SOGANG UNIVERSITY

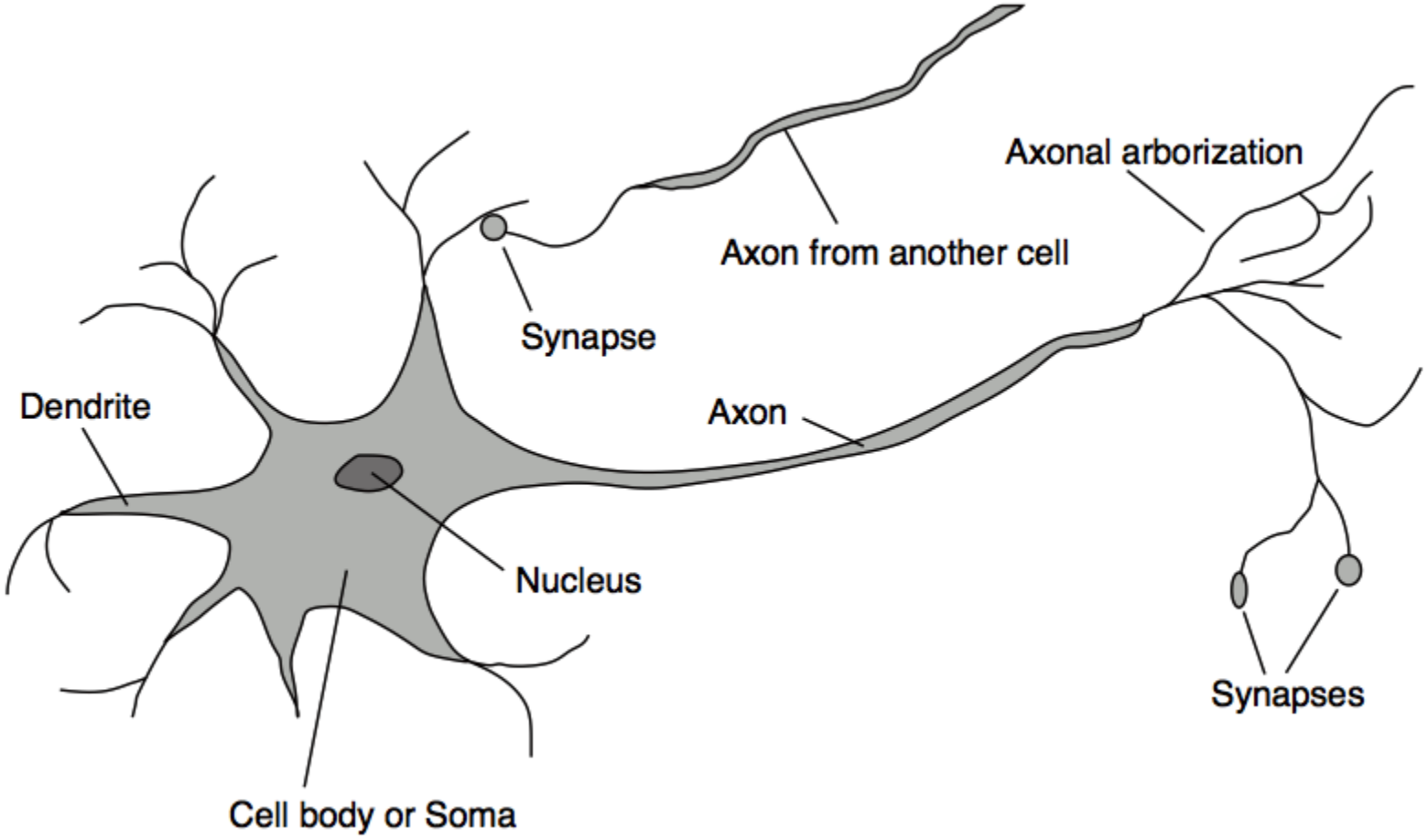
Artificial Neural Networks

Jihoon Yang

Machine Learning Research Laboratory
Department of Computer Science & Engineering
Sogang University
Email: yangjh@sogang.ac.kr
URL: mllab.sogang.ac.kr/people/jhyang.html

