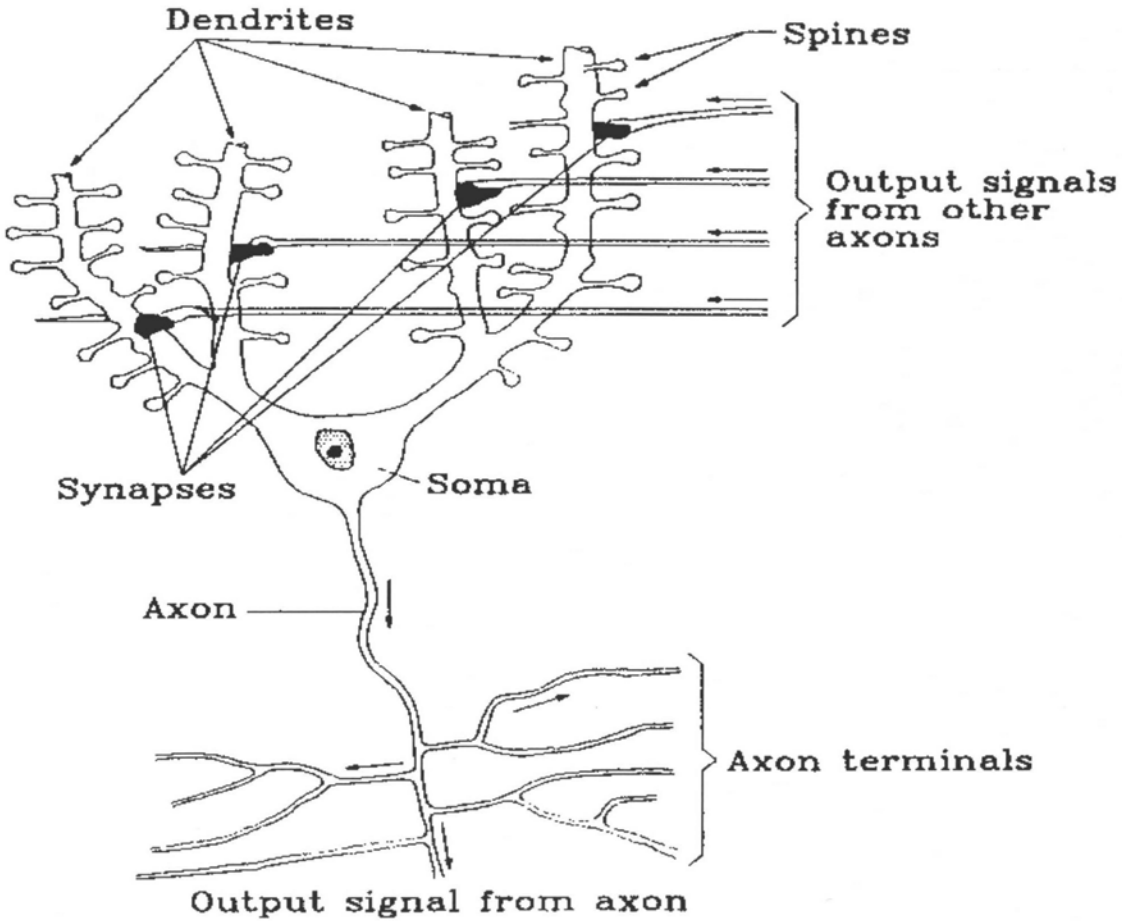
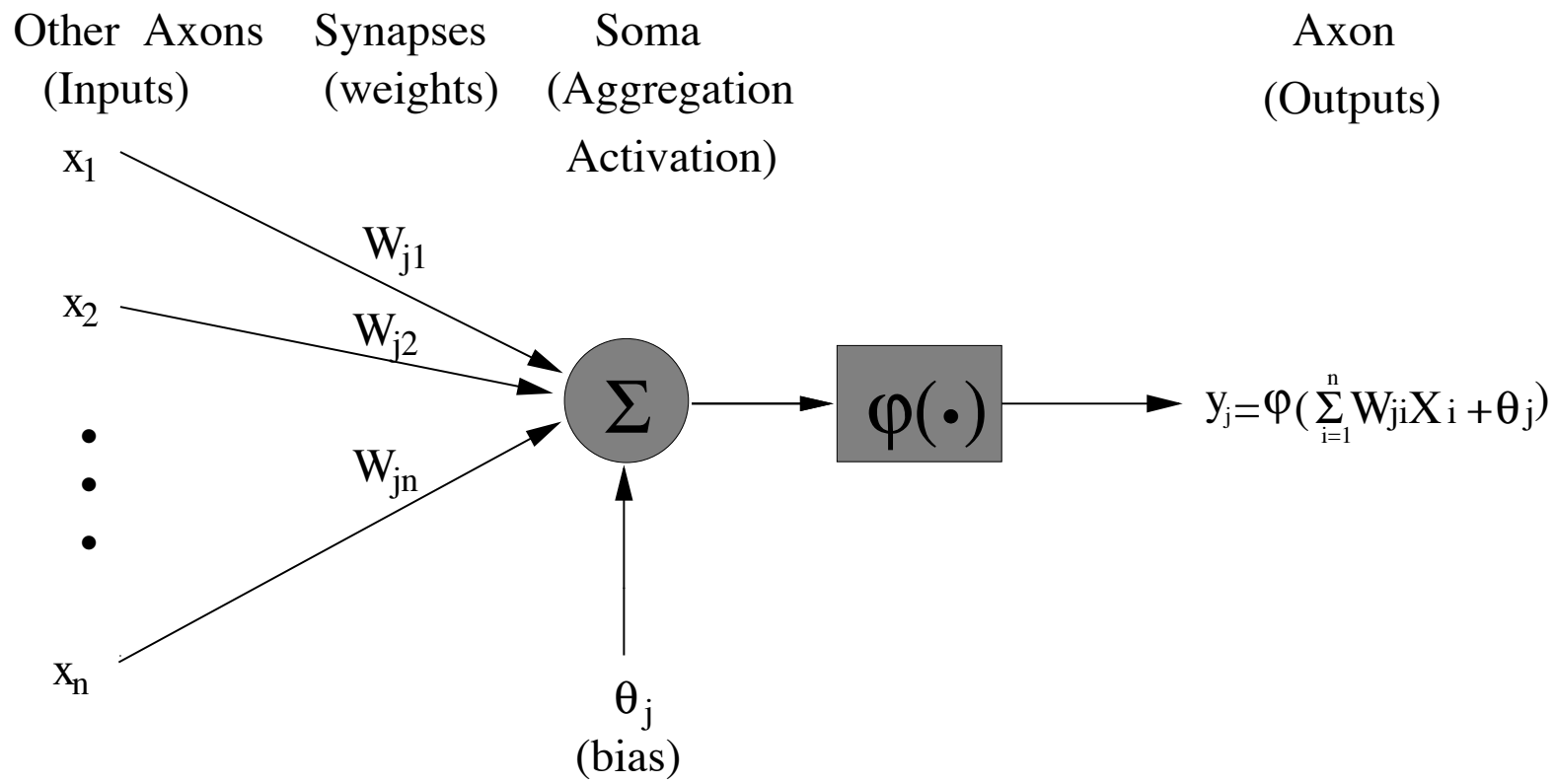
Chapter 1
Neural Network Technology

- What is a Neural Network?
- How does the technology work?
- Where can the technology be applied?
- How to apply the technology?

Schematic Structure of a Biological Neuron

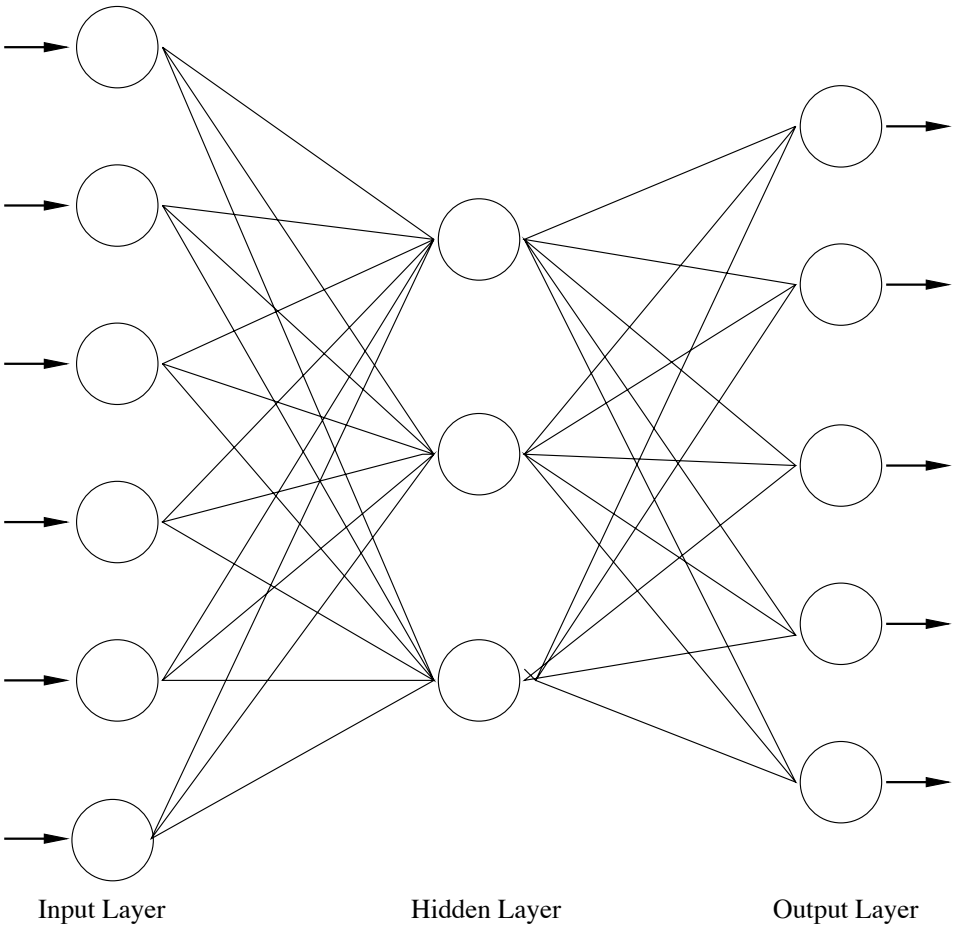


Mathematics of a Conceptual Neuron



What is a Neural Network

- Neural networks mimic human brains to learn the relationship between certain inputs and outputs from experience.



Neural network

Neural Network Architecture

- A neural network consists of several layers of computational units called neurons and a set of data-connections which join neurons in one layer to those in another.
- The network takes inputs and produces outputs through the work of trained neurons.
- Neurons usually calculate their outputs as a sigmoid, or signal activation function of their inputs,

$$f(x) = \frac{1}{1 + e^{-x}}$$

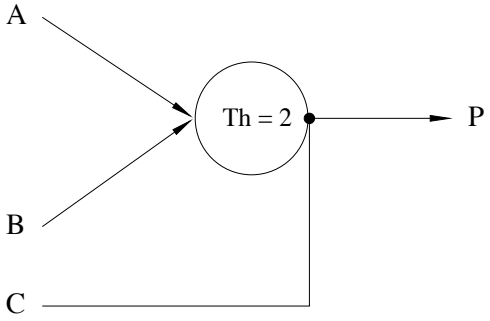
- Using some known results, i.e., input-output pairs for the system being modeled, a weight is assigned to each connection to determine how an activation that travels along it influences the receiving neuron.
- The process of repeatedly exposing the network to known results for proper weight assignment is called "training".

Basic Concepts for Neural Networks

- A brain is composed of networks of neurons.
- A typical neuron receives input - either excitation or inhibition - from many other neurons.
- When its net excitation reaches a certain level, the neuron fires.
- The firing is propagated through a branching axon to many other neurons, where it in turn acts as input to those neurons.
- A neuron always computes the same function.
- We learn because the strength of connections between neurons changes.
- Because the strength of the connections between the neurons in the network can change, the relationship of the network's output to its input can be altered by experience.

