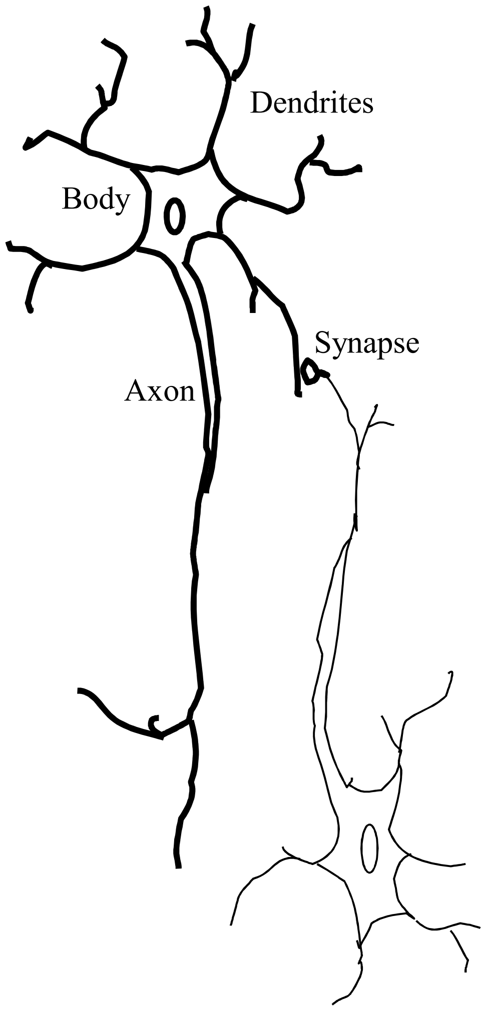
Neural Networks : Basics

Emil M. Petriu
School of Electrical Engineering and Computer Science
University of Ottawa
http://www.site.uottawa.ca/~petriu/
petriu@uottawa.ca

Biological Neurons



- **Dendrites** carry electrical signals in into the neuron body.
  The neuron **body** integrates and thresholds the incoming signals.
  The **axon** is a single long nerve fiber that carries the signal from the neuron body to other neurons.
  A **synapse** is the connection between dendrites of two neurons.

- Incoming signals to a dendrite may be inhibitory or excitatory. The strength of any input signal is determined by the strength of its synaptic connection. A neuron sends an impulse down its axon if excitation exceeds inhibition by a critical amount (threshold/ offset/bias) within a time window (period of latent summation).

- *Memories* are formed by the modification of the **synaptic strengths** which can change during the entire life of the neural systems..

- Biological neurons are rather slow (10^{-3} s) when compared with the modern electronic circuits. ==> The brain is faster than an electronic computer because of its massively parallel structure. The brain has approximately 10^{11} highly connected neurons (approx. 10^{4} connections per neuron).

Historical Sketch of Neural Networks

1940s

Natural components of mind-like machines are simple abstractions based on the behavior of biological nerve cells, and such machines can be built by interconnecting such elements.

- **W. McCulloch & W. Pitts** (1943) the first theory on the fundamentals of neural computing (neuro-logicalnetworks) "A Logical Calculus of the Ideas Immanent in Nervous Activity" ==> *McCulloch-Pitts neuron model*; (1947) "How We Know Universals" - an essay on networks capable of recognizing spatial patterns invariant of geometric transformations.

- **Cybernetics:** attempt to combine concepts from biology, psychology, mathematics, and engineering.

- **D.O. Hebb** (1949) "The Organization of Behavior" the first theory of psychology on conjectures about neural networks (neural networks might learn by constructing internal representations of concepts in the form of "cell-assemblies" - subfamilies of neurons that would learn to support one another's activities). ==> *Hebb's learning rule*: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

1950s

Cybernetic machines developed as specific architectures to perform specific functions.
==> “machines that could learn to do things they aren’t built to do”

- **M. Minsky** (1951) built a reinforcement-based network learning system.
- **IRE Symposium “The Design of Machines to Simulate the Behavior of the Human Brain”** (1955) with four panel members: W.S. McCulloch, A.G. Oettinger, O.H. Schmitt, N. Rochester, invited questioners: M. Minsky, M. Rubinoff, E.L. Gruenberg, J. Mauchly, M.E. Moran, W. Pitts, and the moderator H.E. Tompkins.
- **F. Rosenblatt** (1958) the first practical Artificial Neural Network (ANN) - the *perceptron*, “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.”.

⇨ By the end of 50s, the NN field became dormant because of the new AI advances based on serial processing of symbolic expressions.

1960s

Connectionism (Neural Networks) - versus - **Symbolism** (Formal Reasoning)

- **B. Widrow & M.E. Hoff** (1960) "Adaptive Switching Circuits" presents an adaptive percepton-like network. The weights are adjusted so to minimize the mean square error between the actual and desired output ==> *Least Mean Square (LMS) error algorithm*. (1961) Widrow and his students "Generalization and Information Storage in Newtworks of Adaline "Neurons."

- **M. Minsky & S. Papert** (1969) "Perceptrons" a formal analysis of the percepton networks explaining their limitations and indicating directions for overcoming them ==> *relationship between the perceptron's architecture and what it can learn*: "no machine can learn to recognize X unless it poses some scheme for representing X."

  Limitations of the perceptron networks led to the pessimist view of the NN field as having no future ==> no more interest and funds for NN research!!!

1970s

Memory aspects of the Neural Networks.

- **T. Kohonen** (1972) “Correlation Matrix Memories” a mathematical oriented paper proposing a correlation matrix model for associative memory which is trained, using Hebb’s rule, to learn associations between input and output vectors.

- **J.A. Anderson** (1972) “A Simple Neural Network Generating an Interactive Memory” a physiological oriented paper proposing a “linear associator” model for associative memory, using Hebb’s rule, to learn associations between input and output vectors.

- **S. Grossberg** (1976) “Adaptive Pattern Classification and Universal Recording: I. Parallel Development and Coding of Neural Feature Detectors”describes a self-organizing NN model of the visual system consisting of a short-term and long term memory mechanisms. ==> continuous-time competitive network that forms a *basis for the Adaptive Resonance Theory (ART) networks*.

1980s

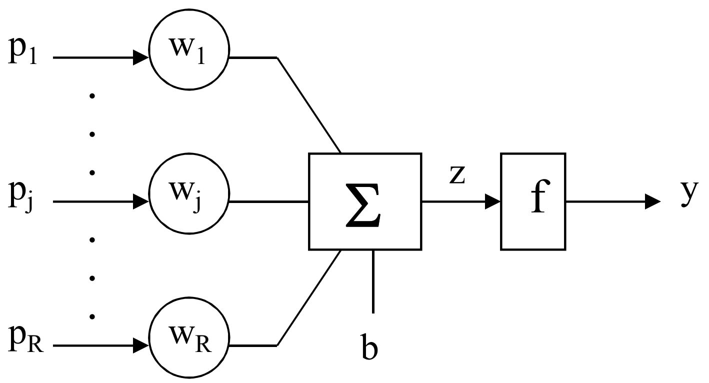
Revival of Learning Machine.

> [Minsky]: "*The marvelous powers of the brain emerge not from any single, uniformly structured connectionst network but from highly evolved arrangements of smaller, specialized networks which are interconnected in very specific ways.*"

- **D.E. Rumelhart & J.L. McClelland**, eds. (1986) "Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Explorations in the Microstructure of Cognition" represents a milestone in the resurgence of NN research.
- **J.A. Anderson & E. Rosenfeld** (1988) "Neurocomputing: Foundations of Research" contains over forty seminal papers in the NN field.
- **DARPA Neural Network Study**(1988) a comprehensive review of the theory and applications of the Neural Networks.
- **International Neural Network Society** (1988) …. **IEEE Tr. Neural Networks** (1990).

Artificial Neural Networks (ANN)

McCulloch-Pitts model of an artificial neuron



$$y = f(w_1 \cdot p_1 + \ldots + w_j \cdot p_j + \ldots w_R \cdot p_R + b)$$

$$\boxed{y = f(\boldsymbol{W} \cdot \boldsymbol{p} + b)}$$

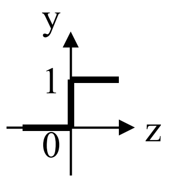
$\boldsymbol{p} = (p_1, \ldots, p_R)^T$ is the input column-vector

$\boldsymbol{W} = (w_1, \ldots, w_R)$ is the weight row-vector

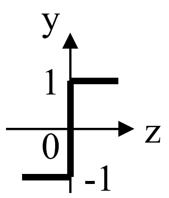
*) The bias *b* can be treated as a weight whose input is always 1.

Some transfer functions "f"

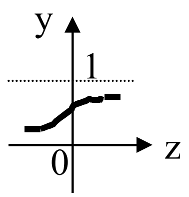
Hard Limit: $y = 0$ if $z<0$
$y = 1$ if $z>=0$



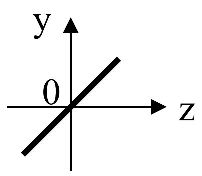
Symmetrical Hard Limit: $y = -1$ if $z<0$
$y = +1$ if $z>=0$



Log-Sigmoid:
$y = 1/(1+e^{-z})$



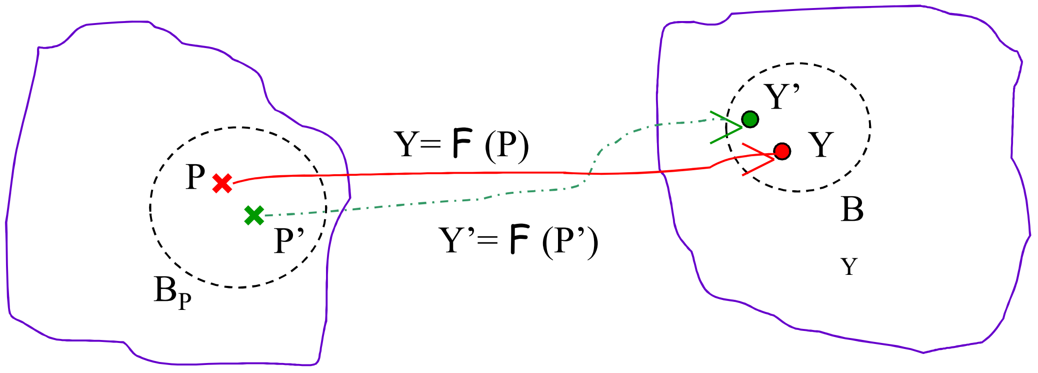
Linear:
$y = z$



The Architecture of an ANN

- Number of inputs and outputs of the network;
- Number of layers;
- How the layers are connected to each other;
- The transfer function of each layer;
- Number of neurons in each layer;

ANNs map input/stimulus values to output/response values: Y= F (P).

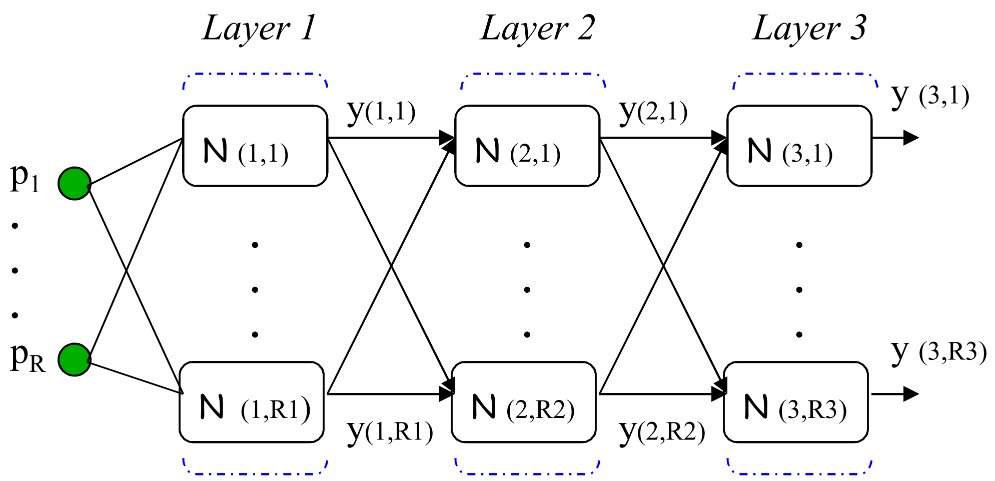


Intelligent systems generalize: their behavioral repertoires exceed their experience. An intelligent system is said to have a creative behaviour if it provides appropriate responses when faced with new stimuli. Usually the new stimuli *P'* resemble known stimuli *P* and their corresponding responses *Y'* resemble known/learned responses *Y*.

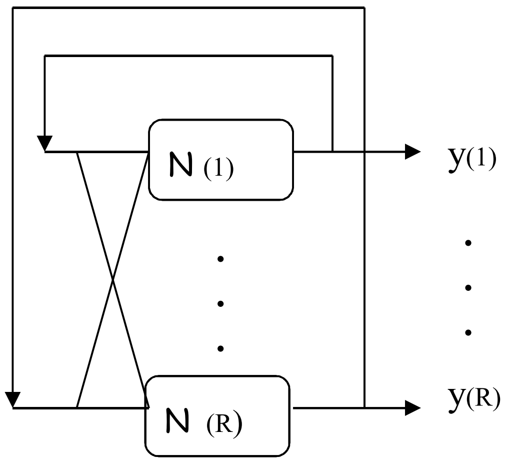
Measure of system's F creativity:

$$\boxed{\frac{\text{Volume of "stimuli ball } B_P\text{ "}}{\text{Volume of "response ball } B_Y\text{"}}}$$

- ✓ Most of the mapping functions can be implemented by a two-layer ANN: a sigmoid layer feeding a linear output layer.
- ✓ ANNs with biases can represent relationships between inputs and outputs than networks without biases.
- ✓ *Feed-forward* ANNs cannot implement temporal relationships. *Recurrent* ANNs have internal feedback paths that allow them to exhibit temporal behaviour.