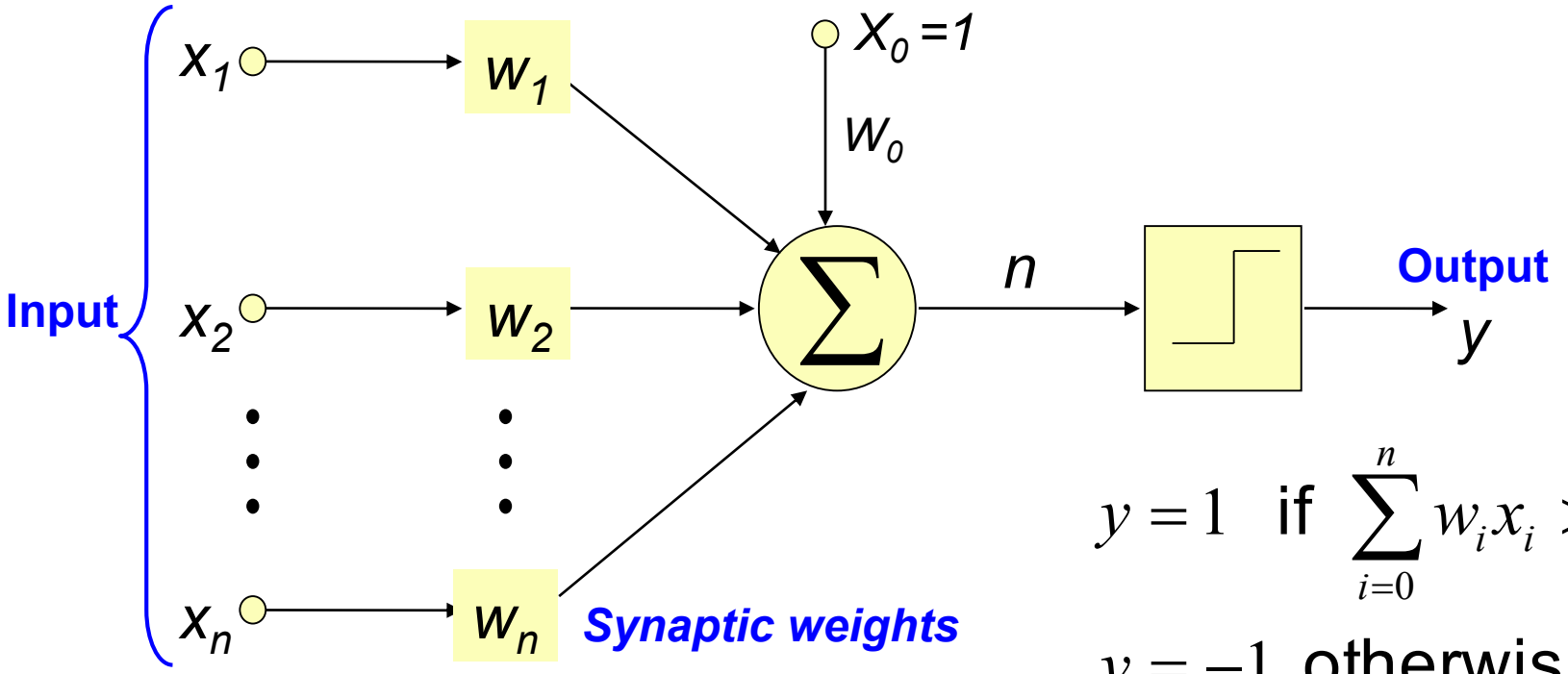
SOGANG UNIVERSITY

Neurons and Computation



SOGANG UNIVERSITY

McCulloch-Pitts computational model of a neuron

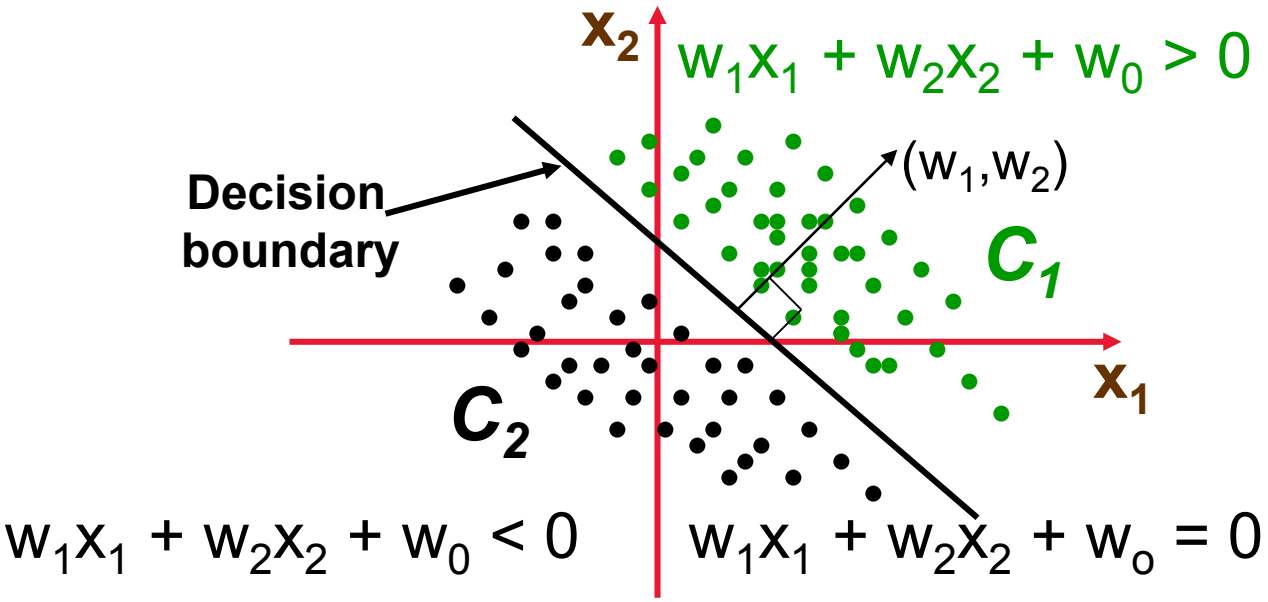


$$y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i x_i > 0$$

$$y = -1 \quad \text{otherwise}$$

When a neuron receives input signals from other neurons, its membrane voltage increases. When it exceeds a certain threshold, the neuron "fires" a burst of pulses.

Threshold neuron – Connection with Geometry



$$\sum_{i=1}^{n} w_i x_i + w_0 = 0$$ **describes a hyperplane which divides the instance space $\Re^n$ into two half–spaces**

$$\chi_+ = \left\{ X_p \in \Re^n \middle| W \bullet X_p + w_0 > 0 \right\} \text{ and } \chi_- = \left\{ X_p \in \Re^n \middle| W \bullet X_p + w_0 < 0 \right\}$$

SOGANG UNIVERSITY

McCulloch-Pitts neuron or Threshold neuron

$$y = sign\left(\mathrm{W} \bullet \mathrm{X} + w_0\right)$$
$$= sign\left(\sum_{i=0}^{n} w_i x_i\right)$$
$$= sign\left(\mathrm{W}^T \mathrm{X} + w_0\right)$$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$$sign\left(v\right) = 1 \text{ if } v > 0$$
$$= 0 \text{ otherwise}$$

SOGANG UNIVERSITY

Threshold neuron – Connection with Geometry

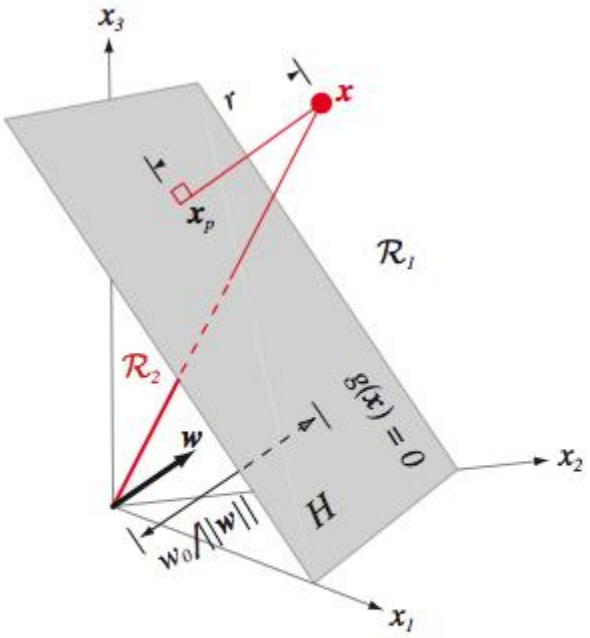


FIGURE 5.2. The linear decision boundary H, where $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(\mathbf{x}) > 0$) and $\mathcal{R}_2$ (where $g(\mathbf{x}) < 0$). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

SOGANG UNIVERSITY

Threshold neuron – Connection with Geometry

- **Instance space** $\Re^n$
- **Hypothesis space is the set of (*n*-1)-dimensional hyperplanes defined in the *n*-dimensional instance space**
- **A hypothesis is defined by** $\sum_{i=0}^{n} w_i x_i = 0$
- **Orientation of the hyperplane is governed by** $(w_1 \ldots w_n)^T$
- ***W* determines the orientation of the hyperplane *H*: given two points *X1* and *X2* on the hyperplane,**

$$W(X_1 - X_2) = 0$$

→ ***W* is normal to any vector lying in *H***

SOGANG UNIVERSITY

Threshold neuron as a pattern classifier

- **The threshold neuron can be used to classify a set of instances into one of two classes C_1, C_2**
- **If the output of the neuron for input pattern X_p is +1 then X_p is assigned to class C_1**
- **If the output is -1 then the pattern X_p is assigned to C_2**
- **Example**

$$[w_0 \; w_1 \; w_2]^T = [-1 \; -1 \; 1]^T$$

$$\mathrm{X}_p^T = [1 \; 0]^T \quad \mathrm{W} \bullet \mathrm{X}_p + w_0 = -1 + (-1) = -2$$

$$\mathrm{X}_p \text{ is assigned to class } C_2$$

SOGANG UNIVERSITY

Threshold neuron – Connection with Logic

- Suppose the input space is $\{0,1\}^n$
- Then threshold neuron computes a Boolean function
  f: $\{0,1\}^n \rightarrow \{-1,1\}$

- **Example**
  - Let $w_0 = -1.5$; $w_1 = w_2 = 1$
  - In this case, the threshold neuron implements the logical AND function

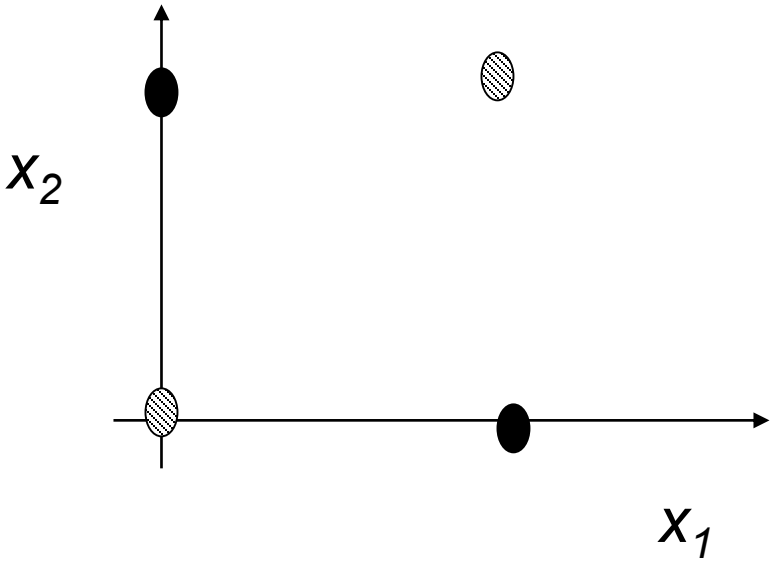
x_1	x_2	$g(X)$	y
0	0	-1.5	-1
0	1	-0.5	-1
1	0	-0.5	-1
1	1	0.5	1

SOGANG UNIVERSITY

Threshold neuron – Connection with Logic

- **Theorem: There exist functions that cannot be implemented by a *single* threshold neuron**

- **Example: Exclusive OR**



Why?

