

1. A system is an **intelligent system** if it exhibits some intelligent behaviors.

2. **Intelligent Behaviors:** Inference: Deduction vs. Induction (generalization); Learning and adaptation: Evolutionary processes; Creativity

3. **Engineering Applications:** Pattern recognition; Control and robotics; Associative memory; Forecasting

4. **Definition of Neural Networks** Massive parallel distributed processors that have a natural property for storing experiential knowledge and making it available for use

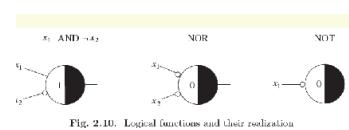


Fig. 2.10. Logical functions and their realization

Perceptron Convergence Algorithm

LMS Learning Algorithm

Gradient Descent Learning Algorithms

- 1) Initialize weights and threshold randomly.
- 2) Calculate actual output of the perceptron:
For all p $y^p = f(\sum_{i=1}^n w_i x_i^p - \theta)$
- 3) Adapt weights:
 $w_i(t+1) = w_i(t) + \eta(z^p - y^p)x_i^p, \eta > 0$
- 4) Repeat until w converges

- 1) Initialize weights and threshold randomly.
- 2) Calculate actual output of the ADALINE:
 $y = \alpha(\sum_i w_i x_i - \theta), \alpha > 0$
- 3) Adapt weights:
 $w_i(t+1) = w_i(t) + \eta(z^p - y^p)x_i^p, \eta > 0$
- 4) Repeat until w converges

$$E = \sum_p E_p = \sum_p (z^p - y^p)^2$$

$$w(t+1) = w(t) - \eta \nabla E_w$$

$$\Delta w(t) = -\eta \nabla E_w$$

$$\nabla E_w = (\partial E / \partial w_1, \partial E / \partial w_2, \dots, \partial E / \partial w_M)^T$$

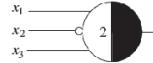


Fig. 2.14. Decoder for the vector (1, 0, 1)

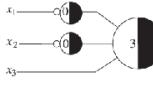


Fig. 2.16. A composite decoder for the vector (0, 0, 1)

5. 6. **Limitations of Perceptron:** Only linearly separable data can be classified. Convergence rate may be low for high-dimensional or large number of data.

Training Modes	Perceptron vs. Adaline
Sequential mode: input training sample pairs one by one orderly or randomly.	Architecture: Perceptron uses bipolar or unipolar hardlimiter activation function, Adaline uses linear activation function.
Batch mode: input training sample pairs in the whole training set at each iteration.	Learning rule: Perceptron learning algorithm is not gradient-descent and can operate in either sequential or batch training mode, whereas Adaline learning (LMS) algorithm is gradient descent, but can only operate in batch mode.
Perceptron learning: either sequential or batch mode.	
ADALINE training: batch mode only.	

Number of Weight Space Regions	Number of Logic Functions vs. Number of Threshold Functions
The number of different regions in weight space defined by m separating hyperplanes in n -dimensional weight space is a polynomial of degree $n-1$ on m : $R(m, n) = 2 \sum_{i=0}^{n-1} \binom{m-1}{i} < 2 \frac{m^n}{n!}$	The number of threshold functions defined by hyperplanes is a function of $2^{n(n-1)}$ whereas that of logical functions is 2^n . The learnability problem: when n is large, there is not enough classification regions in weight space to represent all logical functions.

Backpropagation Algorithm
Also known as generalized delta rule. Invented and reinvented by many researchers, popularized by the PDP group at UC San Diego in 1986. A recursive gradient-descent learning algorithm for multilayer feedforward networks of sigmoid activation function. Compute errors backward from the output layer to input layer. Minimize the mean squares error function.