Feed-forward architecture with three layers



Recurrent architecture (*Hopfield* NN)

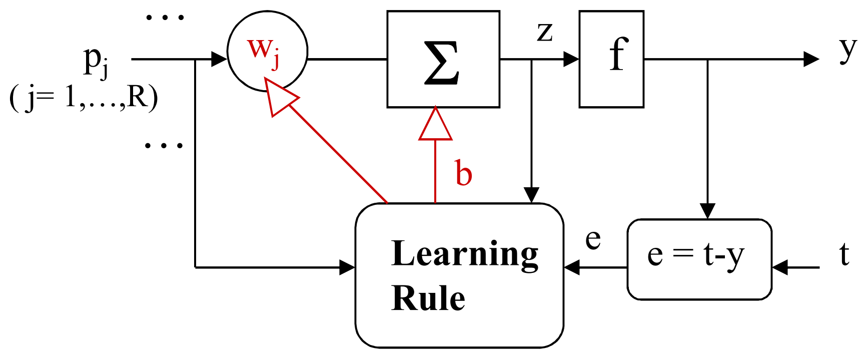
The ANN is usually supplied with an initial input vector and then the outputs are used as inputs for each succeeding cycle.

Learning Rules (Training Algorithms)

Procedure/algorithm to adjust the weights and biases in order for the ANN to perform the desired task.

Supervised Learning

For a given training set of pairs {***p***(1),t(1)},...,{***p***(n),t(n)}, where ***p***(i) is an instance of the input vector and **t**(i) is the corresponding *target* value for the output ***y***, the learning rule calculates the updated value of the neuron weights and bias.



Reinforcement Learning

Similar to supervised learning - instead of being provided with the correct output value for each given input, the algorithm is only provided with a given grade/score as a measure of ANN's performance.

Unsupervised Learning

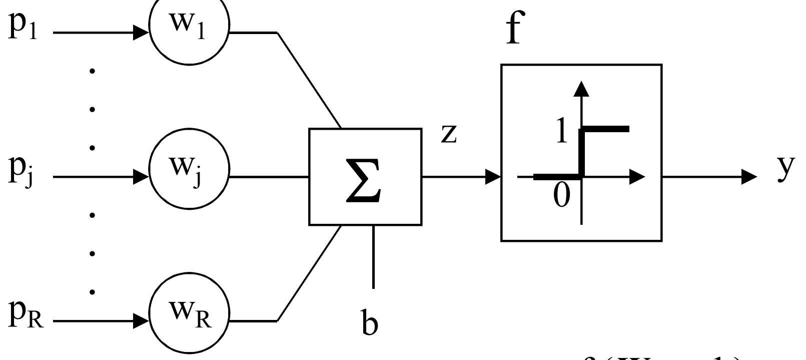
The weight and unbiased are adjusted based on inputs only. Most algorithms of this type learn to cluster input patterns into a finite number of classes. ==> e.g. vector quantization applications

☆ THE PERCEPTRON

- Frank Rosenblatt (1958), Marvin Minski & Seymour Papert (1969)

- [Minski] "Perceptrons make decisions/determine whether or not event fits a certain pattern by adding up evidence obtained from many small experiments"

- The **perceptron** is a neuron with a hard limit transfer function and a weight adjustment mechanism ("learning") by comparing the actual and the expected output responses for any given input /stimulus.



$y = f\,(\boldsymbol{W}\cdot\boldsymbol{p} + b)$

- Perceptrons are well suited for pattern classification/recognition.

- The weight adjustment/training mechanism is called the *perceptron learning rule*.

NB: $\boldsymbol{W}$ is a row-vector and $\boldsymbol{p}$ is a column-vector.

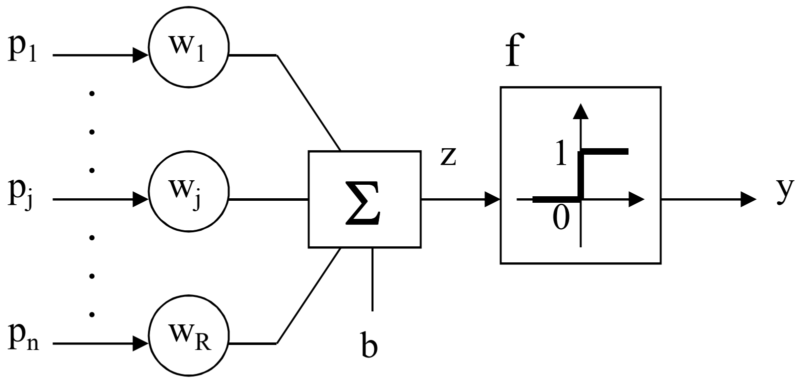
Perceptron Learning Rule

- **Supervised learning**

t <== the target value

e = t-y <== the error

Because of the perceptron's hard limit transfer function *y, t, e* can take only binary values



$\boldsymbol{p} = (p_1, \ldots, p_R)^T$ is the input column-vector

$\boldsymbol{W} = (x_1, \ldots, x_R)$ is the weight row-vector

Perceptron learning rule:

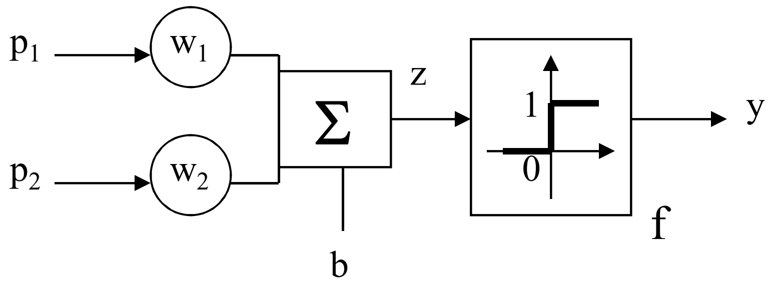
$$\begin{cases} \text{if } e = 1, \text{ then } \boldsymbol{W}^{new} = \boldsymbol{W}^{old} + \boldsymbol{p}, \; b^{new} = b^{old} + 1; \\ \text{if } e = -1, \text{ then } \boldsymbol{W}^{new} = \boldsymbol{W}^{old} - \boldsymbol{p}, \; b^{new} = b^{old} - 1; \\ \text{if } e = 0, \text{ then } \boldsymbol{W}^{new} = \boldsymbol{W}^{old}. \end{cases}$$

$\Rightarrow$

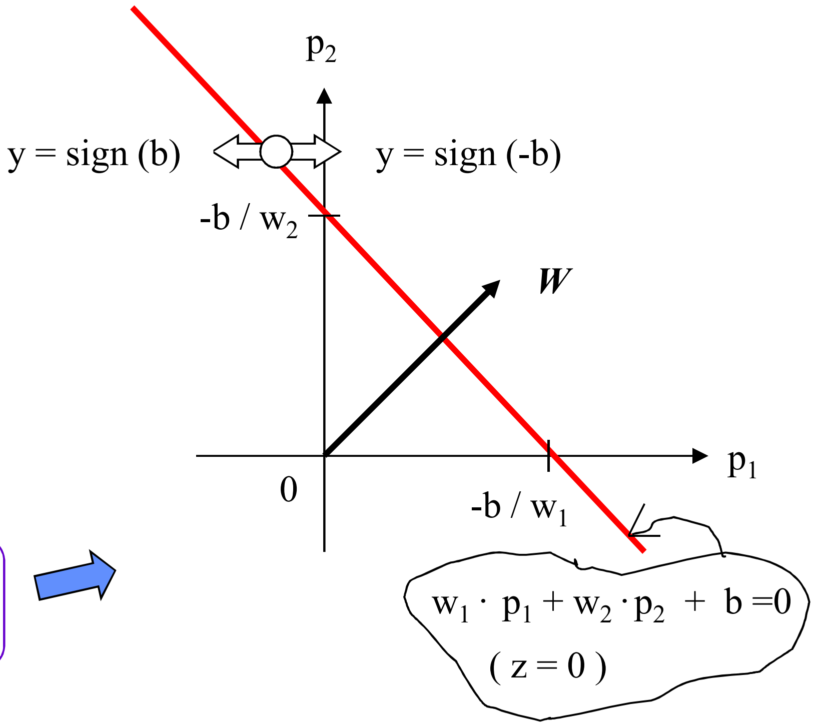
$$\boxed{\begin{aligned} \boldsymbol{W}^{new} &= \boldsymbol{W}^{old} + e \cdot \boldsymbol{p}^T \\ b^{new} &= b^{old} + e \end{aligned}}$$

The hard limit transfer function (threshold function) provides the ability to classify input vectors by deciding whether an input vector belongs to one of two *linearly separable classes.*

Two-Input Perceptron



$$y = \textit{hardlim}\,(z) = \textit{hardlim}\{\,[w_1\,, w_2] \cdot [p_1\,, p_2]^T + b\}$$

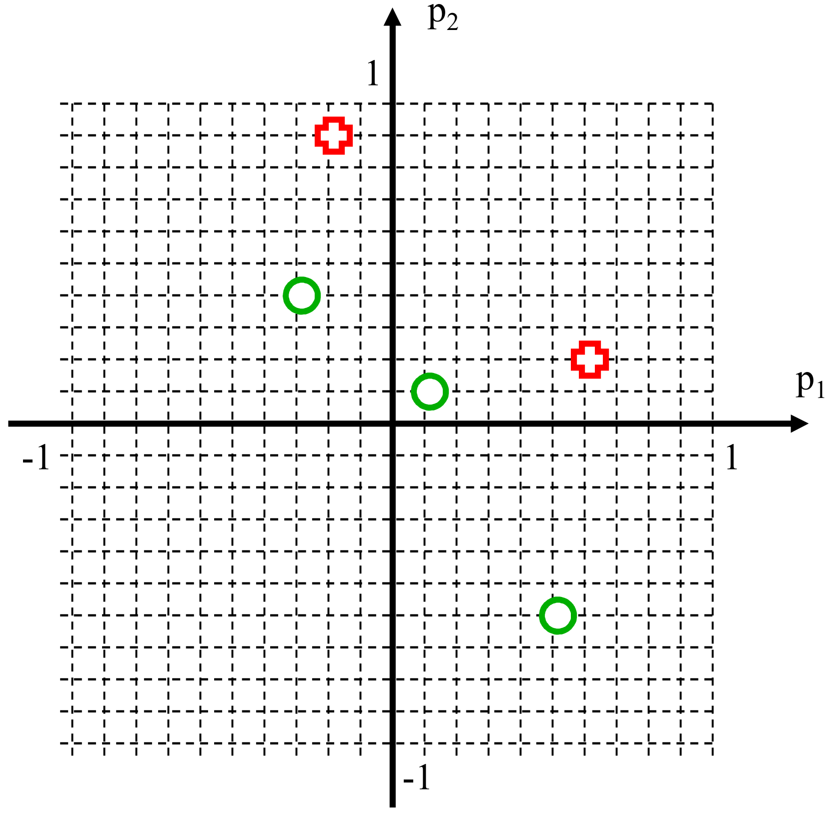


- The two classes (linearly separable regions) in the two-dimensional input space (p_1, p_2) are separated by the line of equation $z = 0$.
- The boundary is always orthogonal to the weight vector ***W***.