SOGANG UNIVERSITY

Terminology and Notation

- **Synonyms: Threshold function, Linearly separable function, Linear discriminant function**

- **Synonyms: Threshold neuron, McCulloch-Pitts neuron, Perceptron, Threshold Logic Unit (TLU)**

- **We often include w_0 as one of the components of W and incorporate x_0 as the corresponding component of X with the understanding that $x_0 = 1$; Then $y = 1$ if $W \cdot X > 0$ and $y = -1$ otherwise**

SOGANG UNIVERSITY

Learning Threshold functions

- **A training example E_k is an ordered pair (X_k, d_k) where**

$X_k = [x_{0k}\ x_{1k}\ ....\ x_{nk}]^T$ **is an (*n+1*) dimensional input pattern, and**

$d_k = f(X_k) \in \{-1, 1\}$ **is the desired output of the classifier and *f* is an unknown target function to be learned**

- **A training set *E* is simply a multi-set of examples**

SOGANG UNIVERSITY

Learning Threshold functions

$$S^+ = \left\{ \mathrm{X}_k \middle| \left(\mathrm{X}_k, d_k \right) \in E \text{ and } d_k = 1 \right\}$$
$$S^- = \left\{ \mathrm{X}_k \middle| \left(\mathrm{X}_k, d_k \right) \in E \text{ and } d_k = -1 \right\}$$

- **We say that a training set *E* is linearly separable if and only if**

$$\exists \mathrm{W}^* \text{ such that } \forall \mathrm{X}_p \in S^+, \mathrm{W}^* \bullet \mathrm{X}_p > 0$$
$$\text{and } \forall \mathrm{X}_p \in S^-, \mathrm{W}^* \bullet \mathrm{X}_p < 0$$

- **<u>Learning task</u>: Given a linearly separable training set *E*, find a solution $\mathbf{W}^*$**

$$\text{such that } \forall \mathbf{X}_p \in S^+, \mathbf{W}^* \bullet \mathbf{X}_p > 0 \text{ and } \forall \mathbf{X}_p \in S^-, \mathbf{W}^* \bullet \mathbf{X}_p < 0$$

SOGANG UNIVERSITY

Rosenblatt's Perceptron Learning Algorithm

1. **Initialize** $\mathrm{W} = [0\ 0 \ldots\ldots 0]^T$

2. **Set learning rate** $\eta > 0$

3. **<u>Repeat until</u> a complete pass through *E* results in no weight updates**

   For each training example $E_k \in E$

   {

   $$y_k \leftarrow sign\left(\mathrm{W} \bullet \mathrm{X}_k\right)$$

   $$\mathrm{W} \leftarrow \mathrm{W} + \eta\left(d_k - y_k\right)\mathrm{X}_k$$

   }

4. $\mathbf{W}^* \leftarrow \mathbf{W}$; **Return** $\mathbf{W}^*$

Perceptron Learning Algorithm – Example

Let

$S^+ = \{(1, 1, 1), (1, 1, -1), (1, 0, -1)\}$

$S^- = \{(1,-1, -1), (1,-1, 1), (1,0, 1)\}$

$W = (0\ 0\ 0)$

$$\eta = \frac{1}{2}$$

$$w^* = w + \eta (d_n - y_n) x_n$$

X_k	d_k	W	$W.X_k$	y_k	Update?	Updated W
(1, 1, 1)	1	(0, 0, 0)	0	-1	Yes	(1, 1, 1)
(1, 1, -1)	1	(1, 1, 1)	1	1	No	(1, 1, 1)
(1,0, -1)	1	(1, 1, 1)	0	-1	Yes	(2, 1, 0)
(1, -1, -1)	-1	(2, 1, 0)	1	1	Yes	(1, 2, 1)
(1,-1, 1)	-1	(1, 2, 1)	0	-1	No	(1, 2, 1)
(1,0, 1)	-1	(1, 2, 1)	2	1	Yes	(0, 2, 0)
(1, 1, 1)	1	(0, 2, 0)	2	1	No	(0, 2, 0)

Perceptron Convergence Theorem (Novikoff)

Theorem:

Let $E = \{(\mathbf{X}_k, d_k)\}$ **be a training set where** $\mathbf{X}_k \in \{1\} \times \Re^n$ **and** $d_k \in \{-1, 1\}$

Let $S^+ = \{\mathbf{X}_k | (\mathbf{X}_k, d_k) \in E \;\&\; d_k = 1\}$ and $S^- = \{\mathbf{X}_k | (\mathbf{X}_k, d_k) \in E \;\&\; d_k = -1\}$

The perceptron algorithm is guaranteed to terminate after a bounded number *t* of weight updates with a weight vector $\mathbf{W}^*$

such that $\forall \mathbf{X}_k \in S^+, \mathbf{W}^* \bullet \mathbf{X}_k \geq \delta$ and $\forall \mathbf{X}_k \in S^-, \mathbf{W}^* \bullet \mathbf{X}_k \leq -\delta$

for some $\delta > 0$, **whenever such** $\mathbf{W}^* \in \Re^{n+1}$ and $\delta > 0$ **exist – that is,** ***E is linearly separable.***

The bound on the number *t* of weight updates is given by

$$t \leq \left(\frac{\|\mathbf{W}^*\| L}{\delta} \right)^2 \quad \text{where } L = \max_{\mathbf{X}_k \in S} \|\mathbf{X}_k\| \quad \text{and } S = S^+ \cup S^-$$

SOGANG UNIVERSITY

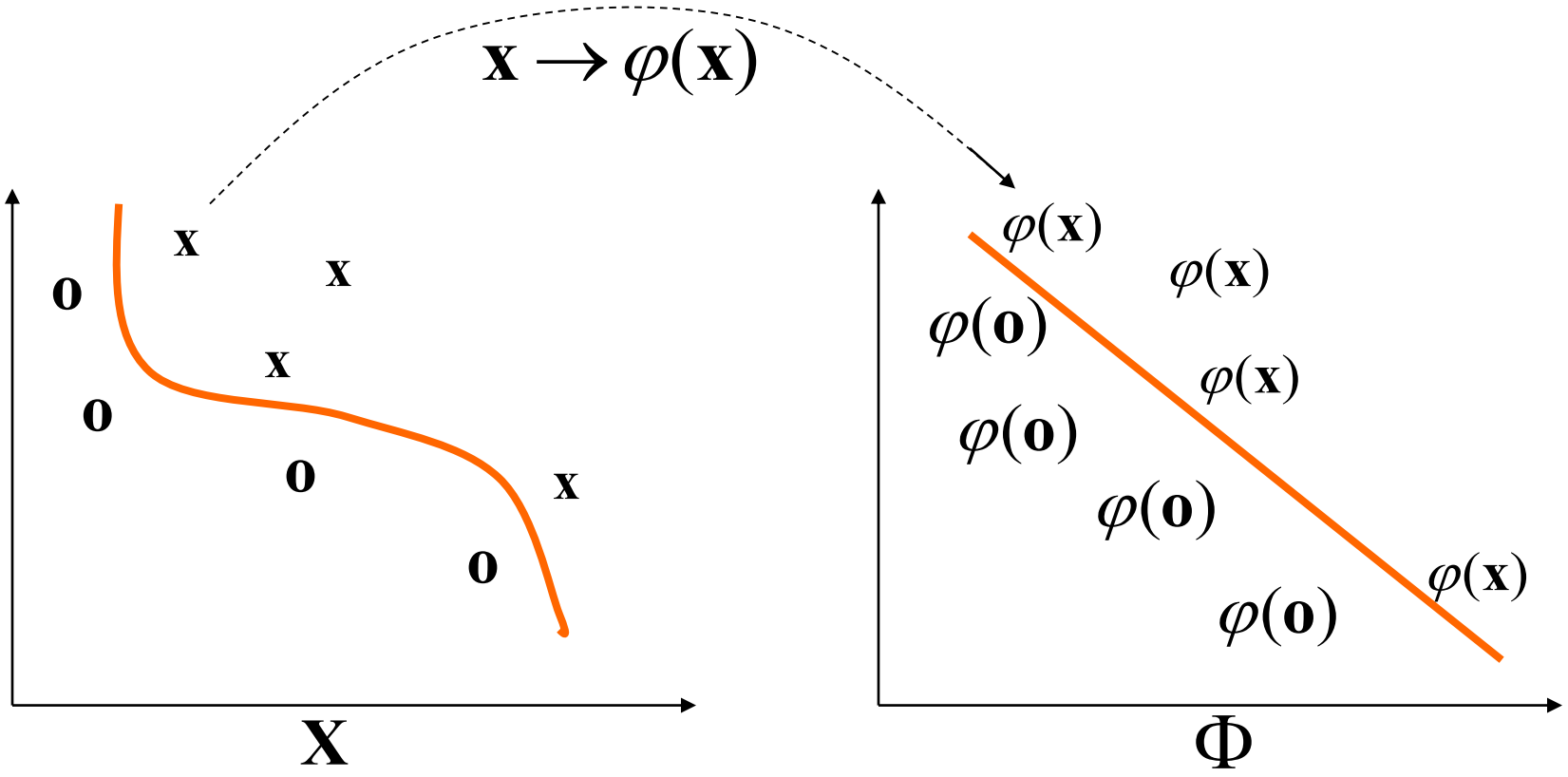
Limitations of Perceptrons

- Perceptrons can only represent threshold functions
- Perceptrons can only learn linear decision boundaries
- What if the data are not linearly separable?
  - Modify the learning procedure or the weight update equation?
    (e.g. Pocket algorithm, Thermal perceptron)
  - More complex networks?
  - Non-linear transformations into a feature space where the data become separable?