History of Neural Networks (Connectionism)

- 1940s Warren McCulloch and Walter Pitts

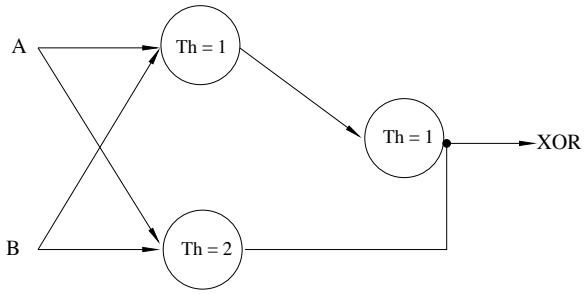


McCulloch-Pitts Neuron

A ,B : excitatory inputs

C: inhibitory input

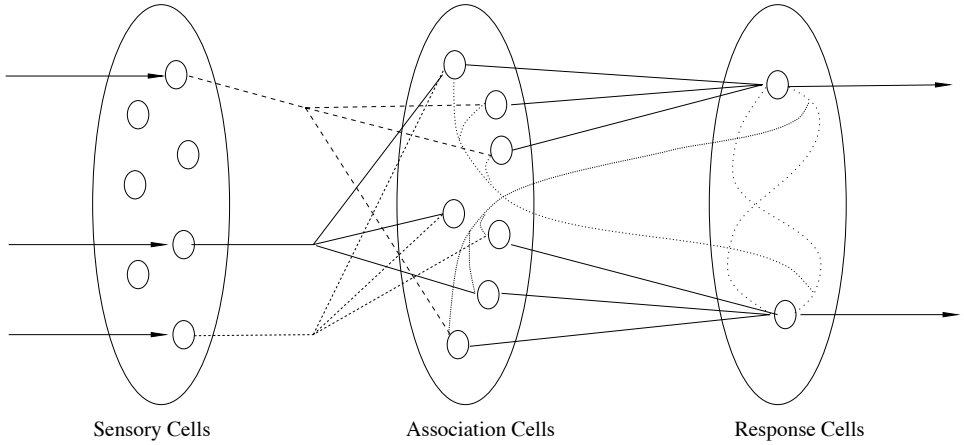
$P = (A \wedge B) \wedge (-C)$



Network of McCulloch-Pitts Neurons

- Good for symbolic logic
- Unbiological
- No-learning

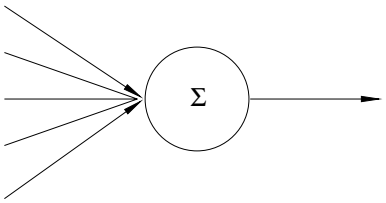
- 1950s Frank Rosenblatt



Perceptron

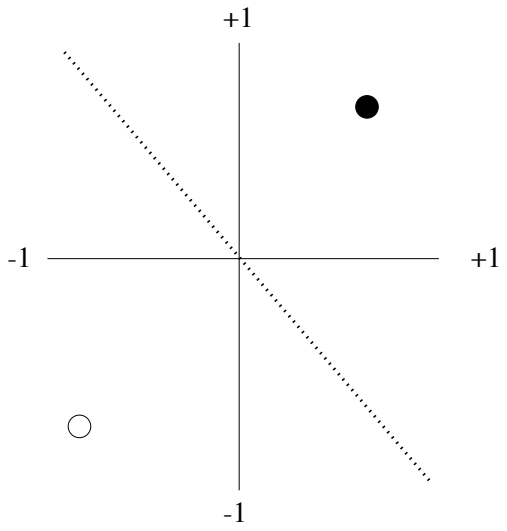
  - Good for categorization
  - Competitive and forced learning

- 1960s Bernard Widrow and Marcian Hoff

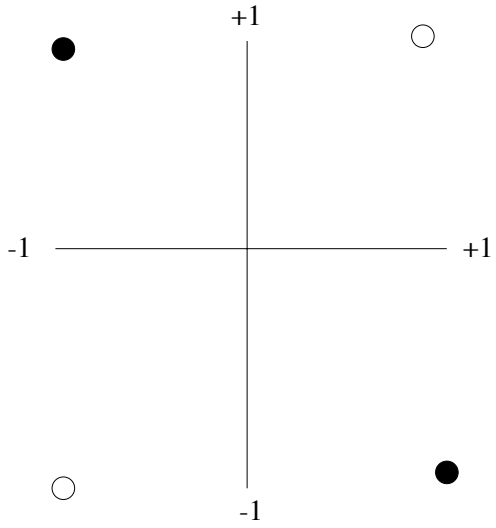


ADAptive LInear NEuron (ADALINE)

  - Adjustable weights
  - Supervised learning
  - Gradient decent
  - Limited to learn linearly separable classes



Linearly separable classes



Classes are not linearly separable

- 1970s Stephen Grossberg
  - Adaptive Resonance Theory (ART)
  - Neural Network models for cognition

- 1980s

  John Hopfield (Feed-Backward)

  – Crossbar network architecture for solving travelling salesman problem

  Feed-Forward with backward propagation of errors

  – David Rumelhart, Geoffrey Hinton, Ronald Williams(1986)
  – David Parker (1982,1985) / Yann Le Cun (1986)
  – First Discovery of back propagation goes to Paul Werbos (1974 Harvard PhD thesis "Beyond Regression")

  Parallel Distributive Processing
  (Rumelhart-McClelland)

  NETalk (Charles Rosenberg and Tery Sejnowski)

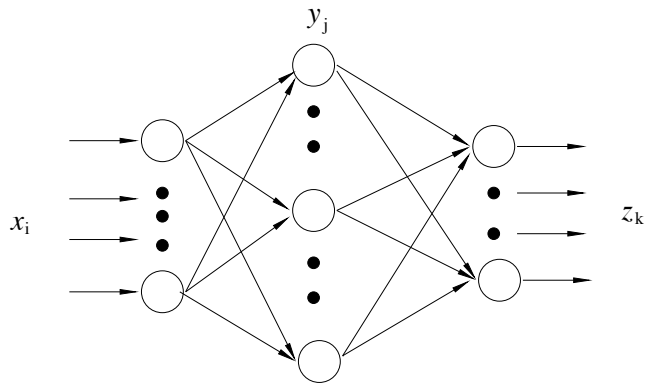
  – demonstration of translating written English into phonetic representation

  First international Conference (1987 IEEE)

Neural Networks and Statistics

- Historically statistics has focused more on linear problems, while Neural Networks has been forced to deal with nonlinearities.
- Neural Networks are very adaptive but less formula-oriented.
- Backpropagation is a non-parametric modeling method: one in which the shape of the relationship between inputs and outputs is decided by the data rather than predetermined by the tool.
- Neural Networks, in theory, approximate any functional form of input-output realation to any degree of accuracy, using backpropagation.
- Neural Networks can estimate both quantitative variables (regression) and class variables (discriminant analysis).

Backpropagation and Learning



$$u_j = a_{0j} + \sum_{i=1}^{I} a_{ij} x_i, \quad v_k = b_{0k} + \sum_{j=1}^{J} b_{jk} y_j,$$

$$y_j = g(u_j), j = 1, \ldots, J, \quad z_k = g(v_k), k = 1, \ldots, K$$

First, instead of changing the weights after all the examples have been processed, it is possible to change the weights as each example is processed.

Second, it is not essential for the output nodes to produce the sigmoid function of their weighted sum of inputs.

Third, it is possible for the network to include direct links from the input nodes to the output nodes.

Fourth, it is possible for the network to have more than one hidden layer.

Finally, there is an issue of the number of hidden nodes.

Fundamentals of Neural Networks with Backpropagation

I. Mapping

- how the neural network mapping function can approximate any target function, so long as the correct coefficients - or link weights - are known.

II. Learning

- how to obtain the coefficients for the best possible fit to the training data.

III. Generalization

- how to obtain the best possible fit to new data.

IV. Pragmatics

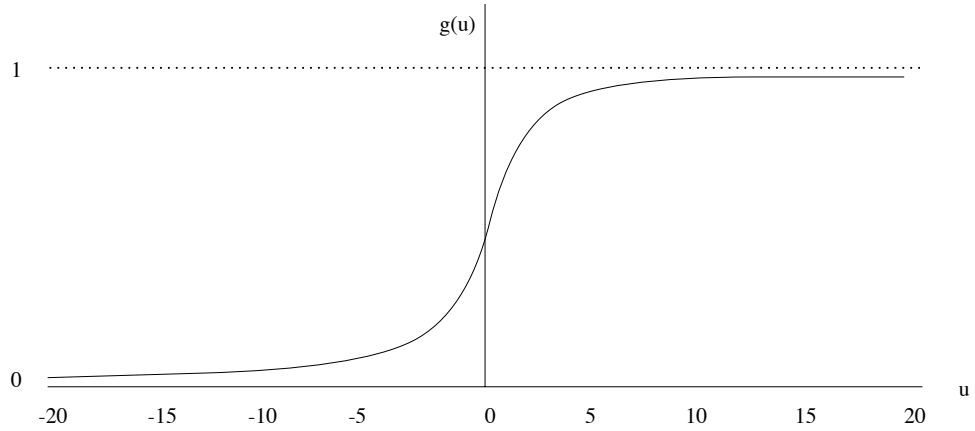
- practical issues such as selecting an application, sampling, selecting and representing variables, and evaluating a model's performance.

I. Mapping Functions

- Calculate output based on input information
- Forward pass through the network
  - in training
  - in using

Sigmoid Functions

(1) $g(u) \equiv \frac{1}{1+\frac{1}{e^u}}$



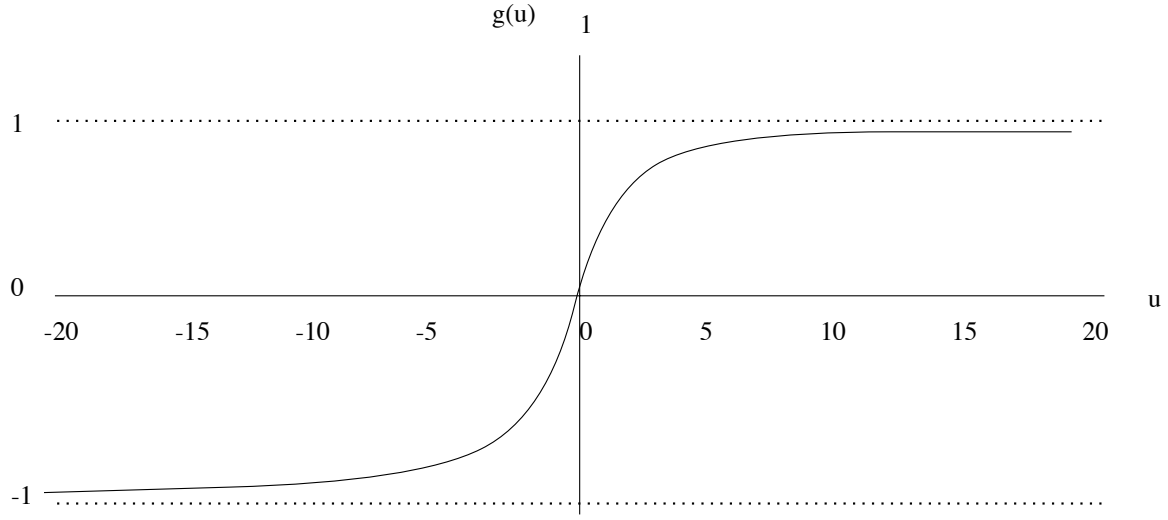
Logistic function

Values of Logistic Function $g(u)$

u	e^u	$1/e^u$	$g(u)$
10	22026.00005	0.00005	0.99995
5	148.41316	0.00674	0.99331
4	54.59815	0.01832	0.98201
3	20.08554	0.04979	0.95257
2	7.38906	0.13534	0.88080
1	2.71828	0.36788	0.73106
0	1.00000	1.00000	0.50000
-1	0.36788	2.71828	0.26894
-2	0.13534	7.38906	0.11920
-3	0.04979	20.08554	0.04743
-4	0.01832	54.59815	0.01799
-5	0.00674	148.41316	0.00669
-10	0.00005	22026.00005	0.00005

(2)

$$h(u) \equiv \frac{1 - e^{-u}}{1 + e^{-u}} = 2g(u) - 1$$

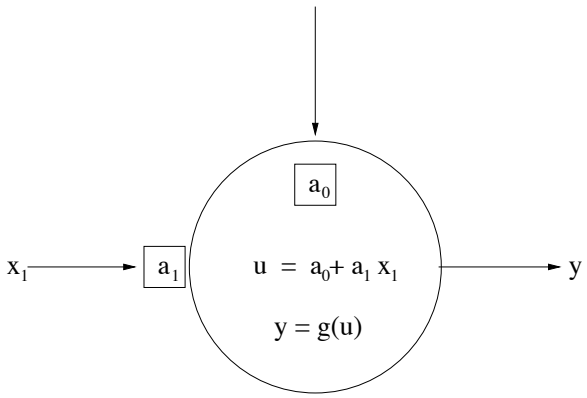


(3)

