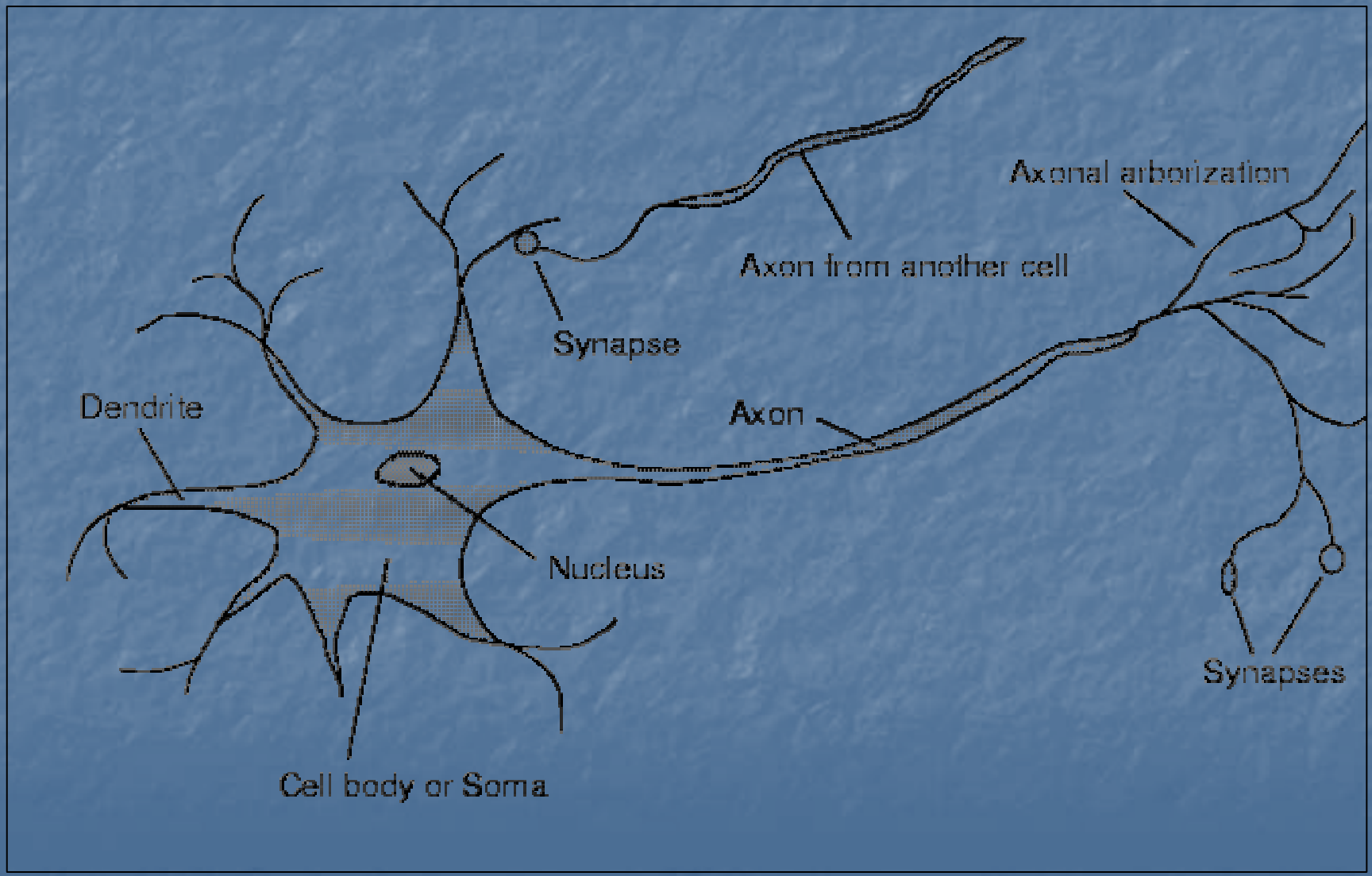
Neuron Model and Network Architectures

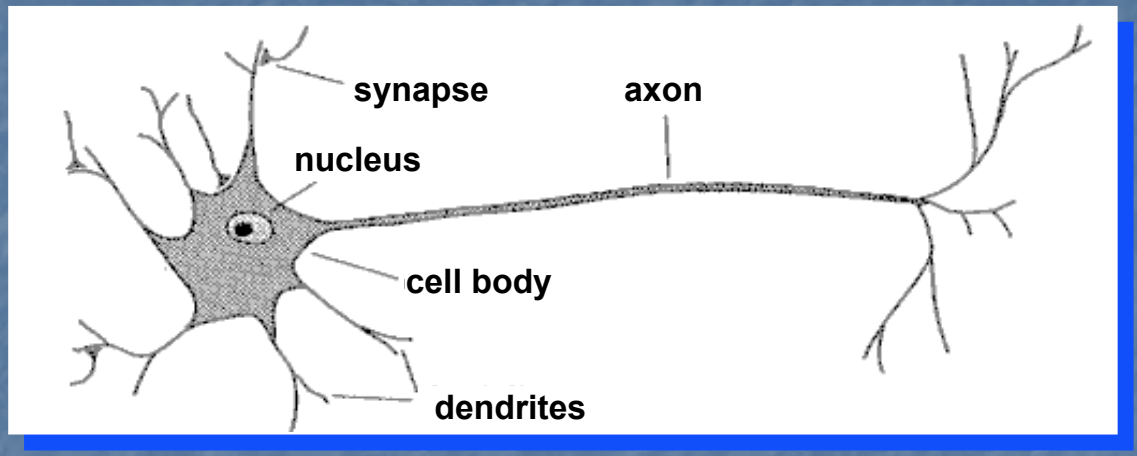
Biological inspirations

- Some numbers:
  - The human brain contains about 10 billion nerve cells (neurons)
  - Each neuron is connected to the others through 10000 synapses
- Properties of the brain:
  - It can learn, reorganize itself from experience
  - It adapts to the environment
  - It is robust and fault tolerant

Human Nerve Cell



Biological neuron



- A neuron has
  - A branching input (dendrites)
  - A branching output (the axon)
- The information circulates from the dendrites to the axon via the cell body
  - Axon connects to dendrites via synapses
    - Synapses vary in strength
    - Synapses may be excitatory or inhibitory

Connectionist Models

Consider humans

- Neuron switching time ~.001 second
- Number of neurons ~10^{10}
- Connections per neuron ~10^{4-5}
- Scene recognition time ~.1 second
- 100 inference step does not seem like enough

must use lots of parallel computation!

Properties of artificial neural nets (ANNs)

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

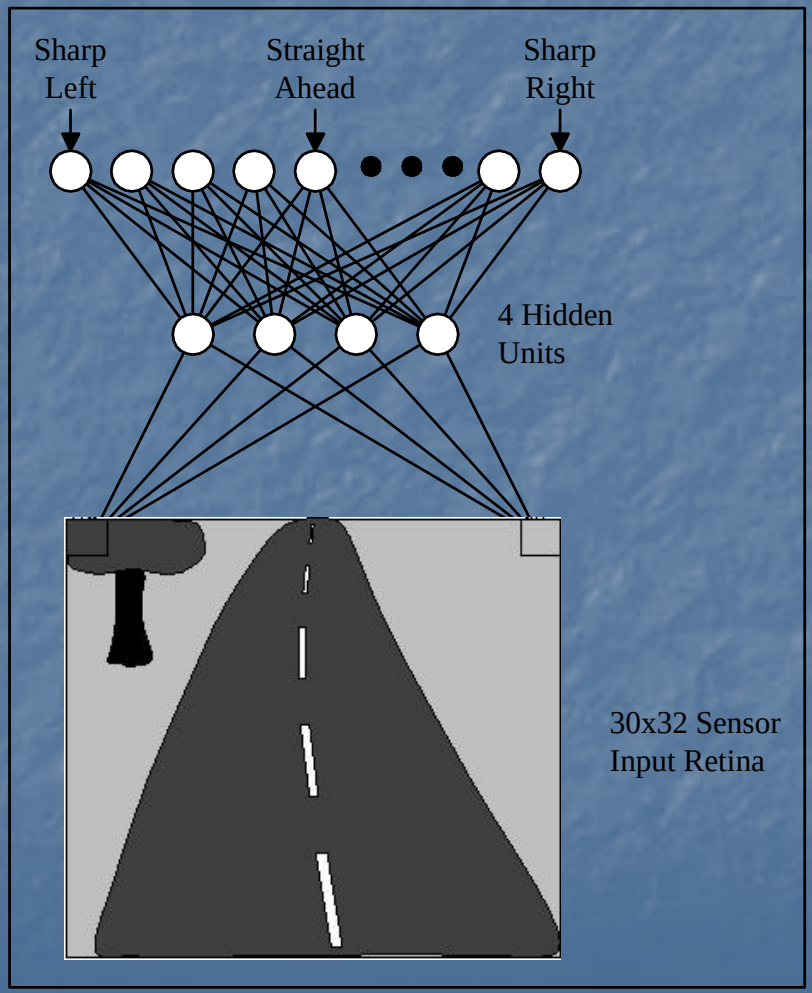
When to Consider Neural Networks

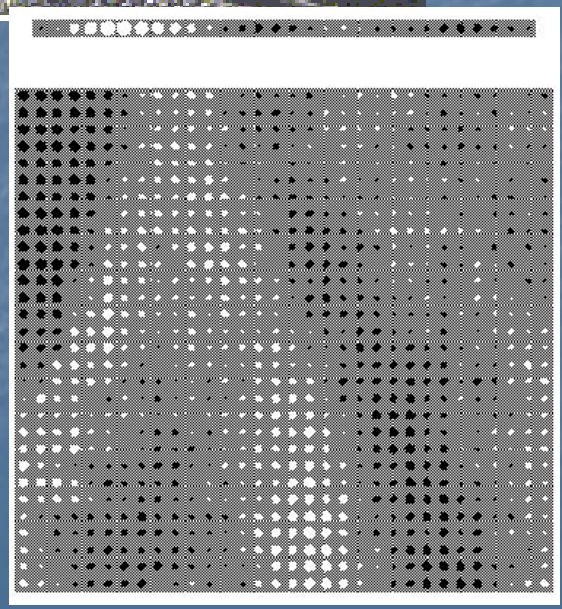
- Input is high-dimensional discrete or real-valued (e.g., raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is *unimportant*

Examples

- Speech phoneme recognition
- Image classification
- Financial prediction

ALVINN drives 70 mph on highways

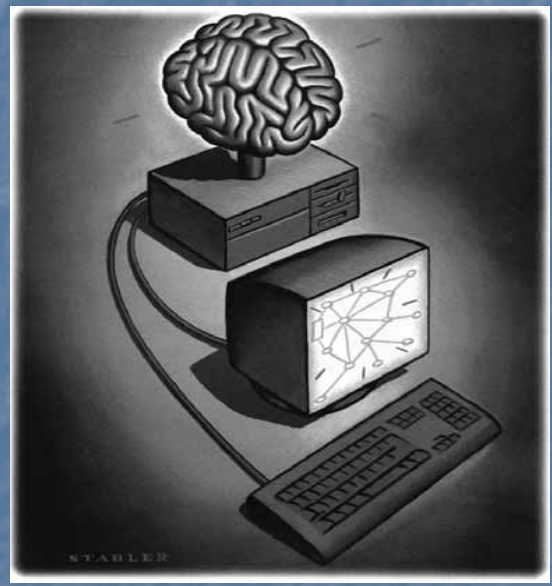




Neural Networks

What is a Neural Network?

- Biologically motivated approach to machine learning

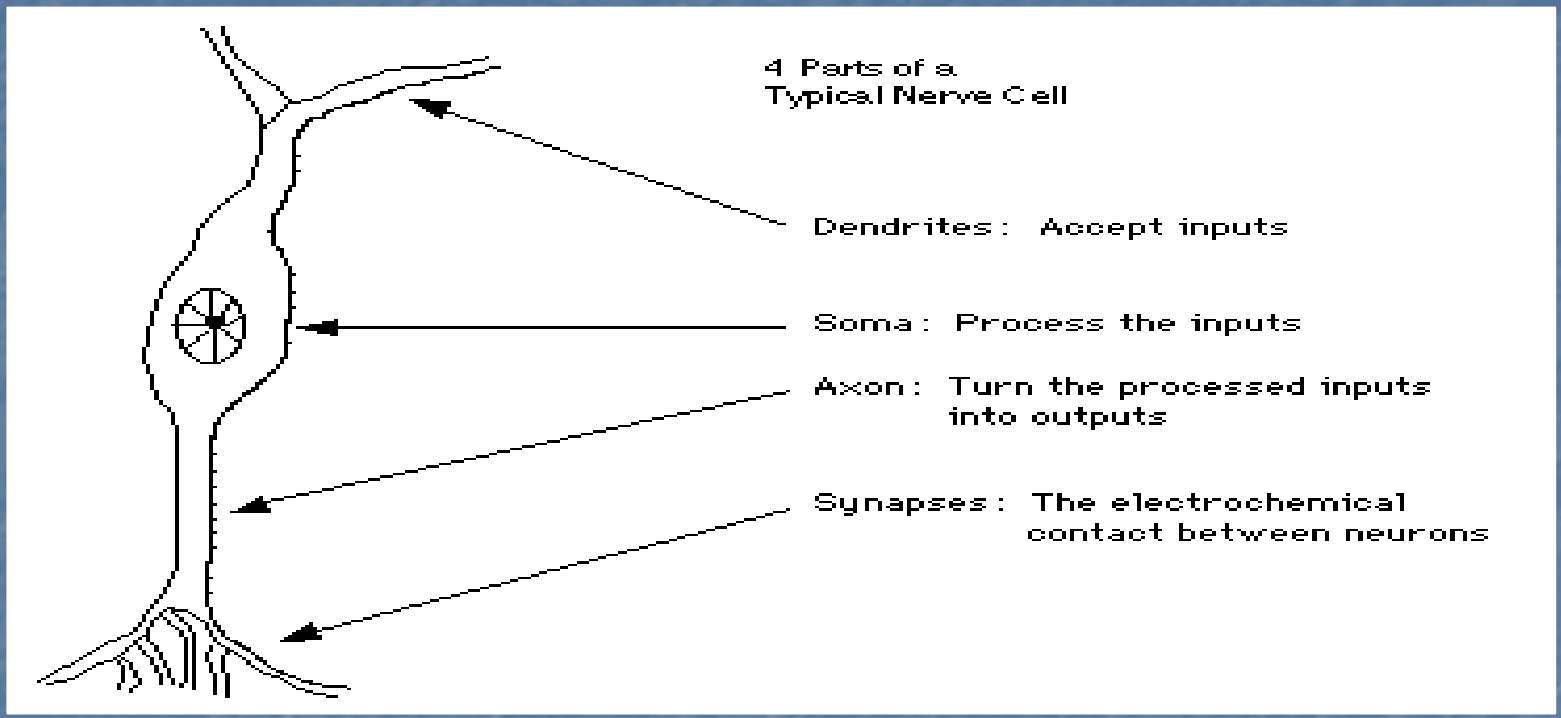


Similarity with biological network

Fundamental processing elements of a neural network is a neuron:

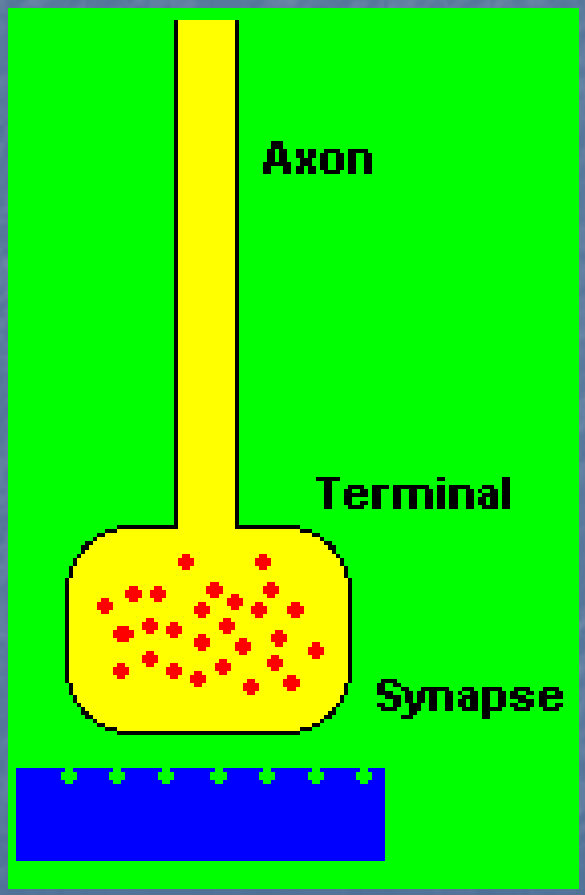
1. Receives inputs from other source
2. Combines them in someway
3. Performs a generally nonlinear operation on the result
4. Outputs the final result

