Southern University of Science and Technology

SDM359 Advanced Machine Learning Spring 2024 Professor Liu

Tuesday 02.20 *Lecture no. 1.1*

Derong Liu, SDIM – Automation

CoE North 215, liudr@sustech.edu.cn, L13810670526

Ph.D. in EE, 1994, University of Notre Dame

Go over the syllabus

http://derongliu.org/sdm359/

Course outline. The book has 13 chapters.

	Chapters	Authors
Intro	1	All 3
NN	2,3,4,5	Liu
FS	6,7,8,9	Keller
EC	10,11,12,13	Fogel

Three former EiCs of IEEE Transactions.

Computational Intelligence = NN + FS + EC, 5 weeks for each subject.

IEEE: Neural Networks Council (Nov. 1989)
→ Neural Networks Society (Nov. 2001)
→ Computational Intelligence Society (Nov. 2003)

International Neural Network Society: Formed in 1987.

– JNNS: Japanese NNS

– ENNS: European NNS

– APNNS: Asia-Pacific NNS

International Fuzzy Systems Association: Founded in 1984.

Chapter 1: Introduction to Computational Intelligence

AI: Artificial Intelligence (AI) is the study of intelligent behavior demonstrated by machines as opposed to the natural intelligence in human beings. It is concerned with the development of technologies that enable a machine or computer to think, behave, or act in a way like human beings.

Artificial Intelligence studies include
– $\underbrace{\text{computer vision, image analysis}}_{eyes\ (brain)}$,
– $\underbrace{\text{natural language processing, speech recognition}}_{mouth,\ ears\ (brain)}$,
– $\underbrace{\text{machine learning}}_{brain}$,
– etc.

Machine Learning:
– Supervised Learning
– Semi-Supervised Learning
– Unsupervised Learning
– Reinforcement Learning

CI: Computational Intelligence (CI) is the theory, design, application and development of biologically and linguistically motivated computational paradigms. It encompasses computing paradigms like ambient intelligence, artificial life, cultural learning, artificial endocrine networks, social reasoning, and artificial hormone networks. The three main pillars of CI are Neural Networks (NN), Fuzzy Systems (FS) and Evolutionary Computation (EC).

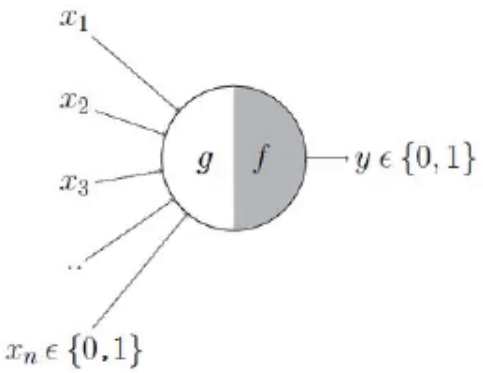
Machine Learning:
– Supervised Learning (NN, FS, EC)
– Semi-Supervised Learning (NN, FS, EC)
– Unsupervised Learning (NN, FS, EC)
– Reinforcement Learning (NN, FS, EC)

Page 3: Things computer cannot do as well as humans (2015):
– Qualitative classification
– Going from specific to general, or vice versa
– Consciousness and emotion
– Driving a car
– Writing a poem
– Chatting
– Shopping
– Handling inaccuracies in problems
– Ethics
– Natural language in conversation, with idioms
– Face recognition
– Aesthetics
– Adaptivity
– Learning (like humans do)

Chapter 2: Single-Layer Neural Networks

2.1 Short History

McCulloch-Pitts Neuron:
A logical calculus of the ideas immanent in nervous activity, *The Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, December 1943.



$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

A linear summation → A nonlinear activation.

Rosenblatt Perceptron:
Rosenblatt, F. (1958) The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 386–408.

2.2 Perceptron

Perceptron = Single-layer neural network

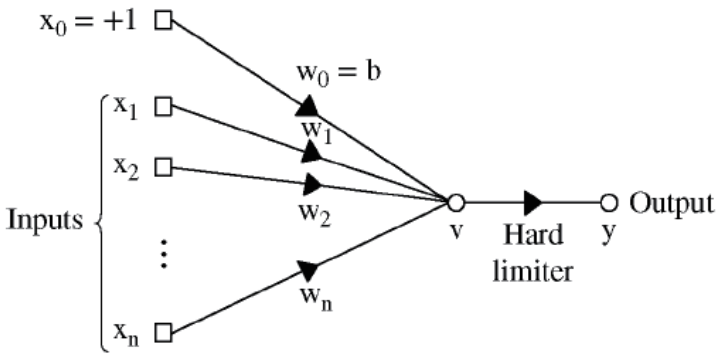


FIGURE 2.7 Equivalent signal flow graph of the perceptron.

ASSIGNMENTS

♠ ♠ ♠ ♠ ♠ ♠ ♠ ♠

Expressions:

$$v = wx + b = \sum_{i=1}^{n} w_i x_i + b,$$

$$y = f(v),$$

$$f(v) = \begin{cases} 1, & \text{if } v > 0, \\ 0, & \text{if } v < 0, \end{cases}$$

$$w = [w_1, w_2, \ldots, w_n],$$

$$x = [x_1, x_2, \ldots, x_n]^T,$$

$$v = \bar{w}\bar{x} = \sum_{i=0}^{n} w_i x_i,$$

$$\bar{w} = [w_0, w_1, w_2, \ldots, w_n] = [b, w_1, w_2, \ldots, w_n],$$

$$\bar{x} = [x_0, x_1, x_2, \ldots, x_n]^T = [1, x_1, x_2, \ldots, x_n]^T.$$

It is an artificial neural network with a single layer: Rosenblatt (1958) called it Perceptron.

x_i – inputs

y – output

w_i – connection weights

$f(\cdot)$ – hard limiter (activation function)

b – bias or threshold

When $n = 2$, it is a two-class pattern classification problem.