SOGANG UNIVERSITY

Extending Linear Classifiers: Learning in feature spaces

- Map data into a feature space where they are linearly separable



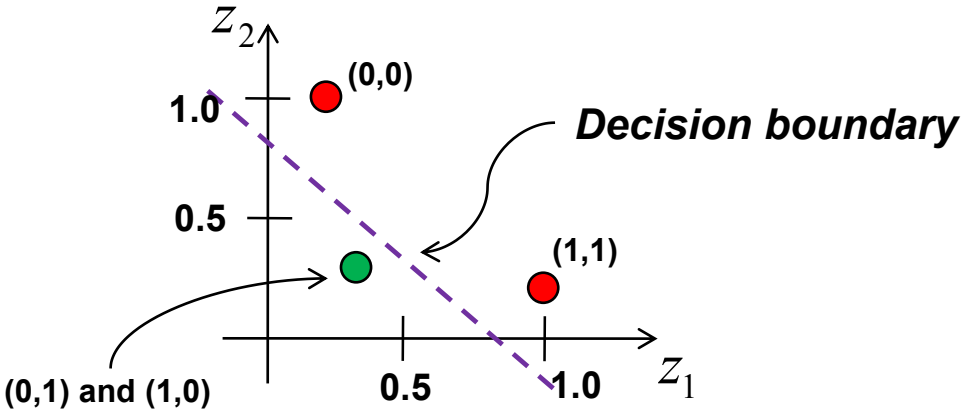
SOGANG UNIVERSITY

Exclusive OR revisited

- **In the feature (hidden) space:**

$$\varphi_1(x_1, x_2) = e^{-\|X - W_1\|^2} = z_1 \qquad W_1 = [1,1]^T$$

$$\varphi_2(x_1, x_2) = e^{-\|X - W_2\|^2} = z_2 \qquad W_2 = [0,0]^T$$



- **When mapped into the feature space <z_1, z_2>, C_1 and C_2 become *linearly separable*. So a linear classifier with $\varphi_1(X)$ and $\varphi_2(X)$ as inputs can be used to solve the XOR problem.**

SOGANG UNIVERSITY

Learning in the Feature Space

- **High dimensional feature spaces**

$$X = (x_1, x_2, \cdots, x_n) \rightarrow \varphi(X) = (\varphi_1(X), \varphi_2(X), \cdots, \varphi_d(X))$$

where typically *d* >> *n* solve the problem of expressing complex functions

- **But this introduces**
  - **Computational problem (working with very large vectors)**
    - **→ Solved using the kernel trick – implicit feature spaces**
  - **Generalization problem (curse of dimensionality)**
    - **→ Solved by maximizing the margin of separation – first implemented in SVM (Vapnik)**

SOGANG UNIVERSITY

Margin Based Bounds on Error of Classification

- **The error ε of classification function h for separable data sets is**

$$\varepsilon = O\left(\frac{d}{l}\right)$$

- **Can prove margin based bound:**

$$\varepsilon = O\left(\frac{1}{l}\left(\frac{L}{\bar{\gamma}}\right)^2\right) \qquad L = \max_p \left\|\mathbf{X}_p\right\|$$

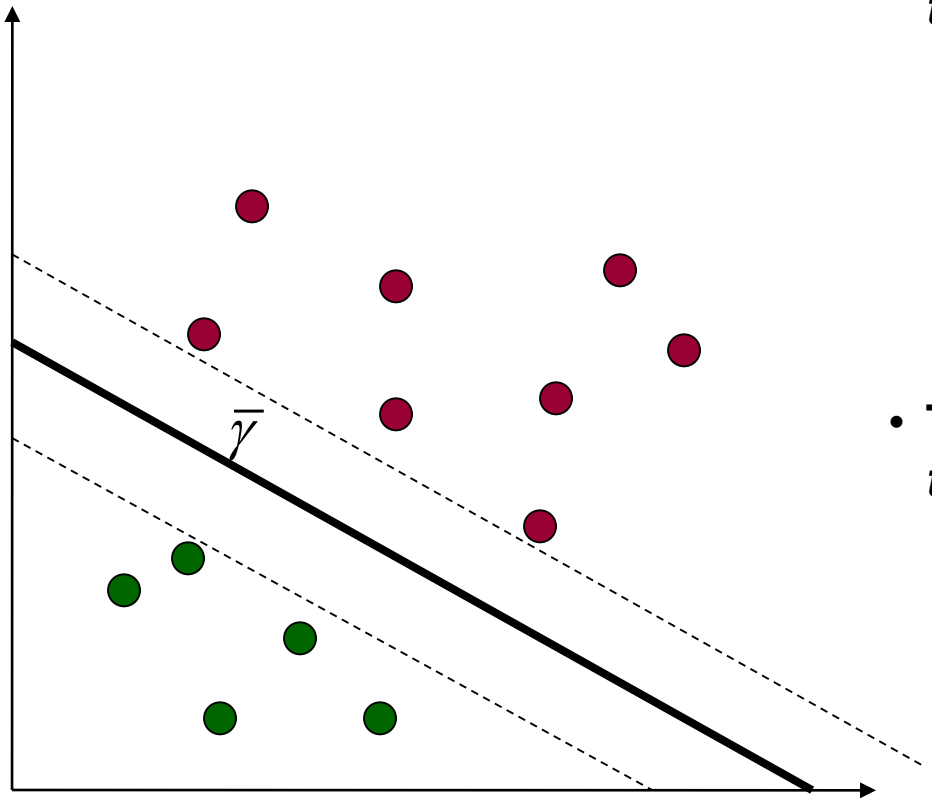
$$\bar{\gamma} = \min_i \frac{y_i f(\mathbf{x}_i)}{\|\mathbf{w}\|} \qquad f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

- **Important insight:**

  Error of the classifier trained on a separable data set is inversely proportional to its margin, and is independent of the dimensionality of the input space!

SOGANG UNIVERSITY

Margin of a Training Set



- **The functional margin of a *training set***

$$\gamma = \min_i \gamma_i$$

- **The geometric margin of a *training set***

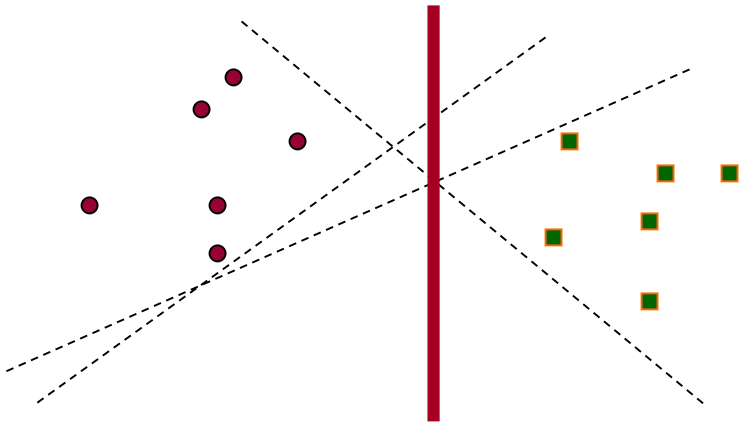
$$\bar{\gamma} = \min_i \bar{\gamma}_i$$

SOGANG UNIVERSITY

Maximum Margin Separating Hyperplane

- $\gamma = \min_i \gamma_i$ **is called the (functional) margin of (*W,b*) w.r.t. the data set *S*={(*Xi,yi*)}**

- **The margin of a training set *S* is the maximum geometric margin over all hyperplanes; A hyperplane realizing this maximum is a maximal margin hyperplane**



Maximal Margin Hyperplane

SOGANG UNIVERSITY

Maximizing Margin → Minimizing ||*W*||

- **Definition of hyperplane (*W,b*) does not change if we rescale it to (σ*W*, σ*b*), for σ>*0*.**
- **Functional margin depends on scaling, but geometric margin $\bar{\gamma}$ does not**
- **If we fix (by rescaling) the functional margin to 1, the geometric margin will be equal 1/||*W*||**
- **Then, we can maximize the margin by minimizing the norm ||*W*||**

SOGANG UNIVERSITY

Learning as Optimization

- **Minimize**

$$\langle \mathbf{W}, \mathbf{W} \rangle$$

Subject to:

$$y_i \left(\langle \mathbf{W}, \mathbf{X}_i \rangle + b \right) \geq 1$$

- **The problem of finding the maximal margin hyperplane: constrained optimization (quadratic programming) problem**

SOGANG UNIVERSITY

Taylor Series Approximation of Functions

Taylor series approximation of $f(x)$

If $f(x)$ is differentiable i.e., its derivatives

$$\frac{df}{dx},\ \frac{d^2 f}{dx^2} = \frac{d}{dx}\left(\frac{df}{dx}\right),\ \ldots \frac{d^n f}{dx^n} \text{ exist at } x = X_0 \text{ and}$$

$f(x)$ is continuous in the neighborhood of $x = X_0$, then

$$f(x) = f(X_0) + \left(\frac{df}{dx}\bigg|_{x=X_0}\right)(x - X_0) + \ldots.. + \frac{1}{n!}\left(\frac{df^n}{dx^n}\bigg|_{x=X_0}\right)(x - X_0)^n$$

$$f(x) \approx f(X_0) + \left(\frac{df}{dx}\bigg|_{x=X_0}\right)(x - X_0)$$

Minimizing/Maximizing Multivariate Functions

To find $\mathbf{X}^*$ that minimizes $f(\mathbf{X})$, we change current guess $\mathbf{X}^C$ in the direction of the negative gradient of $f(\mathbf{X})$ evaluated at $\mathbf{X}^C$

$$\mathbf{X}^C \leftarrow \mathbf{X}^C - \eta \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1} \ldots\ldots\ldots \ldots \frac{\partial f}{\partial x_n} \right) \Bigg|_{\mathbf{X}=\mathbf{X}^C} \quad \text{(why?)}$$

for small (ideally infinitesimally small)

Minimizing/Maximizing Multivariate Functions

Suppose we more from Z_0 to Z_1. We want to ensure $f(Z_1) \leq f(Z_0)$.