Sigmoid Activation Functions
• Unipolar: $f(u) = \frac{1}{1 + \exp(-u)}$
$\frac{df(u)}{du} = \frac{\exp(-u)}{(1 + \exp(-u))^2} = \frac{1}{1 + \exp(-u)} - \frac{1 + \exp(-u) - 1}{(1 + \exp(-u))^2} = f(u)[1 - f(u)]$
• Bipolar: $f(u) = \tanh(u) = \frac{1 - \exp(-u)}{1 + \exp(-u)}$

Backpropagation Algorithm (cont'd)
Error function: $E = \sum_p E_p = \frac{1}{2} \sum_p \sum_{i=1}^m (z_i^p - y_i^p)^2$
General formula: $\Delta w_j(t) = w_j(t+1) - w_j(t) = -\eta \frac{\partial E}{\partial w_j} = -\eta \sum_p \frac{\partial E_p}{\partial w_j}$ where $\delta_j^l = -\frac{\partial E_p}{\partial u_j^l} = (z_j^p - y_j^p)y_j^p(1 - y_j^p)$.
Output layer l :
$\frac{\partial E_p}{\partial w_j^l} = \frac{\partial E_p}{\partial y_j^l} \frac{\partial y_j^l}{\partial u_j^l} \frac{\partial u_j^l}{\partial w_j^l} = -(z_j^p - y_j^p)y_j^p(1 - y_j^p)\delta_j^{l-1} = -\delta_j^l v_j^{l-1}$

Hidden layer $l-1$:	Input layer 1:
$-\frac{\partial E_p}{\partial v_i^{l-1}} = -\sum_j \frac{\partial E_p}{\partial u_j^l} \frac{\partial u_j^l}{\partial v_i^{l-1}} = \sum_j (-\frac{\partial E_p}{\partial u_j^l}) \frac{\partial u_j^l}{\partial v_i^{l-1}} = \sum_j \delta_j^l w_{ji}^l$	$\delta_i^1 = (\sum_j \delta_j^2 w_{ji}^2) v_i^1 (1 - v_i^1)$
$\delta_i^{l-1} = -\frac{\partial E_p}{\partial u_i^{l-1}} = -\frac{\partial E_p}{\partial v_i^{l-1}} \frac{\partial v_i^{l-1}}{\partial u_i^{l-1}} = (\sum_j \delta_j^l w_{ji}^l) v_i^{l-1} (1 - v_i^{l-1})$	$\frac{\partial E_p}{\partial w_{ij}^l} = \frac{\partial E_p}{\partial u_i^l} \frac{\partial u_i^l}{\partial w_{ij}^l} = -\delta_i^l x_j^l$
$\frac{\partial E_p}{\partial w_{ij}^l} = \frac{\partial E_p}{\partial u_i^l} \frac{\partial u_i^l}{\partial w_{ij}^l} = -\delta_i^{l-1} v_j^{l-2}$	
$f(x_1, x_2, \dots, x_n) = \sum_{j=1}^{n+1} g_l(\sum_{i=1}^n w_{ij} h_j(x_i))$	

10. 11. An RBF network can transform the linearly inseparable XOR data in the input space to linearly separable data in the hidden state space.

Backpropagation Algorithm (cont'd)

- 1) Initialize weights and threshold randomly. To avoid local oscillation, a momentum term is sometimes added:
- 2) Calculate actual output of the MLP:
- 3) Adapt weights for all layers:
- 4) Repeat until w converges

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-1)$$

$$0 \leq \alpha < 1$$

Momentum Term

Backpropagation Algorithm (cont'd)

Radial Basis Functions

Cover's Theorem (1965):

- A dichotomy $\{X^+, X^-\}$ is said to be **φ -separable** if there exist an m -dimensional vector w such that
- $w^T \varphi(x) \geq 0$, if x in X^+
 - $w^T \varphi(x) < 0$, if x in X^-
 - The hyperplane defined by $w^T \varphi(x) = 0$ is the separating surface between the two classes.

Kolmogorov Theorem
Let $f: [0, 1]^n \rightarrow [0, 1]$ be a continuous function. There exist functions of one argument g and h_j for $j=1, 2, \dots, 2n+1$ and constant w_i for $i=1, 2, \dots, n$ such that
$f(x_1, x_2, \dots, x_n) = \sum_{j=1}^{n+1} g_l(\sum_{i=1}^n w_{ij} h_j(x_i))$

Universal Approximators

Multilayer feedforward neural networks are universal approximators of continuous functions.