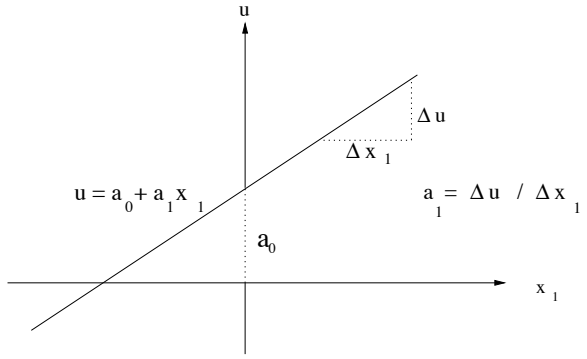
$$\tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}} \quad \text{(deeper slope)}$$

Neural Network with One Input

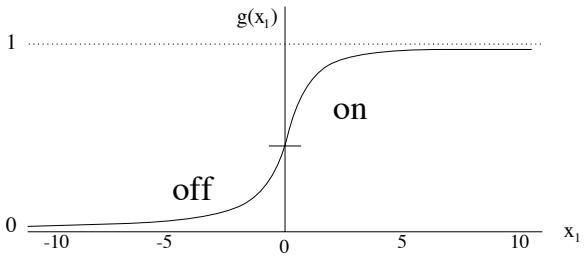
- Hidden node



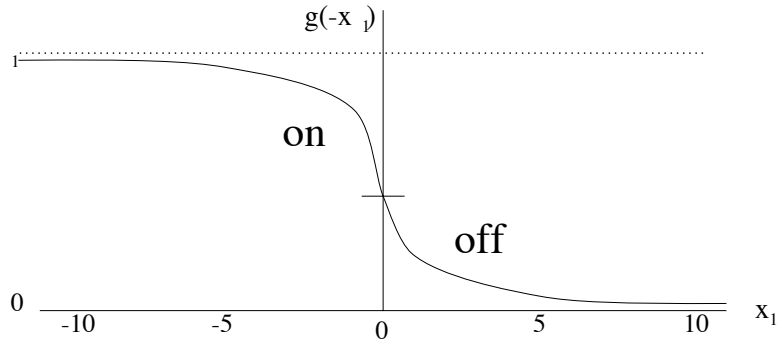
Hidden node



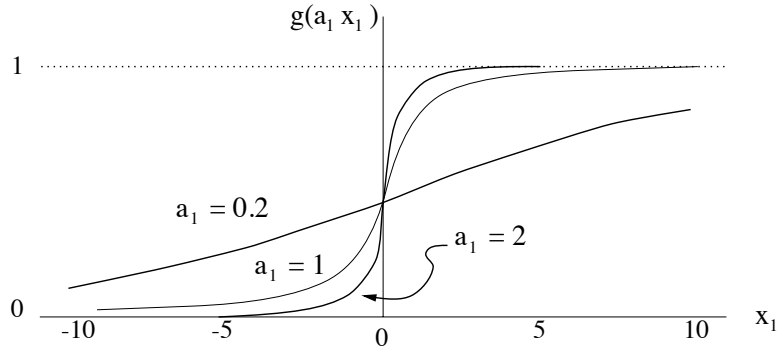
u as a function of x_1



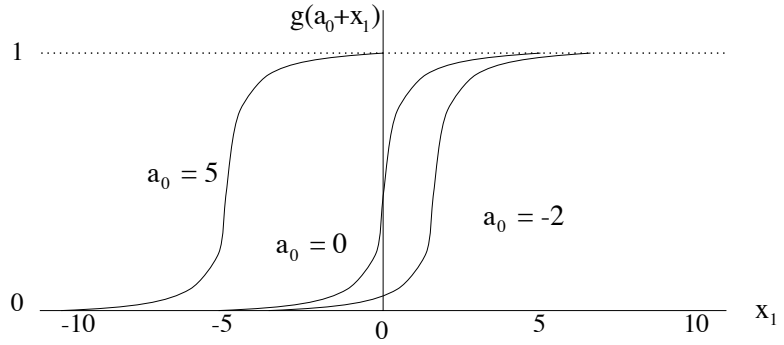
Logistic function of x_1



Logistic function of $-x_1$

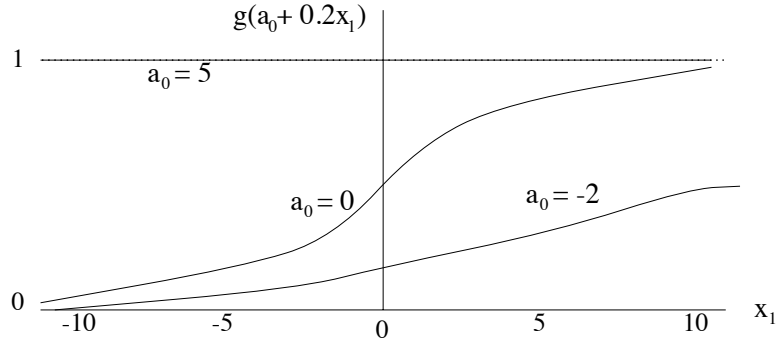


Logistic function of $a_1 x_1$



Logistic function of $a_0 + x_1$

- Boundary point:



Logistic function of $\mathrm{a}_0 + 0.2\mathrm{x}_1$

When $\hat{x}_1 = \frac{-a_0}{a_1}$, $\hat{u} = a_0 + a_1\hat{x}_1 = 0$,

$g(\hat{u}) = \frac{1}{2}$.

The network is on or off when input value crosses $\hat{x}_1$.

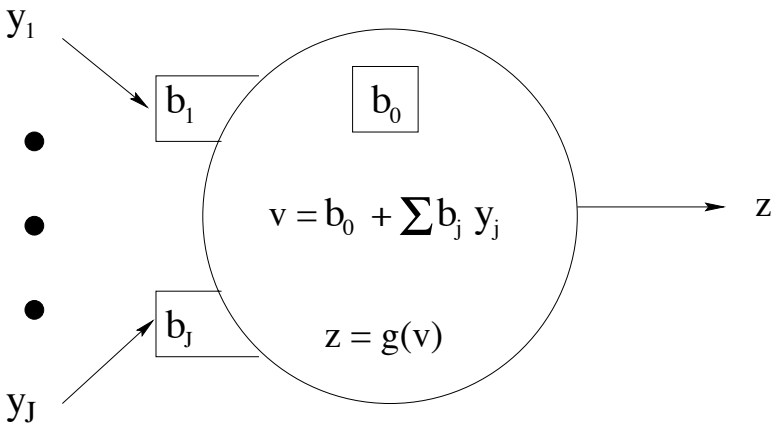
- By varing the weights a_0 and a_1, a hidden node can produce any sigmoid shape we wish.

Step 1: curve moves up $\rightarrow a_1 > 0$
curve moves down $\rightarrow a_1 < 0$

Step 2: ramp-up is steep $\rightarrow a_1$ high
ramp-up is shallow $\rightarrow a_1$ low

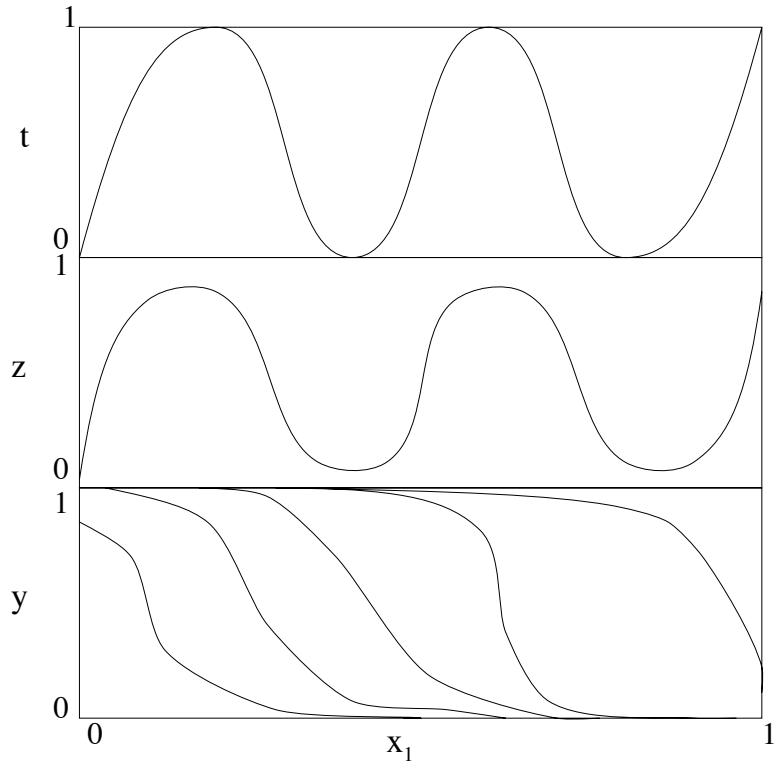
Step 3: let $\hat{x}_1 =$ distance the curve shifts along the x_1-axis and $a_0 \stackrel{\Delta}{=} -a_1\hat{x}_1$

- Output node



Output Node

  - A network consisting of one output node that receives values from several hidden nodes, which in turn receive values from one input node, can approximate functions that are highly nonlinear.



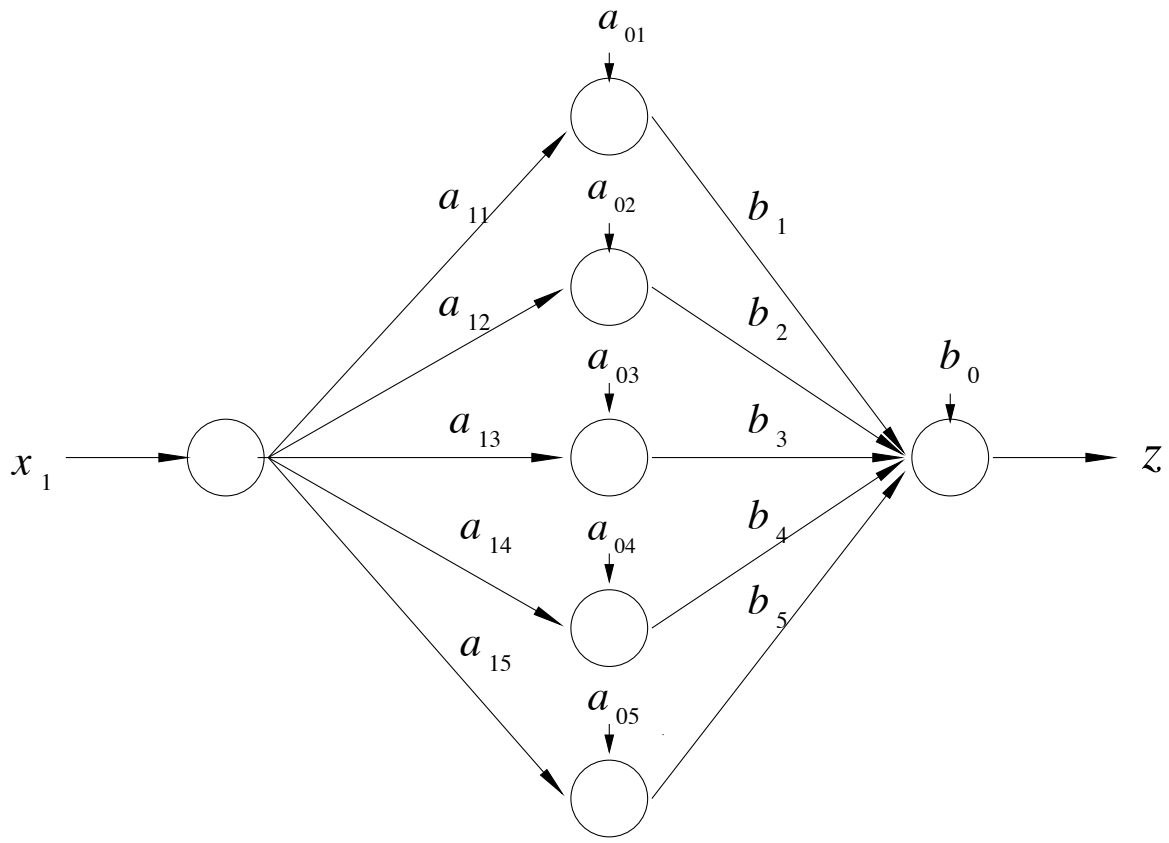
Target function (t), network output (z), and hidden node outputs (y) as function of network input x_1