❑ **Example #1**: Teaching a two-input perceptron to classify five input vectors into two classes

$\left\{ \begin{matrix} \boldsymbol{p}(1) = (0.6, 0.2)^T \\ \boldsymbol{t}(1) = 1 \end{matrix} \right\}$ $\left\{ \begin{matrix} \boldsymbol{p}(2) = (-0.2, 0.9)^T \\ \boldsymbol{t}(2) = 1 \end{matrix} \right\}$ $\left\{ \begin{matrix} \boldsymbol{p}(3) = (-0.3, 0.4)^T \\ \boldsymbol{t}(3) = 0 \end{matrix} \right\}$ $\left\{ \begin{matrix} \boldsymbol{p}(4) = (0.1, 0.1)^T \\ \boldsymbol{t}(4) = 0 \end{matrix} \right\}$ $\left\{ \begin{matrix} \boldsymbol{p}(5) = (0.5, -0.6)^T \\ \boldsymbol{t}(5) = 0 \end{matrix} \right\}$

▶ The MATLAB solution is:



```
P=[0.6 -0.2 -0.3 0.1 0.5;
   0.2  0.9  0.4 0.1 -0.6];
T=[1 1 0 0 0];
W=[-2 2];
b=-1;
plotpv(P,T);
plotpc(W,b);
nepoc=0
Y=hardlim(W*P+b);
while any(Y~=T)
Y=hardlim(W*P+b);
E=T-Y;
[dW,db]= learnp(P,E);
W=W+dW;
b=b+db;
nepoc=nepoc+1;
disp('epochs='),disp(nepoc),
disp(W), disp(b);
plotpv(P,T);
plotpc(W,b);
end
```

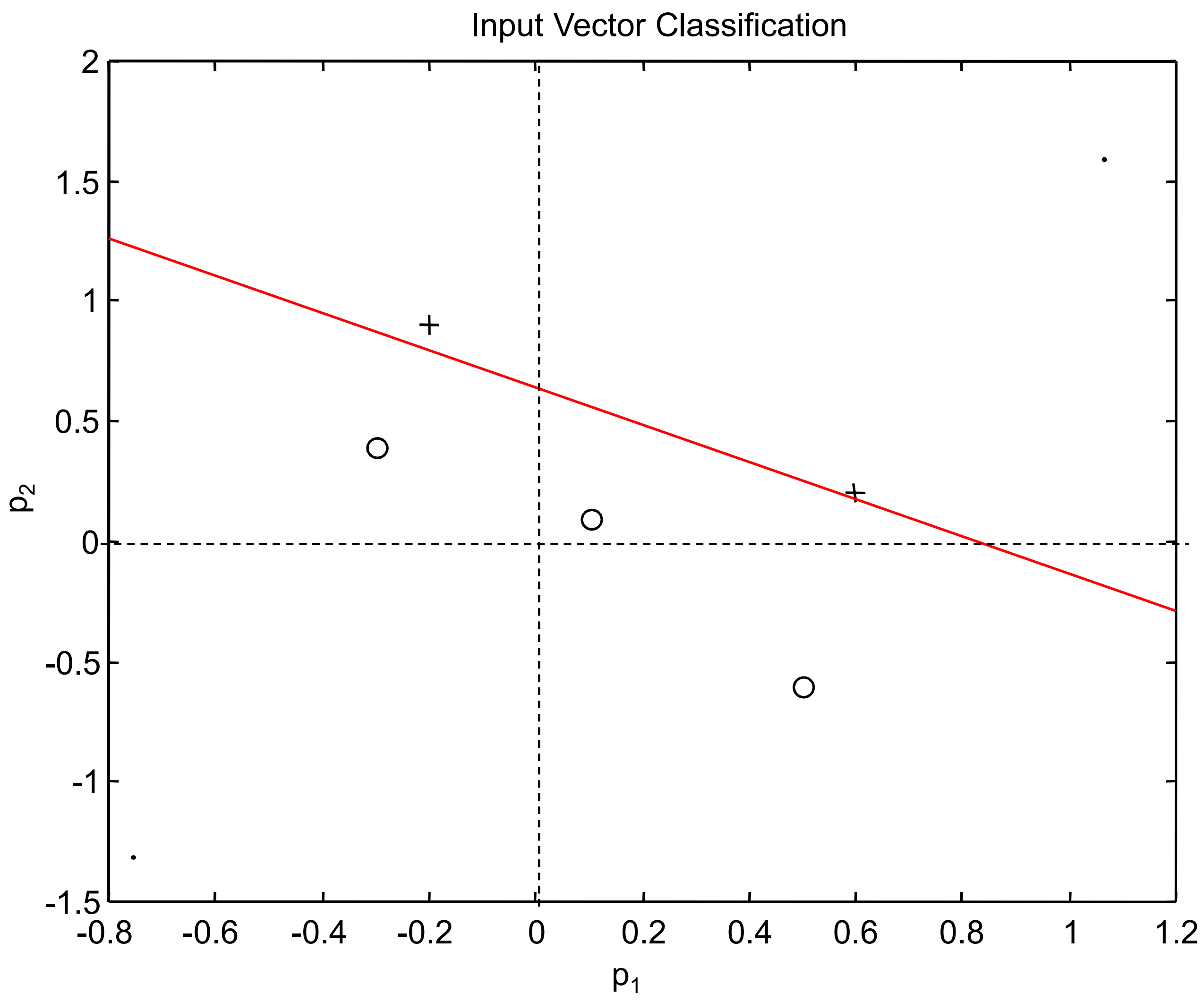
❑ **Example #1:**

After *nepoc* = 11 (epochs of training starting from an initial weight vector `W=[-2 2] and a bias b=-1`) the weights are:

$\boxed{\begin{array}{l} w_1 = 2.4 \\ w_2 = 3.1 \end{array}}$

and the bias is:

$\boxed{b = -2}$



- The larger an input vector $\boldsymbol{p}$ is, the larger is its effect on the weight vector $\boldsymbol{W}$ during the learning process

Long training times can be caused by the presence of an "outlier," i.e. an input vector whose magnitude is much larger, or smaller, than other input vectors.

Normalized perceptron learning rule, the effect of each input vector on the weights is of the same magnitude:

$$W^{new} = W^{old} + e\,\boldsymbol{p}^T / \|\boldsymbol{p}\|$$

$$b^{new} = b^{old} + e$$

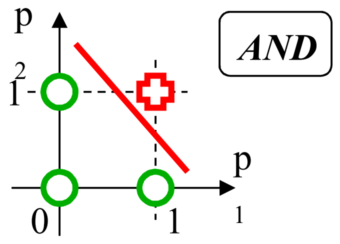
Perceptron Networks for Linearly Separable Vectors

The hard limit transfer function of the perceptron provides the ability to classify input vectors by deciding whether an input vector belongs to one of two *linearly separable classes.*

$$\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 & 1; \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{t}_{AND} = [\,0\ \ 0\ \ 0\ \ 1\,]$$

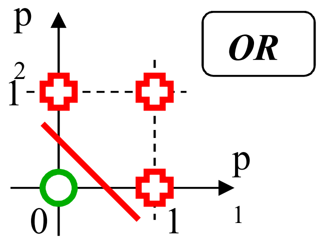
$$\mathbf{W} = [\,2\ \ 2\,]$$
$$b = -3$$



$$\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 & 1; \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

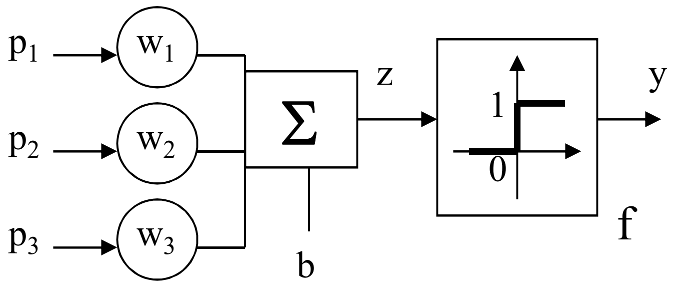
$$\mathbf{t}_{OR} = [\,0\ \ 1\ \ 1\ \ 1\,]$$

$$\mathbf{W} = [\,2\ \ 2\,]$$
$$b = -1$$



Three-Input Perceptron

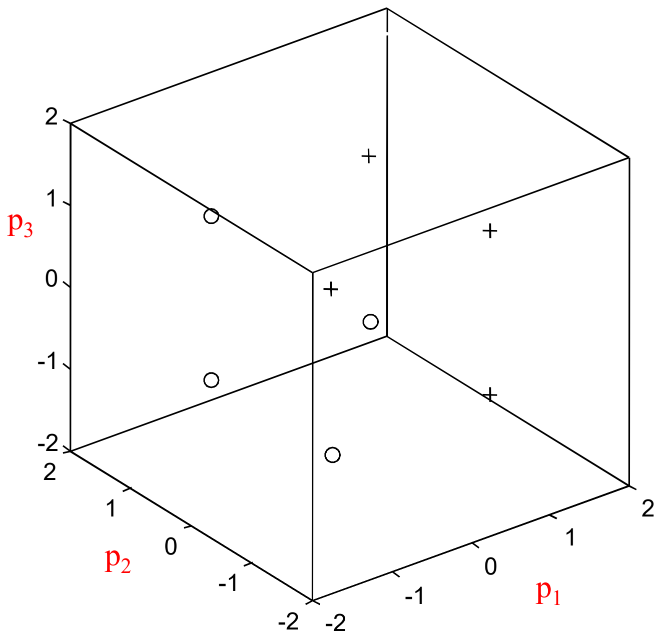
The two classes in the 3-dimensional input space (p_1, p_2, p_3) are separated by the plane of equation $z = 0$.

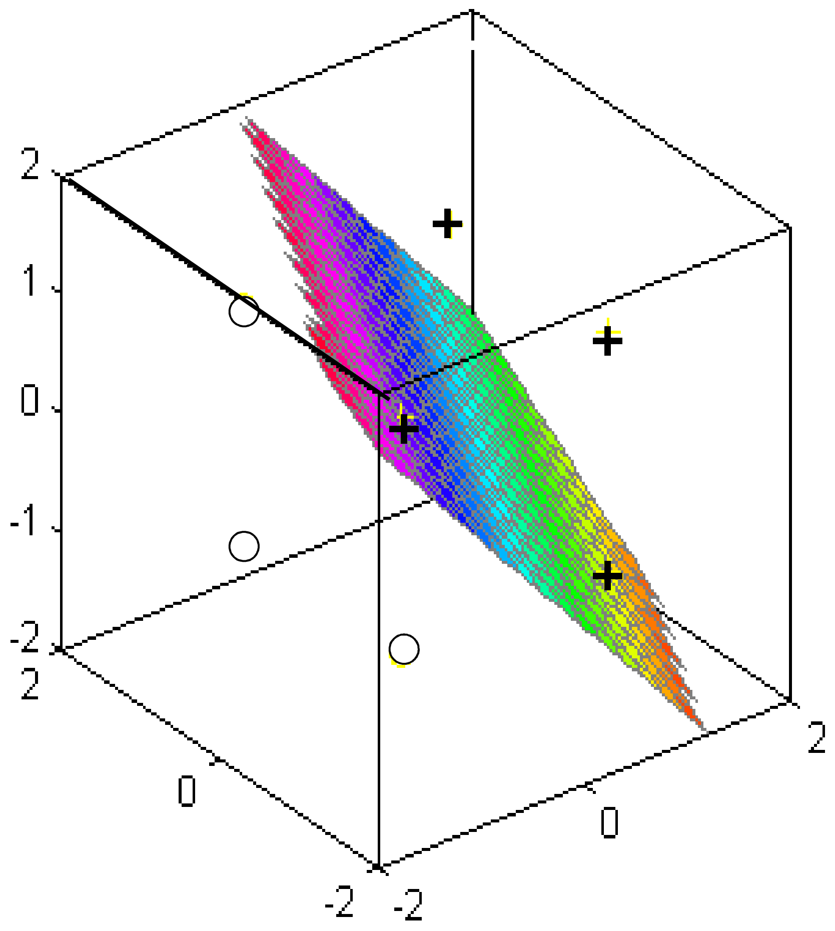


$$
\begin{aligned}
y &= \mathit{hardlim}\,(z) \\
&= \mathit{hardlim}\{\, [w_1, w_2, w_3] \cdot [p_1, p_2\, p_3]^T + b \}
\end{aligned}
$$

EXAMPLE

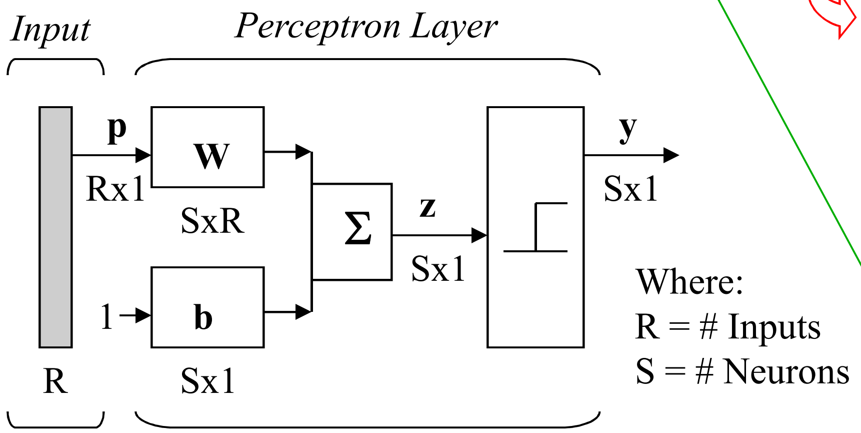
$$
P = \begin{bmatrix} -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1; \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1; \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad T = [\, 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \,]
$$





One-layer multi-perceptron classification of linearly separable patterns

MATLAB representation:



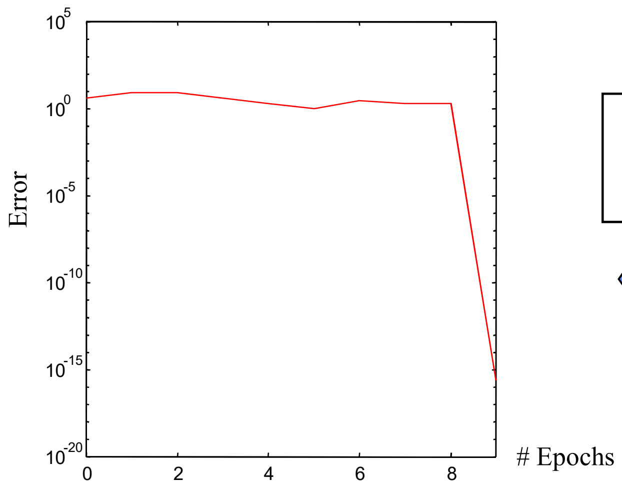
Where:
R = # Inputs
S = # Neurons

$\mathbf{y} = \text{hardlim}(\mathbf{W}^*\mathbf{p}+\mathbf{b})$

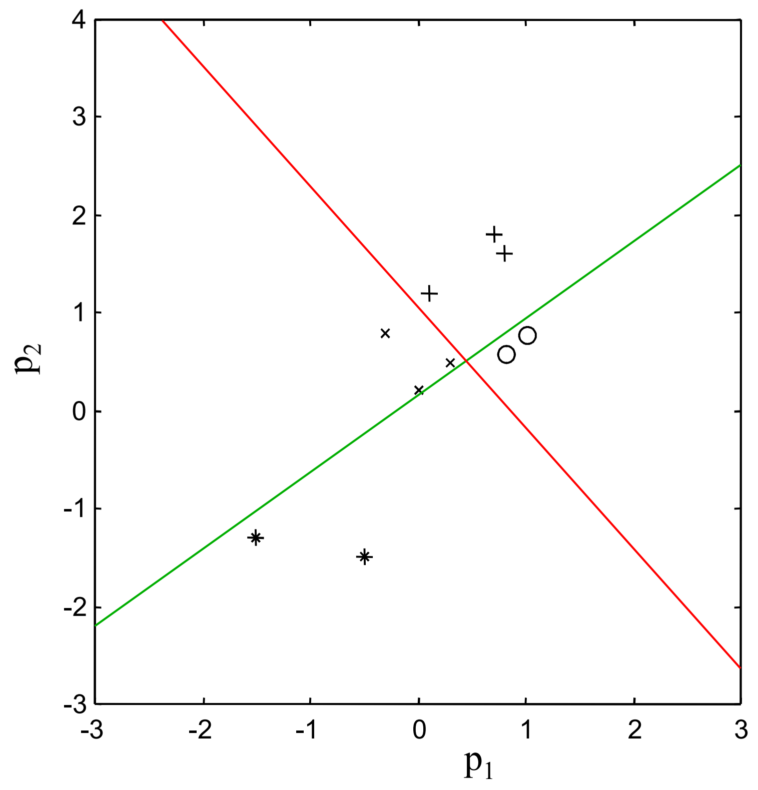
Demo P3 in the "MATLAB Neural Network Toolbox - User's Guide"

$$P = [\ 0.1\ \ 0.7\ \ 0.8\ \ 0.8\ \ 1.0\ \ 0.3\ \ 0.0\ \ -0.3\ \ -0.5\ \ -1.5;\ \ 1.2\ \ 1.8\ \ 1.6\ \ 0.6\ \ 0.8\ \ 0.5\ \ 0.2\ \ 0.8\ \ -1.5\ \ -1.3\ ]$$

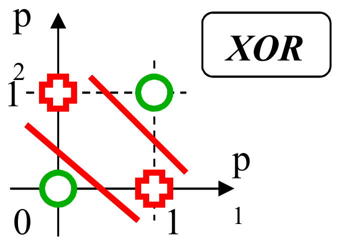
$$T = [\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0;\ \ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ ] \Leftrightarrow \left\{ \begin{array}{l} 00 = \text{O}\ ;\ 10 = + \\ 01 = *\ ;\ \ 11 = \text{x} \end{array} \right\}$$



R = 2 inputs
S = 2 neurons



Perceptron Networks for Linearly Non-Separable Vectors

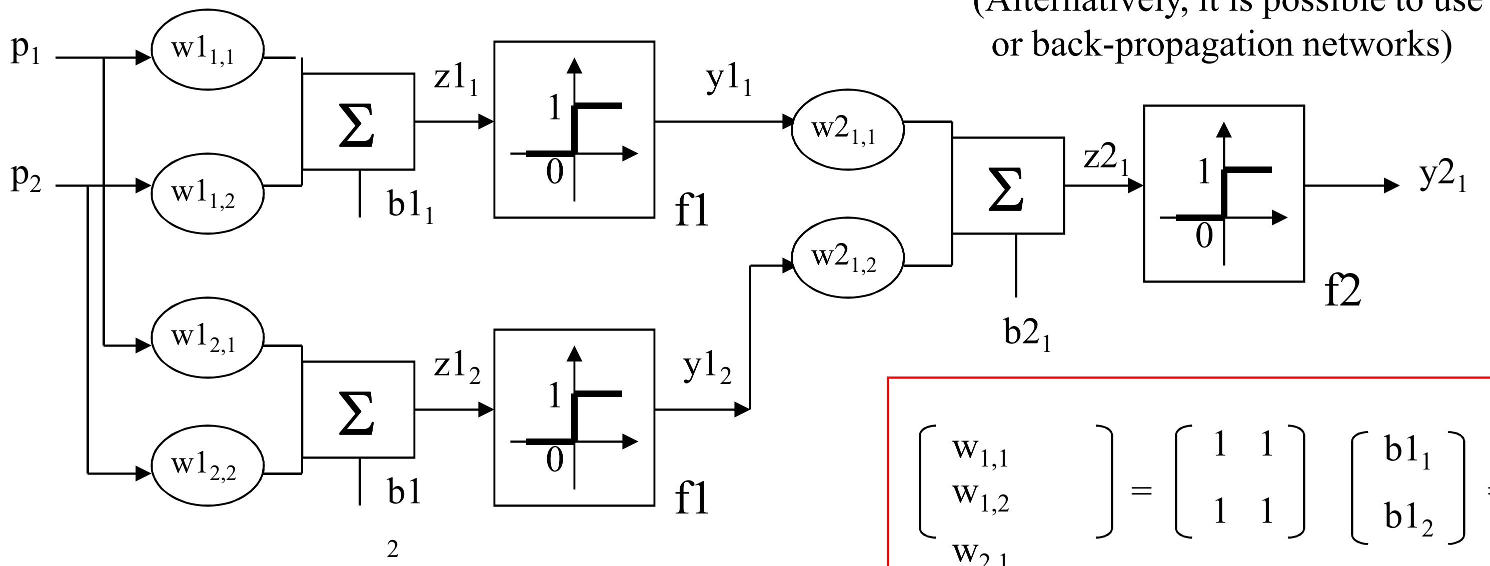


$\mathbf{p}$ = [0 0 1 1 ;
0 1 0 1]
$\mathbf{t}_{XOR}$ = [0 1 1 0]

If a straight line cannot be drawn between the set of input vectors associated with targets of 0 value and the input vectors associated with targets of 1, than a perceptron cannot classify these input vectors.

One solution is to use a two layer architecture, the perceptrons in the first layer are used as preprocessors producing linearly separable vectors for the second layer.

(Alternatively, it is possible to use linear ANN or back-propagation networks)

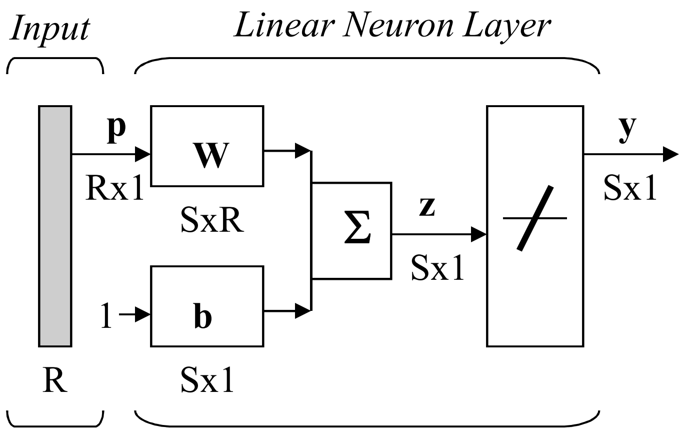


The row index of a weight indicates the destination neuron of the weight and the column index indicates which source is the input for that weight.

$$\begin{bmatrix} w_{1,1} & \\ w_{1,2} & \\ w_{2,1} & \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} b1_1 \\ b1_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$$

$[\ w2_{1,1} \quad w2_{1,2}] = [-1 \quad 1]$ $\qquad$ $[\ b2_1\ ] = [-0.5]$

☆ LINEAR NEURAL NETWORKS (*ADALINE* NETWORKS)



y = purelin(**W*p+b**)

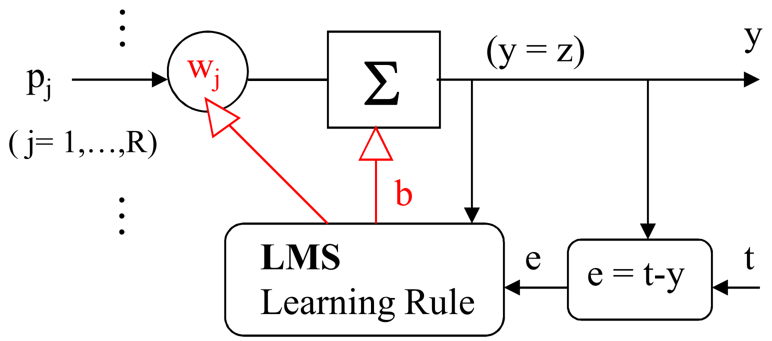
Where: R = # Inputs, S = # Neurons

- Linear neurons have a *linear transfer function*that allows to use a Least Mean-Square (LMS) procedure - *Widrow-Hoff learning rule*- to adjust weights and biases according to the magnitude of errors.

- Linear neurons suffer from the same limitation as the perceptron networks: they can only solve *linearly separable problems*.

(*ADALINE* <== *ADA*ptive *LI*near *NE*uron)

Widrow-Hoff Learning Rule (The δ Rule)



The LMS algorithm will adjust ADALINE's weights and biases in such away to *minimize the* *<u>mean-square-error</u>* *E [e²] between all sets of the desired response and network's actual response*:

$$E\,[\,(t-y)^2\,] = E\,[\,(t - (w_1 \ldots w_R \;\; b)\cdot(p_1 \ldots p_R \;\; 1)^T\,)^2\,]$$
$$= E\,[\,(t - \mathbf{W}\cdot\mathbf{p})^2\,]$$

(<u>NB</u>: E[…] denotes the "expected value"; **p** is column vector)

>> Widrow-Hoff algorithm

$$E[e^2] = E[(t - \mathbf{W} \cdot \mathbf{p})^2] = \{\text{as for deterministic signals the expectation becomes a time-average}\}$$

$$= E[t^2] - 2\cdot\mathbf{W} \cdot \underbrace{E[t\cdot\mathbf{p}]}_{\text{The cross-correlation between the input vector and its associated target.}} + \mathbf{W} \cdot \underbrace{E[\mathbf{p}\cdot\mathbf{p}^T]}_{\text{The input cross-correlation matrix}} \cdot \mathbf{W}^T$$

➡ If the **input correlation matrix is positive** the LMS algorithm will converge as there will a *unique minimum of the mean square error.*

- The W-H rule is an iterative algorithm uses the "steepest-descent" method to reduce the mean-square-error. The key point of the W-H algorithm is that it replaces $E[e^2]$ estimation by the squared error of the iteration *k*: $e^2(k)$. At each iteration step *k* it estimates the gradient of this error ∇_k with respect to **W** as a vector consisting of the partial derivatives of $e^2(k)$ with respect to each weight:

$$\nabla_k^* = \frac{\partial e^2(k)}{\partial W(k)} = \left[\frac{\partial e^2(k)}{\partial w_1(k)} \cdots \frac{\partial e^2(k)}{\partial w_R(k)}, \frac{\partial e^2(k)}{\partial b(k)} \right]$$

The weight vector is then modified in the direction that decreases the error:

$$W(k+1) = W(K) - \mu \bullet \nabla_k^* = W(k) - \mu \bullet \frac{\partial e^2(k)}{\partial W(k)} = W(k) - 2\mu \bullet e(k) \bullet \frac{\partial e(k)}{\partial W(k)}$$

◇ As t(k) and p(k) - both affecting e(k) - are independent of **W**(k), we obtain the final expression of the **Widrow-Hoff learning rule**:

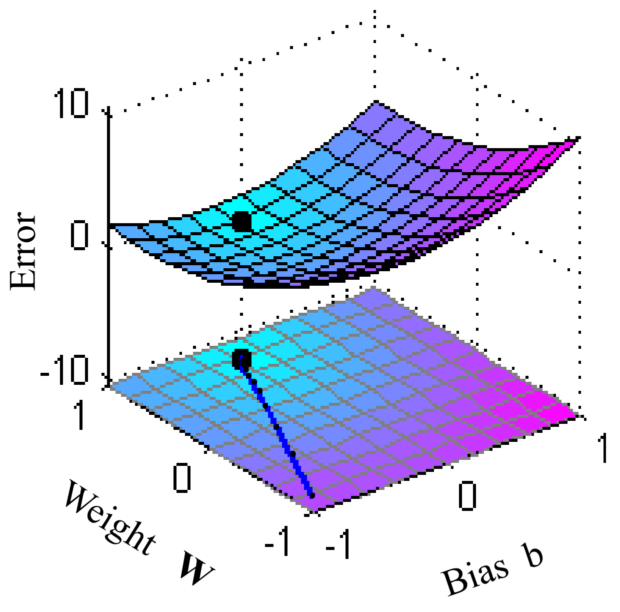
$$\boxed{\mathbf{W}(k+1) = \mathbf{W}(k) + 2\cdot\mu\cdot e(k)\cdot \mathbf{p}(k)} \quad \Rightarrow \quad b(k+1) = b(k) + 2\cdot\mu\cdot e(k)$$

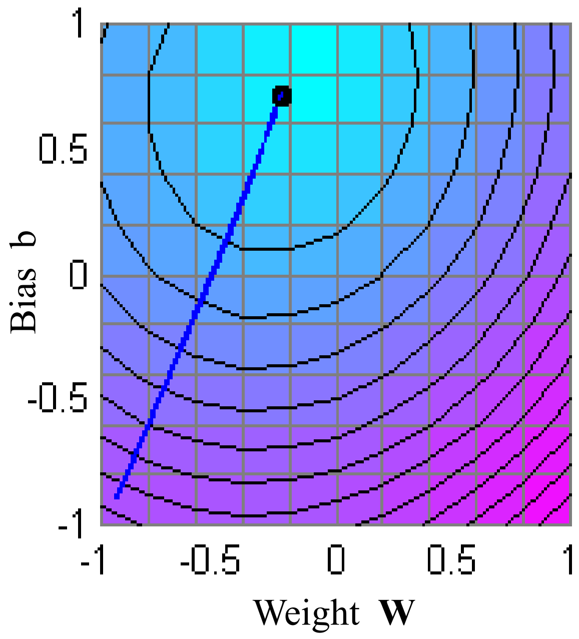
where μ the "learning rate" and $e(k) = t(k) - y(k) = t(k) - \mathbf{W}(k) \cdot \mathbf{p}(k)$

Demo Lin 2 in the "MATLAB Neural Network Toolbox - User's Guide"

P = [1.0 -1.2]
T = [0.5 1.0]

One-neuron one-input ADALINE, starting from some random values for w = -0.96 and b= -0.90 and using the "*trainwh*" MATLAB NN toolbox function, reaches the target after 12 epochs with an error e < 0.001. The solution found for the weight and bias is: w = -0.2354 and b= 0.7066.





