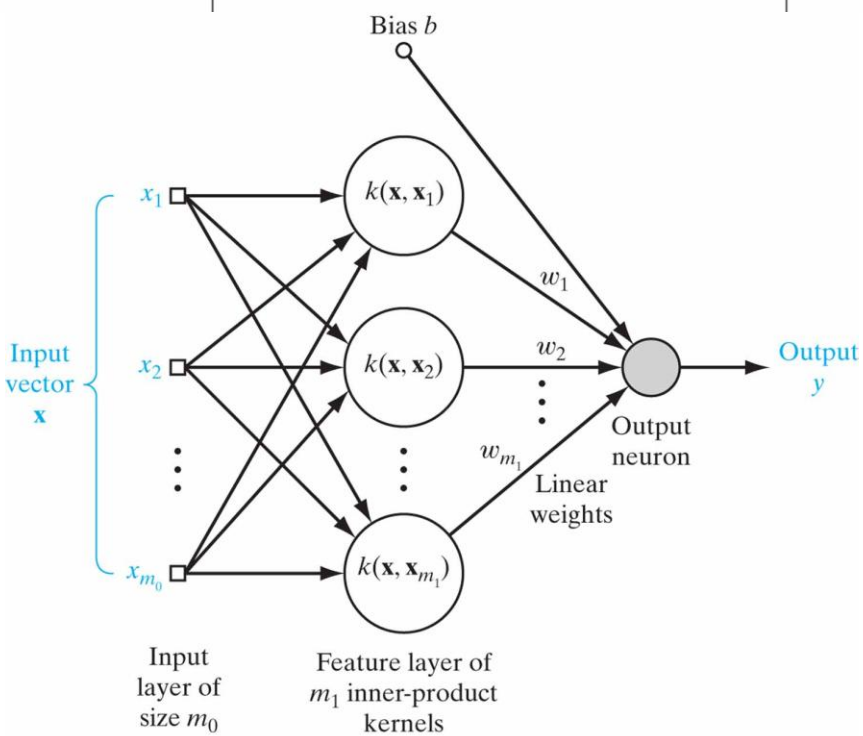
Support Vector Machines (SVM) - Linearly Separable Binary Classifiers (3/3)





Possible Violations of Linear Separability:

- An $\{\mathbf{x}_i, d_i\}$ is located within the separation zone but on the ***right*** side of the optimal hyper-plane
- An $\{\mathbf{x}_i, d_i\}$ is located within the separation zone but on the ***wrong*** side of the optimal hyper-plane



SVM Architecture Using RBF Network Model

Use a great number of *hidden nodes* m_1 (up to the number of training *support* vectors) that transform two *non-linear* separable regions of input vectors $\mathbf{x}$ with dimensionality $m_0 \ll m_1$ into two *linearly* separable hyper-planes