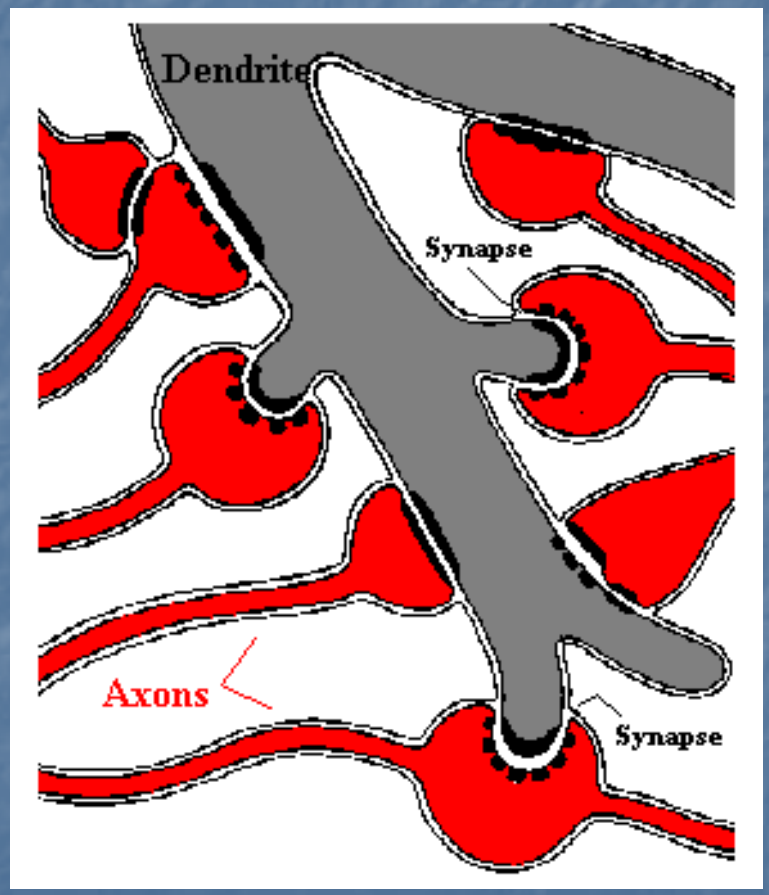
Similarity with Biological Network



- Fundamental processing element of a neural network is a neuron
- A human brain has 100 billion neurons
- An ant brain has 250,000 neurons

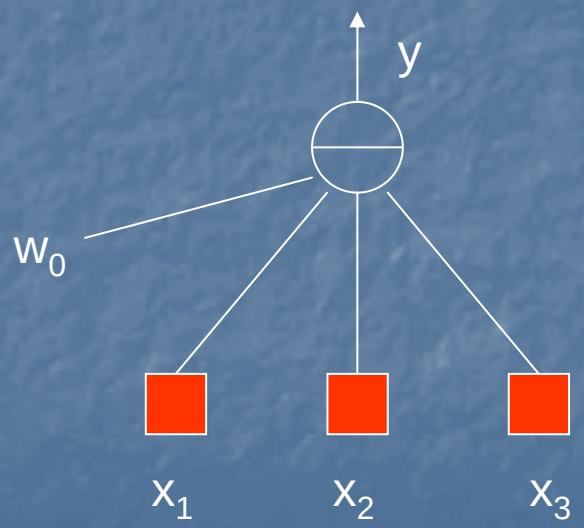
Synapses, the basis of learning and memory





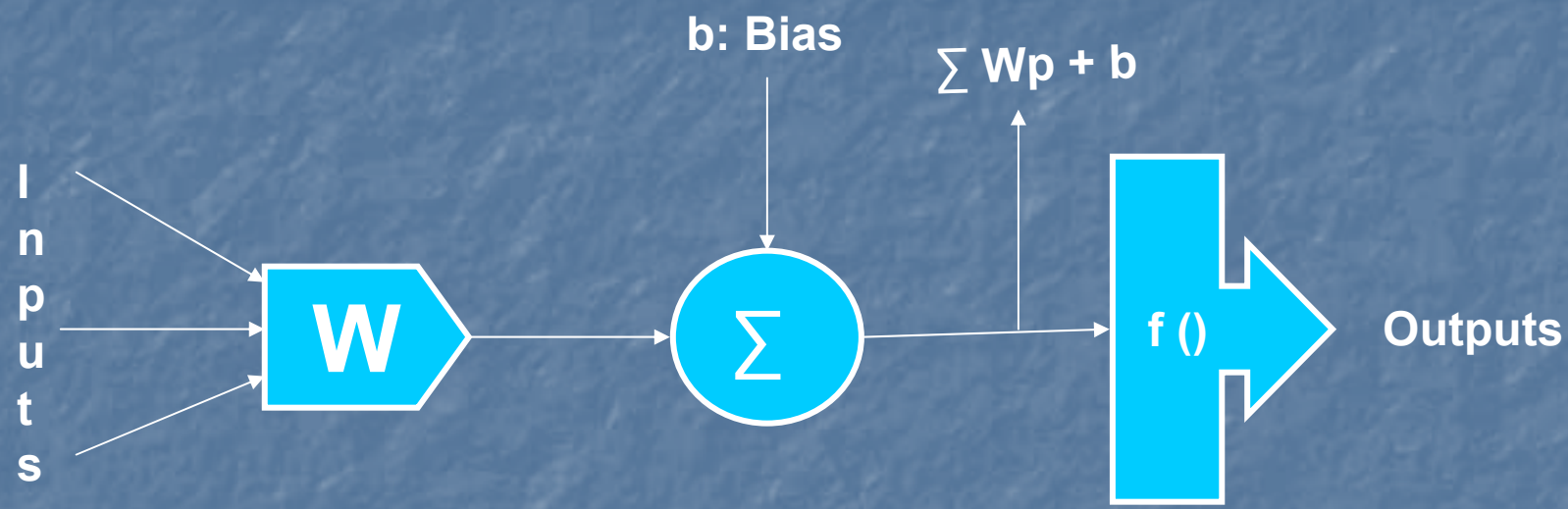
What is an artificial neuron ?

- Definition: Nonlinear, parameterized function with restricted output range



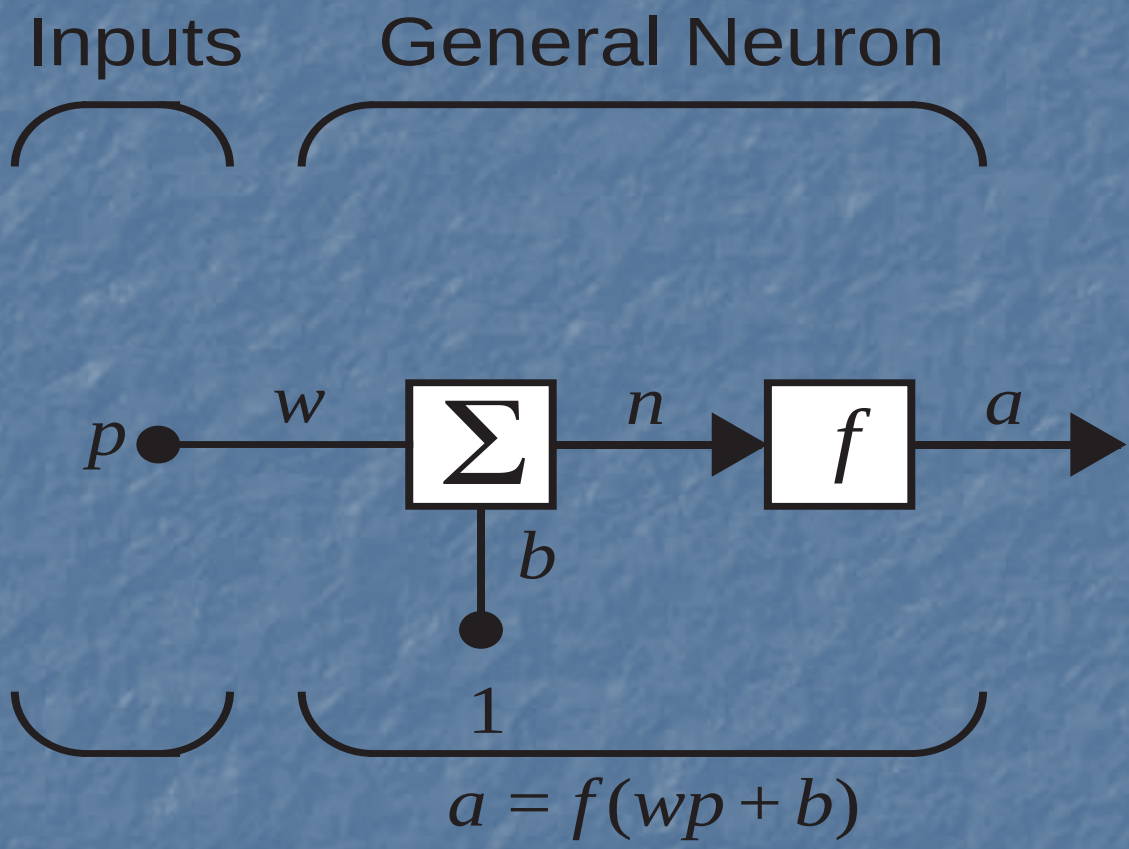
$$y = f\left(w_0 + \sum_{i=1}^{n-1} w_i x_i \right)$$

Basic Neural Network



Inputs – normally a vector of measured parameters
Bias – may/may not be added
f() – transfer or activation function

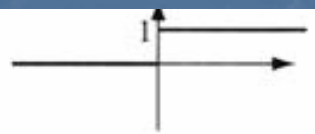
$$\text{Outputs} = \mathbf{f}\left(\sum W^{T} p + b\right)$$


Inputs
General Neuron
p
w
Σ
n
f
a
b
1
a = f(wp + b)

Activation Functions

Hard Limit

$$y = \begin{cases} 1 & if\ net \geq 0 \\ 0 & if\ net < 0 \end{cases}$$

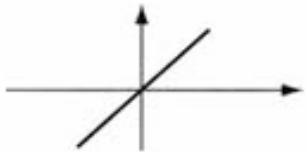


Symmetrical Hard Limit

$$y = \begin{cases} 1 & if\ net \geq 0 \\ -1 & if\ net < 0 \end{cases}$$

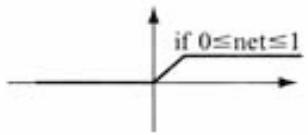
Linear

$$Y = net$$



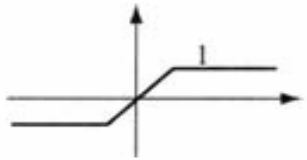
Satauratıng Linear

$$y = \begin{cases} 1 & if\ net > 1 \\ net & \\ 0 & if\ net < 0 \end{cases}$$



Symmetric Saturating LInear

$$y = \begin{cases} 1 & if\ net > 1 \\ net & if\ -1 \leq net \leq 1 \\ -1 & if\ net < -1 \end{cases}$$