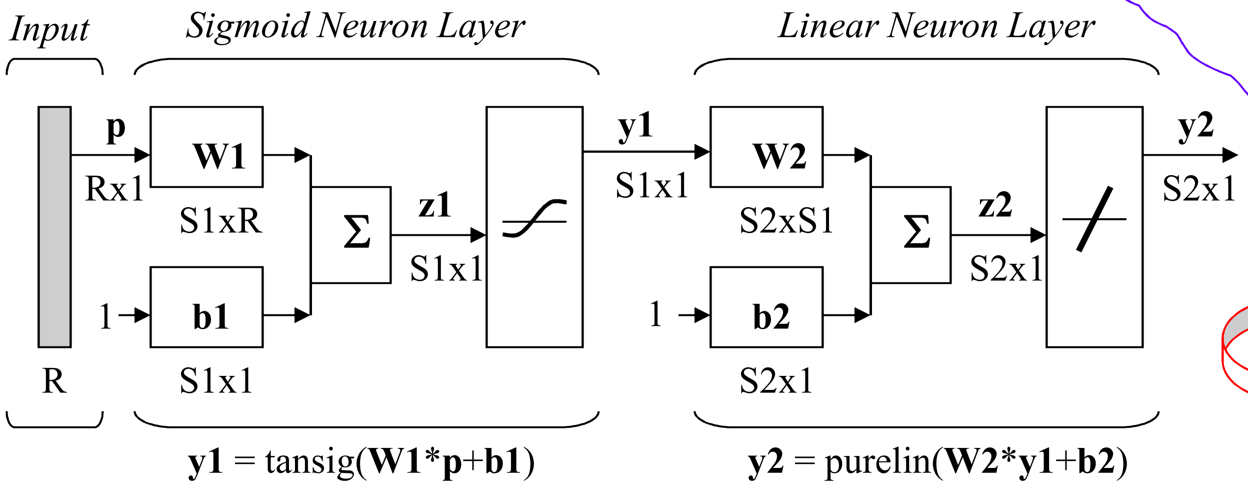
Back-Propagation Learning - The Generalized Δ Rule

P. Werbos (Ph.D. thesis 1974);
D. Parker (1985), Yann Le Cun(1985),
D. Rumelhart, G. Hinton, R. Williams (1986)

- Single layer ANNs are suitable to only solving linearly separable classification problems. Multiple feed-forward layers can give an ANN greater freedom. Any reasonable function can be modeled by a two layer architecture: a sigmoid layer feeding a linear output layer.
- Single layer ANNs are only able to solve linearly Widrow-Hoff learning applies to single layer networks. ==> generalized W-H algorithm (Δ -rule) ==> **back-propagation learning**.

- Back-propagation ANNs often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons.

Two-layer ANN that can approximate any function with a finite number of of discontinuities, arbitrarily well, given sufficient neurons in the hidden layer.



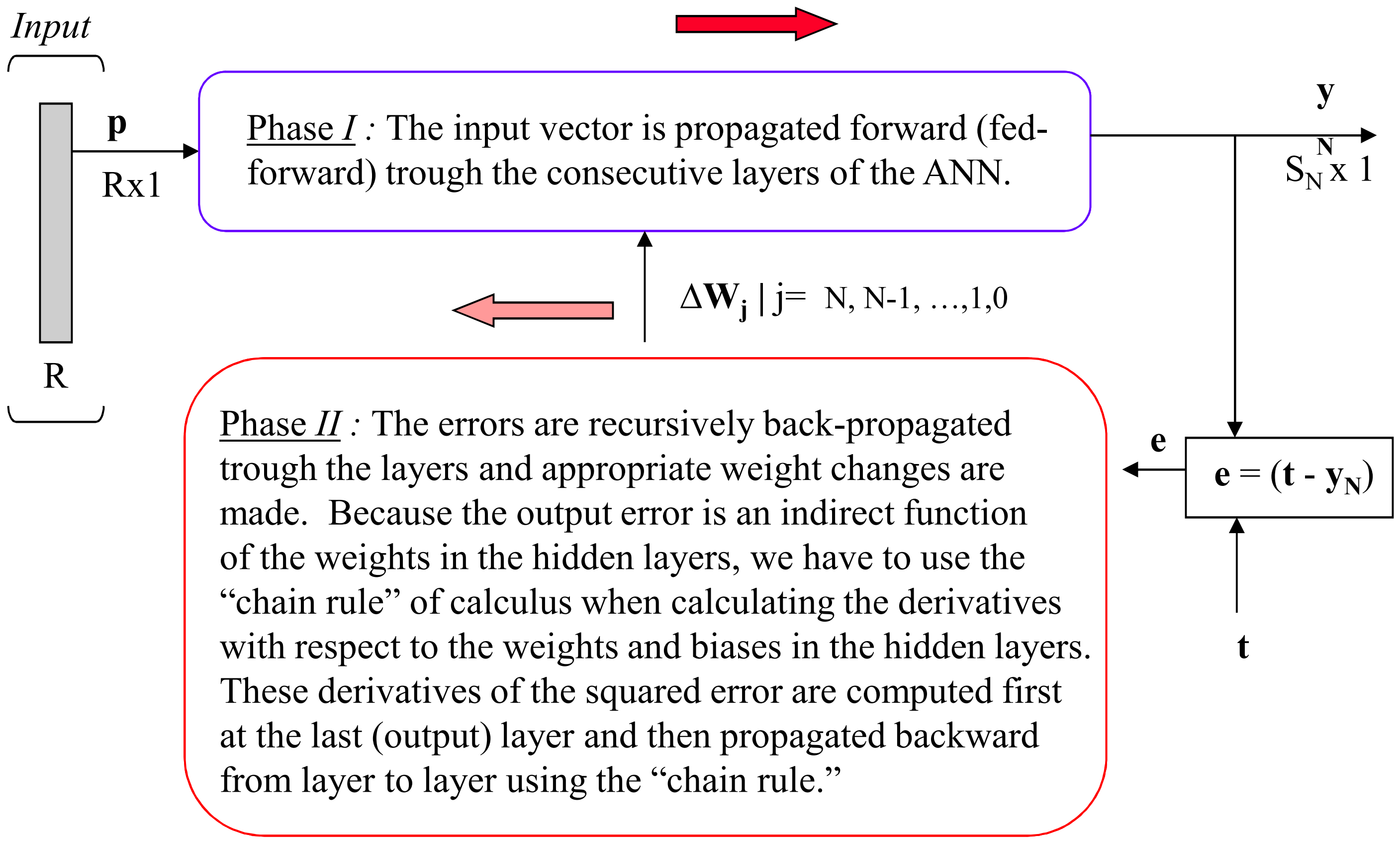
y1 = tansig(**W1*p+b1**)

y2 = purelin(**W2*y1+b2**)

e2= (t-y2) = (t- purelin (**W2***tansig(**W1*p** +**b1**) +**b2**))

The error is an indirect function of the weights in the hidden layers.

- Back-propagation is an iterative steepest descent algorithm, in which the performance index is the mean square error E [e^2] between the desired response and network's actual response:

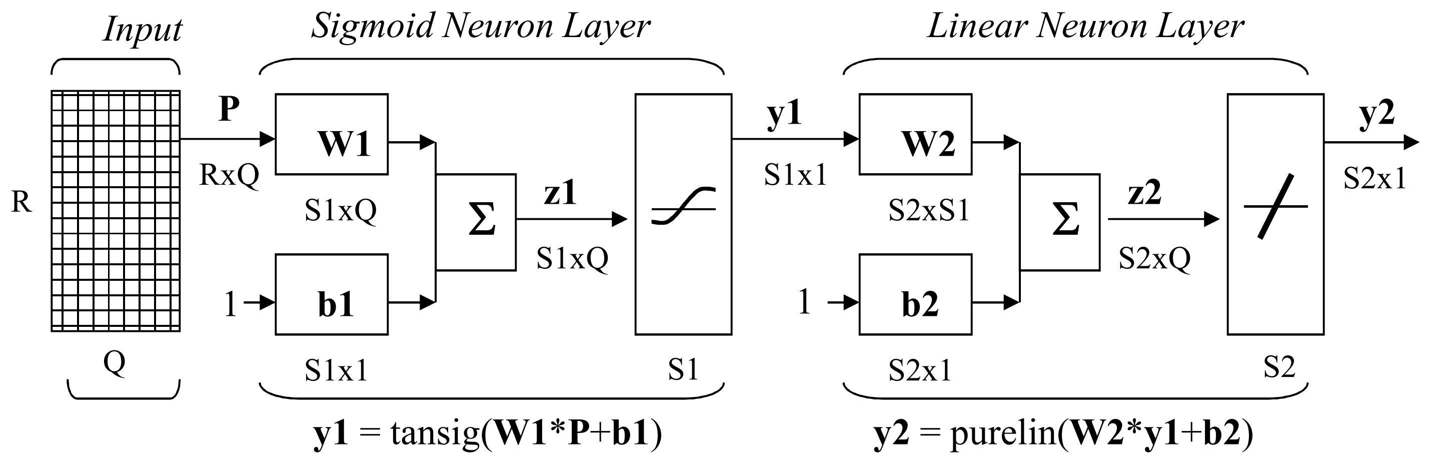

Input
p
Rx1
R
y
S_N^N x 1
$\Delta \mathbf{W}_j$ | j= N, N-1, …,1,0
e
$\mathbf{e} = (\mathbf{t} - \mathbf{y}_N)$
t

Phase *I* : The input vector is propagated forward (fed-forward) trough the consecutive layers of the ANN.

Phase *II* : The errors are recursively back-propagated trough the layers and appropriate weight changes are made. Because the output error is an indirect function of the weights in the hidden layers, we have to use the "chain rule" of calculus when calculating the derivatives with respect to the weights and biases in the hidden layers. These derivatives of the squared error are computed first at the last (output) layer and then propagated backward from layer to layer using the "chain rule."