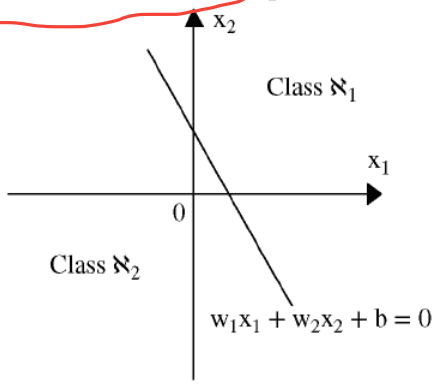


FIGURE 2.6 Illustration of the two-dimensional, two-class pattern classification problem.

Class 1: $v > 0$ $(y = 1)$; Class 2: $v < 0$ $(y = 0)$.

2.3 Perceptron Training Algorithm

Example: $(n = 2)$.

Class 1: $v > 0,\ (0,1),(1,0)$.

Class 2: $v < 0,\ (-1,0),(0,-1)$.

These two classes can easily be illustrated.

We need to determine $w = (w_1, w_2)$ and b such that
when $x = [0,1]^T$ or $[1,0]^T$, we have $y = f(wx + b) = 1$ or $wx + b > 0$, and
when $x = [-1,0]^T$ or $[0,-1]^T$, we have $y = f(wx + b) = 0$ or $wx + b < 0$.

Perceptron Learning Rules:
– Result is correct, no change in weights, or keep the same values.
– Result is incorrect, modify the weights.

How to modify the weights?

There are two possible errors:

1. $y_{\text{target}} = 1$ but $wp + b < 0$.

   In this case, $w^{\text{new}} = w^{\text{old}} + p$.

2. $y_{\text{target}} = 0$ but $wp + b > 0$.

   In this case, $w^{\text{new}} = w^{\text{old}} - p$.

♣ Why such learning rules?

Initial weights are chosen as $w_1 = 0$, $w_2 = 0$, and $b = 0$.

	$w_1x_1 + w_2x_2 + b$	y	Correct?	Update weights
$(0, 1)$	0	?	N	$w_1 = 0, w_2 = 1, b = 1$
$(1, 0)$	1	1	Y	N
$(-1, 0)$	1	1	N	$w_1 = 1, w_2 = 1, b = 0$
$(0, -1)$	-1	0	Y	N
$(0, 1)$	1	1	Y	N
$(1, 0)$	1	1	Y	N
$(-1, 0)$	-1	0	Y	N

Initial weights are chosen as $w_1 = -1$, $w_2 = 2$, and $b = 0$.

	$w_1x_1 + w_2x_2 + b$	y	Correct?	Update weights
$(0, 1)$	2	1	Y	N
$(1, 0)$	-1	0	N	$w_1 = 0, w_2 = 2, b = 1$
$(-1, 0)$	1	1	N	$w_1 = 1, w_2 = 2, b = 0$
$(0, -1)$	-2	0	Y	N
$(0, 1)$	2	1	Y	N
$(1, 0)$	1	1	Y	N
$(-1, 0)$	-1	0	Y	N

Initial weights are chosen as $w_1 = 1$, $w_2 = 1$, and $b = 1$.

	$w_1x_1 + w_2x_2 + b$	y	Correct?	Update weights
$(0, 1)$	2	1	Y	N
$(1, 0)$	2	1	Y	N
$(-1, 0)$	0	?	N	$w_1 = 2, w_2 = 1, b = 0$
$(0, -1)$	-1	0	Y	N
$(0, 1)$	1	1	Y	N
$(1, 0)$	2	1	Y	N
$(-1, 0)$	-2	0	Y	N

♣ Solution is not unique.

♣ Solution depends on the initial conditions.

♣ Solution depends on way patterns are preseted.

♣ It is possible to choose an initial condition which is a solution that requires no learning, e.g., $w_1 = 1, w_2 = 1, b = 0$.

2.19作业 => 2.25交

ASSIGNMENTS

Homework #1: 2.3, 2.6, 2.7. Due 02-29.

♠ ♠ ♠ ♠ ♠ ♠ ♠ ♠

2.3 Perceptron Training Algorithm

Perceptron Learning Rules:

- Denote $\theta = f(wp + b)$ and $t = y_{\text{target}}$. Let $e = t - \theta$.
- Result is correct $\rightarrow e = 0$, no change in weights, or keep the same values.
- Result is incorrect $\rightarrow e \neq 0$, modify the weights.

Two cases for $e \neq 0$:

1. $e = 1$ or $(t = 1$ and $\theta = 0) \rightarrow w^{\text{new}} = w^{\text{old}} + p = w^{\text{old}} + ep$.
2. $e = -1$ or $(t = 0$ and $\theta = 1) \rightarrow w^{\text{new}} = w^{\text{old}} - p = w^{\text{old}} + ep$.