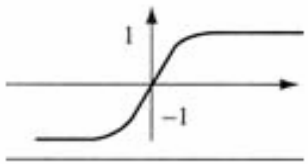


Log-Sigmoid

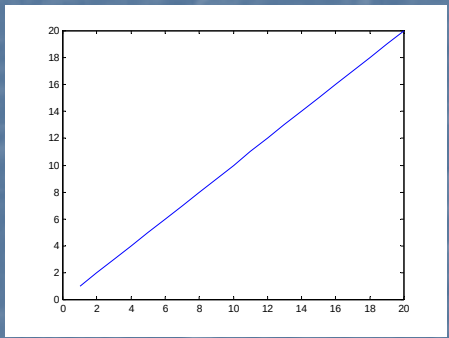
$$y = 1/(1 + e^{-ner})$$

Hyperbolic Tangent Sigmoid

$$y = (e^{net} - e^{-net})/(e^{net} + e^{-net})$$

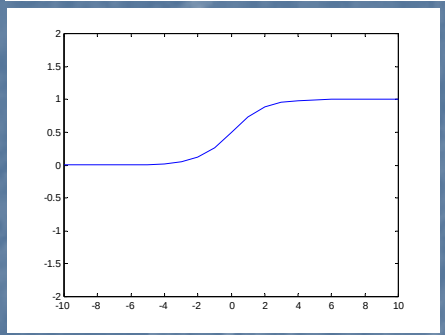


Activation functions



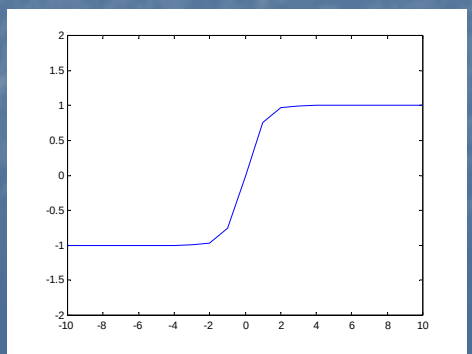
Linear

$$y = x$$



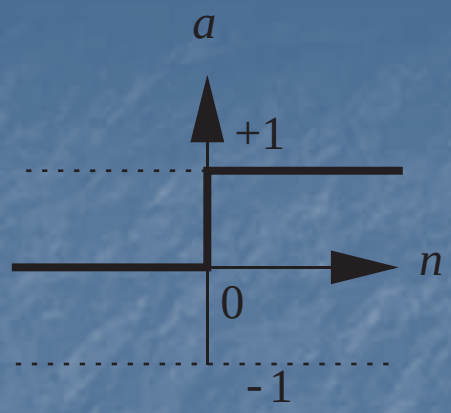
Logistic

$$y = \frac{1}{1+\exp(-x)}$$



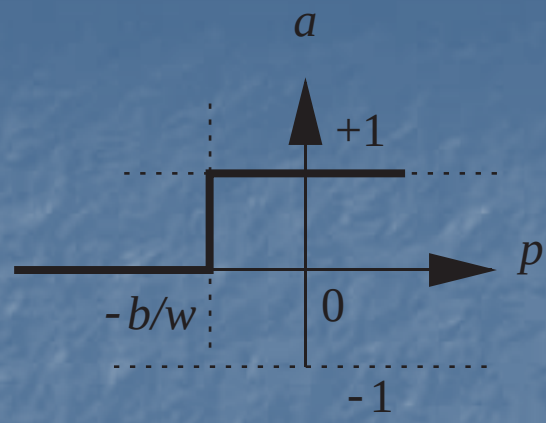
Hyperbolic tangent

$$y = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



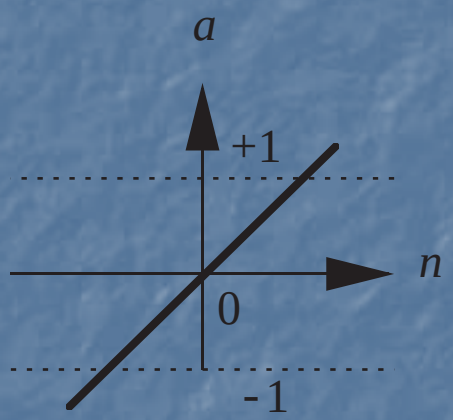
$a = hardlim(n)$

Hard Limit Transfer Function



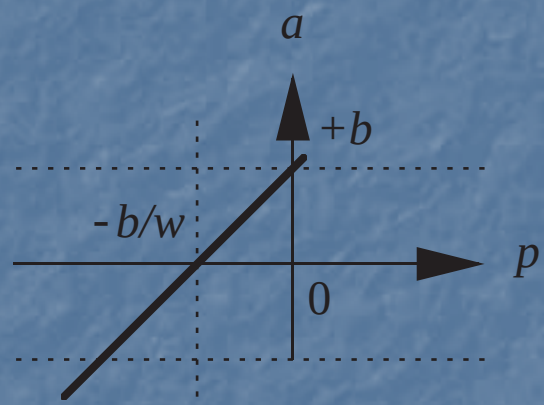
$a = hardlim(wp + b)$

Single-Input *hardlim* Neuron



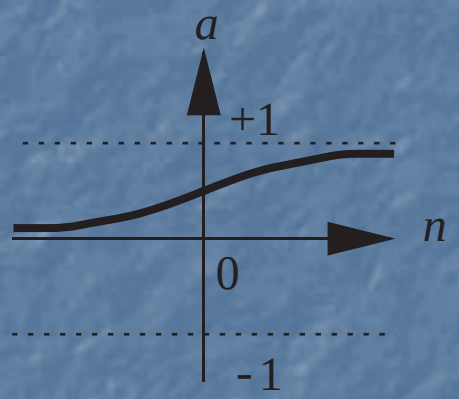
$a = purelin(n)$

Linear Transfer Function



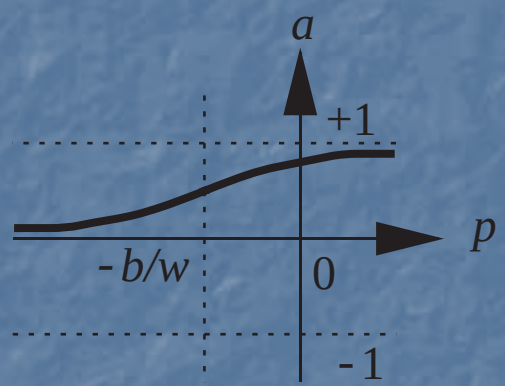
$a = purelin(wp + b)$

Single-Input *purelin* Neuron



$a = logsig(n)$

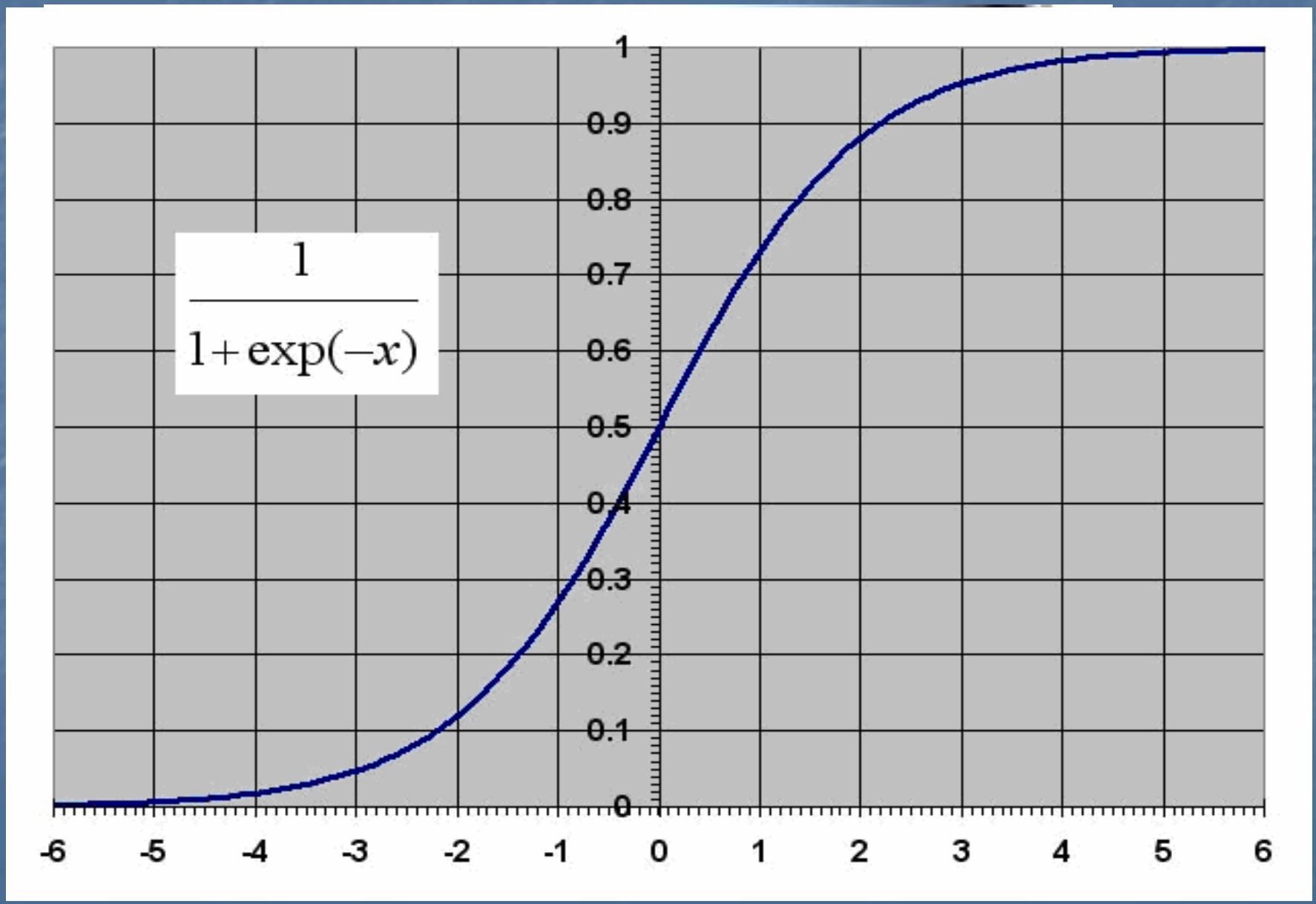
Log-Sigmoid Transfer Function



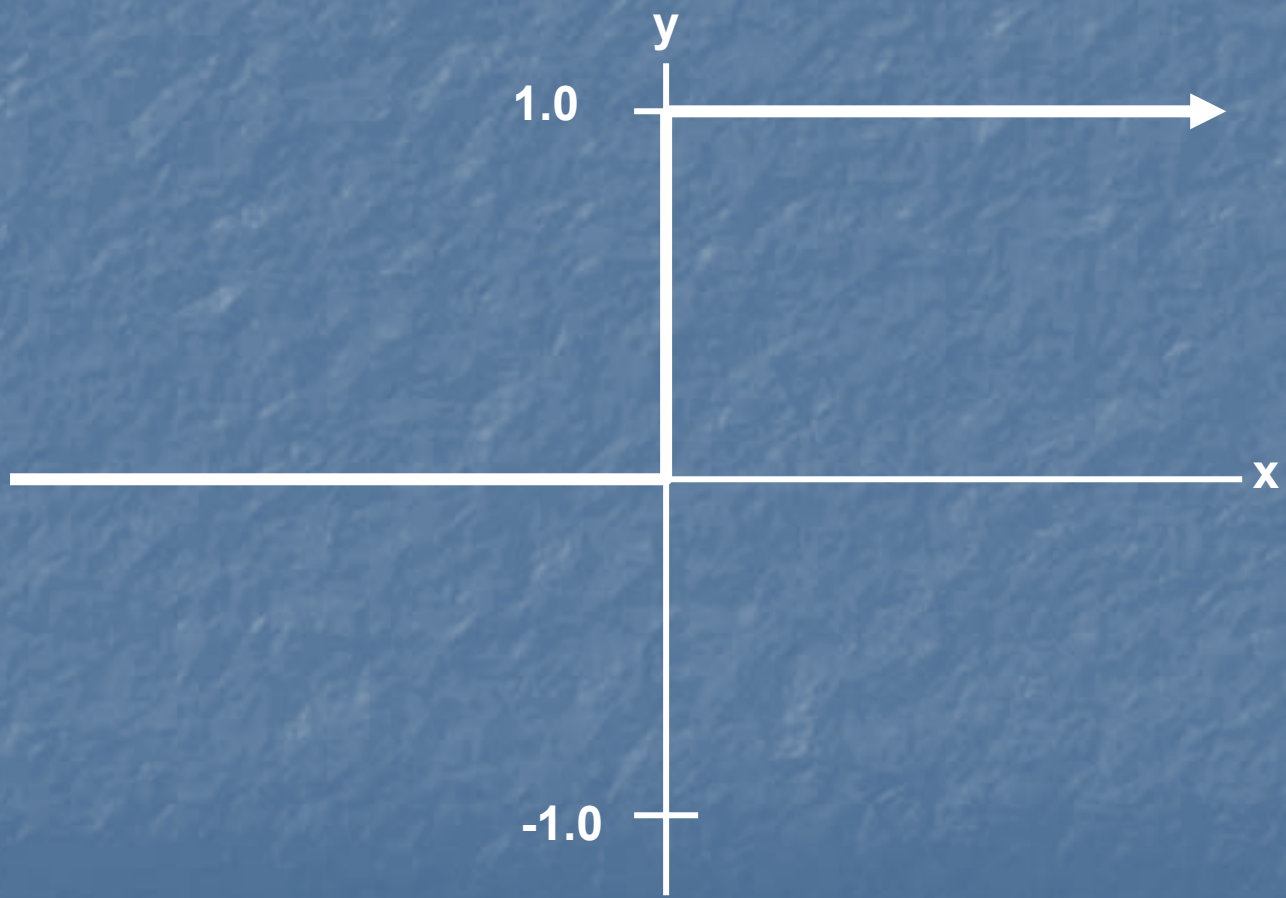
$a = logsig(wp + b)$

Single-Input *logsig* Neuron

Log Sigmoidal Function



Hard Limit Function



Neural Networks

- A mathematical model to solve engineering problems
  - Group of highly connected neurons to realize compositions of non linear functions
- Tasks
  - Classification
  - Discrimination
  - Estimation
- 2 types of networks
  - Feed forward Neural Networks
  - Recurrent Neural Networks

Learning

- The procedure that consists in estimating the parameters of neurons so that the whole network can perform a specific task

- Two general types of learning
  - The supervised learning
  - The unsupervised learning

- The Learning process (supervised):
  - Present the network a number of inputs and their corresponding outputs
  - See how closely the actual outputs match the desired ones
  - Modify the parameters to better approximate the desired outputs

Supervised learning

- The desired response of the neural network in function of particular inputs is well known.
- A “Professor” may provide examples and teach the neural network how to fulfill a certain task

Unsupervised learning

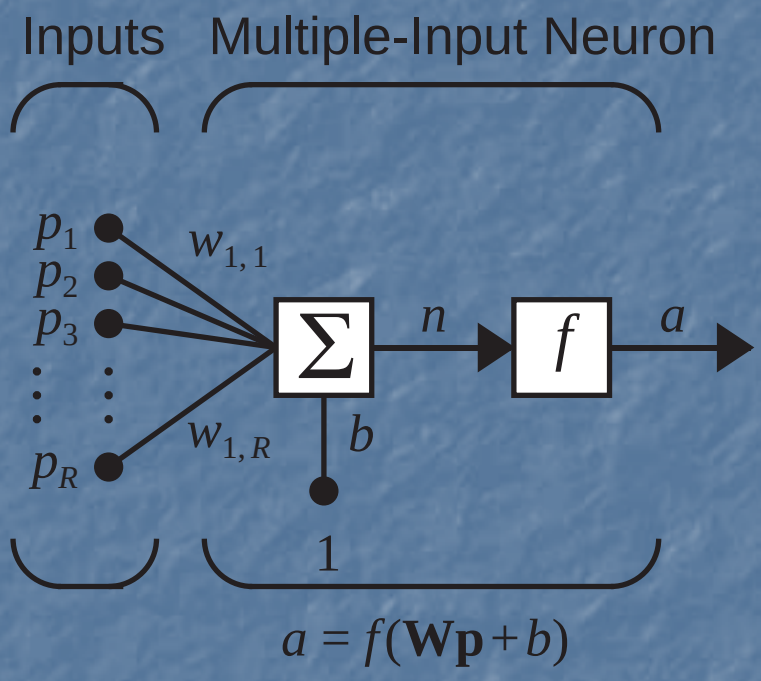
- Idea: group typical input data in function of resemblance criteria un-known a priori
- Data clustering
- No need of a professor:
  - The network finds itself the correlations between the data
  - Examples of such networks:
    - Kohonen feature maps

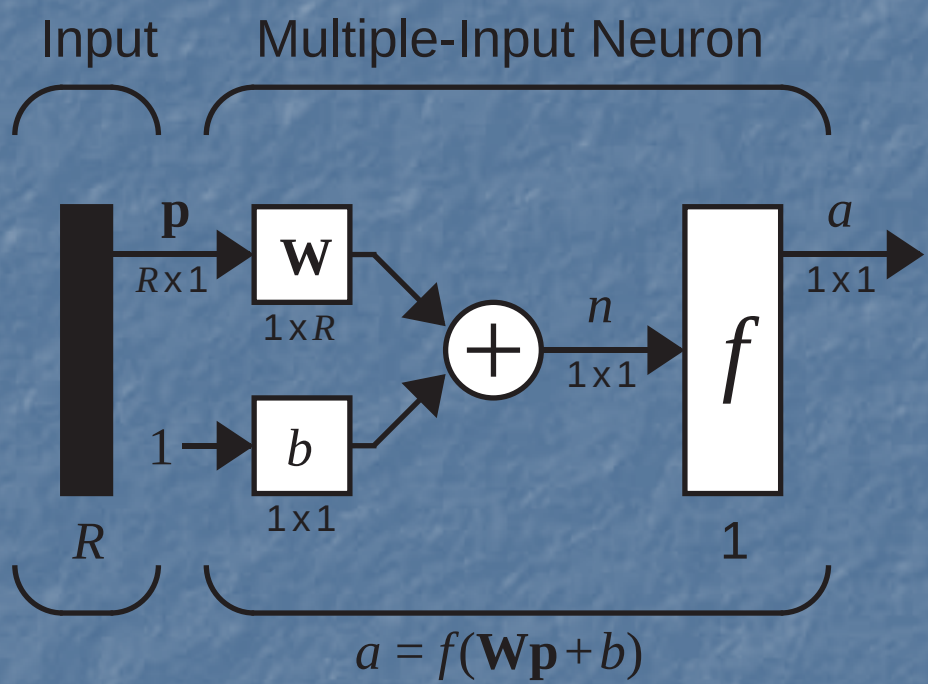
What do we need to use NN ?

(1) Determination of pertinent inputs
(2) Collection of data for the learning and testing phase of the neural network
(3) Finding the optimum number of hidden nodes
(4) Estimate the parameters (Learning)
(5) Evaluate the performances of the network
(6) IF performances are not satisfactory then review all the precedent points

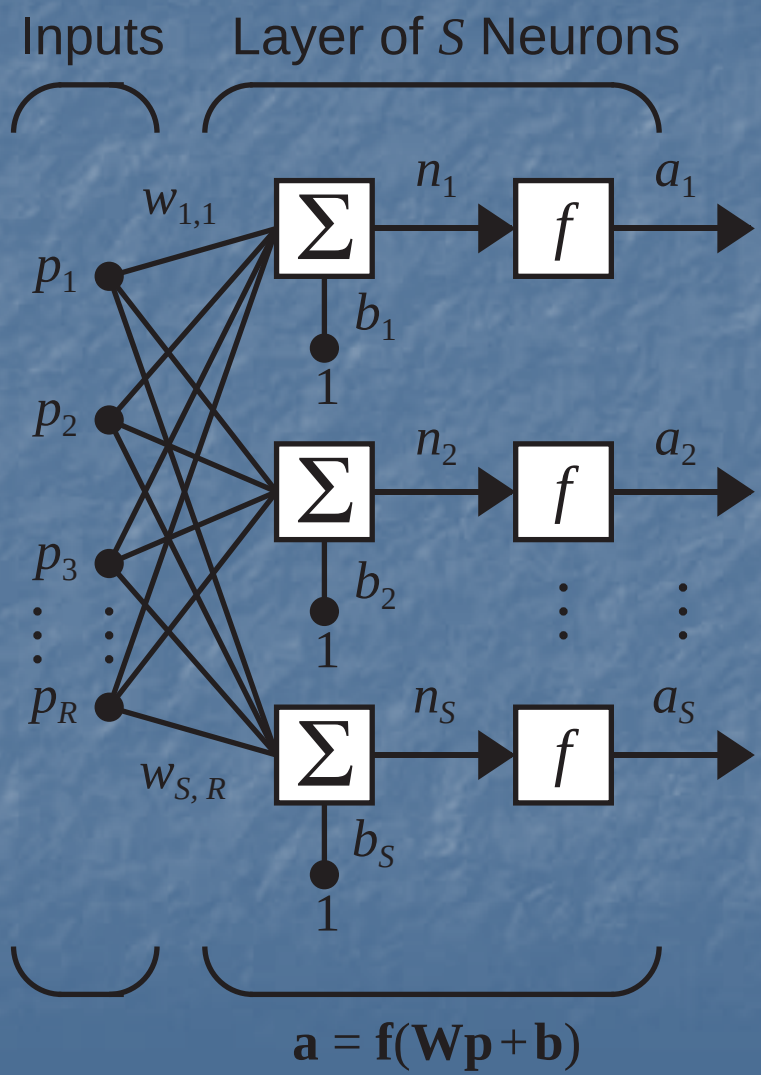
Classical neural architectures

- Perceptron
- Multi-Layer Perceptron
- Radial Basis Function (RBF)
- Kohonen Features maps
- Other architectures
  - An example: Shared weights neural networks

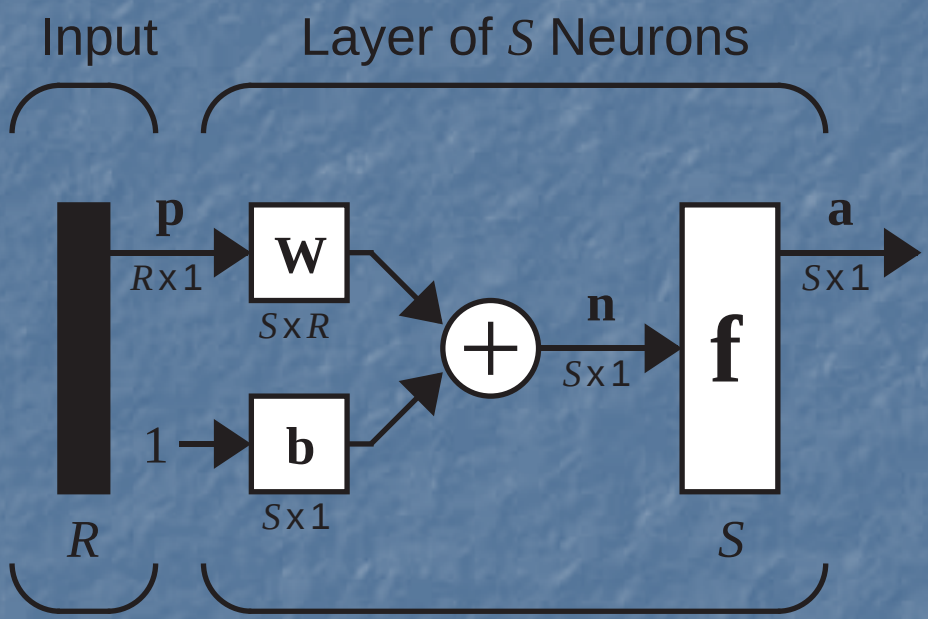

Inputs
Multiple-Input Neuron
p_1
p_2
p_3
p_R
$w_{1,1}$
$w_{1,R}$
Σ
n
f
a
b
1
$a = f(\mathbf{W}\mathbf{p}+b)$


Input
Multiple-Input Neuron
$\mathbf{p}$
Rx1
$\mathbf{w}$
1xR
1
b
1x1
R
n
1x1
f
1
a
1x1
$a = f(\mathbf{W}\mathbf{p}+b)$

Layer of Neurons



Abbreviated Notation

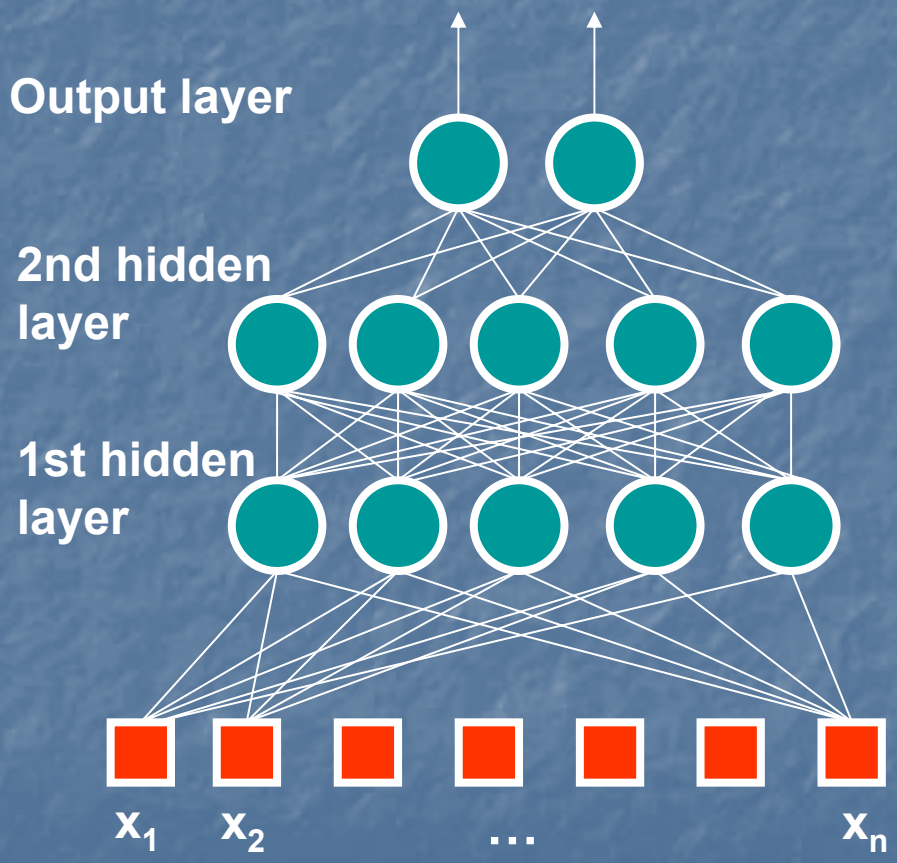


$$\mathbf{a} = \mathbf{f}(\mathbf{W}\mathbf{p} + \mathbf{b})$$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