EXAMPLE: **Function Approximation by Back-Propagation**

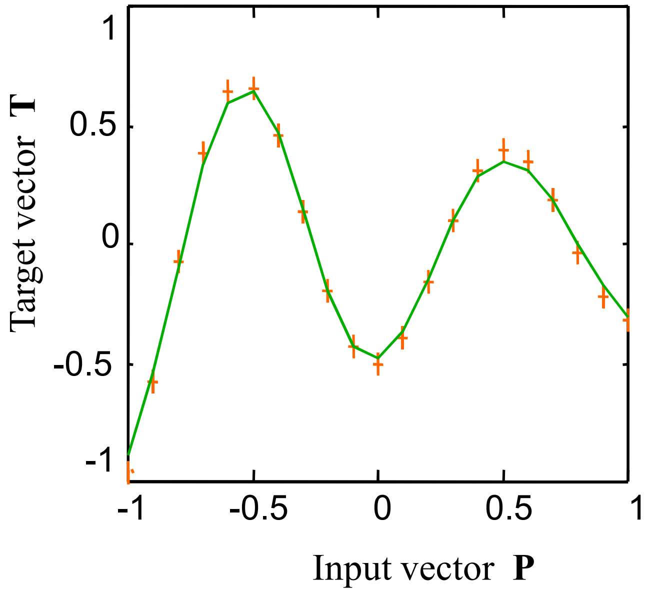
Demo BP4 in the" MATLAB Neural Network Toolbox User's Guide"



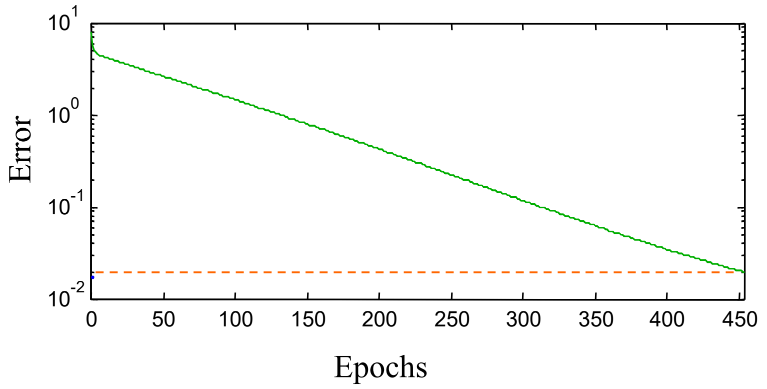
y1 = tansig(**W1*P**+**b1**)

y2 = purelin(**W2*y1**+**b2**)

R = 1 input
S1 = 5 neurons in layer #1
S2 = 1 neuron in layer #2
Q = 21 input vectors



The back-propagation algorithm took 454 epochs to approximate the 21 target vectors with an error < 0.02

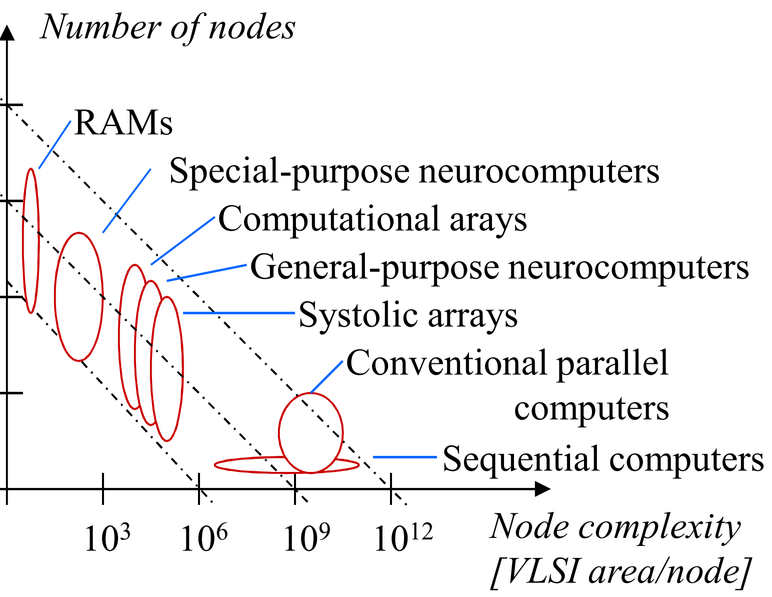


Hardware Neural Network Architectures

ANNs / Neurocomputers ==>architectures optimized for neuron model implementation

- *general-purpose*, able to emulate a wide range of NN models;
- *special-purpose*, dedicated to a specific NN model.

Hardware NNs consisting of a collection of simple neuron circuits provide the massive computational parallelism allowing for a higher modelling speed.



[from P. Treleaven, M. Pacheco, M. Vellasco, "VLSI Architectures for Neural Networks," IEEE Micro, Dec. 1989, pp. 8-27]

ANN VLSI Architectures:

- *analog* ==> compact,high speed, asynchronous, no quantization errors, convenient weight "+"and "X";
- digital ==> more efifcient VLSI technology, robust, convenient weight storage;

Pulse Data Representation:

- *Pulse Amplitude Modulation* (PAM) - not satisfactory for NN processing;
- Pulse Width Modulation (PWM);
- Pulse Frequency Modulation (PFM).

⇩

Pulse Stream ANNs: combination of different pulse data representation methods and opportunistic use of both analog and digital implementation techniques.

*Interactive VE applications require **real-time** rendering of **complex NN models***

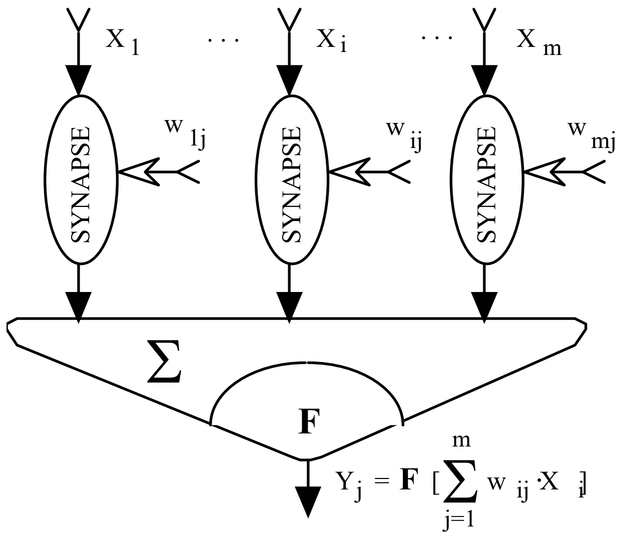
HARDWARE NEURAL NETWORK ARCHITECTURES USING RANDOM-PULSE DATA REPRESENTATION

- Looking for a model to prove that algebraic operations with analog variables can be performed by logical gates, **von Neuman** advanced in 1956 the idea of representing analog variables by the mean rate of random-pulse streams [*J. von Neuman, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in Automata Studies, (C.E. Shannon, Ed.), Princeton, NJ, Princeton University Press, 1956*].

- The "**random-pulse machine**" concept, [*S.T. Ribeiro, "Random-pulse machines," IEEE Trans. Electron. Comp., vol. EC-16, no. 3, pp. 261-276,1967*], a.k.a. "noise computer", "stochastic computing", "dithering" deals with analog variables represented by the mean rate of random-pulse streams allowing to use digital circuits to perform arithmetic operations. This concept presents a good tradeoff between the electronic circuit complexity and the computational accuracy. The resulting neural network architecture has a high packing density and is well suited for very large scale integration (VLSI).

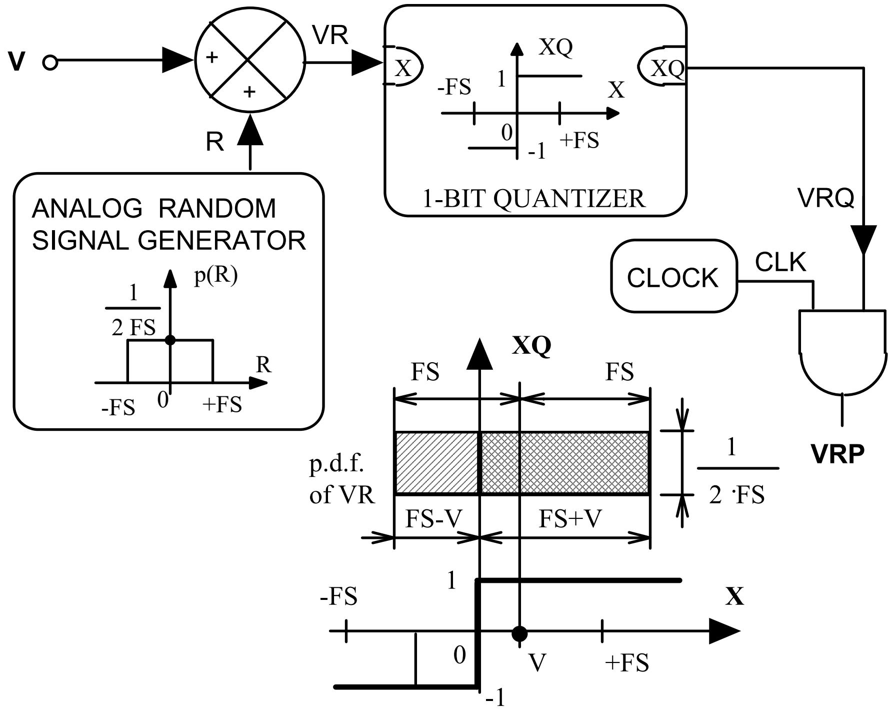
❖ HARDWARE *ANN* USING RANDOM-PULSE DATA REPRESENTATION

[E.M. Petriu, K. Watanabe, T. Yeap, "Applications of Random-Pulse Machine Concept to Neural Network Design," IEEE Trans. Instrum. Meas., Vol. 45, No.2, pp.665-669, 1996.]



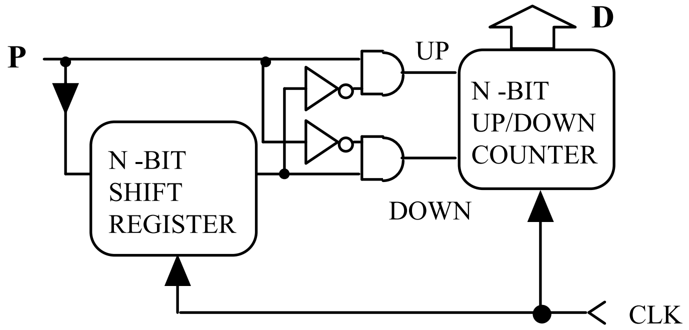
$$Y_j = \mathbf{F}\left[\sum_{j=1}^{m} w_{ij} \cdot X_i\right]$$

Neuron Structure

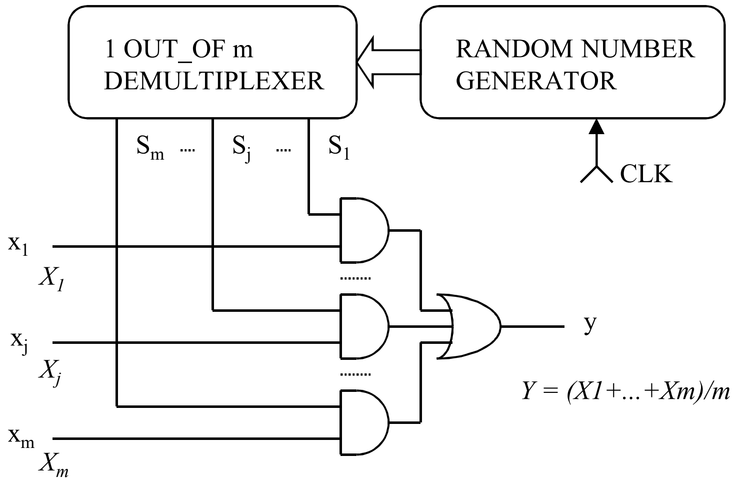


One-Bit "Analog / Random Pulse" Converter

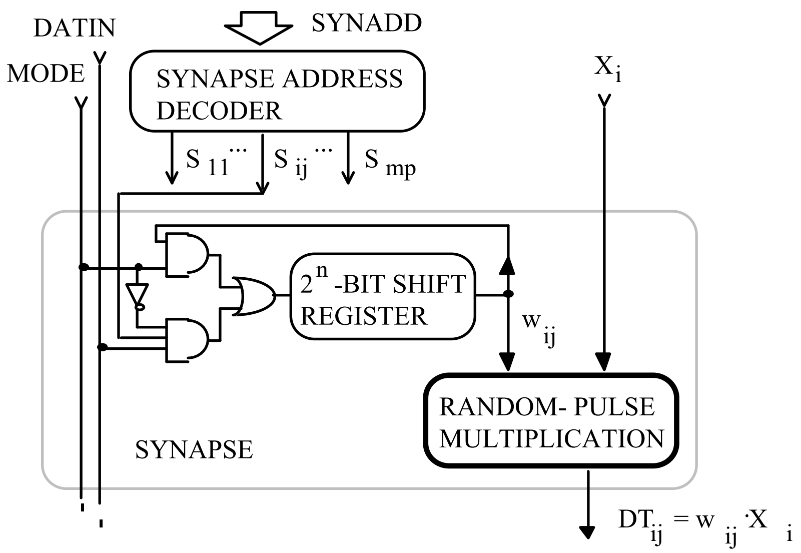
>>> Random-Pulse Hardware ANN



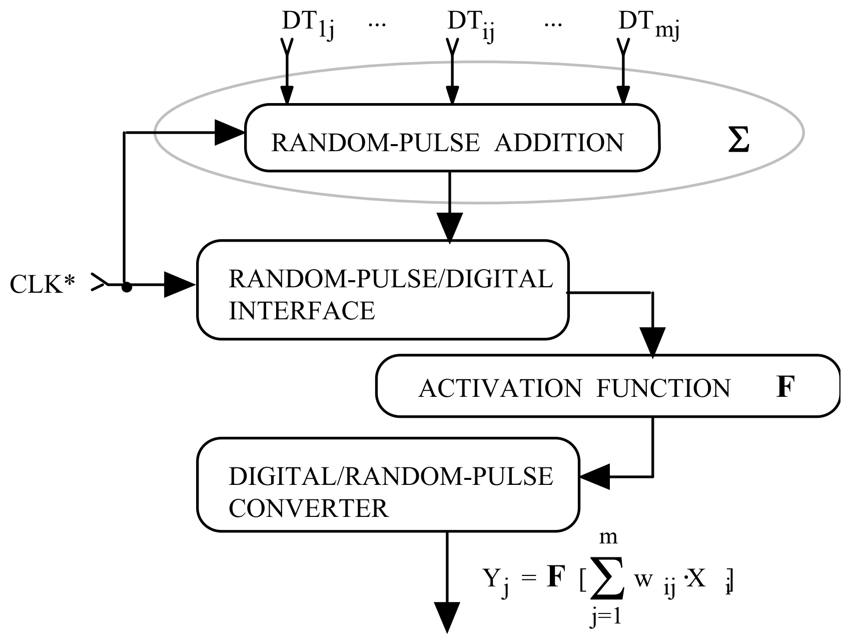
"Random Pulse / Digital" Converter using a Moving Average Algorithm