In the neighborhood of Z_0, using Taylor series expansion, we can write

$$f(Z_1) = f(Z_0 + \Delta Z) = f(Z_0) + \left(\left. \frac{df}{dZ} \right|_{Z=Z_0} \right)(\Delta Z) + \ldots$$

$$\Delta f = f(Z_1) - f(Z_0) \approx \left(\left. \frac{df}{dZ} \right|_{Z=Z_0} \right)(\Delta Z)$$

We want to make sure $\Delta f \leq 0$.

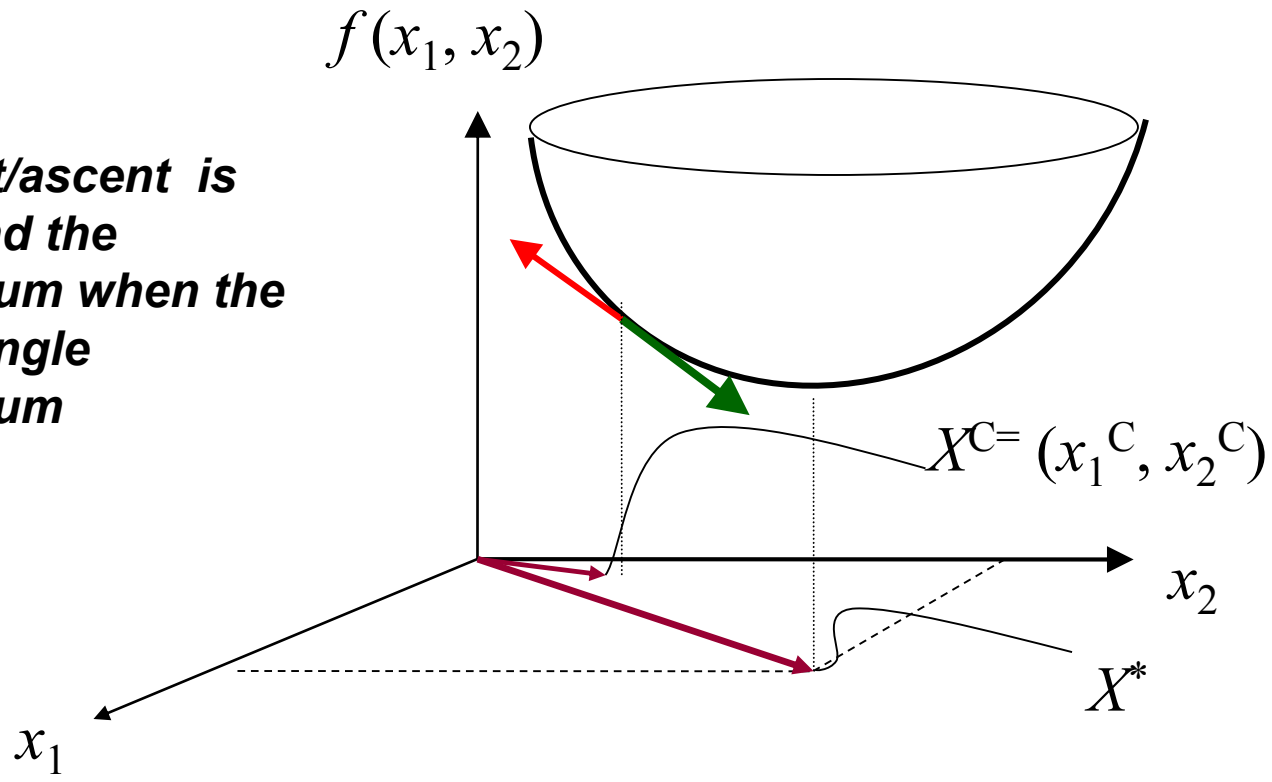
If we choose

$$\Delta Z = -\eta \left(\left. \frac{df}{dZ} \right|_{Z=Z_0} \right) \rightarrow \Delta f = -\eta \left(\left. \frac{df}{dZ} \right|_{Z=Z_0} \right)^2 \leq 0$$

SOGANG UNIVERSITY

Minimizing/Maximizing Functions

Gradient descent/ascent is guaranteed to find the minimum/maximum when the function has a single minimum/maximum



SOGANG UNIVERSITY

Maximum Margin Hyperplane

- The problem of finding the maximal margin hyperplane is a constrained optimization problem

- Use Lagrange theory (extended by Karush, Kuhn, and Tucker – KKT)

- Minimize $\langle \mathbf{W}, \mathbf{W} \rangle$

  subject to

$$y_i\left(\langle \mathbf{W}, \mathbf{X}_i \rangle + b\right) \geq 1$$

- Lagrangian:

$$L_p(\mathbf{w}) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle - \sum \alpha_i [y_i(\langle \mathbf{w}, x_i \rangle + b) - 1] \qquad (4)$$

$$\alpha \geq 0$$

SOGANG UNIVERSITY

From Primal to Dual

- **Minimize *Lp*(*w*) with respect to (*w,b*) requiring that derivatives of *Lp*(*w*) with respect to (*w,b*) all vanish, subject to the constraints $\alpha_i \geq 0$**

- **Differentiating *Lp*(*w*):**

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0, \quad (5)$$

$$\frac{\partial L_p}{\partial b} = 0 \quad (6)$$

$$\mathbf{w} = \sum_i \alpha_i y_i x_i \quad (7)$$

$$\sum_i \alpha_i y_i = 0 \quad (8)$$

- **Substituting this equality constraints back into *Lp*(*w*) we obtain a dual problem**

SOGANG UNIVERSITY

The Dual Problem

- **Maximize:**

$$L_D(\mathbf{w}) = \frac{1}{2}\left\langle \left(\sum_i \alpha_i y_i \mathbf{x}_i\right)\left(\sum_j \alpha_j y_j \mathbf{x}_j\right)\right\rangle$$

$$-\sum_i \alpha_i \left[y_i \left(\left\langle \left(\sum_j \alpha_j y_j \mathbf{x}_j\right), \mathbf{x}_i \right\rangle + b \right) - 1 \right]$$

$$= \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{ij} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - b\sum_i \alpha_i y_i + \sum_i \alpha_i$$

$$= -\frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_i \alpha_i$$

subject to $\sum_i \alpha_i y_i = 0$ **and** $\alpha_i \geq 0$

- **Duality permits the use of kernels!**
- **This is a quadratic optimization problem: convex, no local minima**
- **Solvable in polynomial time**

SOGANG UNIVERSITY

Karush-Kuhn-Tucker Conditions for SVM

- The KKT conditions state that optimal solutions $\alpha_i(\mathbf{w}, b)$ must satisfy

$$\alpha_i\left[y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right) - 1\right] = 0$$