Sine wave network

Example: Sine Curve

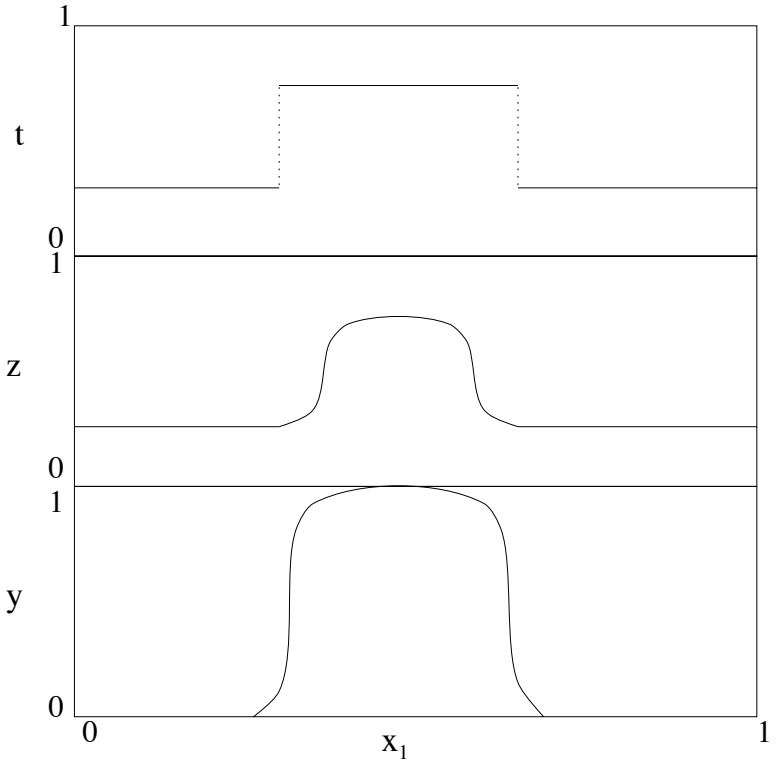
Sine Wave Network $b_0 = 8.1$

j	a_{0j}	a_{1j}	$\hat{x}_1$	b_j
1	1.9	-20.8	0.09	-8.1
2	9.1	-29.1	0.31	7.3
3	13.1	-26.6	0.49	-6.4
4	42.9	-60.0	0.69	5.3
5	19.2	-19.6	0.98	-19.7



Sine Neural Network

Example: Square Wave



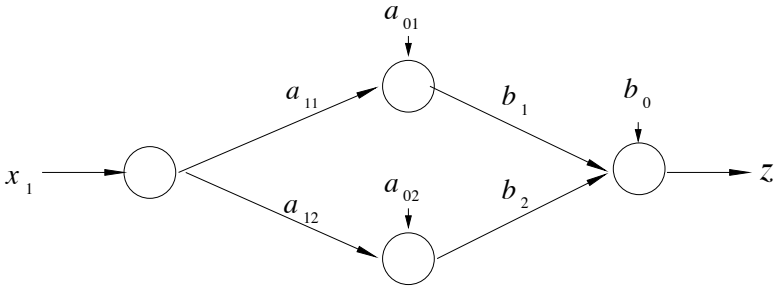
Target function (t), network output (z), and hidden node outputs (y)

as function of network input x_1

Square wave network

Square Wave Network $b_0 = -3.350$

j	a_{0j}	a_{1j}	$\hat{x}_1$	b_j
1	-30	100	0.3	2.225
2	70	-100	0.7	2.225



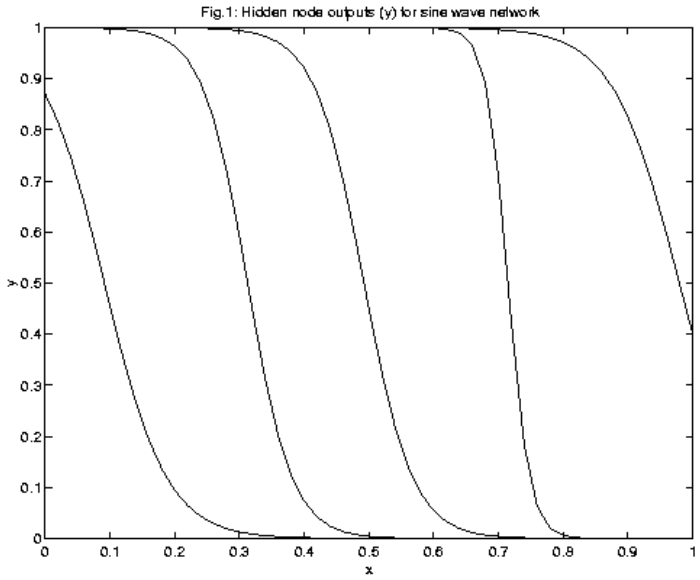


Fig.1: Hidden node outputs (y) for sine wave network
y
x

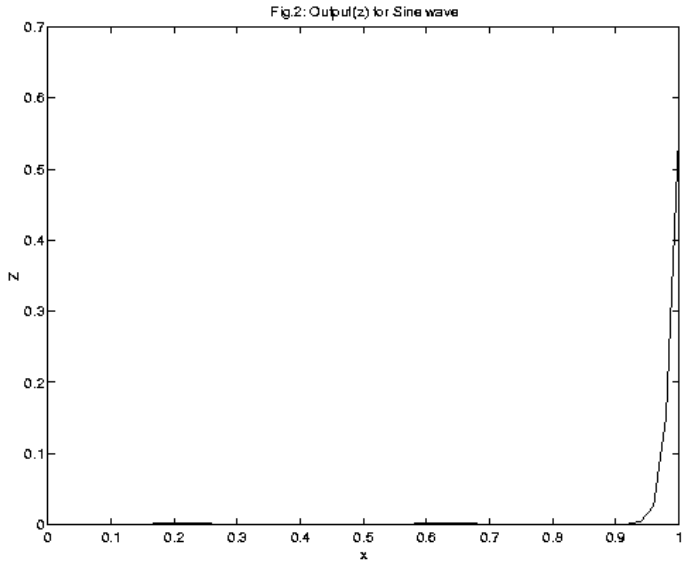


Fig.2: Output(z) for Sine wave
Z
x

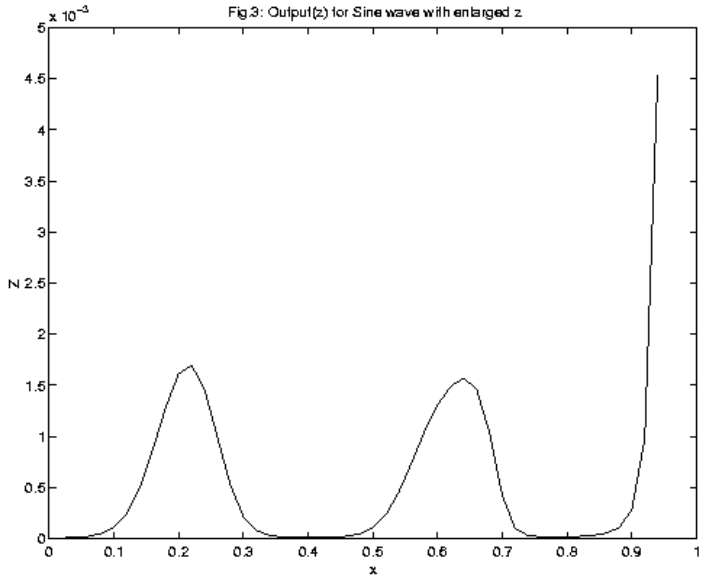


Fig.3: Output(z) for Sine wave with enlarged z
x 10^-3
z
x

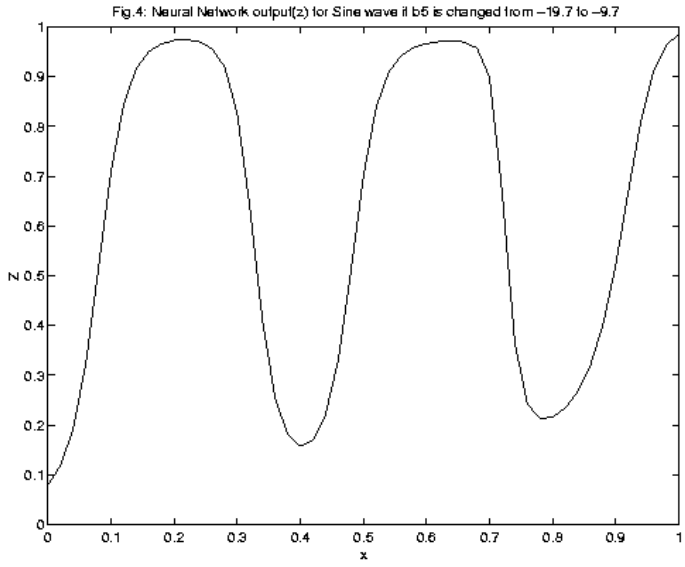


Fig.4: Neural Network output(z) for Sine wave if b5 is changed from −19.7 to −9.7
z
x

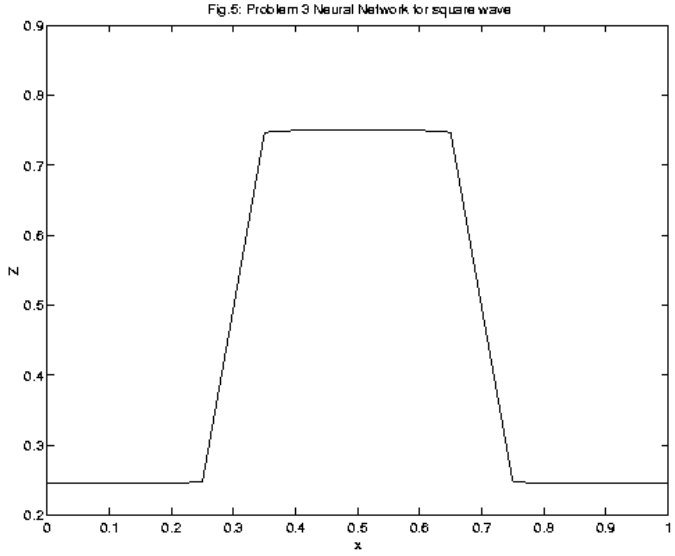
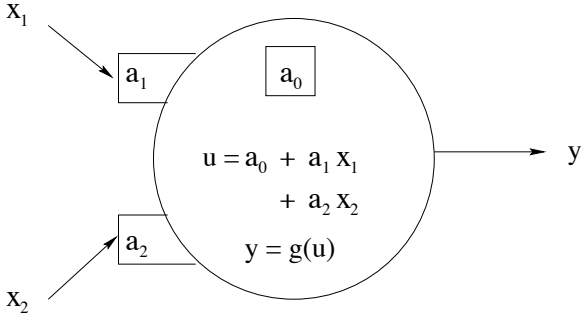


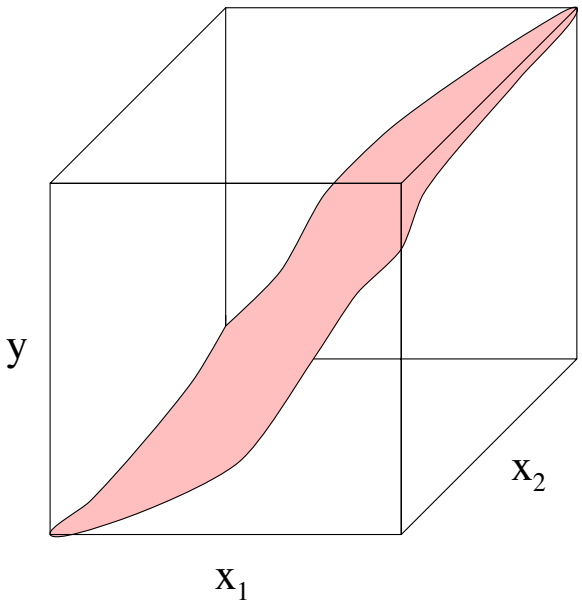
Fig.5: Problem 3 Neural Network for square wave
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
z
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x