A set of weights exist such that the approximation errors can be arbitrarily small.

However, the BP algorithm is not guaranteed to find such a set of weights.

Linear Discriminant Function
$g(\mathbf{x})$ is a linear function:
$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$
A hyper-plane in the feature space
(Unit-length) normal vector of the hyper-plane:
$\mathbf{n} = \frac{\mathbf{w}}{\ \mathbf{w}\ }$

Maximal Margin Classifier

Given a set of data:

$\{(x_i, y_i)\}, i=1, 2, \dots, n$, where

For $y_i = +1$, $\mathbf{w}^T \mathbf{x}_i + b > 0$

For $y_i = -1$, $\mathbf{w}^T \mathbf{x}_i + b < 0$

With a scale transformation on both \mathbf{w} and b , the above is equivalent to

For $y_i = +1$, $\mathbf{w}^T \mathbf{x}_i + b \geq 1$

For $y_i = -1$, $\mathbf{w}^T \mathbf{x}_i + b \leq -1$

Functional Link Network	Functional Link Network	Extreme Learning Machine	Support Vector Machine
Proposed by Yoh-Han Pao at CWRU in late 80's	Proposed by Guangbin Huang at NTU in mid 2000's	One-layer feedforward architecture	Proposed by Vladimir Vapnik based on statistical learning and kernel theory in early 1990's.
One-layer feedforward architecture	Random connection weights from inputs to hidden neurons	Fast learning process for weights in output layer.	Minimization of structural risk.
Higher-order aggregation rule		Local minima eliminated	Maximal generalization power.
Fast learning process			
Local minima could be eliminated			
Many successful stories in applications			

13.

14.

15.

16.

17.

18.

19.

20.

21.

22.

23.

24.

25.

26.

27.

28.

29.

30.

31.

32.

33.

34.

35.

36.

37.

38.

39.

40.

41.

42.

43.

44.

45.

46.

47.

48.

49.

50.

51.

52.

53.

54.

55.

56.

57.

58.

59.

60.

61.

62.

63.

64.

65.

66.

67.

68.

69.

70.

71.

72.

73.

74.

75.

76.

77.

78.

79.

80.

81.

82.

83.

84.

85.

86.

87.

88.

89.

90.

91.

92.

93.

94.

95.

96.

97.

98.

99.

100.

101.

102.

103.

104.

105.

106.

107.

108.

109.

110.

111.

112.

113.

114.

115.

116.

117.

118.

119.

120.

121.

122.

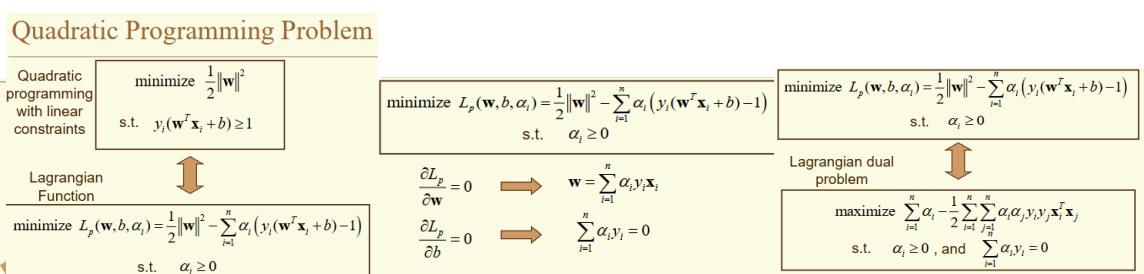
123.

124.

125.

126.

127.



Linear Discriminant Function

The linear discriminant function is:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i \in SV} \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

It is a weighted dot product between the test point \mathbf{x} and the support vectors \mathbf{x}_i .