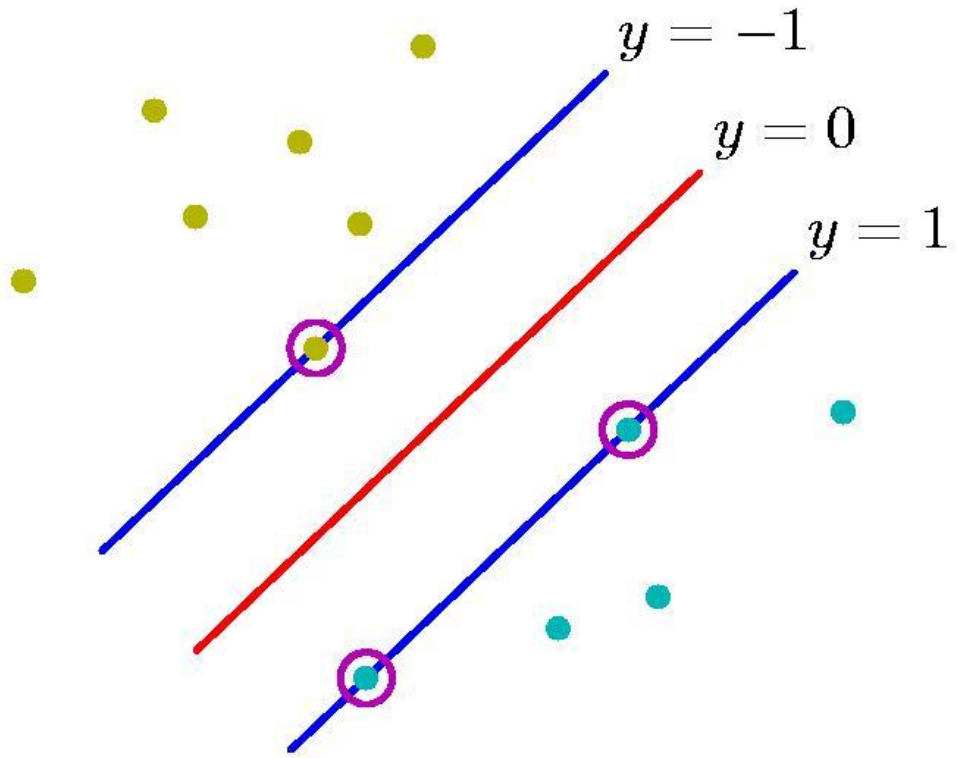
- Only the training samples X_i for which the functional margin = 1 can have nonzero α_i; They are called Support Vectors

- The optimal hyperplane can be expressed in the dual representation in terms of this subset of training samples – the support vectors

$$f(\mathbf{x}, \boldsymbol{\alpha}, b) = \sum_{i=1}^{l} y_i \alpha_i \left\langle \mathbf{x}_i \cdot \mathbf{x} \right\rangle + b = \sum_{i \in \mathrm{SV}} y_i \alpha_i \left\langle \mathbf{x}_i \cdot \mathbf{x} \right\rangle + b$$

SOGANG UNIVERSITY

Support Vectors



Implementation Techniques

- **Maximizing a quadratic function, subject to a linear equality and inequality constraints**

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

$$0 \le \alpha_i \le C$$

$$\sum_i \alpha_i y_i = 0$$

$$\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i \sum_j \alpha_j y_j K(x_i, x_j)$$

SOGANG UNIVERSITY

Strengths and Weaknesses of SVM

- Strengths
  - Training is relatively easy
    - No local optima
  - It scales relatively well to high dimensional data
  - Tradeoff between classifier complexity and error can be controlled explicitly
  - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors

- Weaknesses
  - Need to choose a “good” kernel function

SOGANG UNIVERSITY

Learning Linear Functions

- One approach to regression: directly construct an appropriate function $y(\vec{x})$
- *Linear regression* (linear neuron)

$$y(\vec{x}, \vec{w}) = w_0 + \sum_{i=1}^{d} w_i x_i = \vec{w}^t \vec{x}$$

- We want to find a weight vector $\vec{w}$ such that the linear function is a *good* approximation of unknown $f(\vec{x})$ based on training examples.

Learning Linear Functions

- Define a criterion/error function $J(\vec{w})$, and the learning problem reduces to looking for a weight vector that minimizes $J(\vec{w})$.
- Let the output for input $\vec{x}_k = (x_0^k, x_1^k, \ldots, x_d^k)^t$ be

$$o_k = y(\vec{x}_k, \vec{w}) = \sum_{i=0}^{d} w_i x_i^k = \vec{w}^t \vec{x}_k$$

- The error for the labeled input $(\vec{x}_k, t_k)$: $e_k = o_k - t_k$
- We wish to minimize the magnitude of this error irrespective of the sign. We will use the squared error, and we are interested in the error over all training samples

SOGANG UNIVERSITY

Learning Linear Functions

- The sum-of-squared-error (*Mean Squared Error*) function

$$J_s(\vec{w}) = \frac{1}{2}\sum_{k=1}^{n}(o_k - t_k)^2 = \frac{1}{2}\sum_{k=1}^{n}(\vec{w}^t\vec{x}_k - t_k)^2$$

  where n: number of samples

- The learning problem reduces to looking for a weight vector that minimizes $J_s(\vec{w})$
  — a minimum-squared-error (MSE) solution.

Learning Linear Functions:
Computing Gradient

$$\frac{\partial J}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_k (o_k - t_k)^2 = \frac{1}{2}\sum_k \frac{\partial}{\partial w_i}(o_k - t_k)^2$$

$$= \frac{1}{2}\sum_k 2(o_k - t_k)\frac{\partial}{\partial w_i}(o_k - t_k)$$

$$= \sum_k (o_k - t_k)\frac{\partial o_k}{\partial w_i}$$

$$\frac{\partial o_k}{\partial w_i} = x_i^k$$

Learning Linear Functions:
Computing Gradient

$$\frac{\partial J}{\partial w_i} = \sum_{k=1}^{n} (o_k - t_k) x_i^k$$

$$\nabla J[\vec{w}] = \sum_{k=1}^{n} (o_k - t_k) \vec{x}_k$$

Necessary condition for minimization

$$\frac{\partial J}{\partial w_i} = 0, \qquad i = 0, \ldots, d$$

Equivalently $\nabla J[\vec{w}] = 0$

- This defines a set of linear equations in w_i's. Can be solved explicitly using linear algebra technique.

Learning Linear Functions:
Linear Algebra Solution

- The weight values that minimize the sum-of-squared-error function can be found explicitly by solving the set of linear equations.

$$\sum_{k=1}^{n}\sum_{j=0}^{d} w_j x_j^k x_i^k = \sum_{k=1}^{n} x_i^k t_k$$

$$\mathbf{Y} = \begin{pmatrix} x_0^1 & \cdots & x_d^1 \\ \vdots & \ddots & \vdots \\ x_0^n & \cdots & x_d^n \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} w_0 \\ \vdots \\ w_d \end{pmatrix} \quad \mathbf{t} = \begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix}$$

$$(\mathbf{Y}^t\mathbf{Y})\mathbf{w} = \mathbf{Y}^t\mathbf{t}$$

- *Pseudo-inverse* of Y: $\mathbf{Y}^\dagger = (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t$
- $\vec{w} = \mathbf{Y}^\dagger\mathbf{t}$
- In practice, this solution can lead to numerical difficulties due to the possibility of $\mathbf{Y}^t\mathbf{Y}$ being singular or nearly singular.
- Use the techniques of *singular value decomposition* (SVD)

Learning Linear Functions:
Delta/Adaline/Widrow-Hoff/LMS(Least-Mean-Squared) Rule

$$w_i \leftarrow w_i - \eta \frac{\partial E_S}{\partial w_i}$$

$$\frac{\partial E_S}{\partial w_i} = \frac{1}{2}\frac{\partial}{\partial w_i}\left\{\sum_p e_p^2\right\} = \frac{1}{2}\left(\sum_p \frac{\partial}{\partial w_i}\left(e_p^2\right)\right)$$

$$= \frac{1}{2}\left(\sum_p \left(2e_p\right)\frac{\partial e_p}{\partial w_i}\right) = \sum_p e_p\left(\frac{\partial e_p}{\partial y_p}\right)\left(\frac{\partial y_p}{\partial w_i}\right) = \sum_p e_p\left(-1\right)\left(\frac{\partial}{\partial w_i}\left(\sum_{j=0}^{n} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)\left(\frac{\partial}{\partial w_i}\left(w_i x_{ip} + \sum_{j\neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)\left(\frac{\partial}{\partial w_i}\left(w_i x_{ip}\right) + \frac{\partial}{\partial w_i}\left(\sum_{j\neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)x_{ip}$$

$$w_i \leftarrow w_i + \eta\sum_p \left(d_p - y_p\right)x_{ip}$$

SOGANG UNIVERSITY

Learning Real-Valued Functions

- Universal function approximation theorem
- Learning nonlinear functions using gradient descent in weight space
- Practical considerations and examples

SOGANG UNIVERSITY

Universal function approximation theorem (Cybenko, 1989)

- Let $\varphi : \Re \rightarrow \Re$ be a non-constant, bounded (hence non-linear), monotone, continuous function. Let I_N be the N-dimensional unit hypercube in $\Re^N$.