Neural Network with Two Inputs

- Hidden node

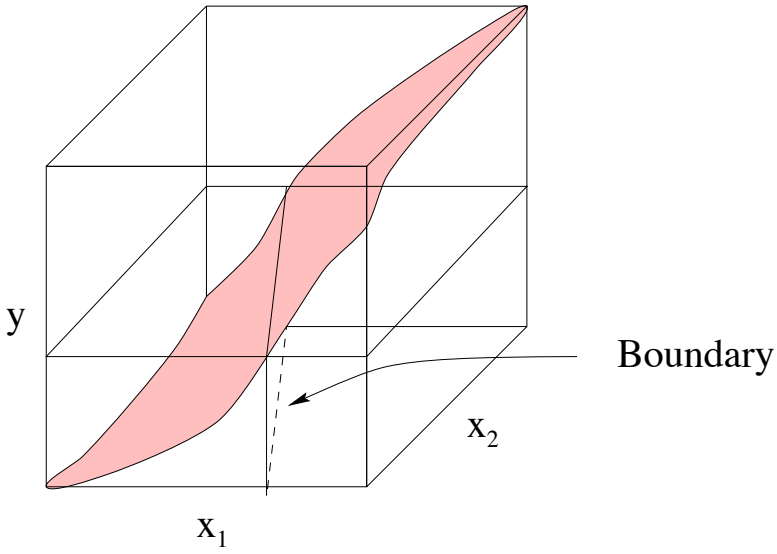


Hidden Node

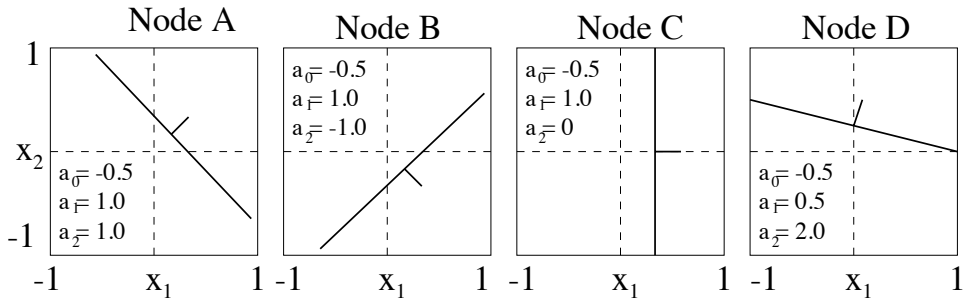


$$y = g(a_0 + a_1 x_1 + a_2 x_2)$$

- Boundary point

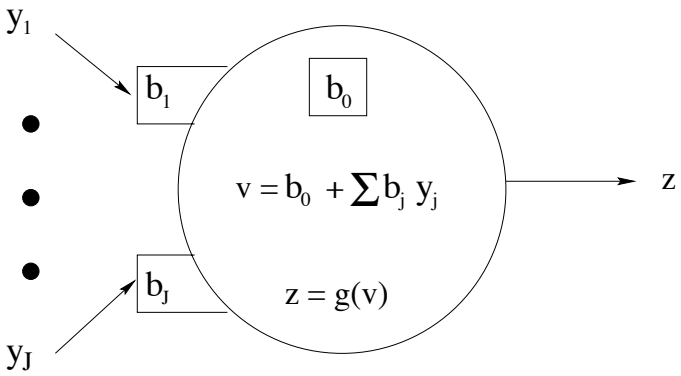


Boundary of sigmoid of two inputs



Boundaries of four hidden nodes with two inputs

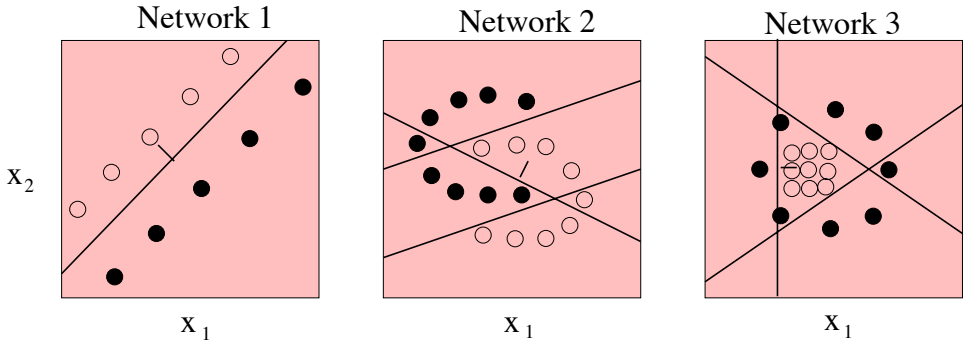
- Output node



Output Node

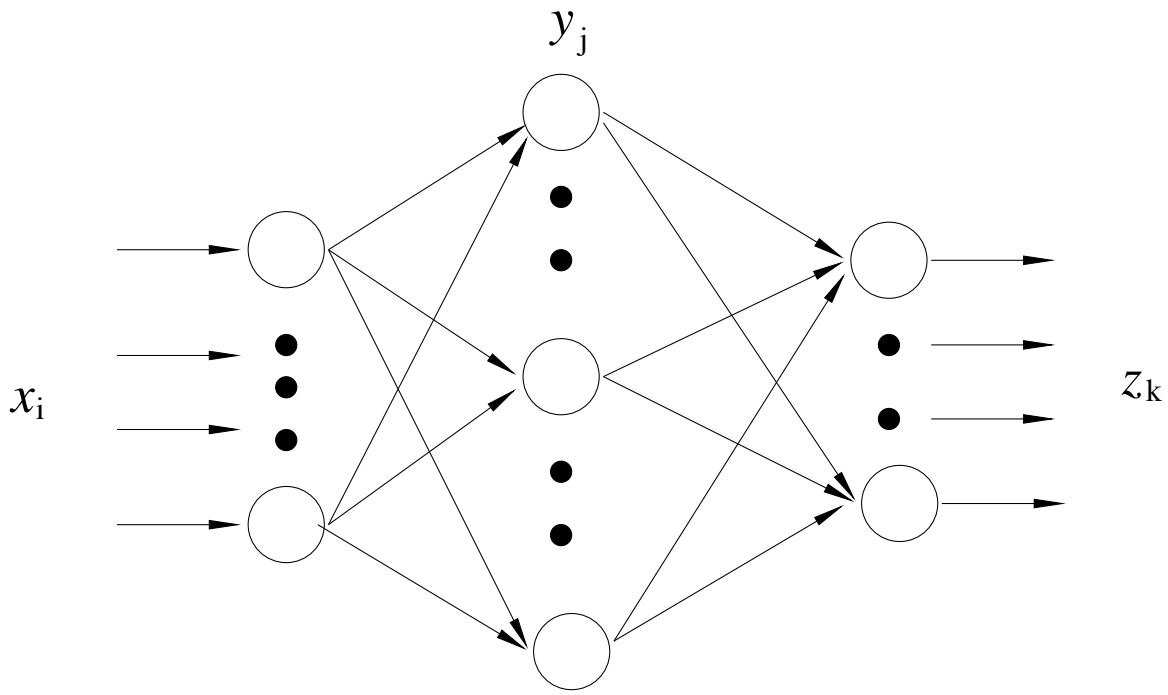
$$v = b_0 + \sum_{j=1}^{J} b_j y_j$$

Examples



Three networks with two inputs

Neural Network with Many Inputs



$$u_j = a_{0j} + \sum_{i=1}^{I} a_{ij} x_i, \quad v_k = b_{0k} + \sum_{j=1}^{J} b_{jk} y_j,$$

$$y_j = g(u_j),\ j = 1, \ldots, J, \quad z_k = g(v_k),\ k = 1, \ldots, K$$

II. Learning

- Finding the coefficients or weights (a_{ij}, b_{jk}) that provides the best fit between the network output (z) and target function value(t).
- Minimizing the mean squared error

$$E = \frac{\frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(z_{kn} - t_{kn})^2}{NK}$$

N: number of examples in the data set

K: number of outputs of the network

t_{kn}: kth target output for the nth example

z_{kn}: kth output for the nth example

- Example by Example learning (on-line)

$$N = 1$$

$$\bar{E} \triangleq KE = \frac{1}{2}\sum_{k=1}^{K}(z_k - t_k)^2$$

- Gradient direction

  (1)

$$\frac{\partial \bar{E}}{\partial b_{jk}} = \frac{\partial \bar{E}}{\partial z_k}\frac{\partial z_k}{\partial v_k}\frac{\partial v_k}{\partial b_{jk}} = \begin{cases} p_k, \ j = 0 \\ p_k y_j, \ j = 1, \ldots, J \end{cases}$$

  where

$$p_k = (z_k - t_k) z_k (1 - z_k)$$

$$y_j = g(a_{0j} + \sum_{i=1}^{I} a_{ij} x_i)$$

  (2)

$$\begin{aligned} \frac{\partial \bar{E}}{\partial a_{ij}} &= \left(\sum_{k=1}^{K} \frac{\partial \bar{E}}{\partial z_k}\frac{\partial z_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}\right)\frac{\partial y_j}{\partial u_j}\frac{\partial u_j}{\partial a_{ij}} \\ &= \begin{cases} q_j, \ i = 0 \\ q_j x_i, \ i = 1, \ldots, I \end{cases} \end{aligned}$$

  where

$$q_j = \left[\sum_{k=1}^{K} p_k b_{jk}\right] y_j (1 - y_j)$$

- Batch learning (off-line)

$$E' \triangleq NKE = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(z_{kn} - t_{kn})^2$$

- Delta Rule:
  - Iteratively updating the weights (a_{ij}, b_{jk})

$$w^{m+1} = w^m - \lambda d^m$$

where

$$d^m = \sum_{n=1}^{N} \left(\frac{\partial E}{\partial w}\bigg|_m \right)_n$$

$$\lambda = \text{step length}$$

- Momentum

$$w^{m+1} = w^m - \lambda[\mu d^m + (1-\mu) d^{m-1}]$$

- Adaptive learning rate / Second order information learning.

- Initial weights
  - Set the hidden node weights to small random numbers distributed evenly around 0.
  - Initialize half of each output node's weights with values of 1 and the other half with -1; if there is an odd number, initialize bias weights at 0.

- Stop learning after a finite number of iterations (epochs), or E becomes small enough, or not much more improvement can be made.

Exercises

$$\bar{E} = \tfrac{1}{2}\sum_{k=1}^{K}(z_k - t_k)^2 \qquad \frac{\partial \bar{E}}{\partial z_k} = (z_k - t_k)$$

$$z_k = g(v_k) \stackrel{\Delta}{=} \frac{1}{1+e^{-v_k}} \qquad \frac{\partial z_k}{\partial v_k} = z_k(1 - z_k)$$

$$v_k = b_{0k} + \sum_{j=1}^{J} b_{jk} y_j \qquad \frac{\partial v_k}{\partial b_{jk}} = \begin{cases} 1, & j = 0 \\ y_j, & j = 1, \ldots, J \end{cases}$$

$$\frac{\partial v_k}{\partial y_j} = b_{jk}$$

$$y_j = g(u_j) \stackrel{\Delta}{=} \frac{1}{1+e^{-u_j}} \qquad \frac{\partial y_j}{\partial u_j} = y_j(1 - y_j)$$

$$u_j = a_{0j} + \sum_{i=1}^{I} a_{ij} x_i \qquad \frac{\partial u_j}{\partial a_{ij}} = \begin{cases} 1, & i = 0 \\ x_i, & i = 1, \ldots, I \end{cases}$$

$$p_k \stackrel{\Delta}{=} \frac{\partial \bar{E}}{\partial z_k}\frac{\partial z_k}{\partial v_k} = (z_k - t_k) z_k (1 - z_k)$$

$$q_j = \left[\sum_{k=1}^{K} \frac{\partial \bar{E}}{\partial z_k}\frac{\partial z_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}\right]\frac{\partial y_j}{\partial u_j} = \left[\sum_{k=1}^{K} p_k b_{jk}\right] y_j(1 - y_j)$$

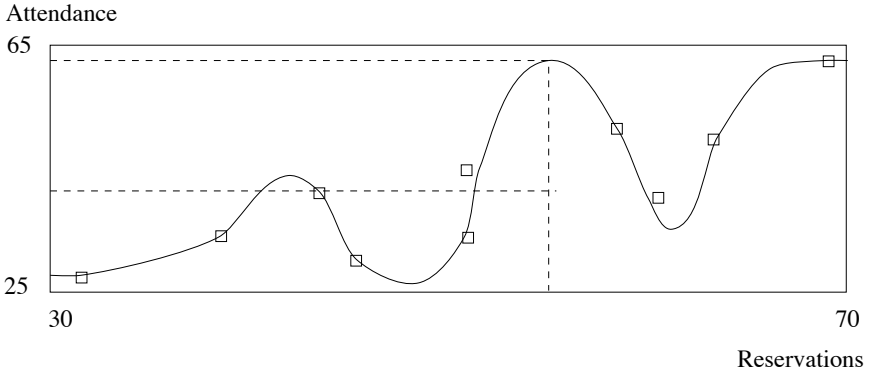
III. Generalization

- Learning provides the best fit for the training examples. Can the good behavior be generalized (or holds valid) for other testing examples?