The update for the bias: $b^{\text{new}} = b^{\text{old}} + e = b^{\text{old}} + ep$.

♣ This is for the case of perceptron with a single output neuron.
It can be extended to multiple perceptrons.

Multiple Perceptrons:

♣ This is for the case of perceptron with multiple output neurons.

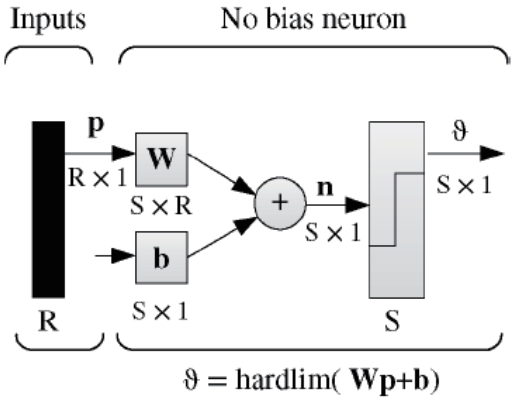


FIGURE 2.17 Test problem multiple-neuron network.

The output has S neurons.

The output $\theta = f(Wp + b) \in R^{S\times 1}$.

Each element of θ is a single perceptron, and there are S such perceptrons.

$p \in R^n \quad p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_s \end{bmatrix}$

$W = \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots \\ w_{1,2} & w_{2,2} & \\ w_{1,3} & w_{3,2} & \\ \vdots & \vdots & \\ w_{1,s} & w_{s,2} & w_{s,s} \end{bmatrix} \} n$

n layer perceptrons

Learning Rules for Multiple Perceptrons:

The connection matrix W has S rows, and each row is updated using

$w^{i,\text{new}} = w^{i,\text{old}} + e_i p,\ i = 1, 2, \ldots, S.$

The ith element of the bias vector is updated as:

$b_i^{\text{new}} = b_i^{\text{old}} + e_i.$

2.4 Perceptron Convergence Theorem

Theorem 2.1: Let the training subsets be linearly separable. Let the inputs be presented to the perceptron from these two subsets. The perceptron converges after some k_0 iterations such that

$$w(k_0) = w(k_0 + 1) = w(k_0 + 2) = \cdots$$

is a solution vector for $k_0 < \infty$.

♣ The proof is simple. $k_0 < \infty$ means convergence in finite number of training steps.

♣ The training sets must be linearly separable. Our previous example is linearly separable.

- The XOR problem is not linearly separable.
- After the 1959 Rosenblatt paper, Minsky published a book in 1969,
  Marvin Minsky and Seymour A. Papert, *Perceptrons – An Introduction to Computational Geometry*, The MIT Press, 1969.
  ♣ They pointed out that perceptrons cannot even solve the simple problem of XOR (because they are not linearly separable).
  ♣ To solve the XOR problem, at least a two-layer neural network is required but there are no training algorithms available in 1969.
- Neural network research was dead until 1980s.

HW2 ddl 3.2

2.5 Computer Experiment

Computer programming for a two-dimensional pattern classification problem.

Class 1: $(2, 2)$ and $(-2, 2)$, where $v = w_1x_1 + w_2x_2 + b < 0$.
Class 2: $(1, -2)$ and $(-1, 0)$, where $v = w_1x_1 + w_2x_2 + b \geq 0$.

♣ Similar to the example we used in the last lecture. Note the choice of activation function.

ASSIGNMENTS

–

♠ ♠ ♠ ♠ ♠ ♠ ♠ ♠

2.6 Activation Functions

1. Threshold function

$$f(v) = \phi(v) = \begin{cases} 1, & \text{if } v \geq 0, \\ 0, & \text{otherwise,} \end{cases}$$

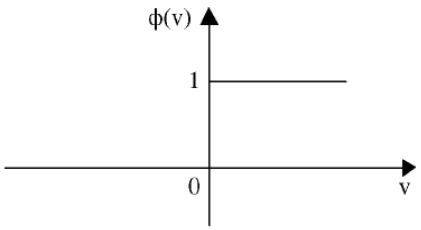


FIGURE 2.20 The threshold function from Eq. 2.57.

2. Signum function

$$\phi(v) = \begin{cases} 1, & \text{if } v > 0, \\ 0, & \text{if } v = 0, \\ -1, & \text{if } v < 0, \end{cases}$$

(可有可无)

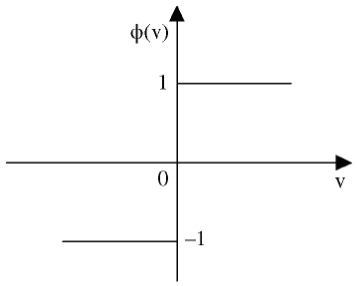


FIGURE 2.22 The signum function.

3. Sigmoid function: ♣ Strictly increasing and differentiable.

Logistic function:

$$\phi(v) = \frac{1}{1 + e^{-av}}$$

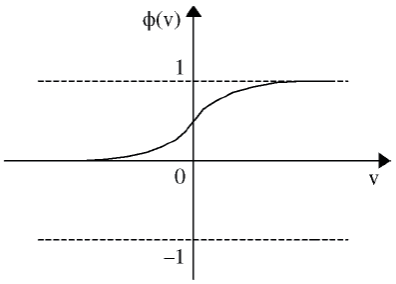


FIGURE 2.21 The logistic function from Eq. 2.60.