Random Pulse Addition

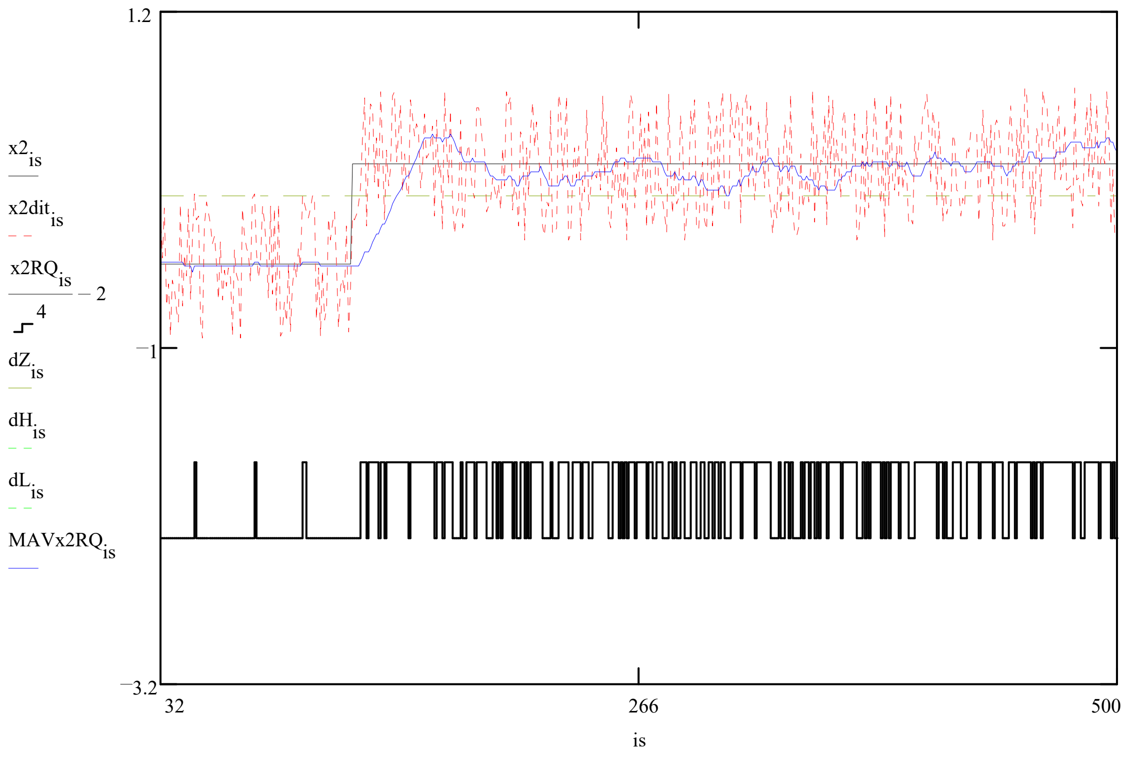

DATIN
MODE
SYNADD
SYNAPSE ADDRESS DECODER
S_{11} ... S_{ij} ... S_{mp}
X_i
2^n-BIT SHIFT REGISTER
w_{ij}
RANDOM- PULSE MULTIPLICATION
SYNAPSE
$DT_{ij} = w_{ij} \cdot X_i$

Random Pulse Implementation of a Synapse

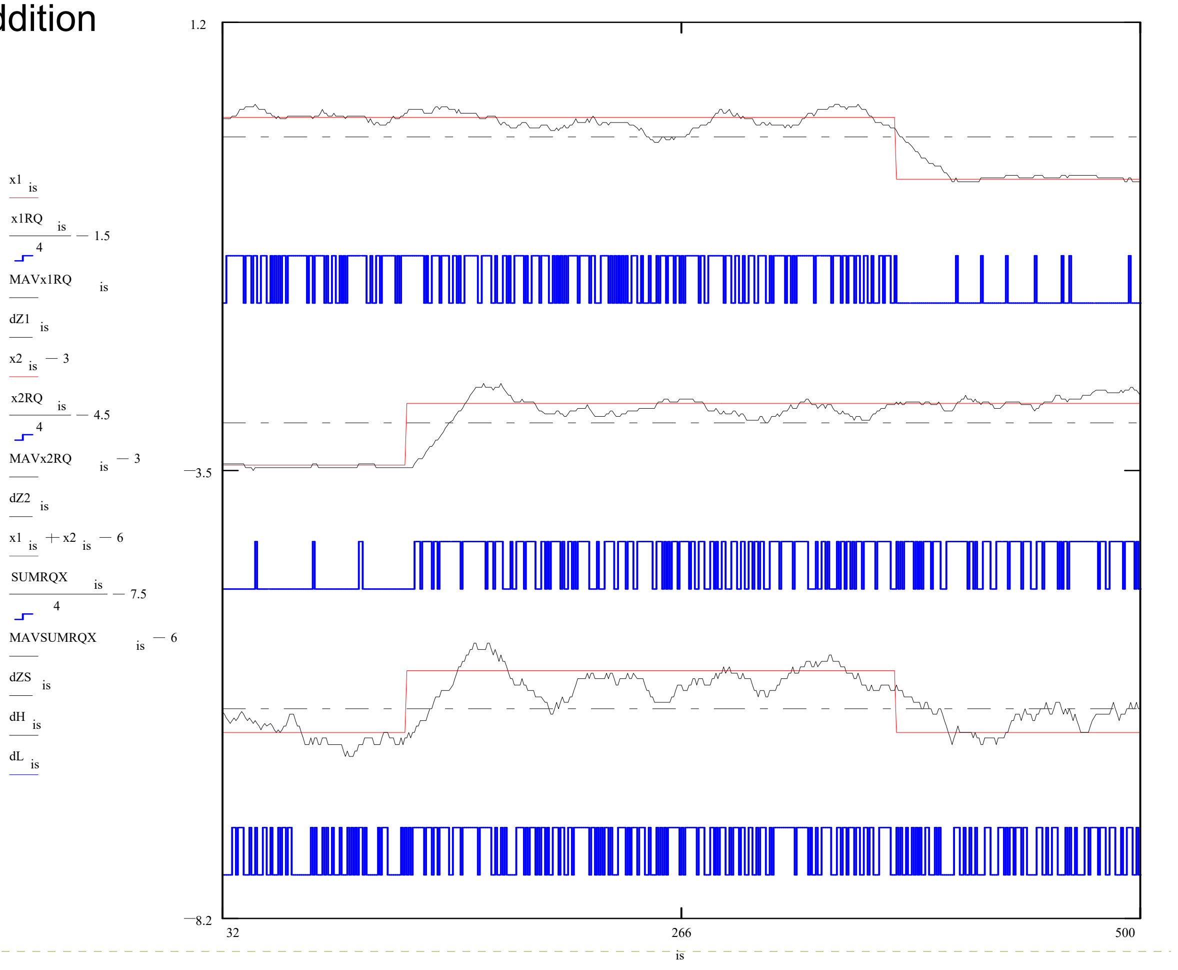

DT_{1j} ... DT_{ij} ... DT_{mj}
RANDOM-PULSE ADDITION
Σ
CLK*
RANDOM-PULSE/DIGITAL INTERFACE
ACTIVATION FUNCTION F
DIGITAL/RANDOM-PULSE CONVERTER
$Y_j = \mathbf{F}\left[\sum_{j=1}^{m} w_{ij} \cdot X_i\right]$

Neuron Body Structure

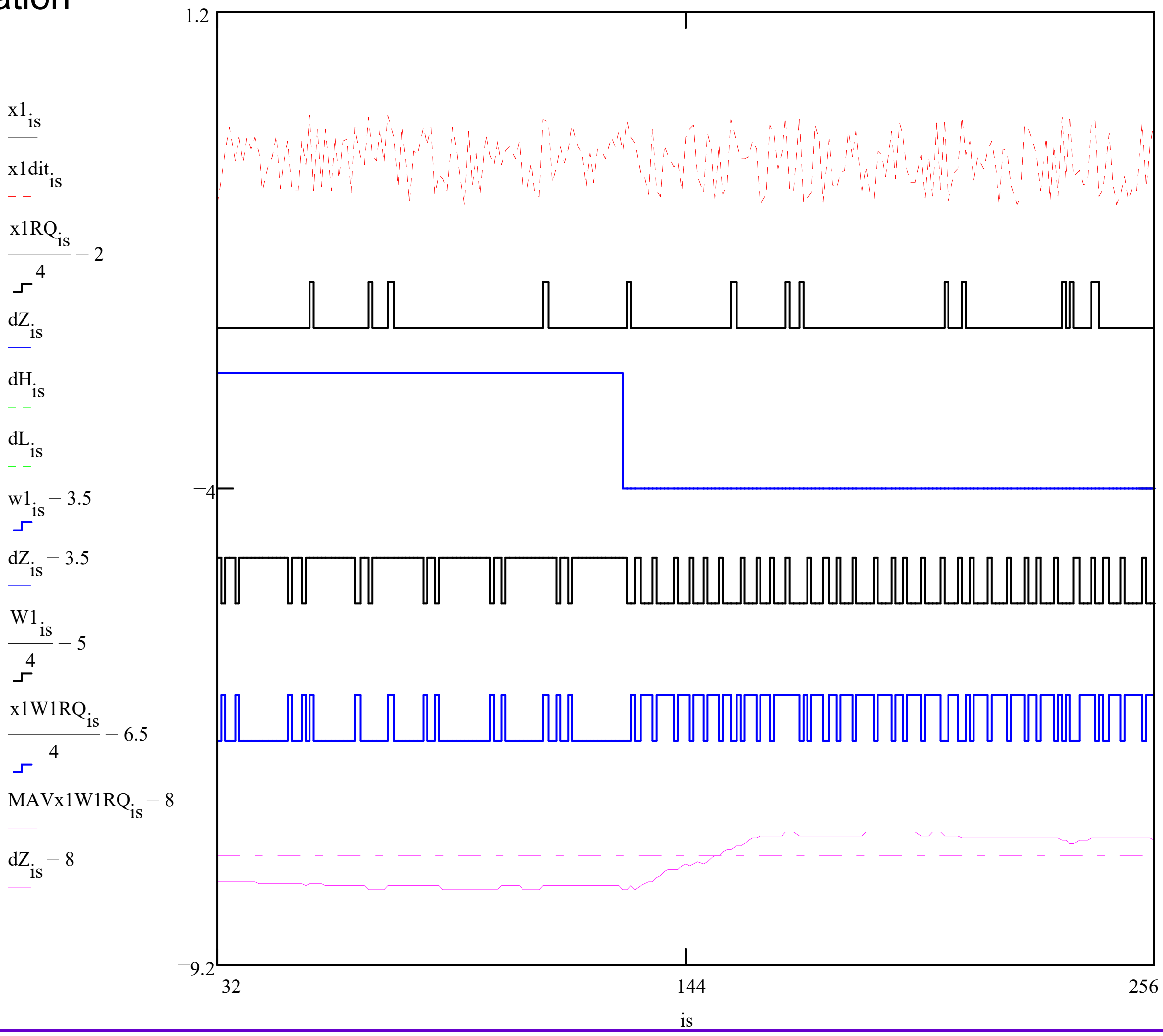
Moving Average 'Random Pulse -to- Digital " Conversion