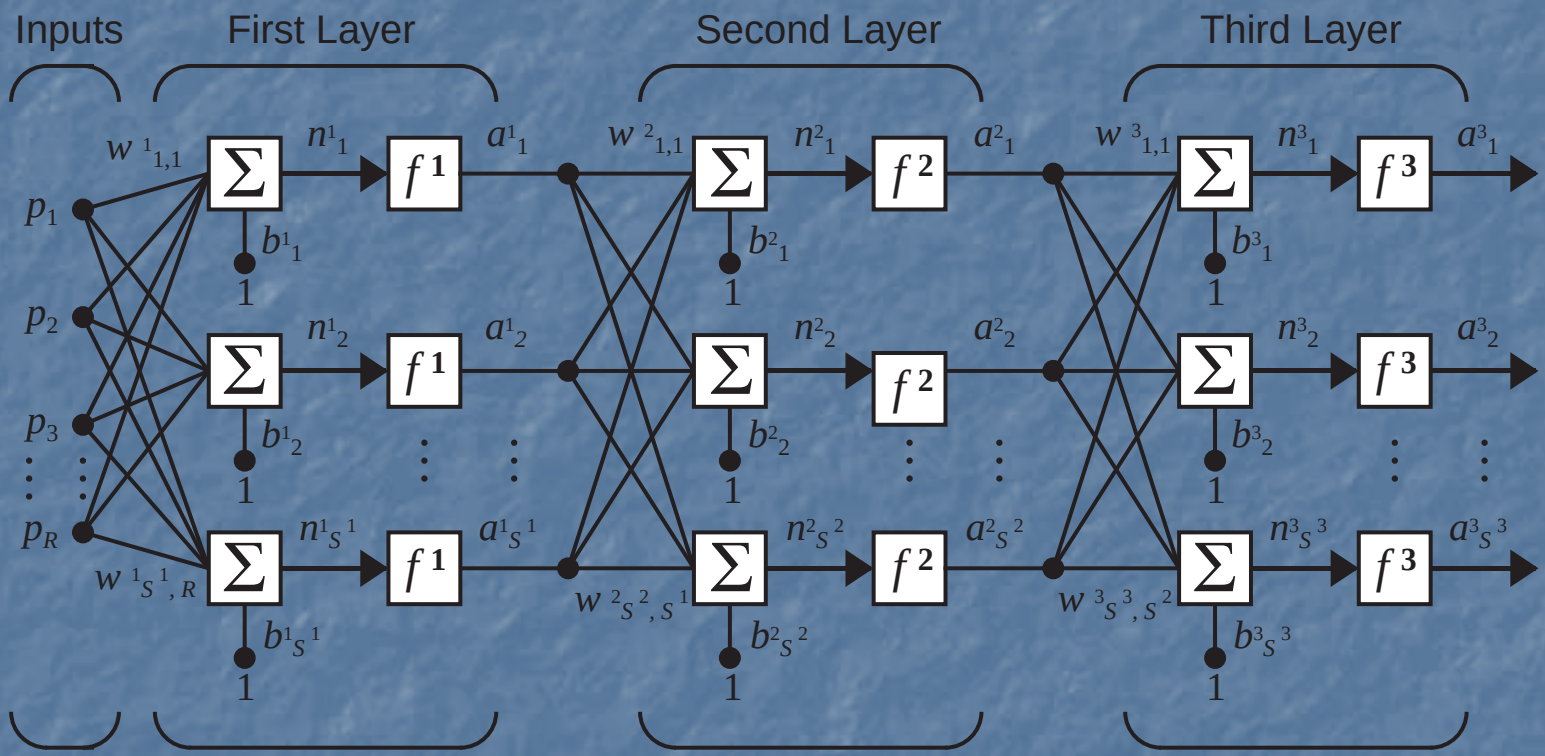
$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

Feedforward Neural Networks



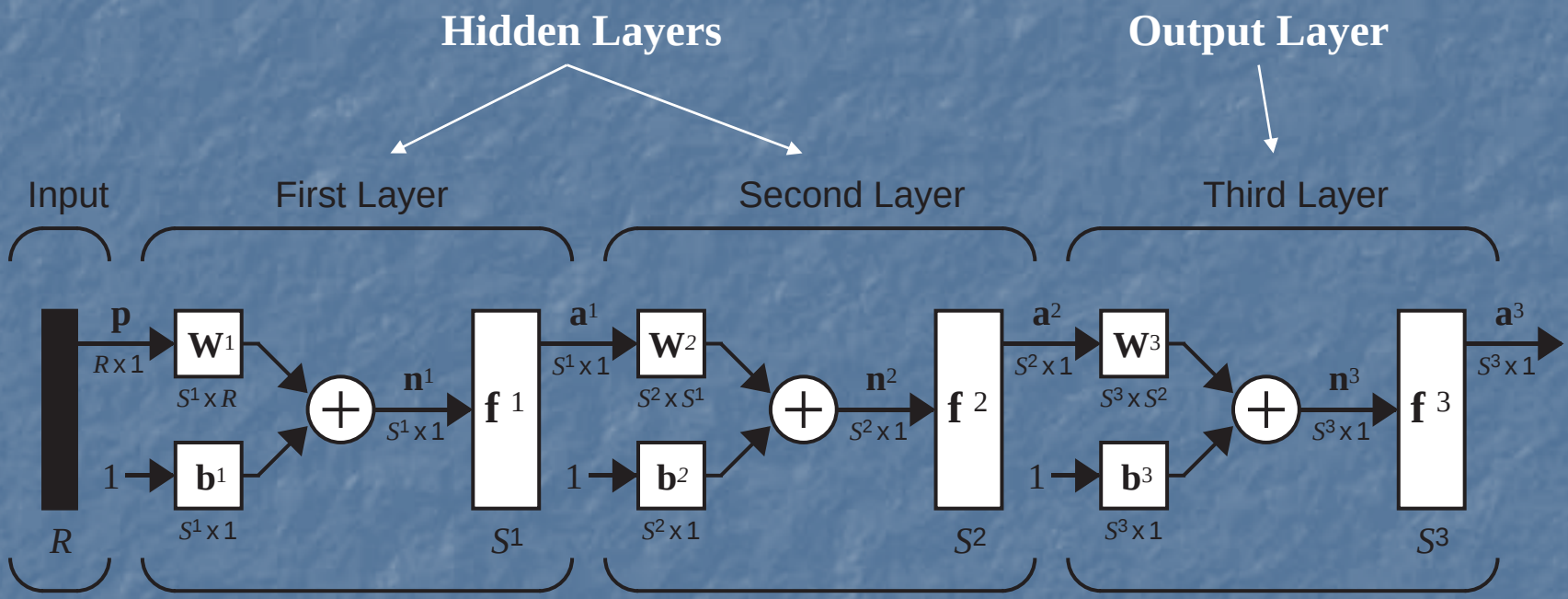
- The information is propagated from the inputs to the outputs
- Computations of No non linear functions from n input variables by compositions of Nc algebraic functions
- Time has no role (NO cycle between outputs and inputs)

Multilayer Network



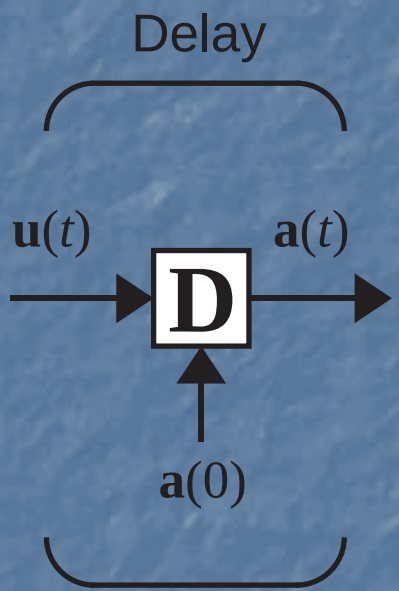
$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

Abreviated Notation

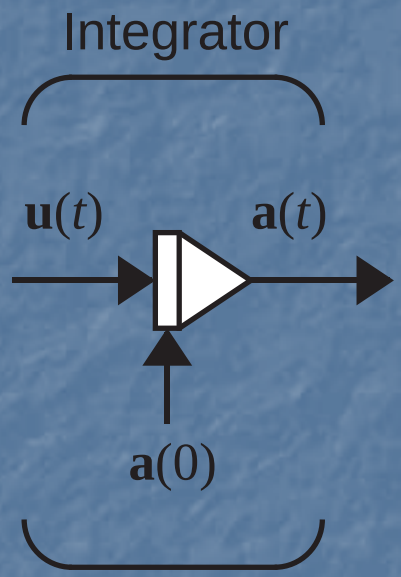


$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Delays and Integrators

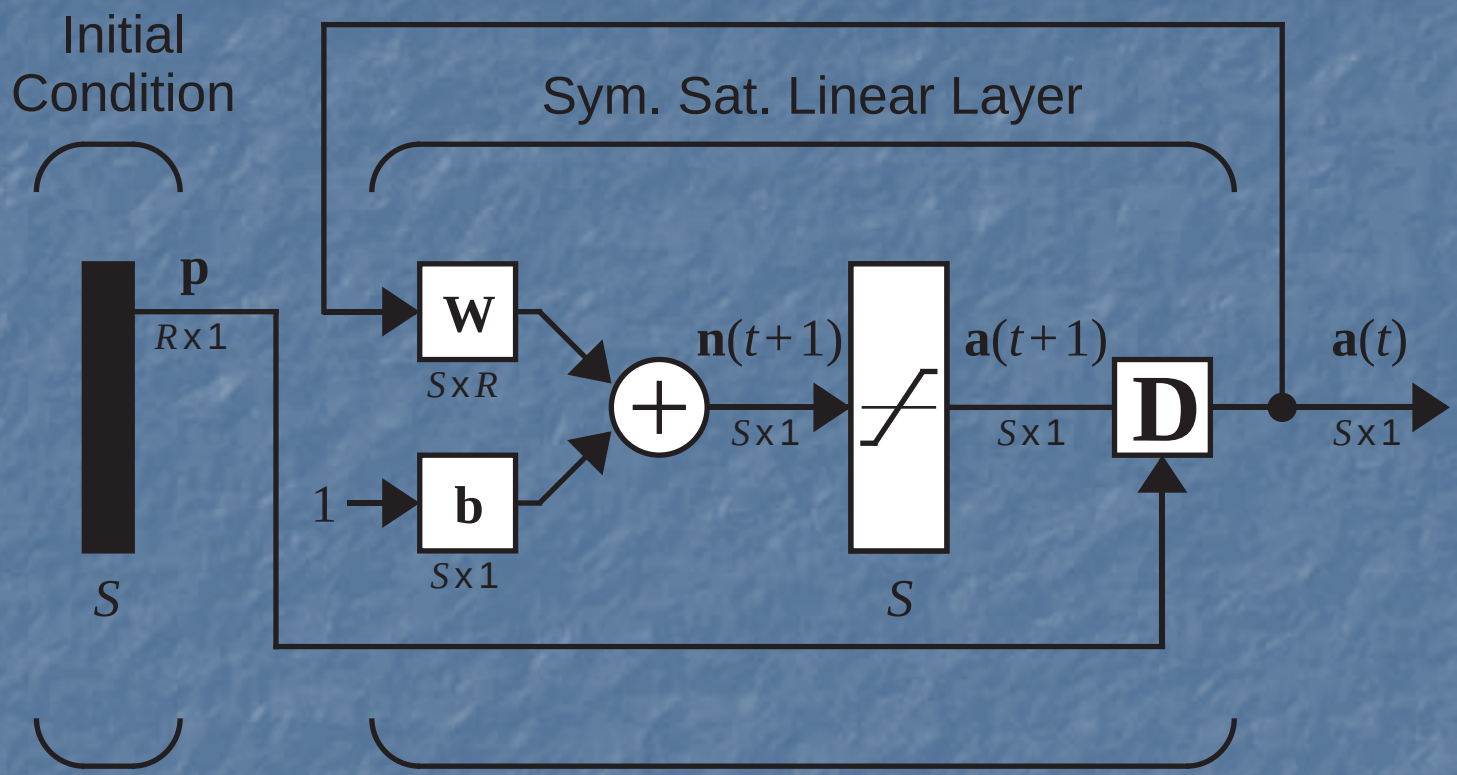


$$\mathbf{a}(t) = \mathbf{u}(t - 1)$$



$$\mathbf{a}(t) = \int_0^t \mathbf{u}(\tau)\, d\tau + \mathbf{a}(0)$$

Recurrent Network



$$\mathbf{a}(0) = \mathbf{p} \qquad \mathbf{a}(t+1) = \mathbf{satlin}(\mathbf{W}\mathbf{a}(t) + \mathbf{b})$$

$$\mathbf{a}(1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(0) + \mathbf{b}) = \mathbf{satlins}(\mathbf{W}\mathbf{p} + \mathbf{b})$$

$$\mathbf{a}(2) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(1) + \mathbf{b})$$

Brief History of ANN

- **McCulloch and Pitts (1943)** designed the first neural network.
- **Hebb (1949)** who developed the first learning rule. If two neurons were active at the same time then the strength between them should be increased.
- **Rosenblatt (1958)** – introduced the concept of a perceptron which performed pattern recognition.
- **Widrow and Hoff (1960)** introduced the concept of the ADALINE (ADAptive Linear Element). The training rule was based on the idea of Least-Mean-Squares learning rule which minimizing the error between the computed output and the desired output.
- **Minsky and Papert (1969)** stated that the perceptron was limited in its ability to recognize features that were separated by linear boundaries. "Neural Net Winter"
- **Kohonen and Anderson** – independently developed neural networks that acted like memories.
- **Webros(1974)** – developed the concept of back propagation of an error to train the weights of the neural network.
- **McCelland and Rumelhart (1986)** published the paper on back propagation algorithm. "Rebirth of neural networks".
- **Today** - they are everywhere a decision can be made.