特点

1. strictly increasing

2. differentiable

Hyperbolic tangent function:

$$\phi(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}.$$

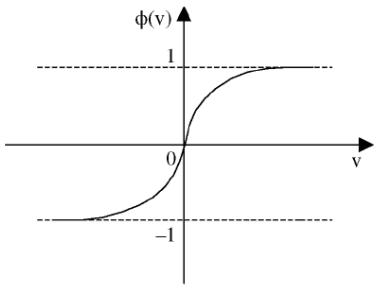


FIGURE 2.23 The hyperbolic tangent function.

Chapter 3: Multilayer Neural Networks and Backpropagation

3.1 Universal Approximation Theory

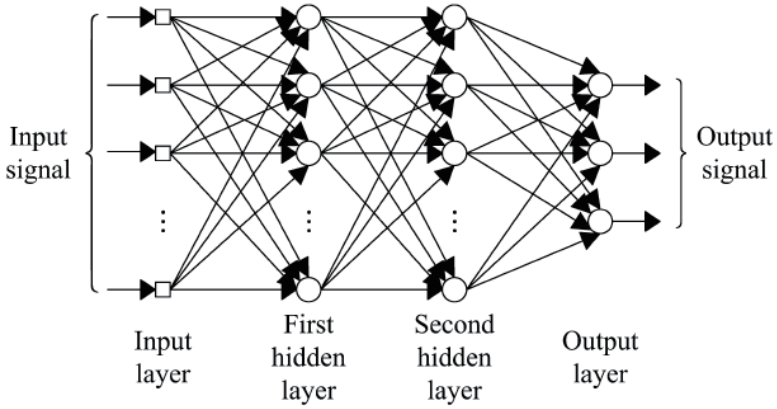


FIGURE 3.1 Architectural graph of a multilayer perceptron with two hidden layers.

Inputs: $x_1, x_2, \ldots, x_{n_0}$.
Outputs: $y_1, y_2, \ldots, y_n$.
Activation function: $\phi(\cdot)$.

Hidden layer (first hidden layer) neurons' inputs: $\sum_{j=1}^{n_0} w_{ij}x_j + b_i,\ i = 1, 2, \ldots, n_1$, or $wx + b = \bar{w}\bar{x}$.

Hidden layer (first hidden layer) neurons' outputs: $\phi(wx + b)$.

– Each neuron may have its own different activation function.
– It is important the hidden layer neurons are nonlinear.

$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n_0} \\ & & & w_{2n_0} \\ & & & \vdots \\ w_{n_1 1} & & & w_{n_1 n_0} \end{bmatrix} \in \mathbb{R}^{n_1 \times n_0}$

$Wx = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + \cdots + w_{1n_0}x_{n_0} \\ \\ w_{n_1}x_1 + \cdots + w_{n_1 n_0}x_{n_0} \end{bmatrix}$

Theorem 3.1: Let $\phi(\cdot)$ be nonconstant, bounded, and monotone-increasing continuous function. Let I_{n_0} denote the n_0-dimensional unit hypercube $[0,1]^{n_0}$. The space of continuous functions on I_{n_0} is denoted by $C(I_{n_0})$. Then, given any function $f \in C(I_{n_0})$ and a small $\epsilon > 0$, there exist an integer n_1 and sets of real constants μ_i and b_i, and w_{ij}, where $i = 1, 2, \ldots, n_1$ and $j = 1, 2, \ldots, n_0$ such that we may define

$$F(x_1, x_2, \ldots, x_{n_0}) = \sum_{i=1}^{n_1} \mu_i \phi \left(\sum_{j=1}^{n_0} w_{ij} x_j + b_i \right)$$

as an approximate realization of the function f, that is,

$$|F(x_1, x_2, \ldots, x_{n_0}) - f(x_1, x2, \ldots, x_{n_0})| < \epsilon$$

for all $x_1, x2, \ldots, x_{n_0}$ that lie in the input space.

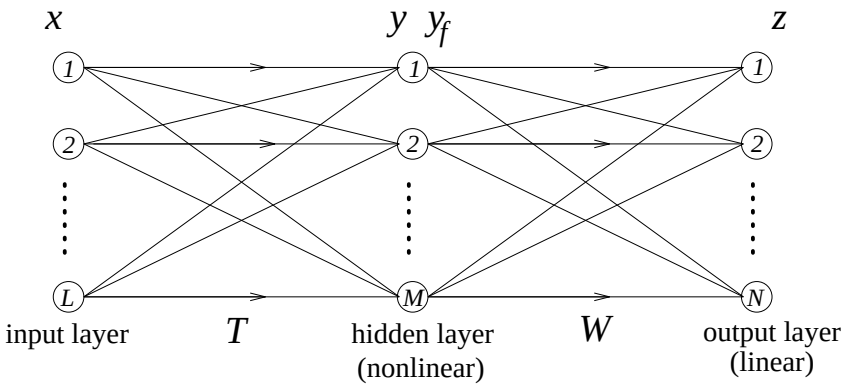
♣ To approximate a nonlinear function to any degree of accuracy, one hidden layer is enough.
♣ The output layer can be linear or nonlinear.
♣ How to obtain such a neural network?