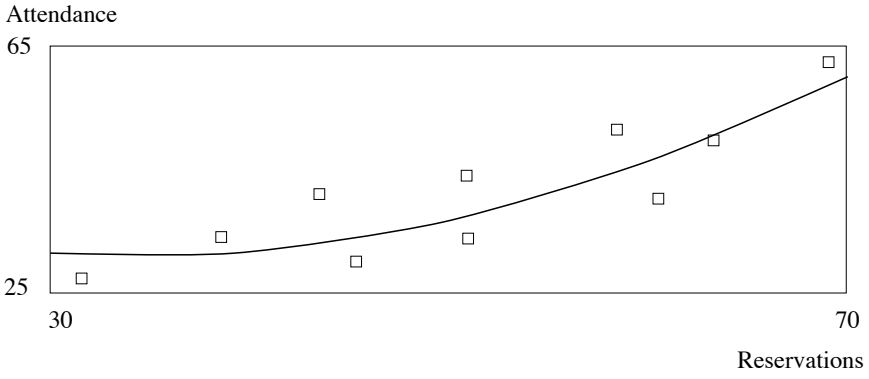
- Noise in the trainng data may cause the overfitting problem, which prevents generalization.

Example:

10 Years Data



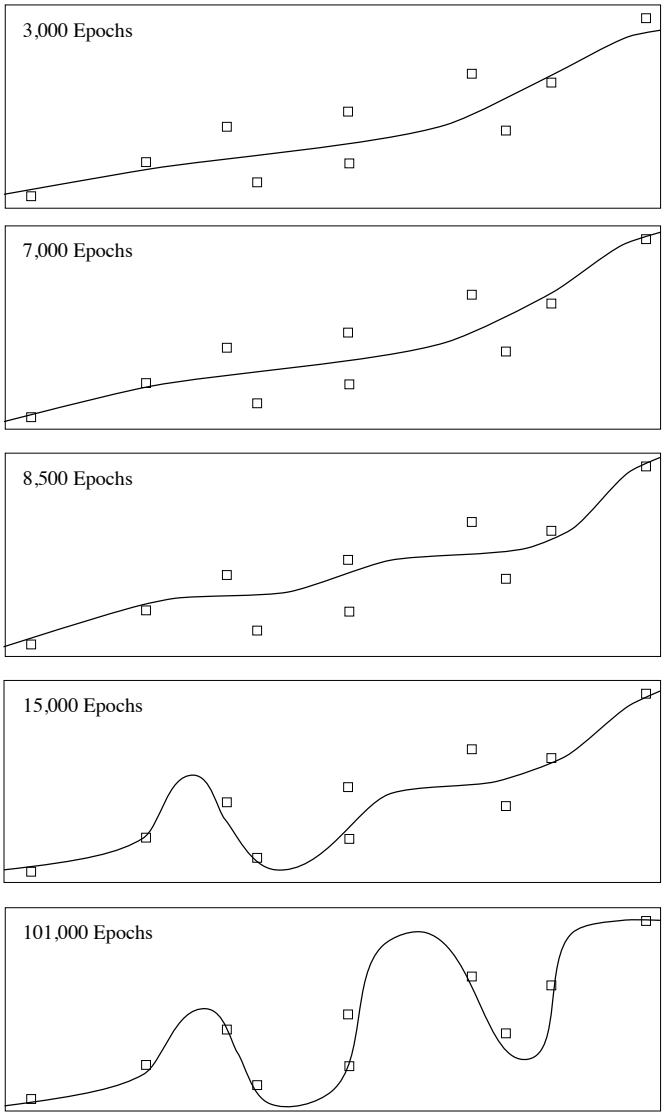
Network Outputs



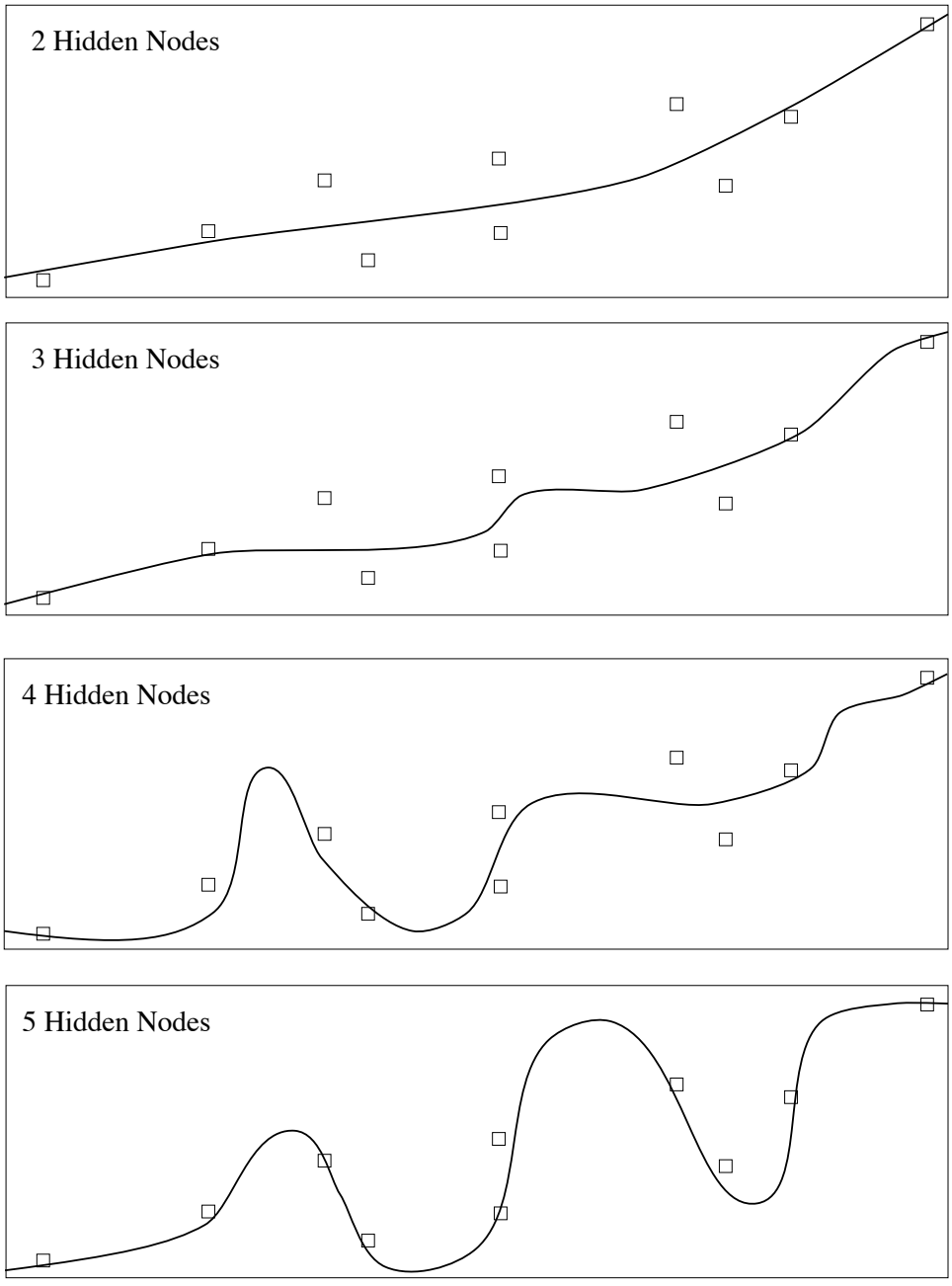
More Real Model

- Prevent from overfitting
  - reduce noise in the data
  - increasing the sample size
  - do not over-train

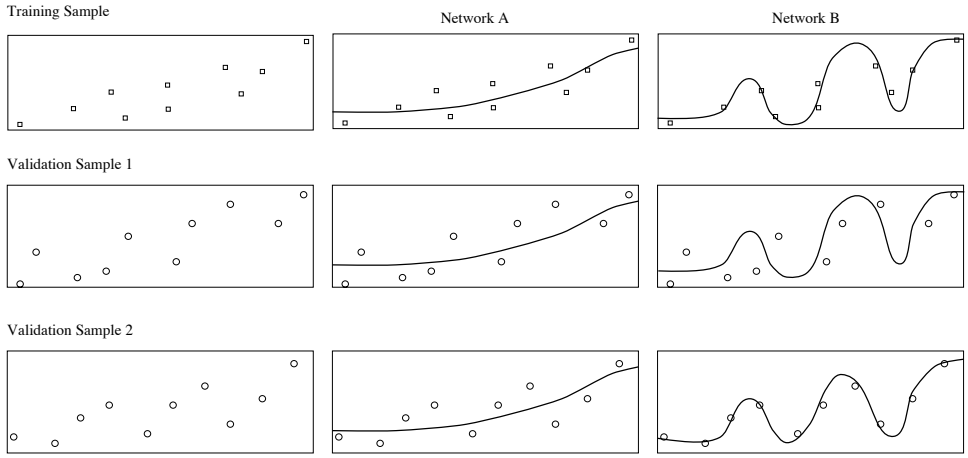
The Course of Learning for a Network with 5 Hidden Nodes



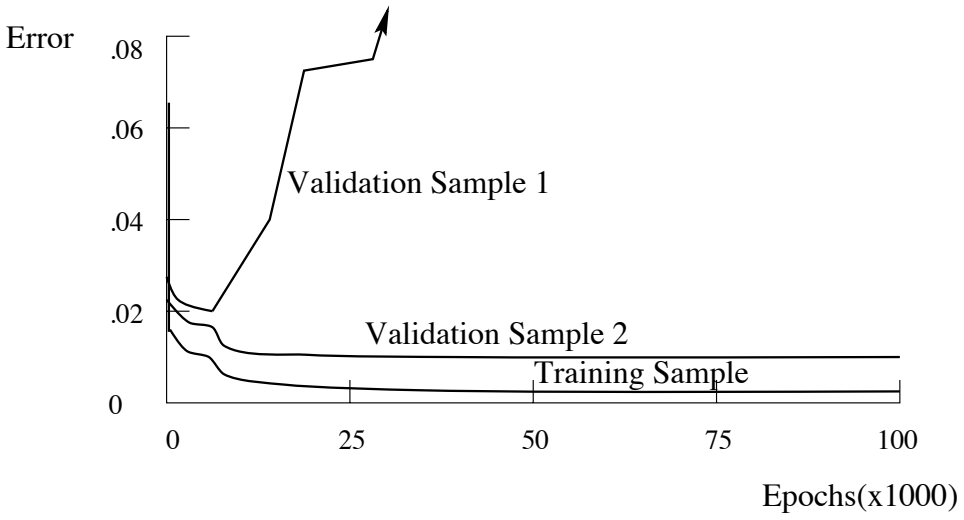
– limit the number of hidden nodes



- Cross validation for the right network



Output of two networks compared to training and validation samples



Error on Two Validation Samples

IV. Pragmatics

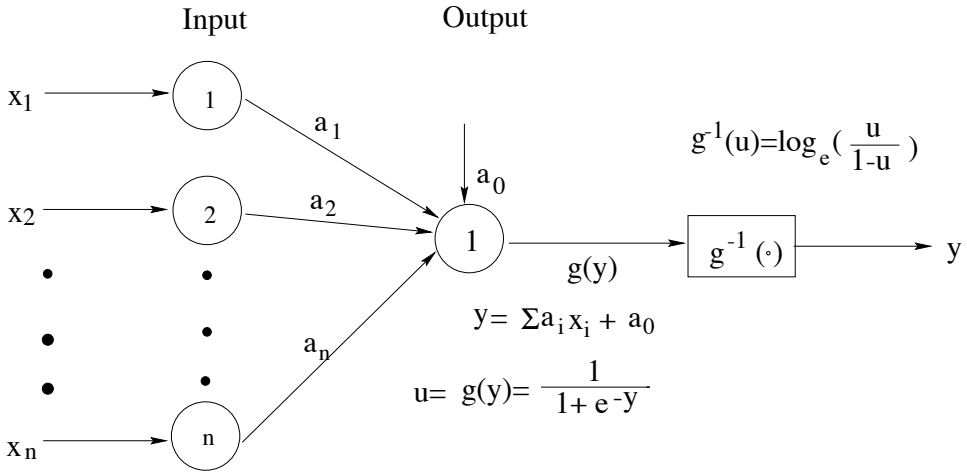
- Neural Network Technology

  1. Choosing an application
  2. Designing the sample
  3. Selecting variables
  4. Representing the variables
  5. Building the model
  6. Using the model

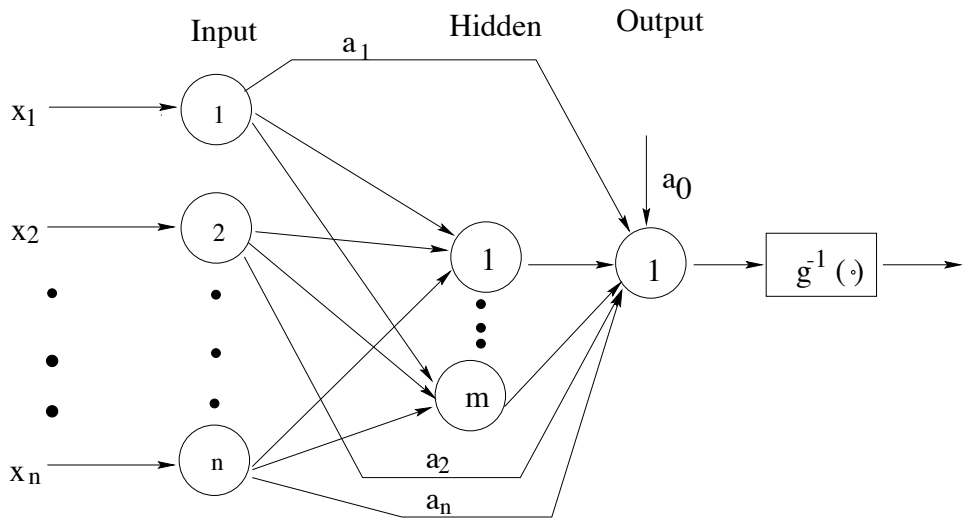
Example

Recall that for $y = f(x) = a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n$, we may use linear regression to get $a_0, a_1, a_2, \ldots, a_n$.