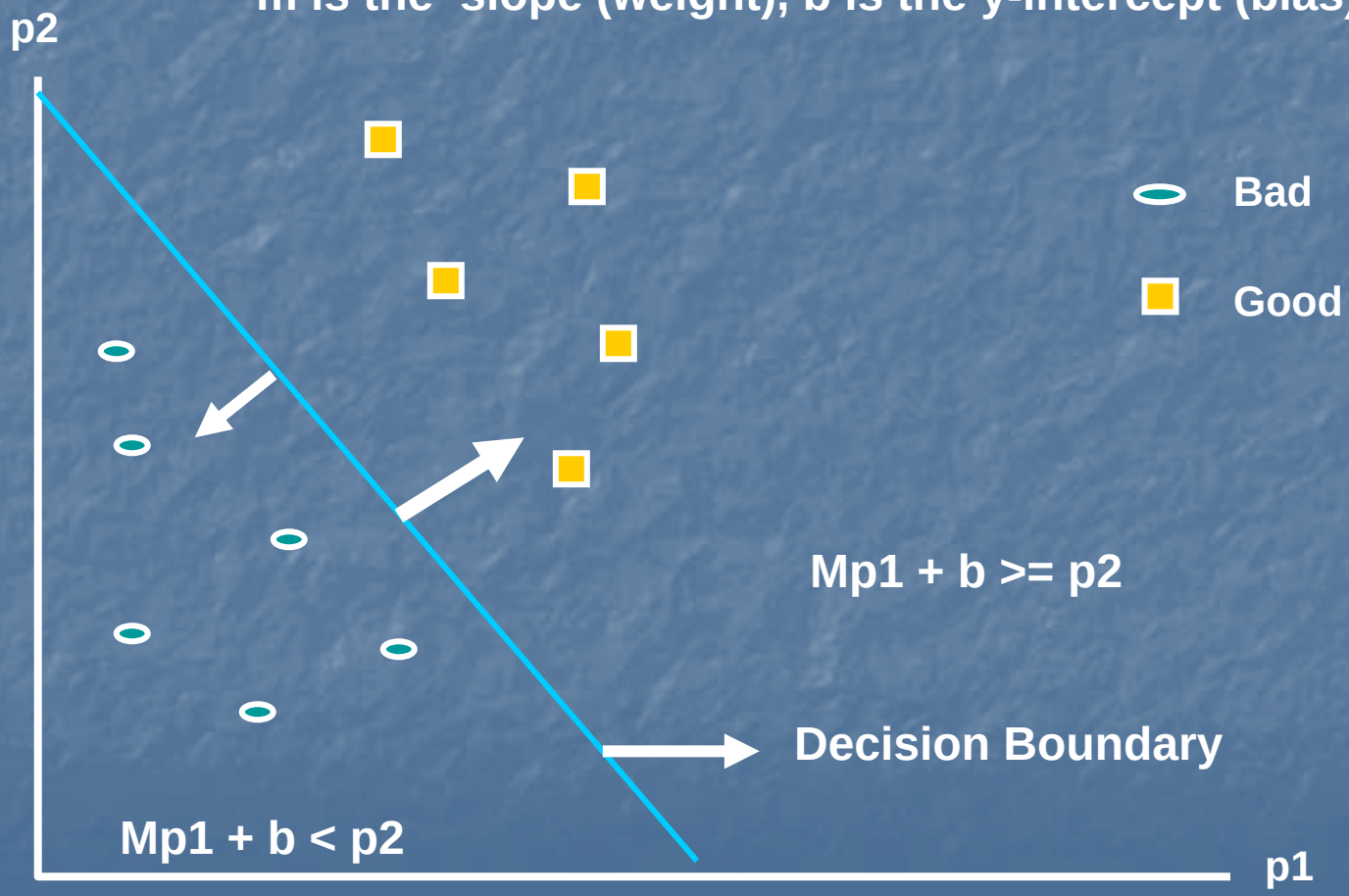
An Illustrative Example

Simple Neural Network

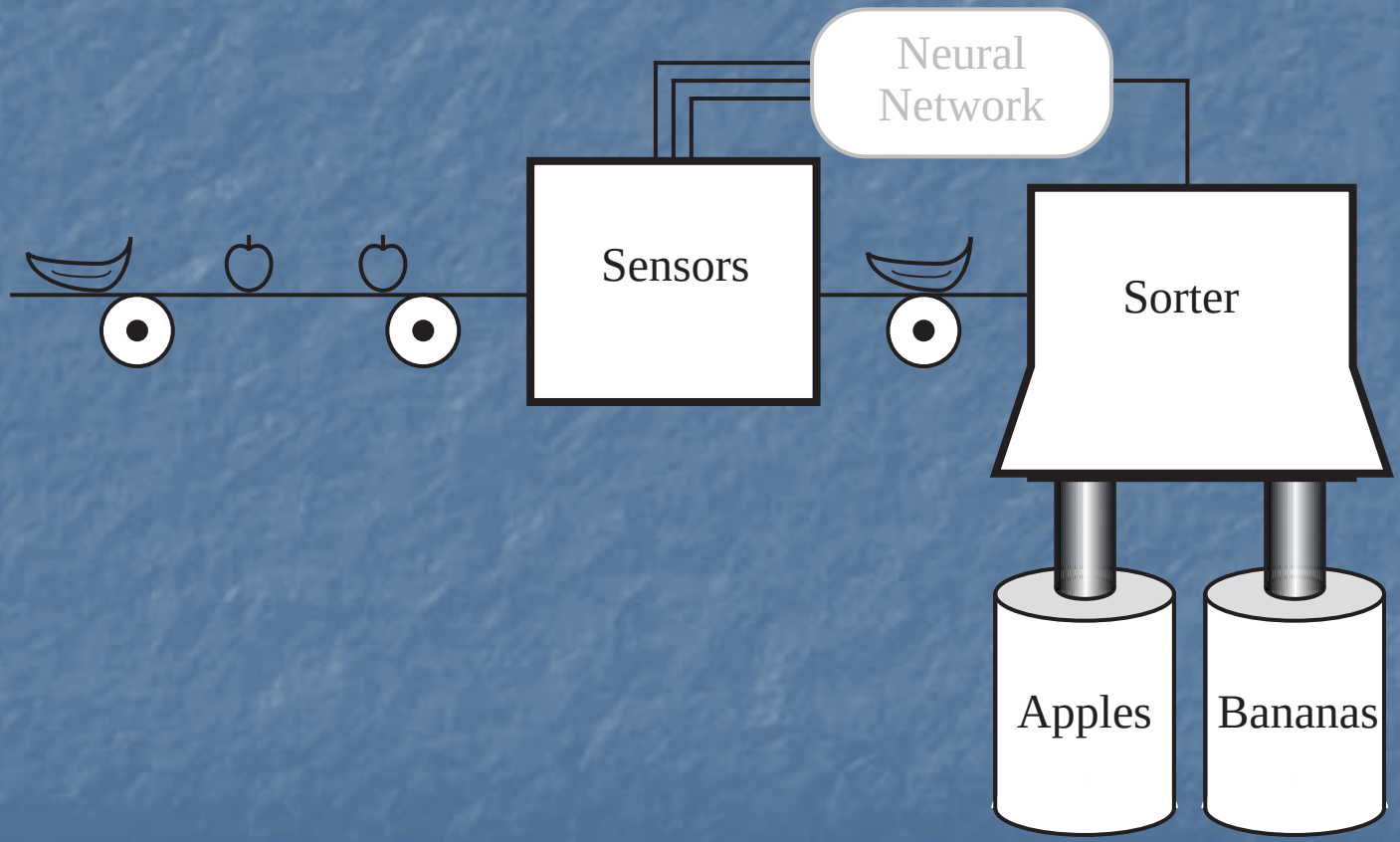
One neuron with a linear activation function => Straight Line

Recall the equation of a straight Line: y = mx +b

m is the slope (weight), b is the y-intercept (bias)



Apple / Banana Sorter



Prototype Vectors

Measurement Vector

$$\mathbf{p} = \begin{bmatrix} \text{shape} \\ \text{texture} \\ \text{weight} \end{bmatrix}$$

Shape: {1: round ; -1: eliptical}
Texture: {1: smooth ; -1: rough}
Weight: {1: > 1 lb. ; -1: < 1 lb.}

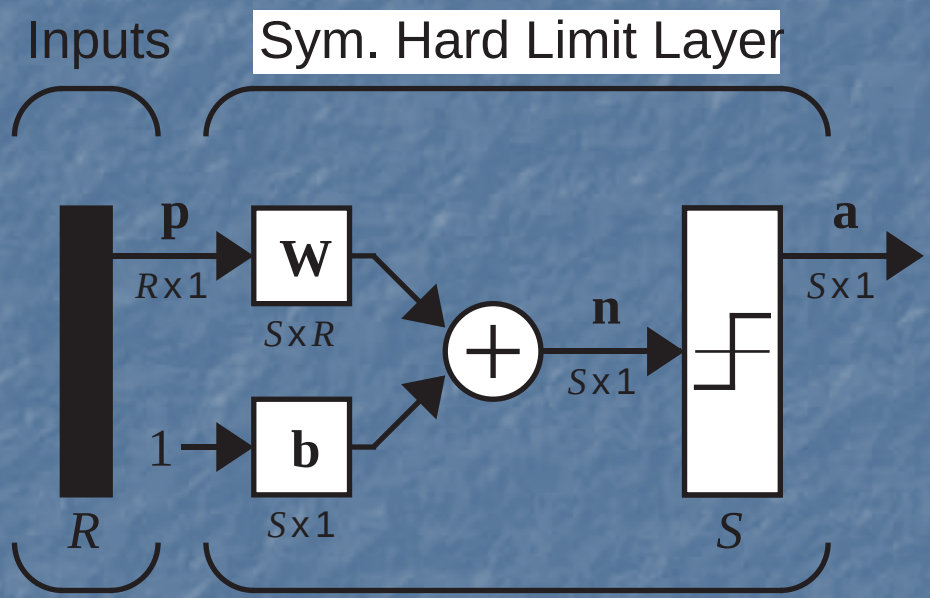
Prototype Banana

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

Prototype Apple

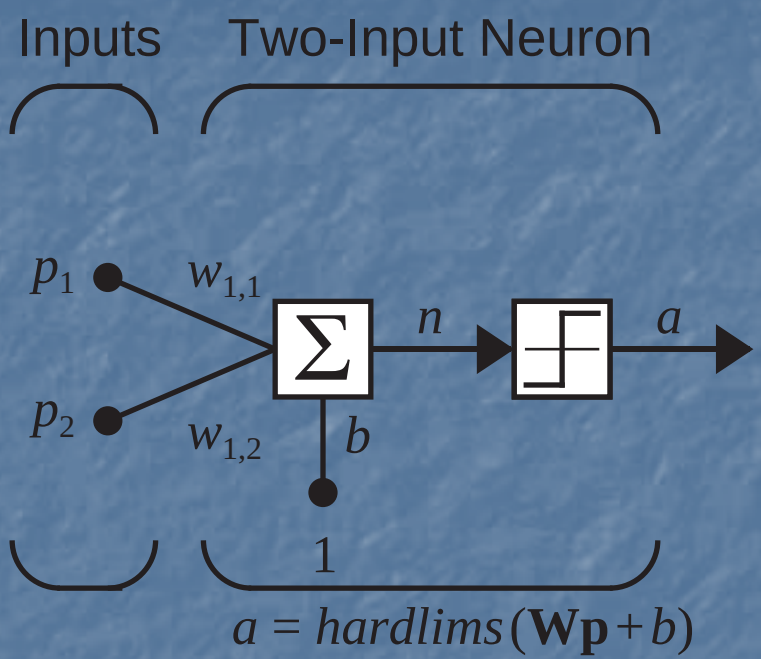
$$\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

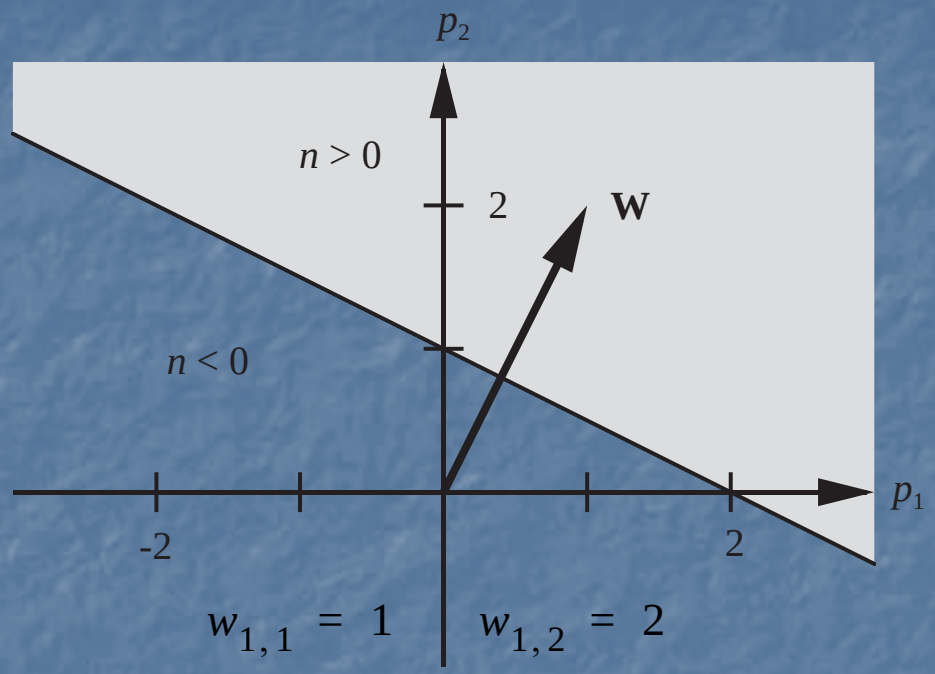
Perceptron



$$\mathbf{a} = \mathbf{hardlims}(\mathbf{Wp} + \mathbf{b})$$

Two-Input Case





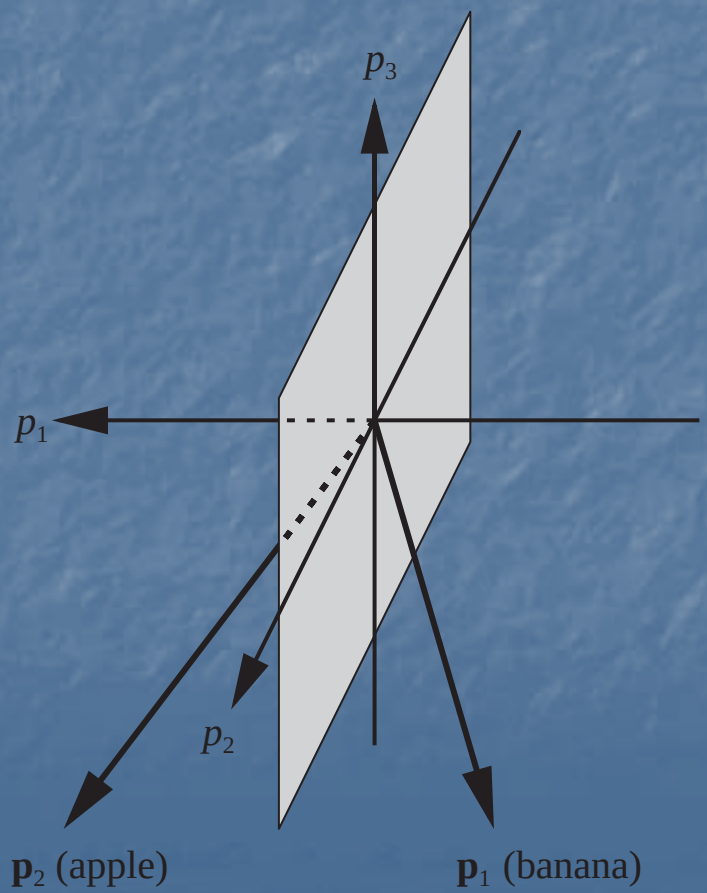
$$a = hardlims(n) = hardlims(\begin{bmatrix}1 & 2\end{bmatrix}\mathbf{p} + (-2))$$

Decision Boundary

$$\mathbf{W}\mathbf{p} + b = 0 \qquad \begin{bmatrix}1 & 2\end{bmatrix}\mathbf{p} + (-2) = 0$$

Apple/Banana Example

$$a = hardlims\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + b\right)$$



The decision boundary should separate the prototype vectors.

$$p_1 = 0$$

The weight vector should be orthogonal to the decision boundary, and should point in the direction of the vector which should produce an output of 1. The bias determines the position of the boundary

$$\begin{bmatrix} -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + 0 = 0$$

Testing the Network

Banana:

$$a = hardlims\left(\begin{bmatrix}-1 & 0 & 0\end{bmatrix}\begin{bmatrix}-1\\ 1\\ -1\end{bmatrix} + 0\right) = 1 (banana)$$

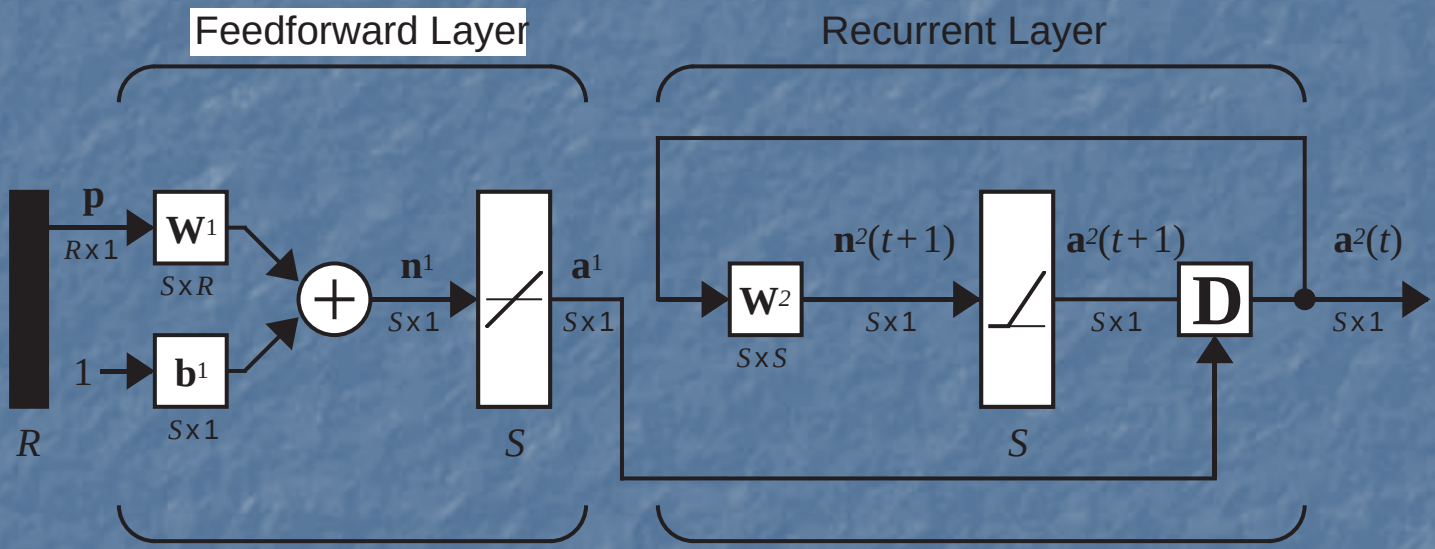
Apple:

$$a = hardlims\left(\begin{bmatrix}-1 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\ 1\\ -1\end{bmatrix} + 0\right) = -1 (apple)$$

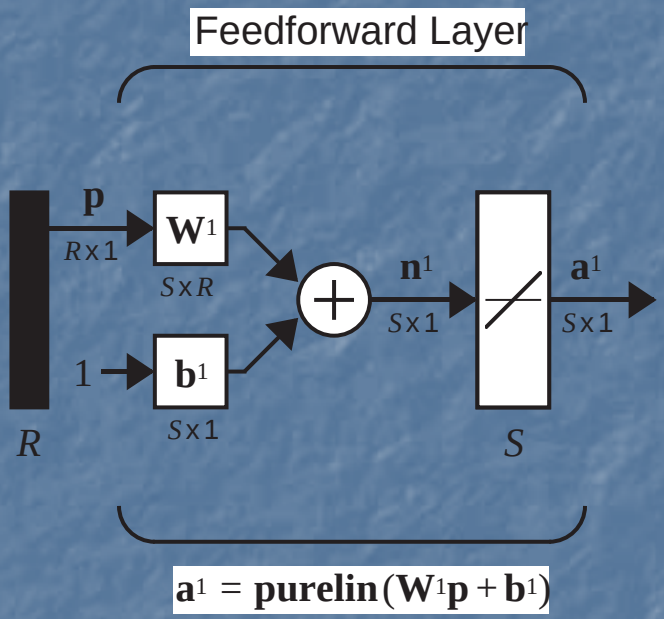
“Rough” Banana:

$$a = hardlims\left(\begin{bmatrix}-1 & 0 & 0\end{bmatrix}\begin{bmatrix}-1\\ -1\\ -1\end{bmatrix} + 0\right) = 1 (banana)$$