3.2 The Backpropagation Training Algorithm

How to train/obtain a multi-layer neural network?
– Multi-layer neural network can solve the XOR problem, but how to train such a neural network?
– It was not known since 1959 (Rosenblatt paper), until 1986, two papers by Hinton:
1. David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams, "Learning representations by back-propagating errors," *Nature*, volume 323, pages 533–536, 1986.
2. David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams, "Learning international representations by error bropagation," in D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Volume 1: Foundations, MIT Press, Cambridge, MA, 1986 (Chapter 8).
– It was later discovered the BP algorithm has appeared earlier in 1974 Paul Werbos PhD thesis. 1986 Hinton's papers did not cite Paul Werbos thesis.

Most important literature for BP algorithm:
♣ Paul Werbos 1974 Harvard PhD thesis, and in 1994 published as a book:
Paul J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, New York: John Wiley, 1994.
♣ Two papers by Hinton: 1986
Nature and the *PDP book*.

Assume a neural network with only one hidden layer and with linear activation function at the output, i.e.,



Relations:

$$y = Tx \quad \text{or} \quad y_i = \sum_{l=1}^{L} t_{il} x_l \text{ for } i = 1, 2, \cdots, M.$$

$$y_f = \phi(y) \quad \text{or} \quad y_{fi} = \phi(y_i) = \phi\left(\sum_{l=1}^{L} t_{il} x_l\right) \text{ for } i = 1, 2, \cdots, M.$$

$$z = W y_f \quad \text{or} \quad z_j = \sum_{i=1}^{M} w_{ji} y_{fi} \text{ for } j = 1, 2, \cdots, N.$$

Error function:

$$E = \frac{1}{2}\sum_{j=1}^{N} e_j^2 \triangleq \frac{1}{2}\sum_{j=1}^{N}(d_j - z_j)^2 = \frac{1}{2}\sum_{j=1}^{N}\left(d_j - \sum_{i=1}^{M} w_{ji} y_{fi}\right)^2$$

$$= \frac{1}{2}\sum_{j=1}^{N}\left[d_j - \sum_{i=1}^{M} w_{ji}\phi\left(\sum_{l=1}^{L} t_{il} x_l\right)\right]^2$$

where d_j is the desired output for neuron j of the output layer.

Updating rules: New value = Old value + Updates.

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

$$t_{il}(k+1) = t_{il}(k) + \Delta t_{il}(k),$$

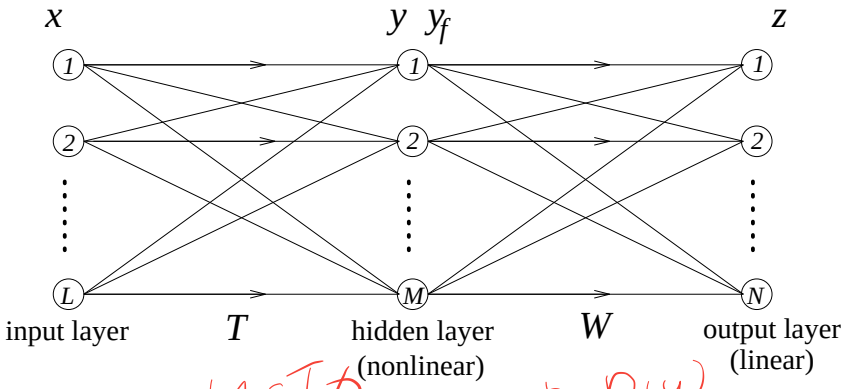
where k indicates the kth updates.

ASSIGNMENTS

Homework #2: Matlab for 2.5 Computer Experiment (Program + display of correct results). Due 02-29.

♠ ♠ ♠ ♠ ♠ ♠ ♠ ♠

3.2 The Backpropagation Training Algorithm



Forward relations:

$$y = Tx, \quad y_f = \phi(y), \quad z = Wy_f.$$

Error function:

$$E = \frac{1}{2}\sum_{j=1}^{N} e_j^2 \triangleq \frac{1}{2}\sum_{j=1}^{N}(d_j - z_j)^2 = \frac{1}{2}\sum_{j=1}^{N}\left(d_j - \sum_{i=1}^{M} w_{ji}y_{fi}\right)^2$$

$$= \frac{1}{2}\sum_{j=1}^{N}\left[d_j - \sum_{i=1}^{M} w_{ji}\phi\left(\sum_{l=1}^{L} t_{il}x_l\right)\right]^2$$

where d_j is the desired output for neuron j of the output layer.

Updating rules: New value = Old value + Updates.

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

$$t_{il}(k+1) = t_{il}(k) + \Delta t_{il}(k),$$

where k indicates the kth updates.

The updates are given by (gradient descent):

$$\Delta w_{ji}(k) = -\eta\frac{\partial E}{\partial w_{ji}} = \eta\delta_j^z(k)y_{fi}(k),$$

$$\Delta t_{il}(k) = -\eta\frac{\partial E}{\partial t_{il}} = \eta\delta_i^y(k)x_l(k),$$

where η is the learning rate.