Random Pulse Addition

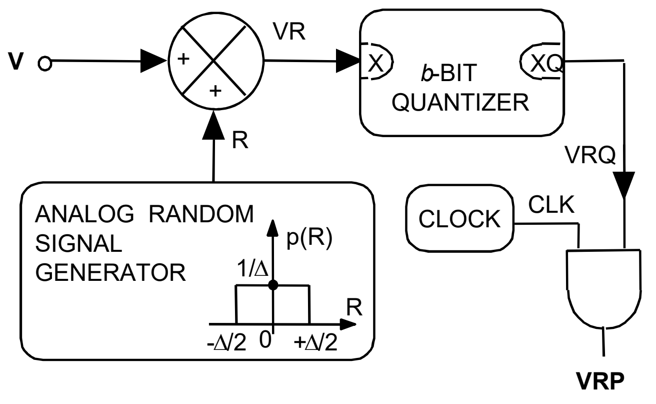


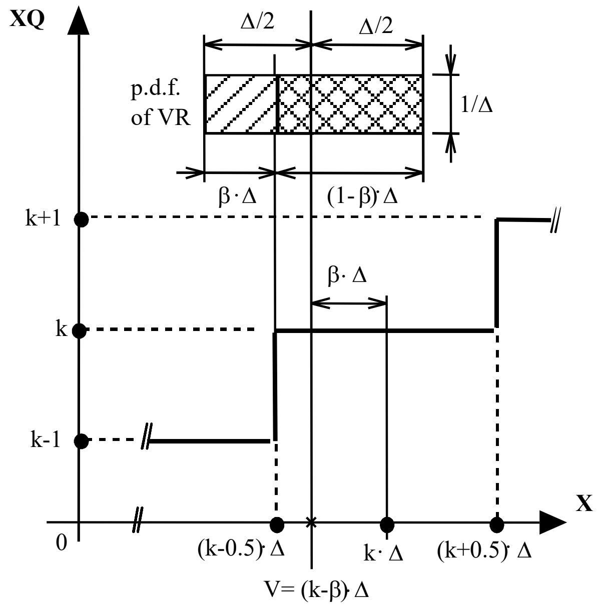
Random Pulse Multiplication



❖ HARDWARE *ANN* USING MULTI-BIT RANDOM-DATA REPRESENTATION

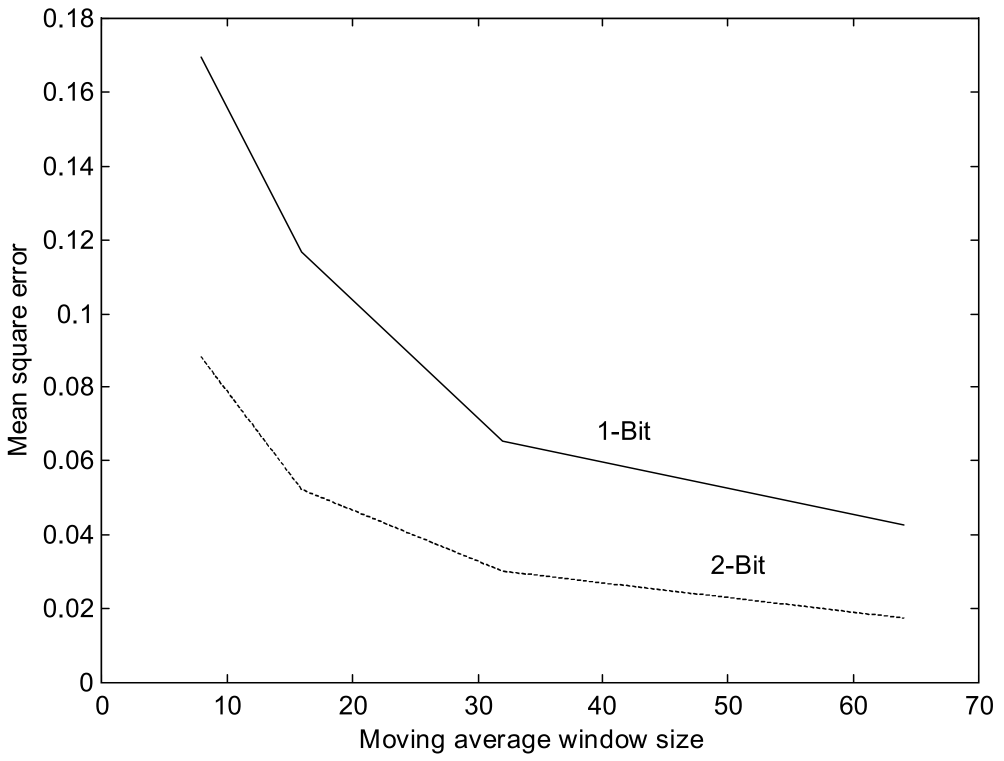
[E.M. Petriu, L. Zhao, S.R. Das, and A. Cornell, "Instrumentation Applications of Random-Data Representation," *Proc. IMTC/2000, IEEE Instrum. Meas. Technol. Conf.,* pp. 872-877, Baltimore, MD, May 2000]
[L. Zhao, "Random Pulse Artificial Neural Network Architecture," *M.A.Sc. Thesis, University of Ottawa,* 1998]



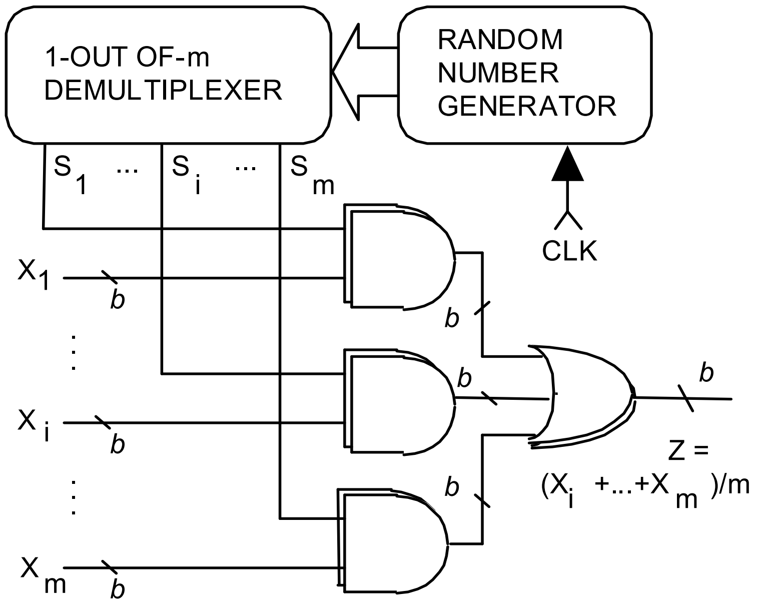


Generalized *b*-bit analog/random-data conversion and its quantization characteristics

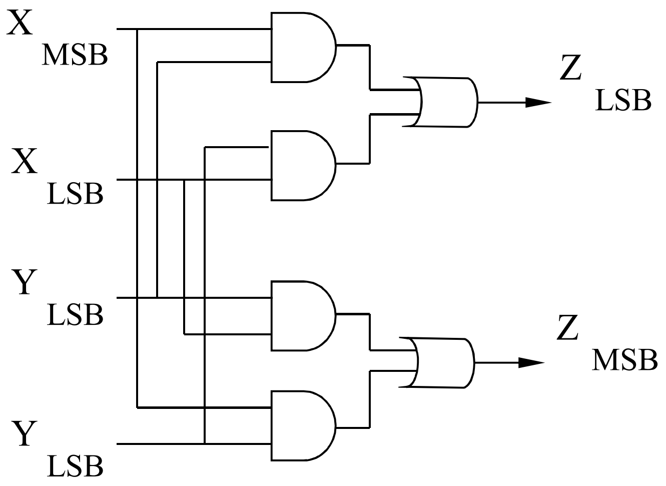
Quantization levels	Relative mean square error
2	72.23
3	5.75
4	2.75
...	...
8	1.23
...	...
analog	1



Mean square errors function of the moving average window size

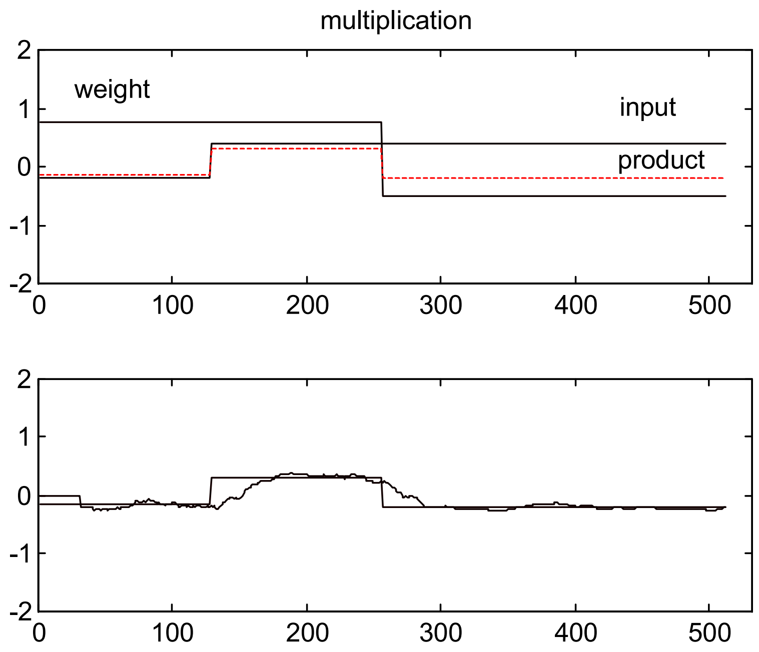


Stochastic adder for random-data.

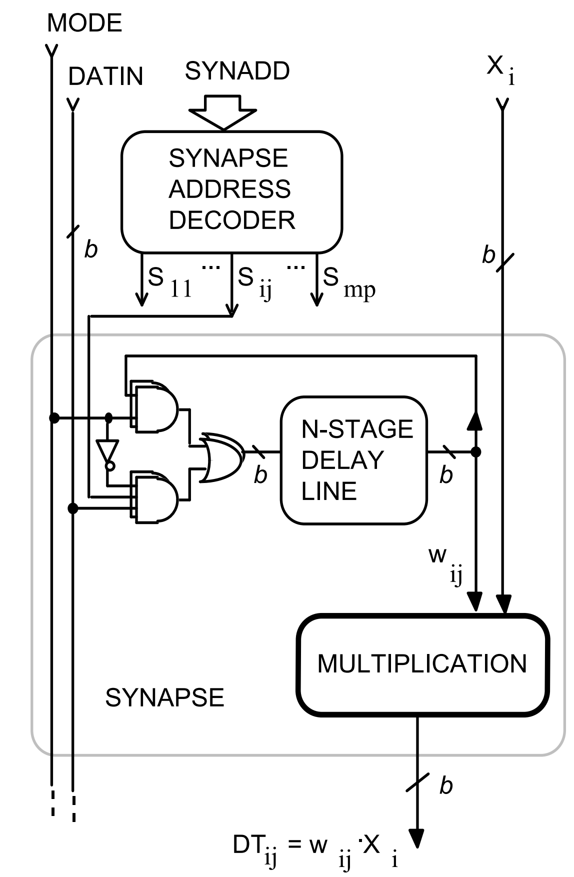


X \ Y		0 *00*	1 *01*	-1 *10*
0	*00*	**0** *00*	**0** *00*	**0** *00*
1	*01*	**0** *00*	**1** *01*	**-1** *10*
-1	*10*	**0** *00*	**-1** *10*	**1** *01*

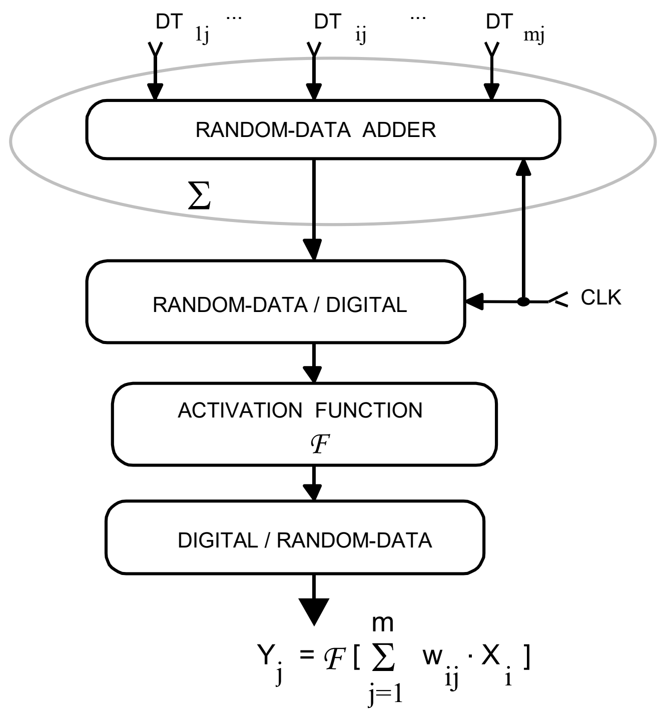
2-bit random-data multiplier.



Example of 2-bit random-data multiplication.

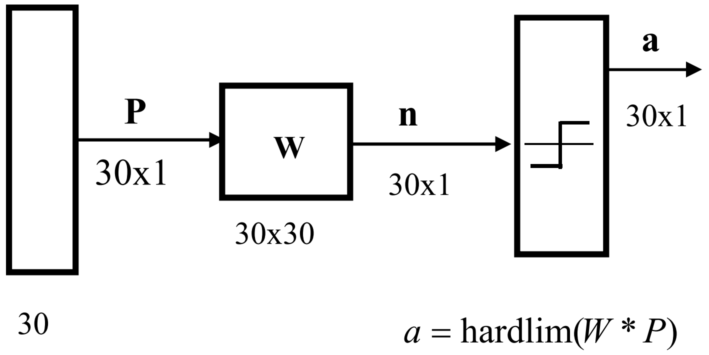


Multi-bit random-data implementation of a synapse

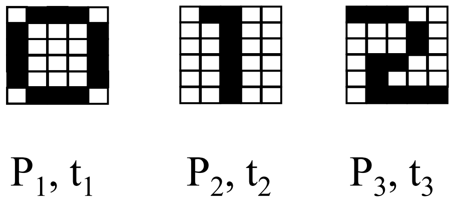


$$Y_j = \mathcal{F}\left[\sum_{j=1}^{m} w_{ij} \cdot X_i\right]$$

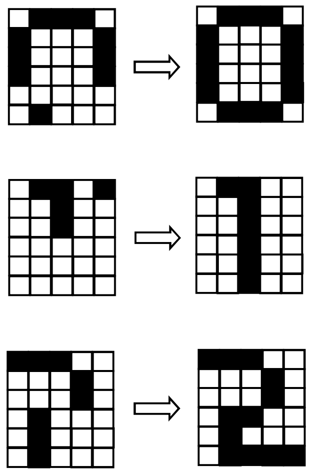
Multi-bit random-data implementation of a neuron body.



Auto-associative memory NN architecture



Training set



Recovery of 30% occluded patterns