Solving the optimization problem involved computing the dot products $\mathbf{x}_i^T \mathbf{x}$ between all pairs of training samples

Feature Space

General idea: the original input space can be mapped to a higher-dimensional feature space where the training set is linearly separable

For some functions $K(\mathbf{x}_i, \mathbf{x}_j)$ checking that

$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ can be cumbersome.

Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

$$\begin{matrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & K(\mathbf{x}_1, \mathbf{x}_3) & \dots & K(\mathbf{x}_1, \mathbf{x}_N) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & K(\mathbf{x}_2, \mathbf{x}_3) & & K(\mathbf{x}_2, \mathbf{x}_N) \\ \dots & \dots & \dots & \dots & \dots \\ K(\mathbf{x}_N, \mathbf{x}_1) & K(\mathbf{x}_N, \mathbf{x}_2) & K(\mathbf{x}_N, \mathbf{x}_3) & \dots & K(\mathbf{x}_N, \mathbf{x}_N) \end{matrix}$$

SVM Learning

- Choose a kernel function
- Choose a value for C
- Solve the quadratic programming problem (many software packages available)
- Construct the discriminant function from the support vectors

Choice of kernel

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

Choice of kernel parameters

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

Optimization criterion – Hard margin v.s. Soft margin

- a lengthy series of experiments in which various parameters are tested

$$\begin{aligned} L &= \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^l (\xi_i + \xi_i^*) - \sum_{i=1}^l (\eta_i \xi_i + \eta_i^* \xi_i^*) \\ &\quad - \sum_{i=1}^l \alpha_i (\varepsilon + \xi_i - y_i + \mathbf{w} \cdot \mathbf{x}_i + b) \\ &\quad - \sum_{i=1}^l \alpha_i^* (\varepsilon + \xi_i^* + y_i - \mathbf{w} \cdot \mathbf{x}_i - b) \end{aligned}$$

$$\min_{w, b, \xi} L$$

subject to $\alpha, \eta \geq 0$

Common Kernel Functions

Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$

Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$

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

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

Design Issues

Choice of kernel

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

Choice of kernel parameters

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

Optimization criterion – Hard margin v.s. Soft margin

- a lengthy series of experiments in which various parameters are tested

$$\frac{\partial L}{\partial b} = \sum_{i=1}^l (\alpha_i^* - \alpha_i) = 0$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^l (\alpha_i - \alpha_i^*) \mathbf{x}_i = 0$$

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0$$

$$\max_{\alpha, \eta} -\frac{1}{2} \sum_{i,j=1}^l (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \mathbf{x}_i \cdot \mathbf{x}_j - \varepsilon \sum_{i=1}^l (\alpha_i + \alpha_i^*) + \sum_{i=1}^l y_i (\alpha_i - \alpha_i^*)$$

subject to $\sum_{i=1}^l (\alpha_i - \alpha_i^*) = 0$ and $\alpha_i, \alpha_i^* \in [0, C]$

Least-squares SVM

Proposed by Suykens and Vandewalle at KUL in 1999.

Let $y_i = \mathbf{w}^T \varphi(\mathbf{x}_i) + b + e_i$

Primal problem formulation

$$\min_{w, b, e_i} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \gamma \frac{1}{2} \sum_{i=1}^N e_i^2$$

$$y_i = \mathbf{w}^T \varphi(\mathbf{x}_i) + b + e_i, \quad i = 1, \dots, N.$$

Least-squares SVM

Lagrangian

$$\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \gamma \frac{1}{2} \sum_{i=1}^N e_i^2 - \sum_{i=1}^N \alpha_i (y_i(\mathbf{w}^T \varphi(\mathbf{x}_i) + b + e_i) - y_i)$$

$\frac{\partial \mathcal{L}}{\partial w} = 0, \frac{\partial \mathcal{L}}{\partial b} = 0, \frac{\partial \mathcal{L}}{\partial e_i} = 0, \frac{\partial \mathcal{L}}{\partial \alpha_i} = 0$ gives

$$\begin{bmatrix} \Omega + \gamma^{-1} \mathbf{I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix}$$

Least-squares SVM

Final model

$$y(\mathbf{x}) = \sum_{i=1}^N \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$