We can also build a neural network to get $a_0, a_1, a_2, \ldots, a_n$, using the same set of data with the same order of computational complexity.

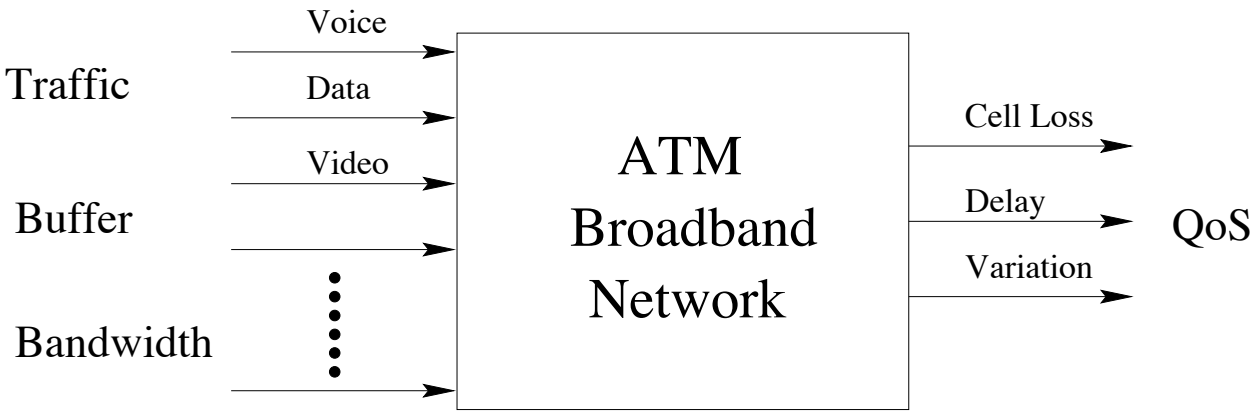


$u = \frac{1}{1+e^{-y}}$
$u + ue^{-y} = 1$
$e^{-y} = \frac{1-u}{u}$
$y = \ln \frac{u}{1-u}$

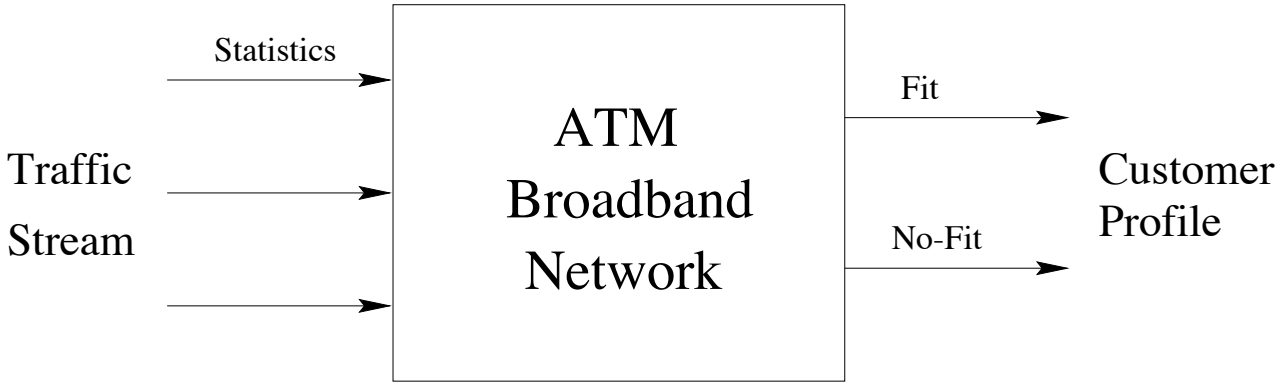


By incorporating hidden layers into this network, we can even achieve nonlinear regression.

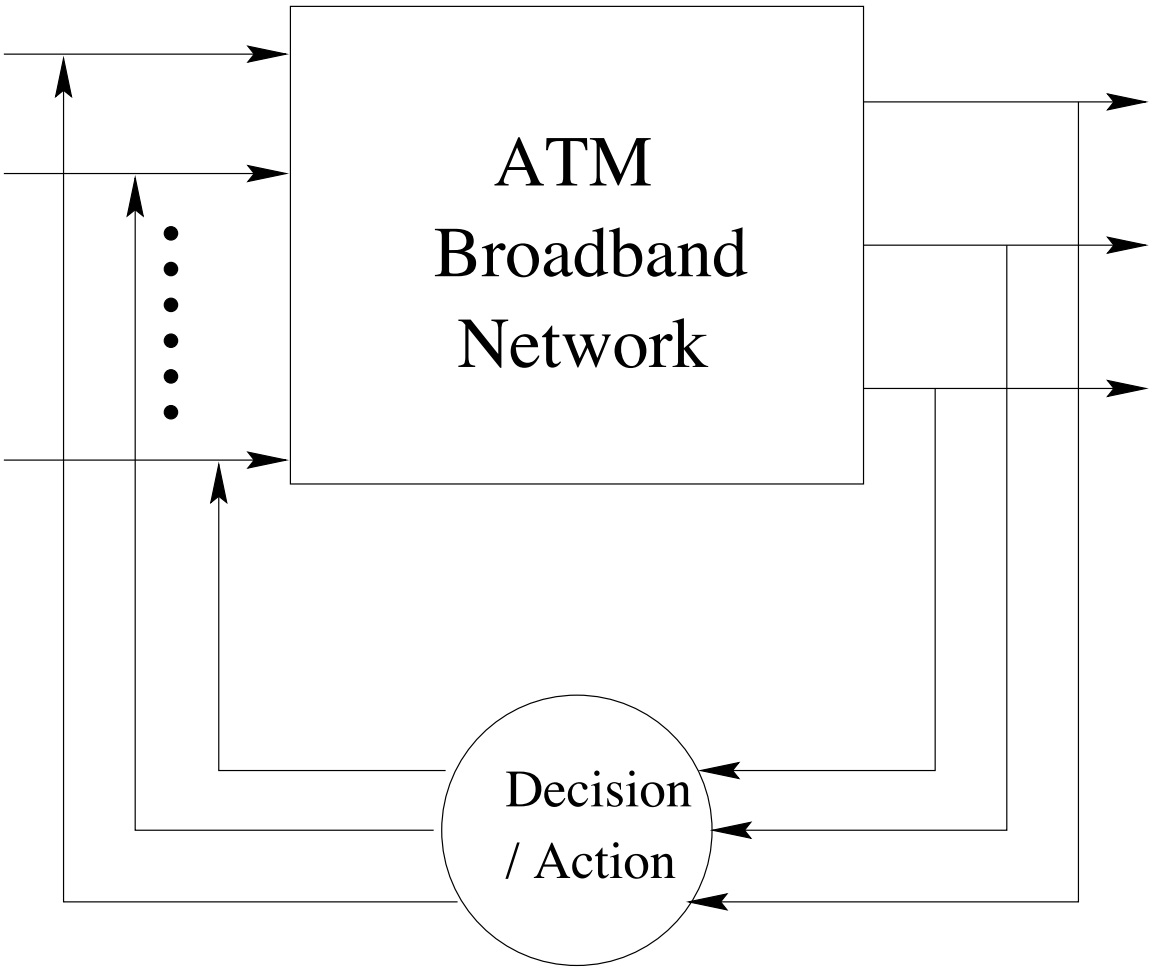
Applications



(Estimation / Prediction)



(Classification)



(Control / Management)

Feed-Backward Neural Network

- Non-supervised learning
- Finding an optimal solution of a mathematical optimization problem by reaching an equilibrium point of corresponding neural dynamics
- Massive parallel processing
- Hardware analogue circuits

Basic Framework

- Optimization Problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad i = 1, \ldots, m \\ & x \in X \end{aligned}$$

- Energy Function

$$E(x(t), s) = f(x(t)) + s \sum_{i=1}^{m} \phi(g_i(x(t)))$$

- Neural Dynamics

$$\begin{aligned} \dot{x}(t) &= \frac{dx(t)}{dt} \\ &= -\eta \nabla_{x(t)} E(x(t), s) \\ &= -\eta \nabla_{x(t)} f(x(t)) - \eta s \sum_{i=1}^{m} \phi(g_i(x(t))) \nabla_{x(t)} g_i(x(t)) \end{aligned}$$

- **Two types of neurons:**

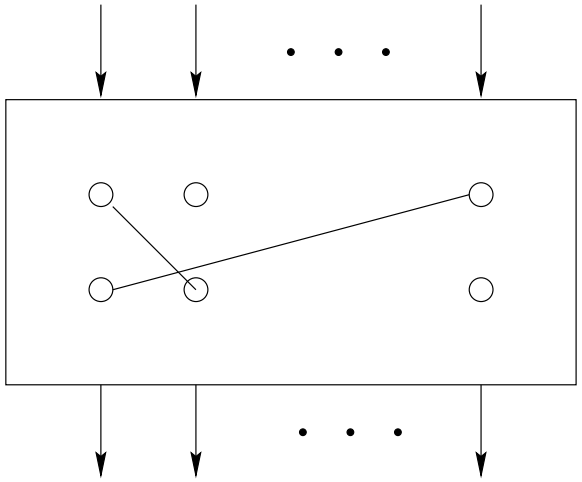
  (1) First type has a neuron potential $x(t)$ with coefficients of $x(t)$ in $g(x(t))$ as connection weights, some negative values of constant terms of $g(x(t))$ as thresholds, and the derivative of the penalty function Φ as neuron activation functions.

  (2) The output of each neuron is computed as the derivative of the penalty function evaluated at the value of $g(x(t))$.

  (3) By taking the output values of the first type of neurons as inputs, the second type has coefficients of $\phi(g(x(t)))$ in $-\eta s \sum_{i=1}^{m} \phi(g_i(x(t)))\nabla_{x(t)} g_i(x(t))$ as connection weights, values of $\eta \nabla_{x(t)} f(x(t))$ as thresholds, and integrators as neuron activation functions to provide $x(t)$ for the input of the first type of neurons in the next iteration.

- This process is repeated, and the solutions computed by the neural network can be obtained by measuring the output voltage of the second type of neurons in the corresponding neural circuit after the neural network reaches its equillibrium.