- Let $C(I_N) = \{f: I_N \rightarrow \Re\}$ be the set of all continuous functions with domain I_N and range $\Re$. Then for any function $f \in C(I_N)$ and any $\varepsilon > 0$, $\exists$ an integer L and a set of real values $\theta, \alpha_j, \theta_j, w_{ji}$ $(1 \le j \le L; 1 \le i \le N)$ such that

$$F(x_1, x_2 \ldots x_n) = \sum_{j=1}^{L} \alpha_j \varphi\left(\sum_{i=1}^{N} w_{ji} x_i - \theta_j \right) - \theta$$

  is a uniform approximation of f – that is,

$$\forall (x_1, \ldots x_N) \in I_N, \quad \left| F(x_1, \ldots x_N) - f(x_1, \ldots x_N) \right| < \varepsilon$$

SOGANG UNIVERSITY

Universal function approximation theorem (UFAT)

$$F(x_1, x_2 \ldots x_n) = \sum_{j=1}^{L} \alpha_j \varphi\left(\sum_{i=1}^{N} w_{ji} x_i - \theta_j \right) - \theta$$

- **Unlike Kolmogorov's theorem, UFAT requires only one kind of nonlinearity to approximate any arbitrary nonlinear function to any desired accuracy**

- **The sigmoid function satisfies the UFAT requirements**

$$\varphi(z) = \frac{1}{1 + e^{-az}}; a > 0 \qquad \lim_{z \to -\infty} \varphi(z) = 0; \lim_{z \to +\infty} \varphi(z) = 1$$

- **Similar universal approximation properties can be guaranteed for other functions (e.g. radial basis functions)**

SOGANG UNIVERSITY

Feed-forward neural networks

- A feed-forward 3-layer network consists of 3 layers of nodes
  - *Input* nodes
  - *Hidden* nodes
  - *Output* nodes

- Interconnected by *modifiable weights* from input nodes to the hidden nodes and the hidden nodes to the output nodes

- More general topologies (with more than 3 layers of nodes, or connections that skip layers – e.g., direct connections between input and output nodes) are also possible

SOGANG UNIVERSITY

Three-layer feed-forward neural network

- **A single *bias* unit is connected to each unit other than the input units**

- **Net *input***

$$n_j = \sum_{i=1}^{N} x_i w_{ji} + w_{j0} = \sum_{i=0}^{N} x_i w_{ji} \equiv \mathbf{W}_j \bullet \mathbf{X},$$

where the subscript i indexes units in the input layer, j in the hidden; w_{ji} denotes the input-to-hidden layer weights at the hidden unit j.

- **The output of a hidden unit is a nonlinear function of its net input. That is, $y_j = f(n_j)$ e.g.,**

$$y_j = \frac{1}{1 + e^{-n_j}}$$

SOGANG UNIVERSITY

Three-layer feed-forward neural network

- **Each output unit similarly computes its net activation based on the hidden unit signals as:**

$$n_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{W}_k \bullet \mathbf{Y},$$

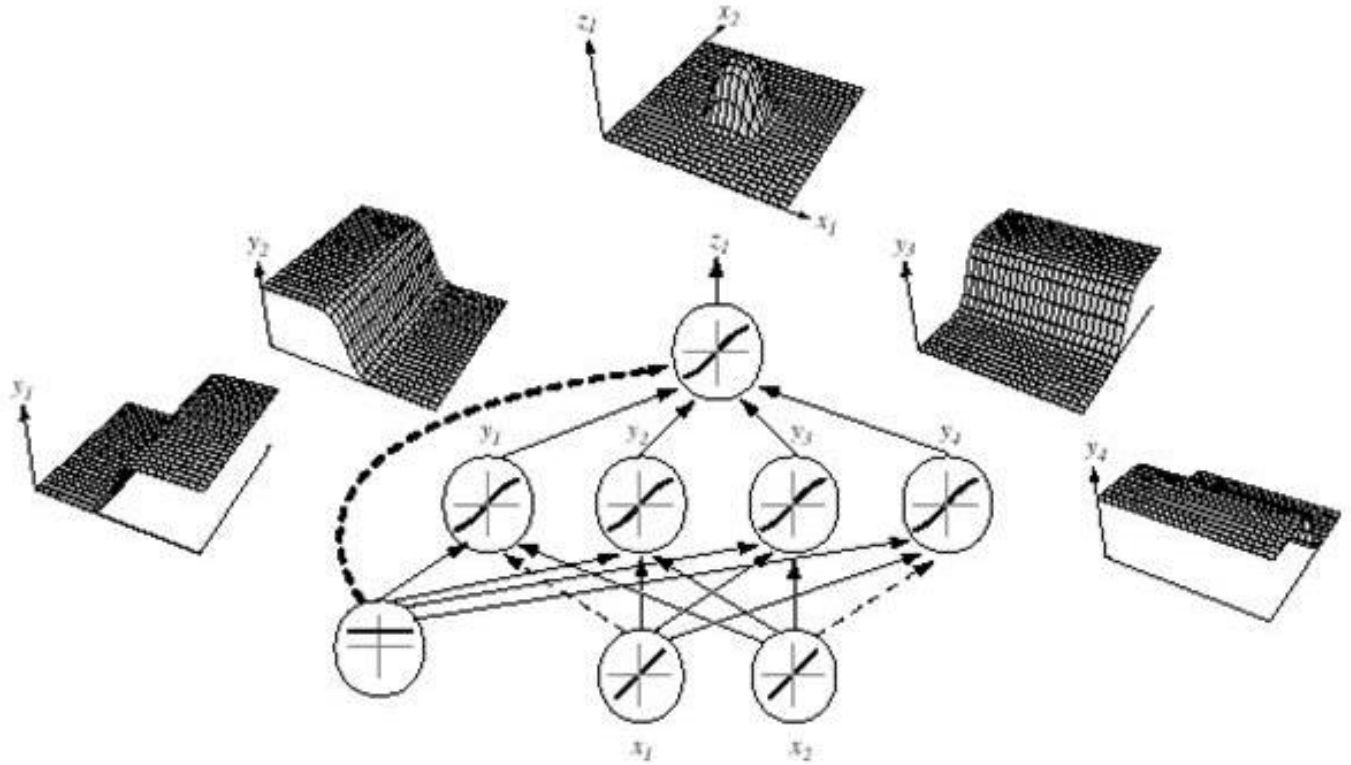
where the subscript k indexes units in the ouput layer and n_H denotes the number of hidden units

- **The output can be a linear or nonlinear function of the net input e.g.,**

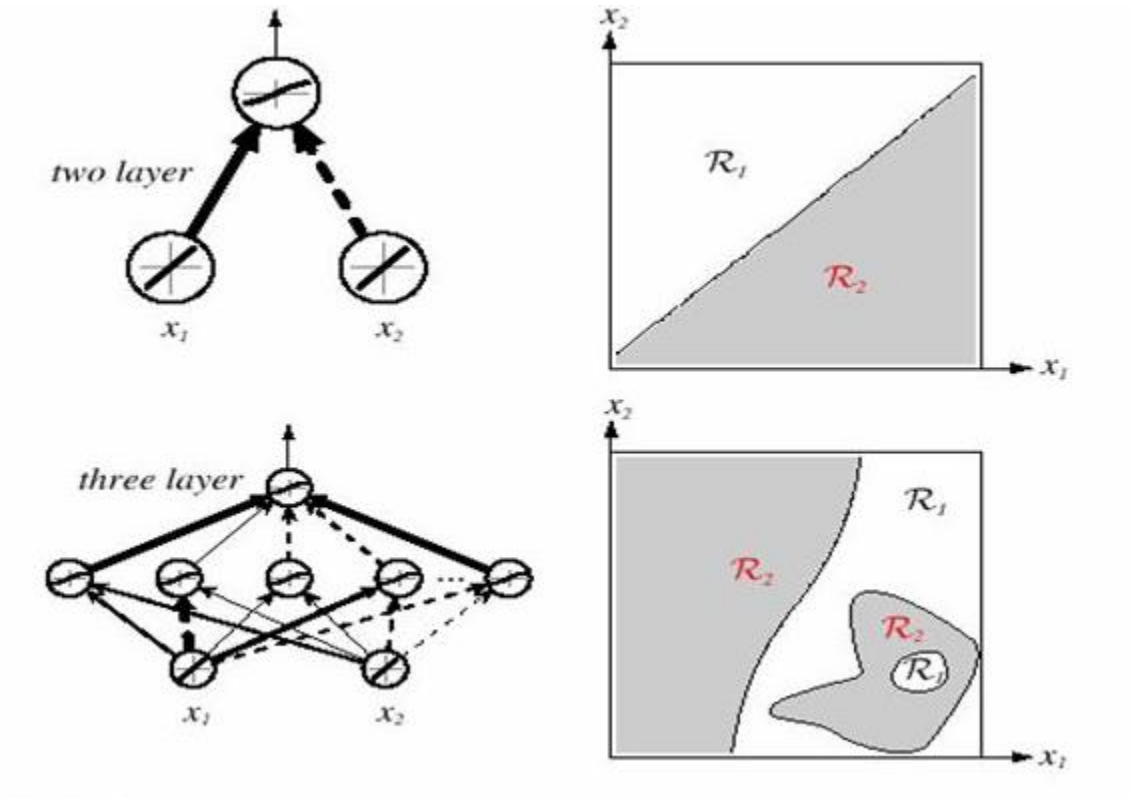
$$y_k = n_k$$

SOGANG UNIVERSITY

Computing nonlinear functions using a feed-forward neural network



Realizing non linearly separable class boundaries using a 3-layer feed-forward neural network