The gradients:

$$\begin{aligned}\frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial e_j} \cdot \frac{\partial e_j}{\partial w_{ji}} \\ &= (d_j(k) - z_j(k)) \cdot (-y_{fi}(k)) \\ &= -\delta_j^z(k) y_{fi}(k),\end{aligned}$$

where $\delta_j^z(k) = d_j(k) - z_j(k)$, and

$$\begin{aligned}\frac{\partial E}{\partial t_{il}} &= \sum_{j=1}^{N} \frac{\partial E}{\partial e_j} \cdot \frac{\partial e_j}{\partial t_{il}} \\ &= \sum_{j=1}^{N} (d_j(k) - z_j(k)) \cdot (-w_{ji}(k)) \cdot \phi'(y_i)(k) \cdot x_l(k) \\ &= -\phi'(y_i)(k) \left(\sum_{i=1}^{N} w_{ji}(k) \delta_j^z(k) \right) x_l(k) \\ &= -\delta_i^y(k) x_l(k),\end{aligned}$$

where

$$\delta_i^y(k) = \phi'(y_i)(k) \left(\sum_{i=1}^{N} w_{ji}(k) \delta_j^z(k) \right).$$

Note that:

Signals: $x \to y \to z$ forward propagation, but error signals: $\delta^z \to \delta^y$ back-propagation.

Easy calculation of derivatives of the activation function.

1. Logistic function.

$$\phi(v) = \frac{1}{1+e^{-av}}.$$

The derivative is

$$\begin{aligned}\phi'(v) &= \frac{ae^{-av}}{(1+e^{-av})^2} \\ &= a \cdot \frac{1}{1+e^{-av}} \cdot \left(1 - \frac{1}{1+e^{-av}}\right) \\ &= a\phi(v)[1-\phi(v)].\end{aligned}$$

---

2. Hyperbolic tangent function.

$$\phi(v) = a\tanh(bv) = a\frac{e^{bv}-e^{-bv}}{e^{bv}+e^{-bv}}.$$

The derivative is

$$\begin{aligned}\phi'(v) &= a\frac{be^{bv}+be^{-bv}}{e^{bv}+e^{-bv}} - a\frac{(e^{bv}-e^{-bv})(be^{bv}-be^{-bv})}{(e^{bv}+e^{-bv})^2} \\ &= ba\left(1+\frac{e^{bv}-e^{-bv}}{e^{bv}+e^{-bv}}\right)\left(1-\frac{e^{bv}-e^{-bv}}{e^{bv}+e^{-bv}}\right) \\ &= \frac{b}{a}\left(a+a\frac{e^{bv}-e^{-bv}}{e^{bv}-e^{-bv}}\right)\left(a+a\frac{e^{bv}+e^{-bv}}{e^{bv}-e^{-bv}}\right) \\ &= \frac{b}{a}[a+\phi(v)][a-\phi(v)].\end{aligned}$$

When $a = b = 1$,

$$\phi'(v) = [1+\phi(v)][1-\phi(v)].$$

Improve the algorithm with momentum: The original updating rule,

$$\Delta w_{ji}(k) = \delta_j^z(k) y_{fi}(k),$$

can be modified as

Gradient descent

$$\underbrace{\Delta w_{ji}(k) = \beta \Delta w_{ji}(k-1) + \alpha \Delta w_{ji}(k)}_{\text{You can see the momentum.}} = \beta \Delta w_{ji}(k-1) + \alpha \delta_j^z(k) y_{fi}(k)$$

where $\beta > 0$ is called a momentum constant.

In this case,

$$\Delta w_{ji}(k) = \alpha \sum_{t=0}^{k} \beta^{k-t} \delta_j^z(t) y_{fi}(t).$$

♣ We need to have $0 \leq \beta < 1$ in order for the summation to converge when $k \to \infty$.

♣ When $\beta = 0$ it goes back to the backpropagation algorithm (without momentum).

ASSIGNMENTS

-

♠ ♠ ♠ ♠ ♠ ♠ ♠ ♠

Thursday 02.29 *Lecture no. 3.2*

The design procedure for BP algorithm:

1. Initialization: Choose network configuration, including number of layers, number of neurons on each layer, activation function, and initial weight values.
2. Presentation of training samples: Present each training sample and do the following computations. One epoch of training = one complete cycle of presentation of all training samples.
2.1 Forward computation: Computation goes from inputs to outputs, to compute the output values corresponding to the given training input pattern. Denote the training sample as $(x(k), d(k))$, and the computed output as $o(k)$. The error signal is obtained as

$$e(k) = d(k) - o(k).$$

2.2 Backward computation: Compute the gradients for each layer, to go from the output layer backward to the input layer.

$$\frac{\partial E}{\partial w_{ji}} = -\delta_j^z(k) y_{fi}(k),$$

$$\frac{\partial E}{\partial t_{il}} = -\delta_i^y(k) x_l(k),$$

$$\vdots \quad \text{if } x_l \text{ is not the input layer.}$$

2.3 Update connection weights.

3. Iteration: Let $k = k + 1$, and go back to Step 2 with the next training sample. Start a new epoch/cycle of training if needed, until satisfactory results are obtained.

♣ What satisfactory results?
Either $E(k) < \epsilon$ for all k in an epoch or the average $\bar{E} < \epsilon$.

♣ What order of presentation of samples? Better be random!

♣ Momentum and others. Suggestions in the literature to improve the backproagation algorithm:

– Momentum

– Adjustable learning rate

– Others?

♣ Update after each epoch or each sample? Sometimes epoch/batch training is better.
Epoch/Batch learning: Update is not performed after each sample pair is presented. But the update is only performed after a complete cycle of presentation of all samples.