Hamming Network



Feedforward Layer



For Banana/Apple Recognition

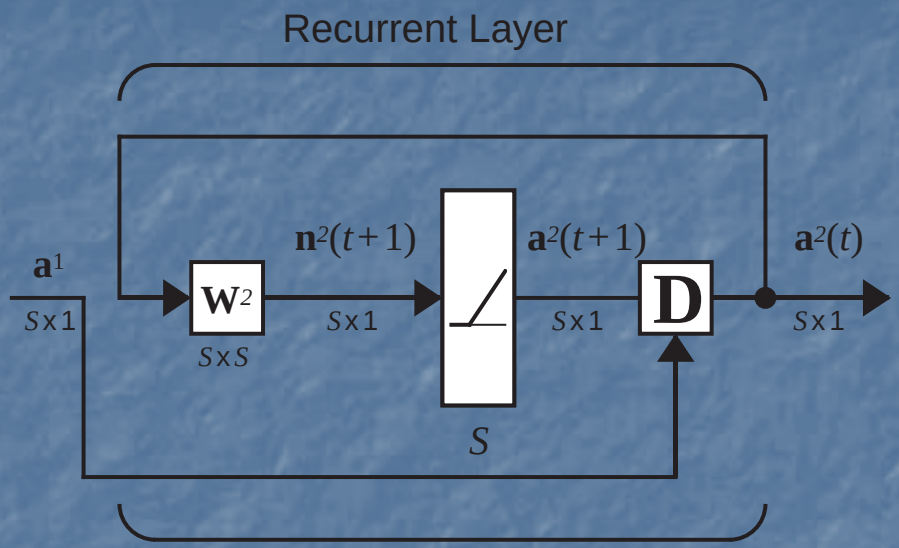
$$S = 2$$

$$\mathbf{W}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{b}^1 = \begin{bmatrix} R \\ R \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix}\mathbf{p} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + 3 \\ \mathbf{p}_2^T\mathbf{p} + 3 \end{bmatrix}$$

Recurrent Layer



$\mathbf{a}^2(0) = \mathbf{a}^1 \qquad \mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix} \qquad \varepsilon < \frac{1}{S-1}$$

$$\mathbf{a}^2(t+1) = \mathbf{poslin}\left(\begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}\mathbf{a}^2(t)\right) = \mathbf{poslin}\left(\begin{bmatrix} a_1^2(t) - \varepsilon a_2^2(t) \\ a_2^2(t) - \varepsilon a_1^2(t) \end{bmatrix}\right)$$

Hamming Operation

First Layer

Input (Rough Banana)

$$\mathbf{p} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$\mathbf{a}^1 = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} (1+3) \\ (-1+3) \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

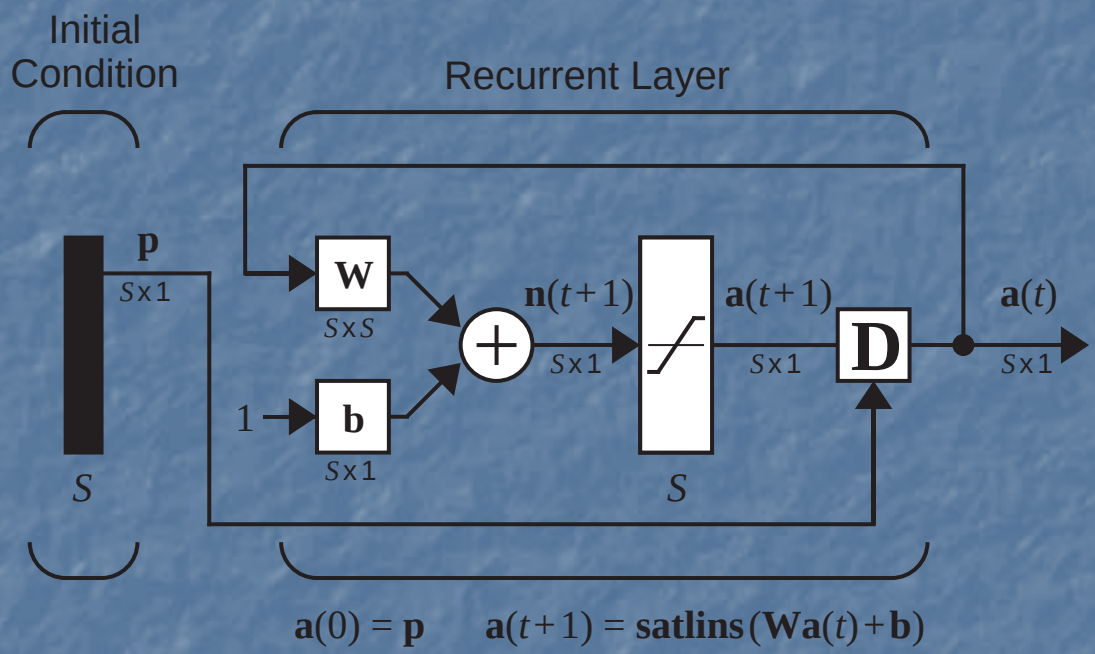
Hamming Operation

Second Layer

$$\mathbf{a}^2(1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(0)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 4 \\ 2 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

$$\mathbf{a}^2(2) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(1)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ -1.5 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

Hopfield Network



Apple/Banana Problem

$$\mathbf{W} = \begin{bmatrix} 1.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 0.9 \\ -0.9 \end{bmatrix}$$

$$a_1(t+1) = satlins(1.2\,a_1(t))$$

$$a_2(t+1) = satlins(0.2\,a_2(t) + 0.9)$$

$$a_3(t+1) = \text{satlins}(0.2\,a_3(t) - 0.9)$$

Test: “Rough” Banana

$$\mathbf{a}(0) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \quad \mathbf{a}(1) = \begin{bmatrix} -1 \\ 0.7 \\ -1 \end{bmatrix} \quad \mathbf{a}(2) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{a}(3) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$ (Banana)

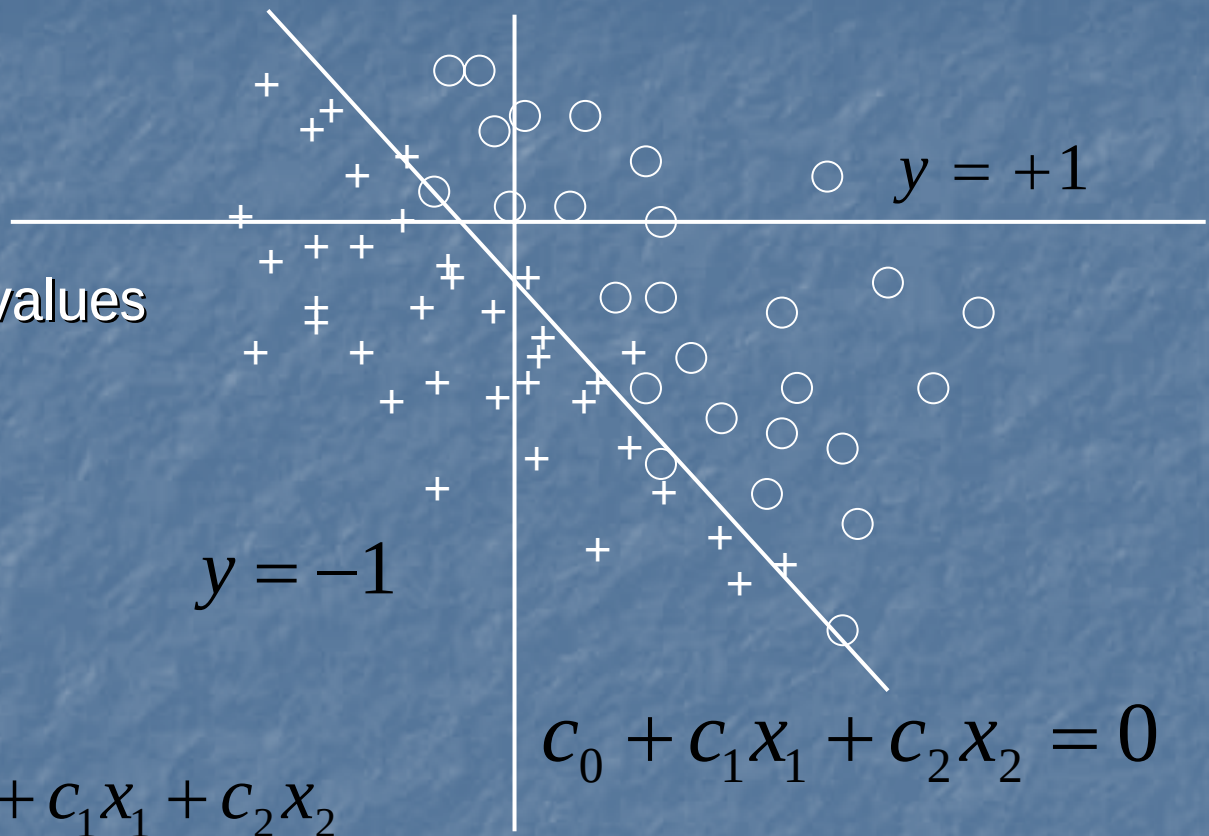
Summary

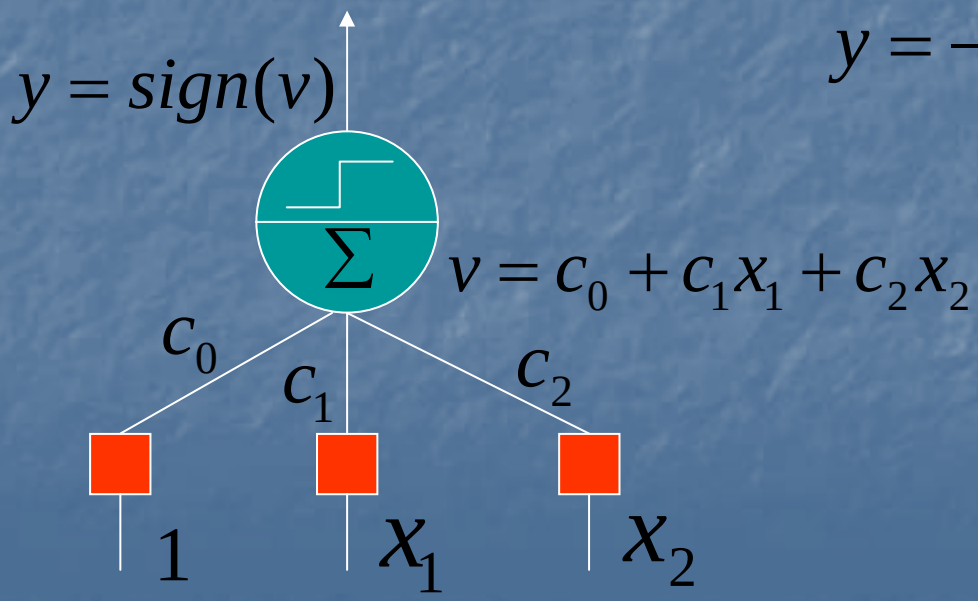
- Perceptron
  - Feedforward Network
  - Linear Decision Boundary
  - One Neuron for Each Decision
- Hamming Network
  - Competitive Network
  - First Layer – Pattern Matching (Inner Product)
  - Second Layer – Competition (Winner-Take-All)
  - # Neurons = # Prototype Patterns
- Hopfield Network
  - Dynamic Associative Memory Network
  - Network Output Converges to a Prototype Pattern
  - # Neurons = # Elements in each Prototype Pattern

Perceptron Learning Rule

Perceptron

- Rosenblatt (1962)
- Linear separation
- Inputs: Vector of real values
- Outputs: 1 or -1





Learning Rules

- **Supervised Learning**

  Network is provided with a set of examples of proper network behavior (inputs/targets)

$$\{\mathbf{p}_1,\mathbf{t}_1\},\{\mathbf{p}_2,\mathbf{t}_2\},\ldots,\{\mathbf{p}_Q,\mathbf{t}_Q\}$$

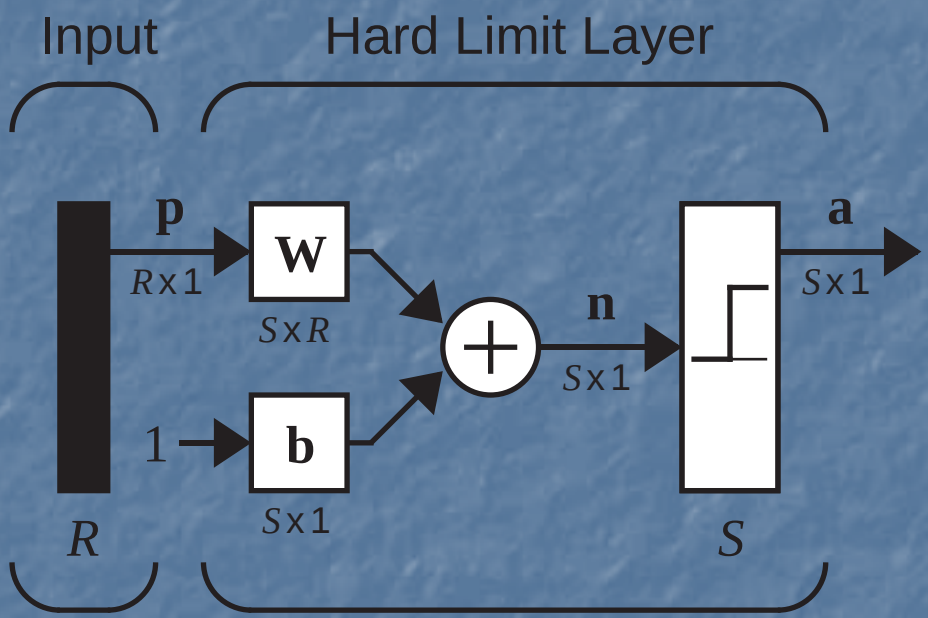
- **Reinforcement Learning**

  Network is only provided with a grade, or score, which indicates network performance

- **Unsupervised Learning**

  Only network inputs are available to the learning algorithm. Network learns to categorize (cluster) the inputs.

Perceptron Architecture



$$\mathbf{a} = \mathbf{hardlim}(\mathbf{Wp} + \mathbf{b})$$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

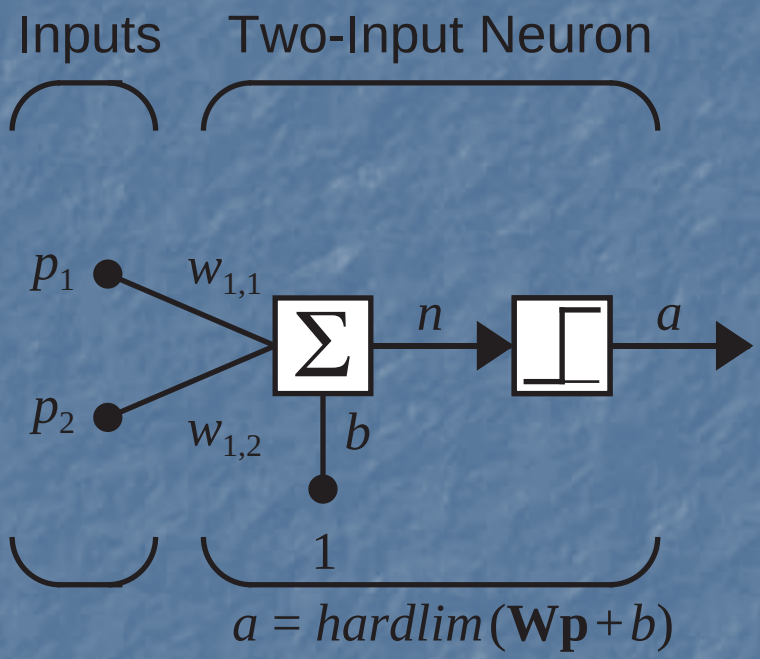
$${}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}$$