- **Example (Crossbar Switching)**



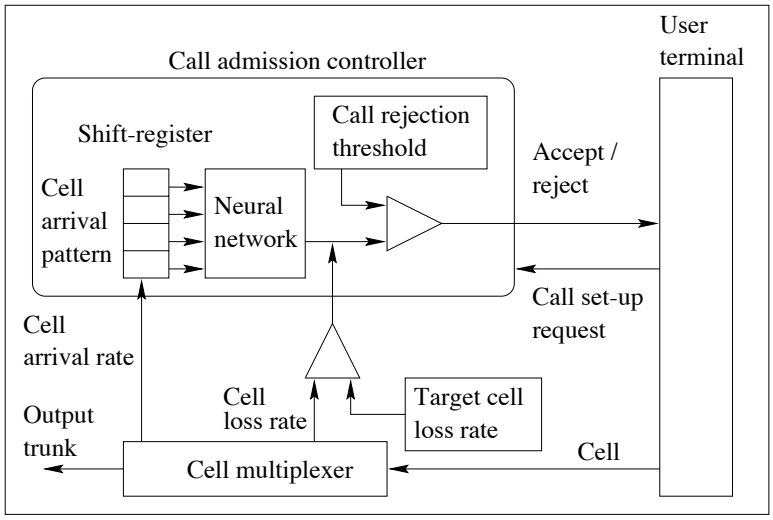
$T = [t_{ij}]$ traffic matrix

$V = [v_{ij}]$ output of neurons

$$E = -A\sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij}v_{ij} + B\sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij}[\sum_{p\neq i} v_{pj} + \sum_{q\neq j} v_{iq}]$$

- CMOS VLSI implementation takes only 120 *nsec* for an 8×8 crossbar switch optimal routing.

- **Example (Call Admission Control)**



- Input to NN

  (1) Observed status of multiplexer
  - cell arrival rate
  - cell loss rate
  - trunk utilization rate
  - ...

  (2) Call set-up parameters
  - average bit rate
  - rate variation
  - ...

- Output of NN

- predicted QoS
- acceptance / rejection decision

Multi-layer Perceptron Approximation Power

- There is very little theoretical guidance for determining network size in terms of, say, the number of hidden nodes and hidden layers it should contain.

- Cybenko showed that a backpropagation MLP, with one hidden layer and any fixed continuous sigmoidal nonlinear function, can approximate any continuous function arbitrarily well on a compact set.

  G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems, 2:303-314, 1989.

- When used as a binary-valued neural network with the hard-limiter (step) activation function, a backpropagation MLP with two hidden layers can form arbitrary complex decision regions to separate different classes, as Lippmann pointed out.

  R. P. Lippmann. An introduction to computing with neural networks. IEEE Acoustics, Speech, and Signal Processing Magazine, 4(2):4-22, 1987.

- For function approximation as well as data classification, two hidden layers may be required to learn a piecewise-continuous function.

  Timothy Masters. Practical neural network recipe in C++. Academic Press, Inc., 1993.

- In their book, Hertz et al. introduced an intuitive explanation that MLPs with two hidden layers may be able to construct localized receptive fields out of logistic functions.

  J.Hertz, A. Krogh, and R. G. Palmer. Introduction to the theory of neural computation. Addison-wesley, Reading, MA, 1991.

- The work of Leshno et al. could be one of the most general results. They showed that if the sigmoidal activation function used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial, then a three-layered neural network can approximate any continuous function with respect to the uniform norm.

  Leshno, M., Lin, V., Pinkus, A., Shochen, S. (1993). Multilayer feedforward networks with a polynomial activation function can approximate any function. Neural Networks, 6, 861-867.

Radial Basis Function Networks (RBFN)

- A radial basis function has the form

$$g(x, \theta, b) = \phi(\frac{x-\theta}{b})$$

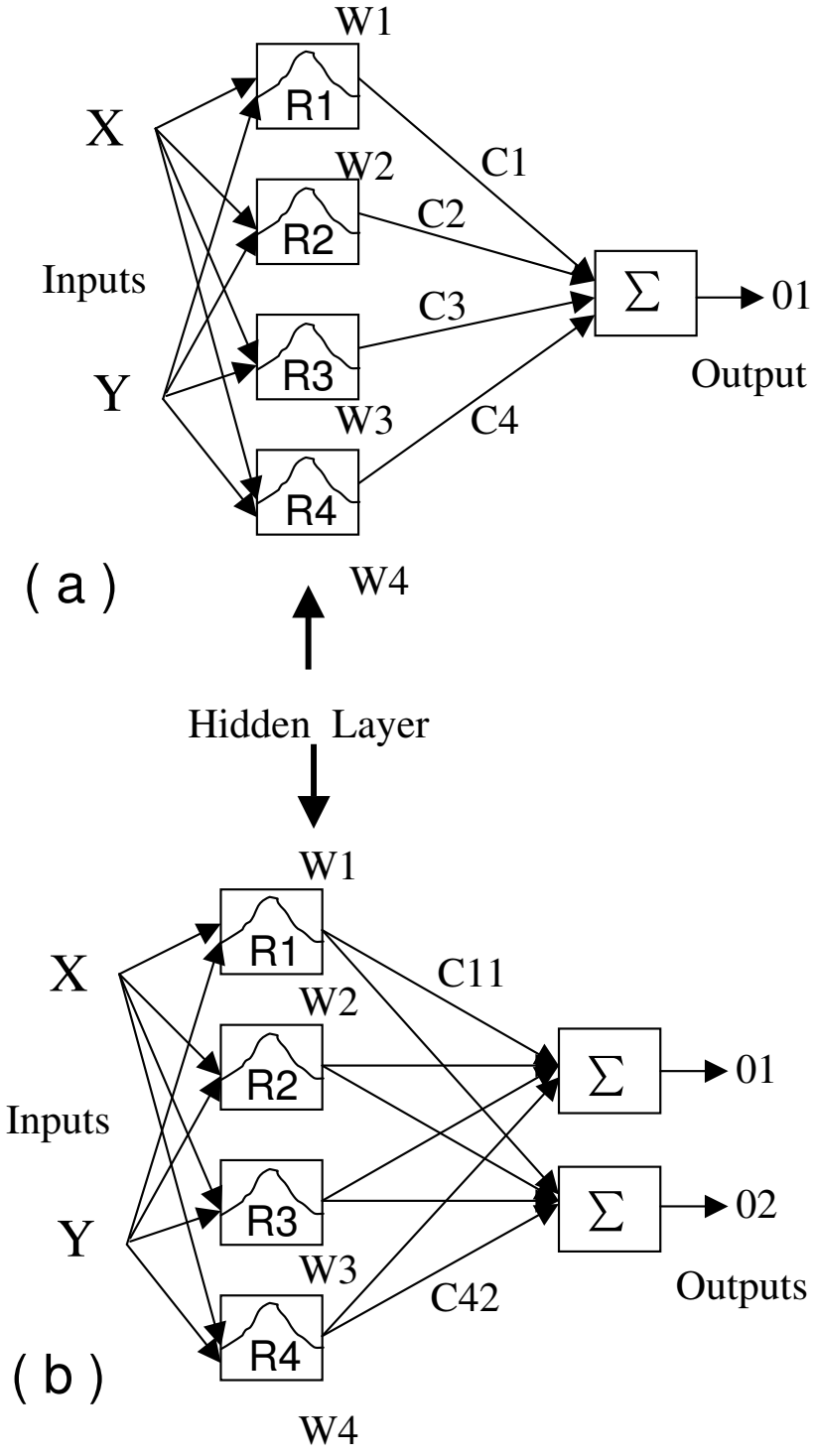
  where $\phi : R^n \rightarrow R$, $x \in R^n$, $\theta \in R^n$ in a "center vector", and $b \in R$ is a "spread parameter".

- The Gaussian function

$$g(x) = \exp(-\frac{\|x-\theta\|}{b})$$

  is a typical example.

- RBFN is a neural network using radial basis functions as activation functions.


W1
X
R1
W2
C1
C2
R2
Inputs
Σ
01
C3
R3
Output
Y
W3
C4
R4
(a)
W4
Hidden Layer
W1
R1
X
C11
W2
R2
Σ
01
Inputs
R3
Σ
02
Y
W3
C42
Outputs
R4
(b)
W4

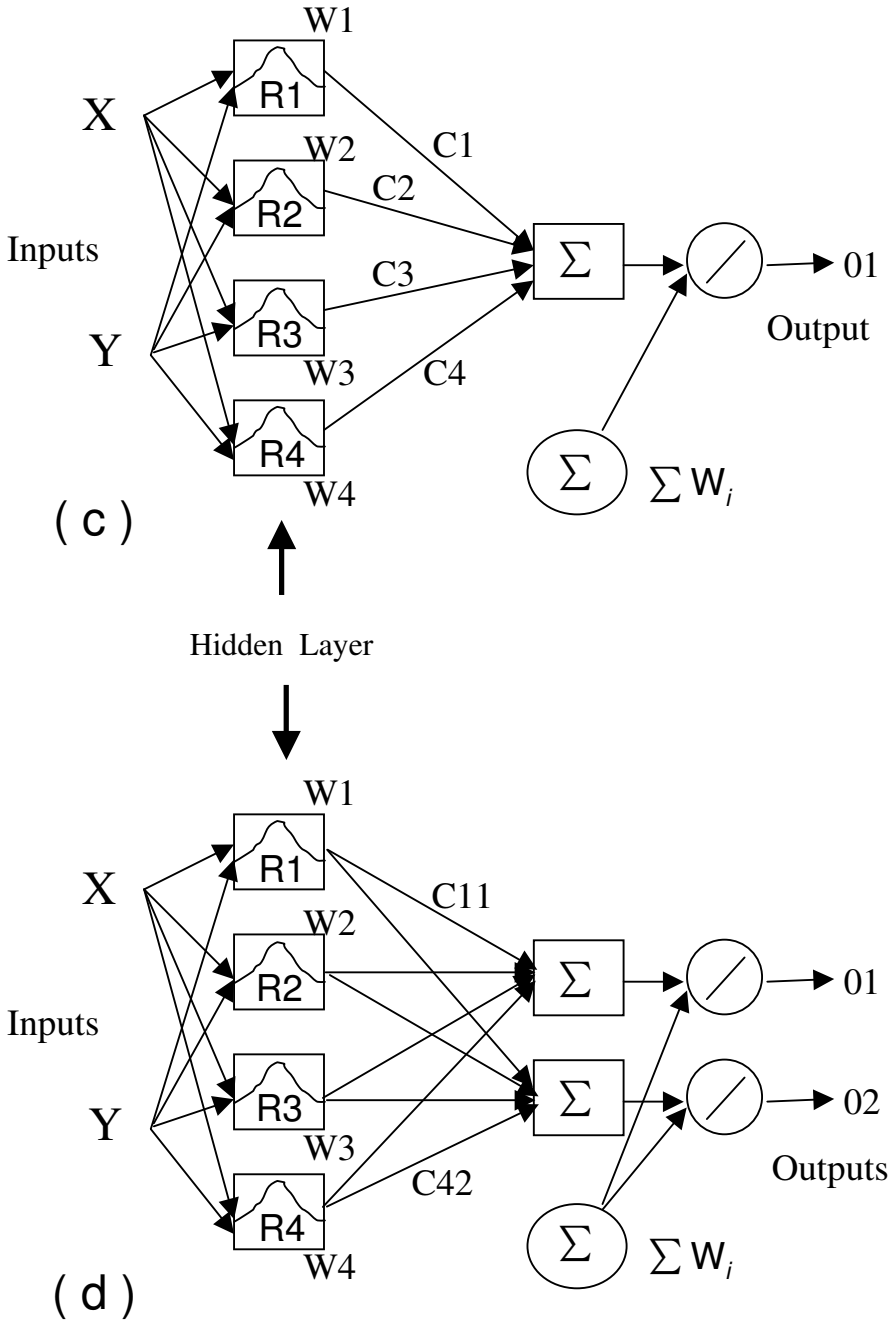


Figure1. (from Neuro-Fuzzy and Soft Computing) Four RBFNs that possess four basis functions: (a) single-output RBFN that uses weighted sum; (b) two-output RBFN that uses weighted sum; (c) single-output RBFN that uses weighted average; (d) two-output RBFN that uses weighted average.

Approximation Power of RBFN

- The most well-known result is due to Park and Sandberg. They showed that if the radial-basis activation function used in the hidden layer is continuous almost everywhere, bounded and integrable on R^n, and the integration is not zero, then a three-layered neural network can approximate any function in $L^p(R^n)$ with respect to the L^p norm with $1 \leq p < \infty$.

  Park, J., Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks. Neural Computation, 3(2), 246-257.

  Park, J., Sandberg, I. W. (1993). Approximation and radial-basis-function networks. Neural Computation, 5, 305-316.

- The most general result is due to Liao, Fang and Nuttle. They showed that, if the radial-basis activation function used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial, then the three-layered radial-basis function network can approximate any continuous function with respect to the uniform norm. Moreover, Radial Basis Function Networks (RBFN) can approximate any function in $L^p(\mu)$, where $1 \leq p < \infty$ and μ is any finite measure, if the radial-basis activation function used in the hidden layer is essentially bounded and not a polynomial.

  Liao, Y., Fang, S. C., Nuttle, H. L. W. (2003). Relaxed conditions for radial-basis function networks to be universal approximators. Neural Networks, 16, 1019-1028.