SOGANG UNIVERSITY

Learning nonlinear functions

Given a training set determine:

- **Network structure – number of hidden nodes or more generally, network topology**
  - **Start small and grow the network**
  - **Start with a sufficiently large network and prune away the unnecessary connections**
- **For a given structure, determine the parameters (weights) that minimize the error on the training samples (e.g., the mean squared error)**
- **For now, we focus on the *latter***

SOGANG UNIVERSITY

Generalized delta rule – error back-propagation

- Challenge – we know the desired outputs for nodes in the output layer, but not the hidden layer
- Need to solve the credit assignment problem – dividing the credit or blame for the performance of the output nodes among hidden nodes
- Generalized delta rule offers an elegant solution to the credit assignment problem in feed-forward neural networks in which each neuron computes a differentiable function of its inputs
- Solution can be generalized to other kinds of networks, including networks with cycles

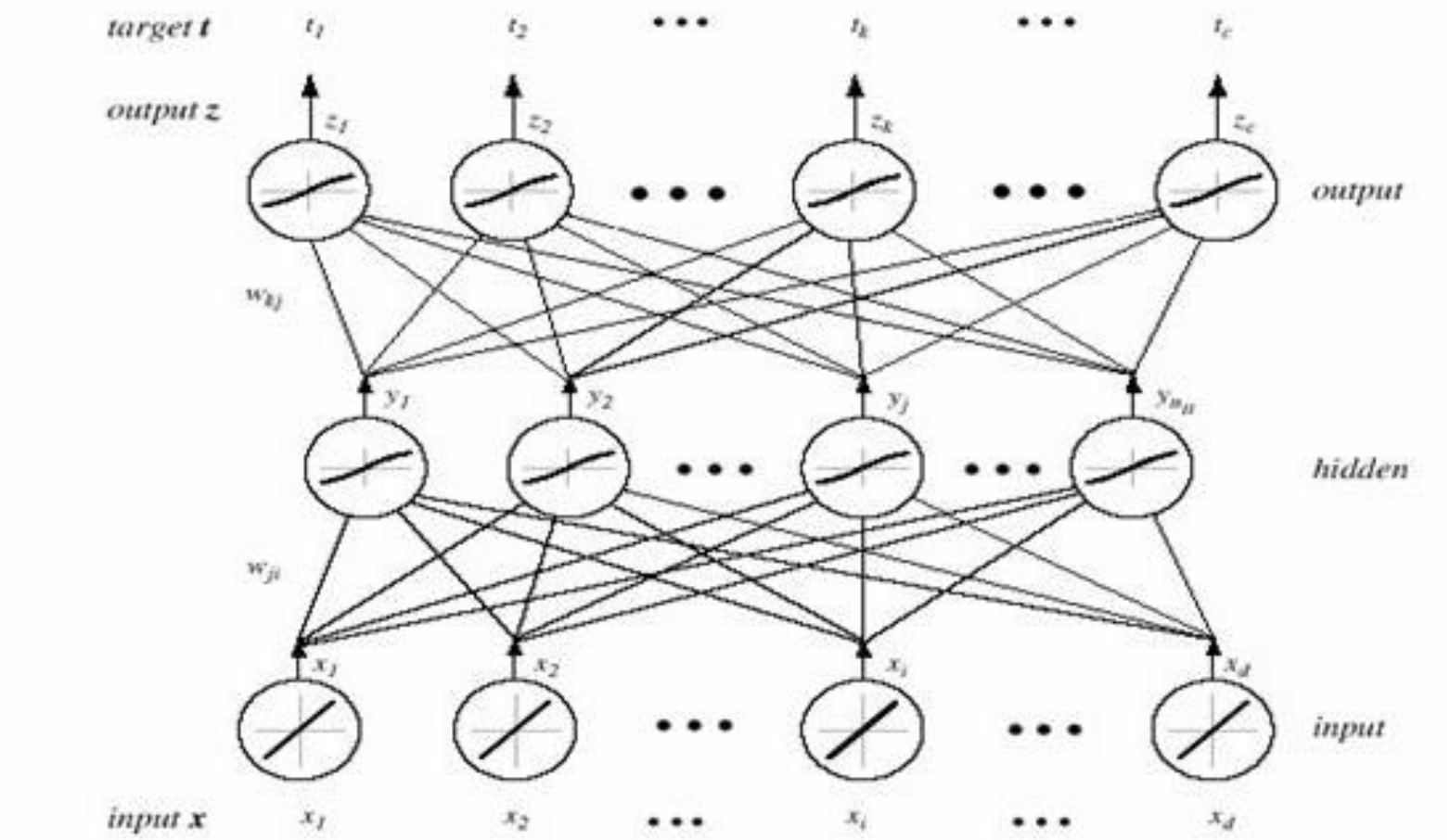
SOGANG UNIVERSITY

Feed-forward networks

- **Forward operation** (computing output for a given input based on the current weights)
- **Learning –** modification of the network parameters (weights) to minimize an appropriate error measure
- Because each neuron computes a differentiable function of its inputs if error is a differentiable function of the network outputs, the error is a differentiable function of the weights in the network – so we can perform gradient descent!

SOGANG UNIVERSITY

A fully connected 3-layer network



SOGANG UNIVERSITY

Generalized delta rule

- Let t_{kp} be the k-th target (or desired) output for input pattern $\mathbf{X}_p$ and z_{kp} be the output produced by k-th output node and let $\mathbf{W}$ represent all the weights in the network

- Training error:

$$E_S(\mathbf{W}) = \frac{1}{2}\sum_p \sum_{k=1}^{M}(t_{kp} - z_{kp})^2 = \sum_p E_p(\mathbf{W})$$

- The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

$$\Delta w_{ji} = -\eta \frac{\partial E_S}{\partial w_{ji}} \qquad \Delta w_{kj} = -\eta \frac{\partial E_S}{\partial w_{kj}}$$

SOGANG UNIVERSITY

Generalized delta rule

$\eta>0$ is a suitable the learning rate $W \leftarrow W + \Delta W$

Hidden–to-output weights

$$\frac{\partial E_p}{\partial w_{kj}} = \frac{\partial E_p}{\partial n_{kp}} \cdot \frac{\partial n_{kp}}{\partial w_{kj}}$$

$$\frac{\partial n_{kp}}{\partial w_{kj}} = y_{jp}$$

$$\frac{\partial E_p}{\partial n_{kp}} = \frac{\partial E_p}{\partial z_{kp}} \cdot \frac{\partial z_{kp}}{\partial n_{kp}} = -(t_{kp} - z_{kp})(1)$$

$$w_{kj} \leftarrow w_{kj} - \eta \frac{\partial E_p}{\partial w_{kj}} = w_{kj} + (t_{kp} - z_{kp}) y_{jp} = w_{kj} + \delta_{kp} y_{jp}$$

Generalized delta rule

Weights from input to hidden units

$$\frac{\partial E_p}{\partial w_{ji}} = \sum_{k=1}^{M} \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial w_{ji}} = \sum_{k=1}^{M} \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial y_{jp}} \cdot \frac{\partial y_{jp}}{\partial n_{jp}} \cdot \frac{\partial n_{jp}}{\partial w_{ji}}$$

$$= \sum_{k=1}^{M} \frac{\partial}{\partial z_{kp}} \left[\frac{1}{2} \sum_{l=1}^{M} (t_{lp} - z_{lp})^2 \right] (w_{kj})(y_{jp})(1 - y_{jp})(x_{ip})$$

$$= -\sum_{k=1}^{M} (t_{kp} - z_{kp})(w_{kj})(y_{jp})(1 - y_{jp})(x_{ip})$$

$$= -\underbrace{\left(\sum_{k=1}^{M} \delta_{kp} (w_{kj})(y_{jp})(1 - y_{jp}) \right)}_{\delta_{jp}} (x_{ip}) = -\delta_{jp} x_{ip}$$

$$w_{ji} \leftarrow w_{ji} + \eta \delta_{jp} x_{ip}$$

SOGANG UNIVERSITY

Back propagation algorithm

Start with small random initial weights

Until desired stopping criterion is satisfied do

- Select a training sample from S
- Compute the outputs of all nodes based on current weights and the input sample
- Compute the weight updates for output nodes
- Compute the weight updates for hidden nodes
- Update the weights

SOGANG UNIVERSITY

Using neural networks for classification

Network outputs are real valued.

How can we use the networks for classification?

$$F(\mathbf{X}_p) = \underset{k}{\operatorname{argmax}} z_{kp}$$

Classify a pattern by assigning it to the class that corresponds to the index of the output node with the largest output for the pattern