3.3 Batch Learning and Online Learning

Batch Learning: Adjustments to the synaptic weights of the multi-layer perceptron are performed after all training sample pairs are presented. A complete cycle of presentation of all training samples is called one epoch of training. The cost function in this case is defined as the average of error

$$\bar{E} = \sum_{k=1}^{P} E(k).$$

♣ A more accurate estimation of the gradient vector.

♣ Calculation for each training pair can be performed in parallel.

♣ It requires more storage space.

Online Learning: Adjustments to the synaptic weights of the multi-layer perceptron are performed on the sample-by-sample basis. The cost function is the instantaneous error $E(k)$.

♣ Training patterns are presented in a random order. → The online learning is also referred to as stochastic method. → Less likely to get trapped in a local minimum.

♣ When training data are redundant (repeating the same training pairs), online learning can save some efforts since samples are presented one-by-one.

♣ Online learning can track small changes in the training data, while batch learning tend to ignore such changes,.

3.4 Cross Validation and Generalization

Cross Validation:

– Divide the data set into two parts: A training set and a test set.
– The training subset is then divided into two disjoint sets: An estimation set, used to perform the training to obtain the model, and a validation set to validate the model.
– The BP training is implemented using the estimation set but the training result is determined using the validation set.
– Network complexity, or the sizes of neural network, may lead to overfitting.
Underfitting = Too few parameters, not possible to achieve the learning goal;
Overfitting = Too many parameter, difficult to learning the task.

Generalization: The test subset is used for measuring the generalization performance.

A network is said to generalize well if the input-output mapping computed by the network is correct (or nearly so) for test data never used in the training.

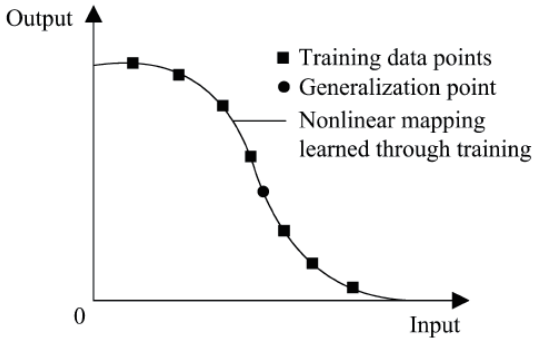


FIGURE 3.5 Properly fitted nonlinear mapping with good generalization.

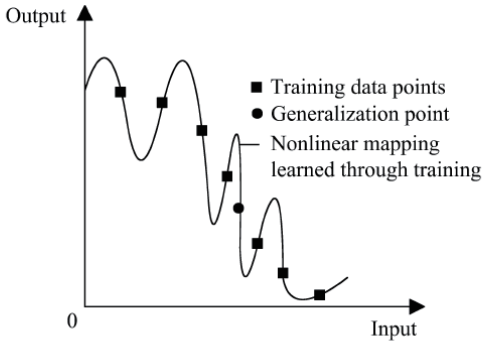


FIGURE 3.6 Overfitted nonlinear mapping with poor generalization.

This figure should show it does not generalize well.

Generalization is influenced by:

1. The size of the training sample, and how representative the sample is.
2. The architecture of the neural network.
3. The complexity of the problem.

ASSIGNMENTS

Homework #3: 3.1, 3.2, 3.3. Due 03-07.

HW4

3.5 Computer Experiment

The XOR problem: Class 0: (0,0) and (1,1); Class 1: (0,1) and (1,0).

One of the solutions is shown:

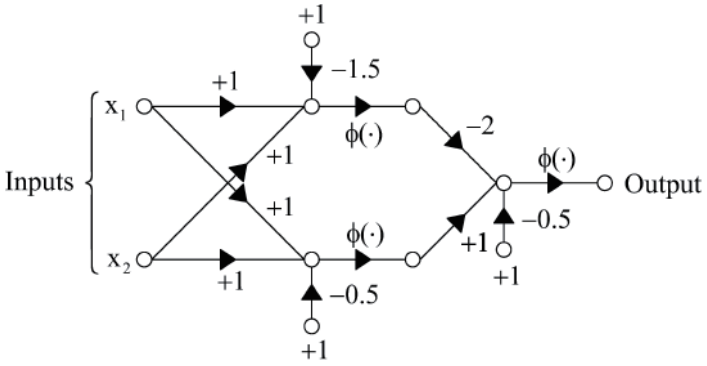


FIGURE 3.8 Signal flow graph of network for solving the XOR problem.

$\phi(\cdot)$ = The threshold function.

The computer experiment is modified as follows:

1. Choose the structure and initial weights:
From input layer to hidden layer: $w_{11}, w_{12}, w_{21}, w_{22}, b_1 = w_{10}, b_2 = w_{20}$.
From hidden layer to output layer: v_1, v_2.
Randomly choose initial values.