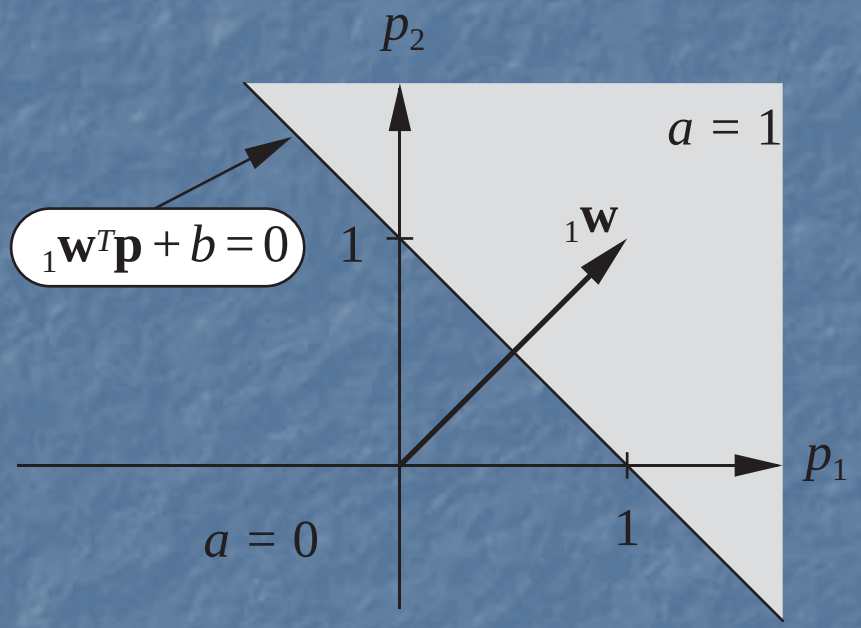
$$\mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix}$$

$$a_i = hardlim(n_i) = hardlim({}_i\mathbf{w}^T\mathbf{p} + b_i)$$

Single-Neuron Perceptron

$w_{1,1} = 1 \qquad w_{1,2} = 1 \qquad b = -1$



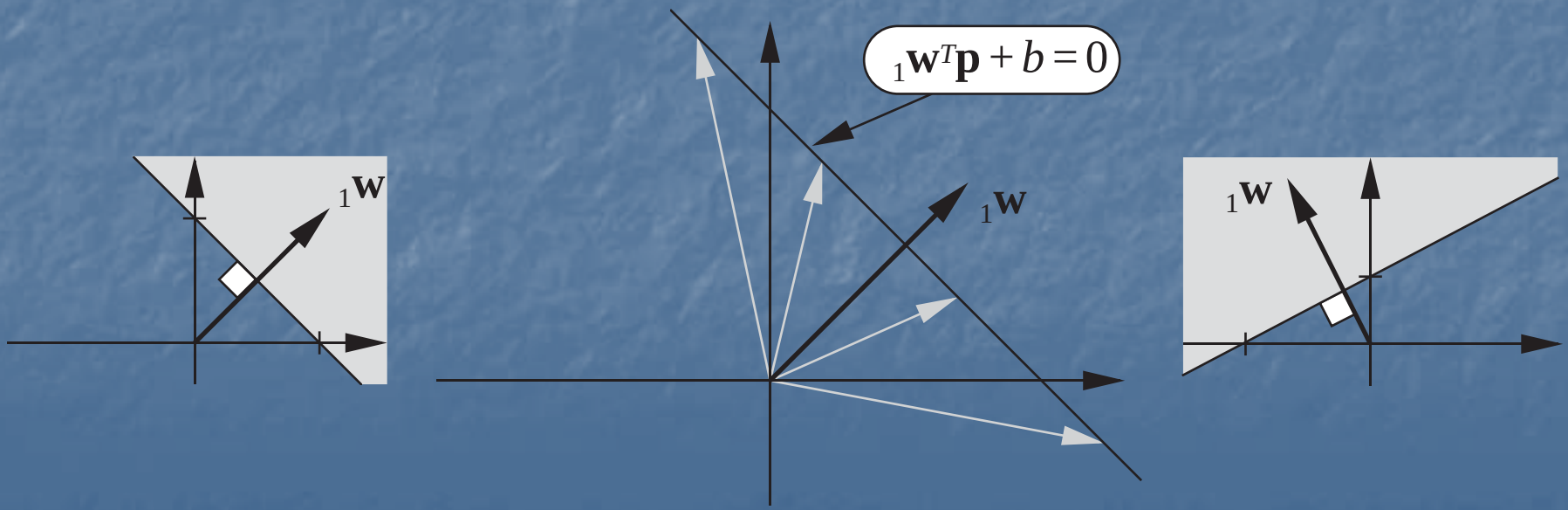


$$a = hardlim(_1\mathbf{w}^T\mathbf{p} + b) = hardlim(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

Decision Boundary

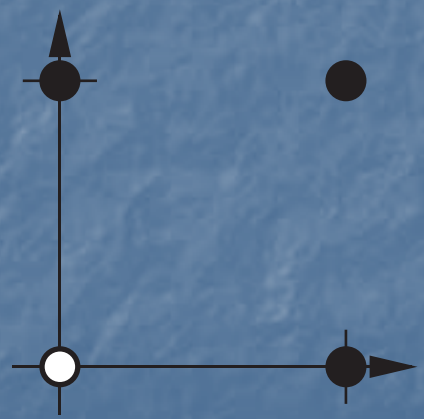
$$ {}_1\mathbf{w}^T\mathbf{p} + b = 0 \qquad {}_1\mathbf{w}^T\mathbf{p} = -b $$

- All points on the decision boundary have the same inner product with the weight vector.
- Therefore they have the same projection onto the weight vector, and they must lie on a line orthogonal to the weight vector.

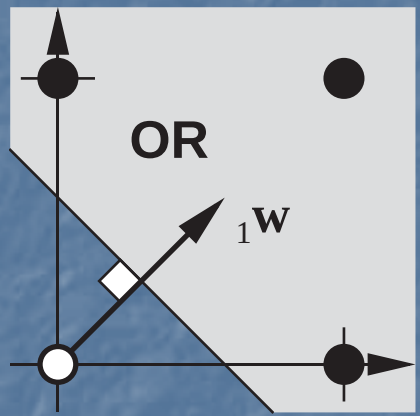


Example - OR

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 1\right\}$$



OR Solution



Weight vector should be orthogonal to the decision boundary

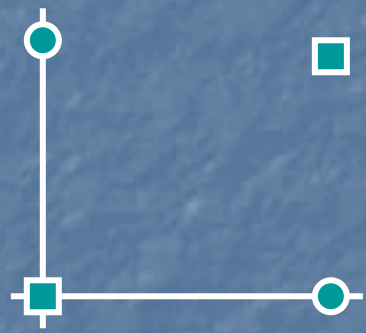
$$ {}_1\mathbf{w} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} $$

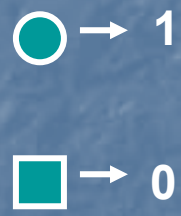
Pick a point on the decision boundary to find the bias

$$ {}_1\mathbf{w}^T\mathbf{p} + b = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + b = 0.25 + b = 0 \quad \Rightarrow \quad b = -0.25 $$

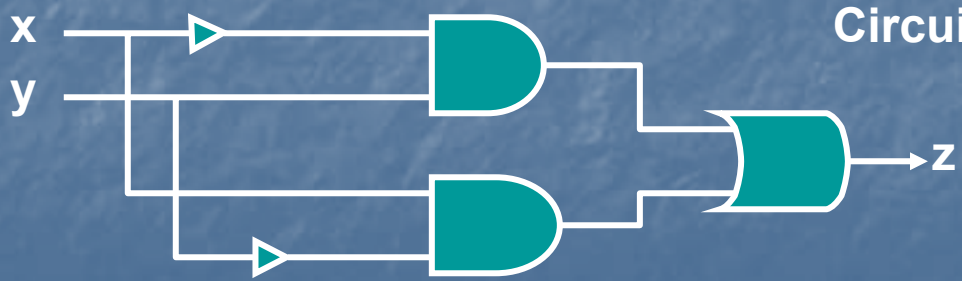
XOR Function

X	Y	Z = (X and (not Y)) or ((not X) and Y)
0	0	0
0	1	1
1	0	1
1	1	0





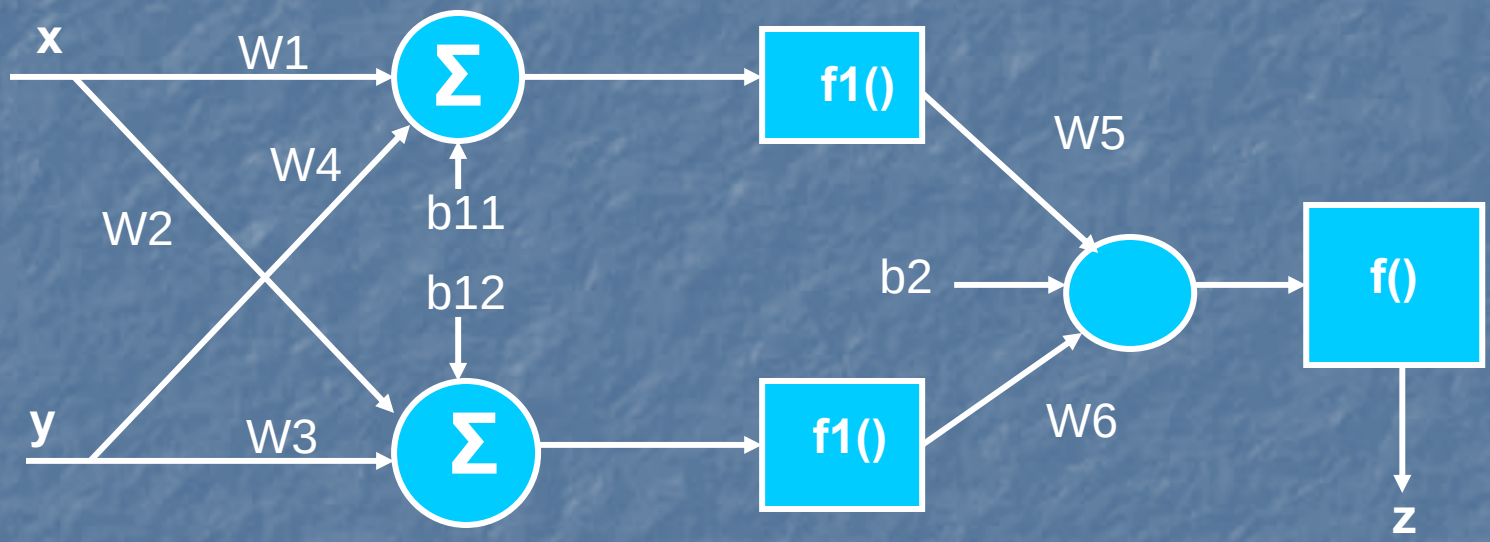
No single decision boundary can separate the favorable and unfavorable outcomes



Circuit Diagram

We will need a more complicated neural net to realize this function

XOR Function – Multilayer Perceptron



$$Z = f\,(W5*f1(W1*x + W4*y+b11) + W6*f1(W2*x + W3*y+b12)+b2)$$

Multiple-Neuron Perceptron

Each neuron will have its own decision boundary

$$_i\mathbf{w}^T\mathbf{p} + b_i = 0$$

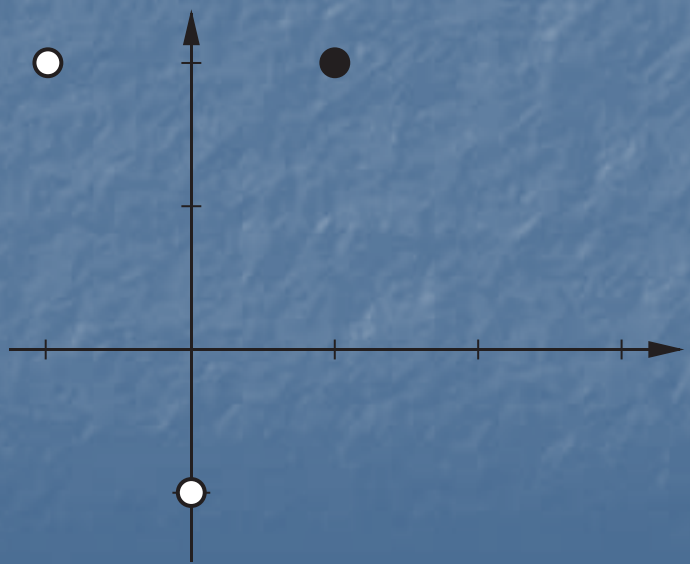
A single neuron can classify input vectors into two categories.

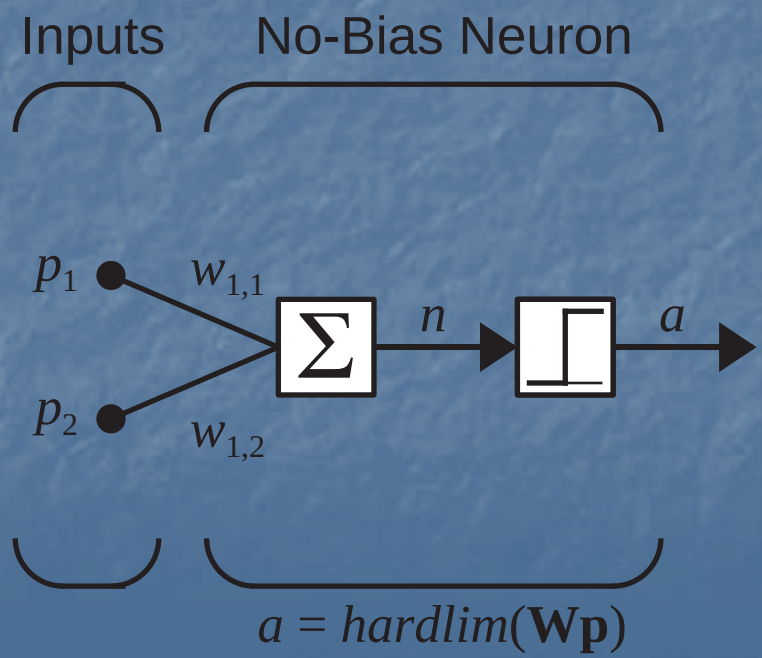
A multi-neuron perceptron can classify input vectors into 2^S categories.

Learning Rule Test Problem

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}$$

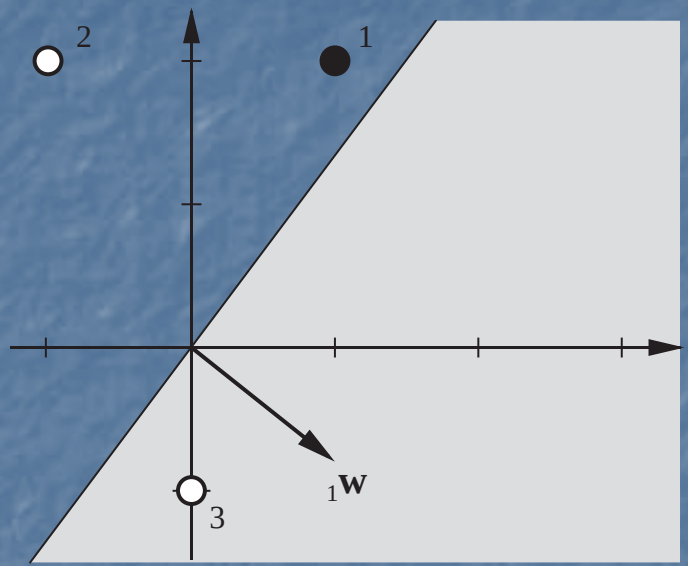




Starting Point

Random initial weight:

$$ {}_1\mathbf{w} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} $$



Present $\mathbf{p}_1$ to the network:

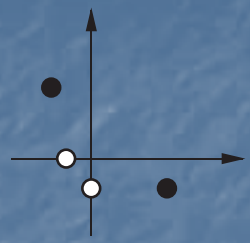
$$ a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) $$

$$ a = hardlim(-0.6) = 0 $$

Incorrect Classification

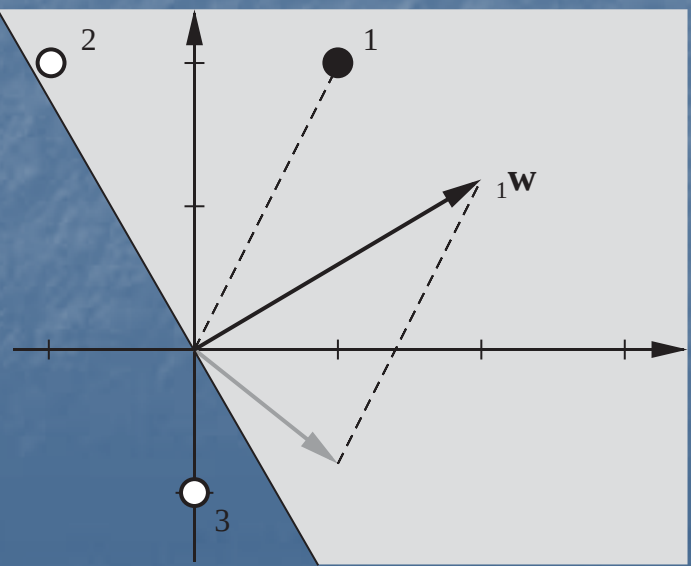
Tentative Learning Rule

- Set ${}_1\mathbf{w}$ to $\mathbf{p}_1$
  - Not stable ×
- Add $\mathbf{p}_1$ to ${}_1\mathbf{w}$ √



Tentative Rule: If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}$$



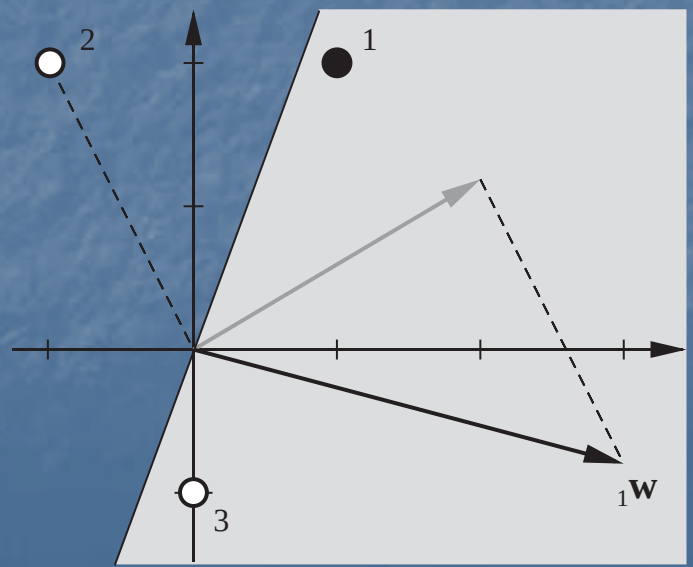
Second Input Vector

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_2) = hardlim\left(\begin{bmatrix}2.0 & 1.2\end{bmatrix}\begin{bmatrix}-1\\2\end{bmatrix}\right)$$

$$a = hardlim(0.4) = 1 \quad \text{(Incorrect Classification)}$$

Modification to Rule: If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix}2.0\\1.2\end{bmatrix} - \begin{bmatrix}-1\\2\end{bmatrix} = \begin{bmatrix}3.0\\-0.8\end{bmatrix}$$

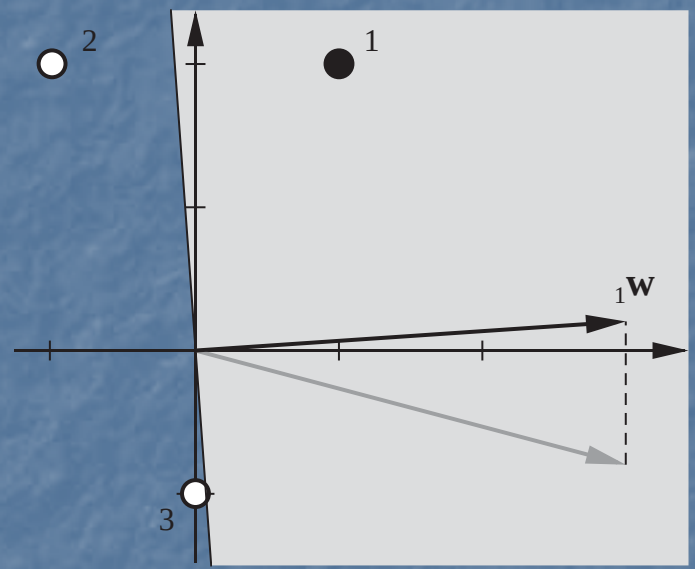


Third Input Vector

$$a = hardlim({}_1\mathbf{w}^T \mathbf{p}_3) = hardlim\left(\begin{bmatrix}3.0 & -0.8\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}\right)$$

$$a = hardlim(0.8) = 1$$ (Incorrect Classification)

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix}3.0\\-0.8\end{bmatrix} - \begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}3.0\\0.2\end{bmatrix}$$



Patterns are now correctly classified

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$.