2. Training data:
Class 0: (1,0,0) and (1,1,1) $\rightarrow d = 0$;
Class 1: (1,0,1) and (1,1,0) $\rightarrow d = 1$.

$\phi(y) =$ hyperbolic tangent

3. Relations:
Hidden layer in:

$$y_i = w_{i0}x_0 + w_{i1}x_1 + w_{i2}x_2, \quad i = 1, 2,$$

where $x_0 = 1$.
Hidden layer out:

$$y_{fi} = \phi(y_i), \ i = 1, 2.$$

Output neuron - linear with no bias:

$$z = v_1 y_{f1} + v_2 y_{f2}.$$

---

4. Error function:

$$E = \frac{1}{2}e^2 = \frac{1}{2}(d-z)^2 = \frac{1}{2}[d-(v_1 y_{f1} + v_2 y_{f2})]^2$$
$$= \frac{1}{2}\left[d - \sum_{i=1}^{2} v_i \phi\left(\sum_{l=0}^{2} w_{il} x_l\right)\right]^2.$$

5. Updating rules:

$$w_{il}(k+1) = w_{il}(k) + \Delta w_{il}(k),$$
$$\Delta w_{il}(k) = -\eta \frac{\partial E}{\partial w_{il}} = \eta \delta_i^y(k) x_l(k),$$
$$v_i(k+1) = v_i(k) + \Delta v_i(k) = v_i(k) + \eta \delta^z(k) y_{fi}(k),$$

6. The gradients:

♣ Please derive $\dfrac{\partial E}{\partial w_{il}}$, $\dfrac{\partial E}{\partial v_i}$, $\delta_i^y(k)$, and $\delta^z(k)$.

– Choices of the activation function: $\phi(\cdot)$.
It can be either the logistic function or the hyperbolic tangent function.

– Why do we choose linear output layer?
Otherwise we may not be able to reach the desired output values 0 and 1 using the sigmoid function.

ASSIGNMENTS

-

♠ ♠ ♠ ♠ ♠ ♠ ♠ ♠

Tuesday 03.05 *Lecture no. 4.2*

Chapter 4: Radial-Basis Function Networks

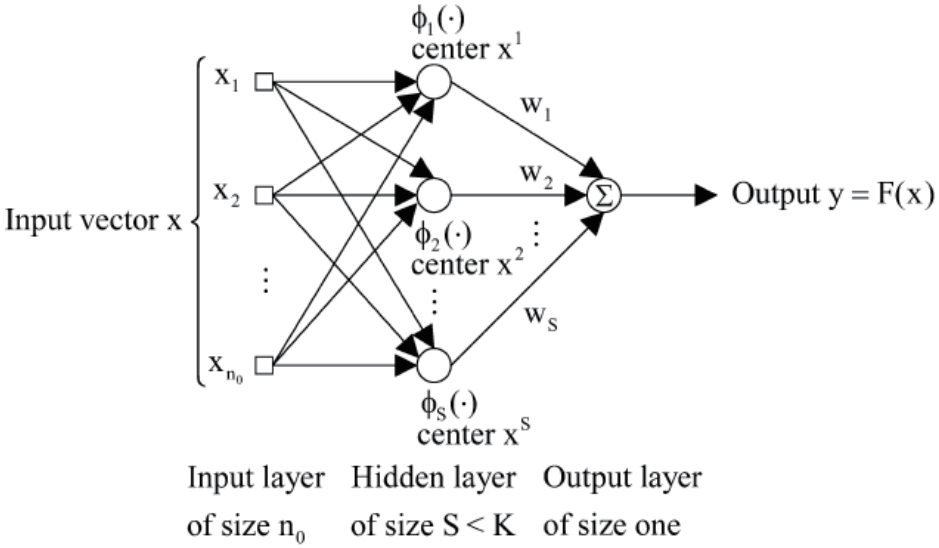
4.1 Radial-Basis Functions

Radial-basis function: A real-valued function whose value depends only on the distance from the origin/center, $\phi(x, c) = \phi(\|x - c\|)$. Examples, for $c > 0$, $\sigma > 0$,

1. Multiquadrics: $\phi(r) = \sqrt{r^2 + c^2}$.
2. Inverse multiquadrics: $\phi(r) = \dfrac{1}{\sqrt{r^2 + c^2}}$.
3. Gaussian functions: $\phi(r) = e^{-r^2/2\sigma^2}$.

4.2 The Interpolation Problem

Radial-basis function (RBF) network: A feedforward neural network with an input layer with n_0 input nodes, a single hidden layer with S neurons, and an output layer.



– The dimensionality of the input vector x is n_0.

– How many hidden layer units? S = the number of given center points. Each center point corresponds to a hidden layer unit, and each unit describes a radial-basis function:

$$\phi(x, x^j) = \phi_j(x) = \phi(\|x - x^j\|), \ j = 1, 2, \ldots, S,$$

or

$$\phi(x, c^j) = \phi_j(x) = \phi(\|x - c^j\|), \ j = 1, 2, \ldots, S.$$

– Output layer can have as many units as you wish. The ith component of the output y is computed as

$$y_i = F_i(x) = \sum_{j=1}^{S} W_{ij}\phi(x, c^j) = \sum_{j=1}^{S} W_{ij} \exp\left(-\frac{1}{2\sigma^2}\|x - c^j\|^2\right)$$

with the popular choice of Gaussian function as the radial-basis function.

– Each of the output component describes a function that interpoplates all the center points.

– The training will be done by applying each training pair with a desired corresponding output value.

L_p-norm:

$$L_p(x) = \|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}},$$

$$\|x - y\|_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}},$$

The commonly used ones are for $p = 1, 2, \infty$,

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|,$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2},$$

$$\|x\|_\infty = \max_{i=1} |x_i|.$$

Homework #4:

3.5 Computer Experiment as described in class. Due 03-18.

ASSIGNMENTS