Unified Learning Rule

If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$$e = t - a$$

If $e = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $e = -1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $e = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t-a)\mathbf{p}$$

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

A bias is a weight with an input of 1

Multiple-Neuron Perceptrons

To update the i^{th} row of the weight matrix:

$$_i\mathbf{w}^{new} = {}_i\mathbf{w}^{old} + e_i\mathbf{p}$$

$$b_i^{new} = b_i^{old} + e_i$$

Matrix form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T$$

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

Apple/Banana Example

Training Set

$$\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}, t_1 = \begin{bmatrix}1\end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\\-1\end{bmatrix}, t_2 = \begin{bmatrix}0\end{bmatrix}\right\}$$

Initial Weights

$$\mathbf{W} = \begin{bmatrix}0.5 & -1 & -0.5\end{bmatrix} \qquad b = 0.5$$

First Iteration

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix}0.5 & -1 & -0.5\end{bmatrix}\begin{bmatrix}-1\\1\\-1\end{bmatrix} + 0.5\right)$$

$$a = hardlim(-0.5) = 0 \qquad e = t_1 - a = 1 - 0 = 1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix}0.5 & -1 & -0.5\end{bmatrix} + (1)\begin{bmatrix}-1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + (1) = 1.5$$

Second Iteration

$$a = hardlim\,(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix} + (1.5)\right)$$

$$a = hardlim\,(2.5) = 1$$

$$e = t_2 - a = 0 - 1 = -1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix} + (-1)\begin{bmatrix}1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-1.5 & -1 & -0.5\end{bmatrix}$$

$$b^{new} = b^{old} + e = 1.5 + (-1) = 0.5$$

Check

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix}-1.5 & -1 & -0.5\end{bmatrix}\begin{bmatrix}-1\\ 1\\ -1\end{bmatrix} + 0.5\right)$$

$$a = hardlim(1.5) = 1 = t_1$$

$$a = hardlim(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix}-1.5 & -1 & -0.5\end{bmatrix}\begin{bmatrix}1\\ 1\\ -1\end{bmatrix} + 0.5\right)$$

$$a = hardlim(-1.5) = 0 = t_2$$

Perceptron Rule Capability

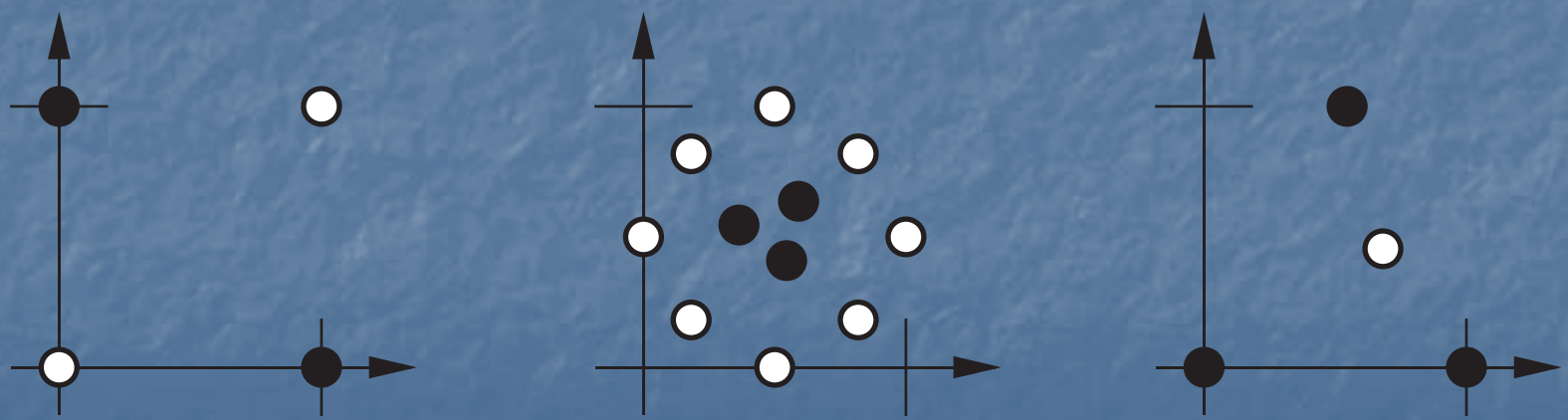
The Perceptron rule will always converge to weights which accomplish the desired classification, assuming that such weights exist

Perceptron Limitations

Linear Decision Boundary

$$_1\mathbf{w}^T\mathbf{p} + b = 0$$

Linearly Inseparable Problems

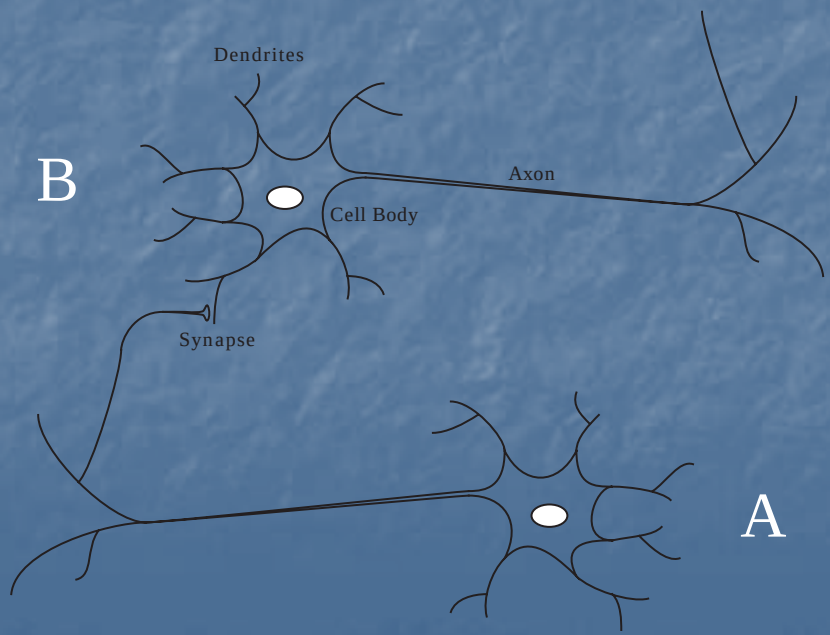


Supervised Hebbian Learning

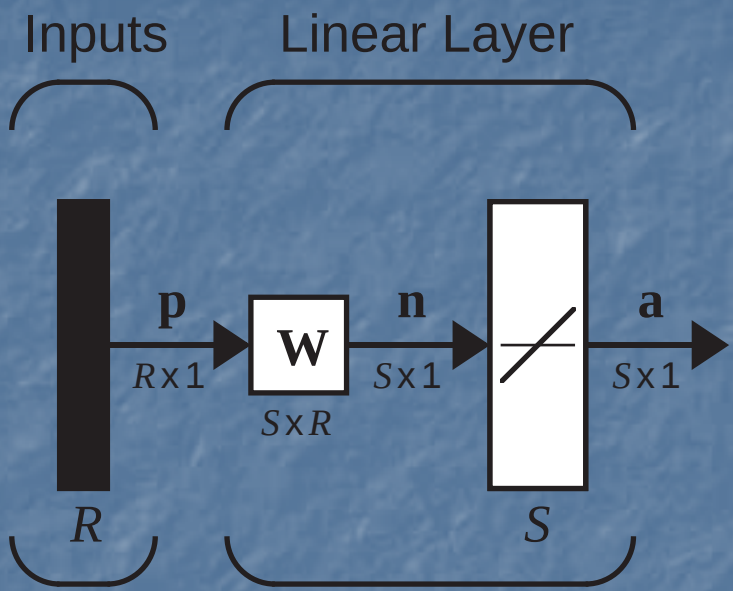
Hebb's Postulate

"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."

D. O. Hebb, 1949



Linear Associator



$$\mathbf{a} = \mathbf{W}\mathbf{p} \qquad a_i = \sum_{j=1}^{R} w_{ij} p_j$$

Training Set:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Hebb Rule

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, f_i(a_{iq}) g_j(p_{jq})$$

Presynaptic Signal

Postsynaptic Signal

Simplified Form:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq} p_{jq}$$

Supervised Form:

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq} p_{jq}$$

Matrix Form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q \mathbf{p}_q^T$$

Batch Operation

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T = \sum_{q=1}^{Q} \mathbf{t}_q\mathbf{p}_q^T$$

(Zero Initial Weights)

Matrix Form:

$$\mathbf{W} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix} \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} = \mathbf{T}\mathbf{P}^T$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix}$$

Performance Analysis

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \left(\sum_{q=1}^{Q} \mathbf{t}_q \mathbf{p}_q^T\right)\mathbf{p}_k = \sum_{q=1}^{Q} \mathbf{t}_q(\mathbf{p}_q^T\mathbf{p}_k)$$

Case I, input patterns are orthogonal.

$$(\mathbf{p}_q^T\mathbf{p}_k) = 1 \qquad q = k$$

$$= 0 \qquad q \neq k$$

Therefore the network output equals the target:

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k$$

Case II, input patterns are normalized, but not orthogonal.

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k + \boxed{\sum_{q \neq k} \mathbf{t}_q(\mathbf{p}_q^T\mathbf{p}_k)}$$

Error

Example

Banana Apple Normalized Prototype Patterns

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \quad \left\{ \mathbf{p}_1 = \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$$

Weight Matrix (Hebb Rule):

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -0.5774 & 0.5774 & -0.5774 \\ 0.5774 & 0.5774 & -0.5774 \end{bmatrix} = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix}$$

Tests:

Banana $\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} -0.6668 \end{bmatrix}$

Apple $\mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 0 & 1.1548 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} 0.6668 \end{bmatrix}$

Pseudoinverse Rule - (1)

Performance Index: $\mathbf{W}\mathbf{p}_q = \mathbf{t}_q \qquad q = 1, 2, \ldots, Q$

$$F(\mathbf{W}) = \sum_{q=1}^{Q} \|\mathbf{t}_q - \mathbf{W}\mathbf{p}_q\|^2$$

Matrix Form:

$$\mathbf{W}\mathbf{P} = \mathbf{T}$$

$$\mathbf{T} = \left[\mathbf{t}_1\ \mathbf{t}_2\ \cdots\ \mathbf{t}_Q\right] \quad \mathbf{P} = \left[\mathbf{p}_1\ \mathbf{p}_2\ \cdots\ \mathbf{p}_Q\right]$$

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2$$

$$\|\mathbf{E}\|^2 = \sum_i \sum_j e_{ij}^2$$

Pseudoinverse Rule - (2)

$$\mathbf{W}\mathbf{P} = \mathbf{T}$$

Minimize:

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2$$

If an inverse exists for **P**, $F(\mathbf{W})$ can be made zero:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{-1}$$

When an inverse does not exist $F(\mathbf{W})$ can be minimized using the pseudoinverse:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+}$$

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T}$$

Relationship to the Hebb Rule

Hebb Rule

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T$$

Pseudoinverse Rule

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$$

If the prototype patterns are orthonormal:

$$\mathbf{P}^T\mathbf{P} = \mathbf{I}$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T = \mathbf{P}^T$$

Example

$$\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}-1\end{bmatrix}\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\\-1\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\end{bmatrix}\right\} \qquad \mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \begin{bmatrix}-1 & 1\end{bmatrix}\left(\begin{bmatrix}-1 & 1\\1 & 1\\-1 & -1\end{bmatrix}\right)^{+}$$

$$\mathbf{P}^{+} = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T = \begin{bmatrix}3 & 1\\1 & 3\end{bmatrix}^{-1}\begin{bmatrix}-1 & 1 & -1\\1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-0.5 & 0.25 & -0.25\\0.5 & 0.25 & -0.25\end{bmatrix}$$

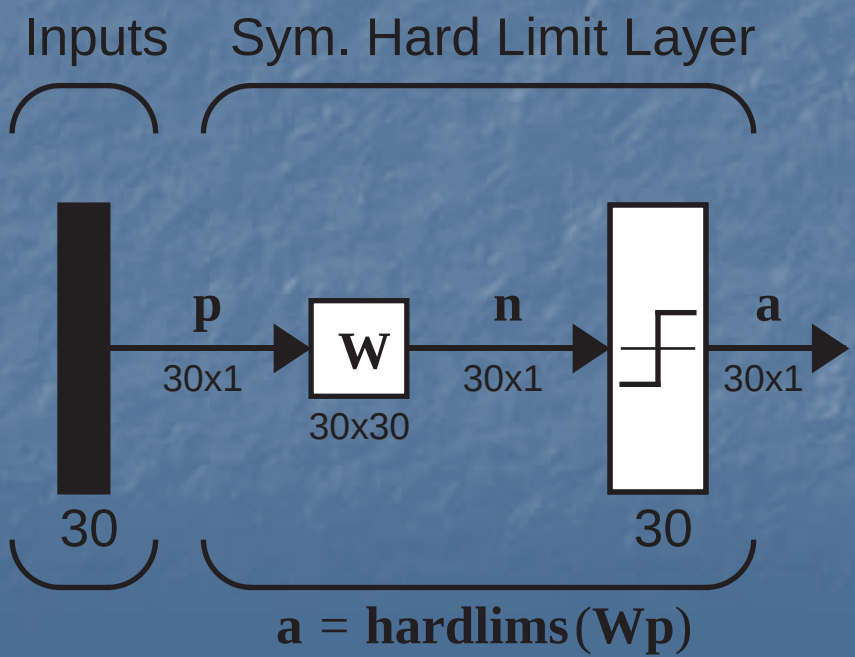
$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}-0.5 & 0.25 & -0.25\\0.5 & 0.25 & -0.25\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}$$

$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}-1\\1\\-1\end{bmatrix} = \begin{bmatrix}-1\end{bmatrix} \qquad \mathbf{W}\mathbf{p}_2 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix} = \begin{bmatrix}1\end{bmatrix}$$

Autoassociative Memory

$\mathbf{p}_1,\mathbf{t}_1$ $\mathbf{p}_2,\mathbf{t}_2$ $\mathbf{p}_3,\mathbf{t}_3$

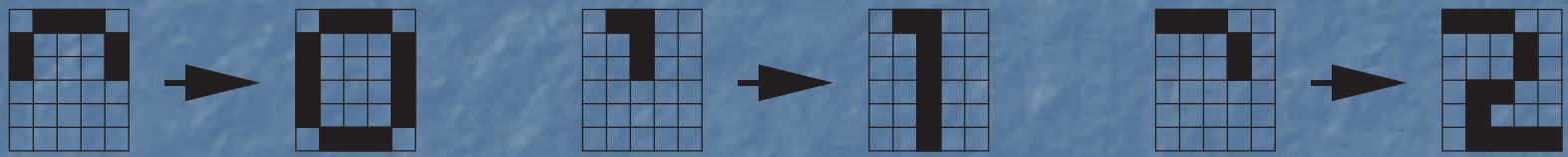
$$\mathbf{p}_1 = \begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & \ldots & 1 & -1 \end{bmatrix}^T$$



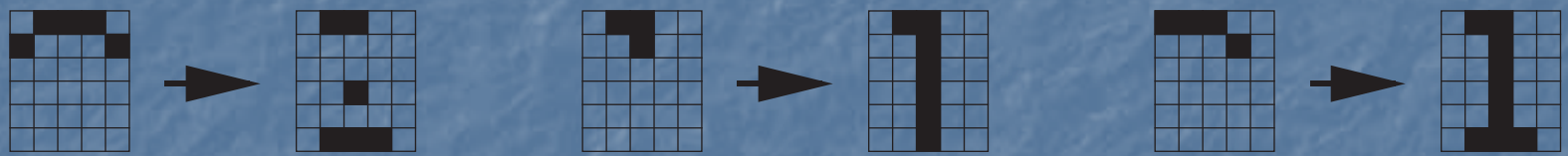
$$\mathbf{W} = \mathbf{p}_1\mathbf{p}_1^T + \mathbf{p}_2\mathbf{p}_2^T + \mathbf{p}_3\mathbf{p}_3^T$$

Tests

50% Occluded



67% Occluded



Noisy Patterns (7 pixels)

Variations of Hebbian Learning

Basic Rule: $\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q \mathbf{p}_q^T$

Learning Rate: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T$

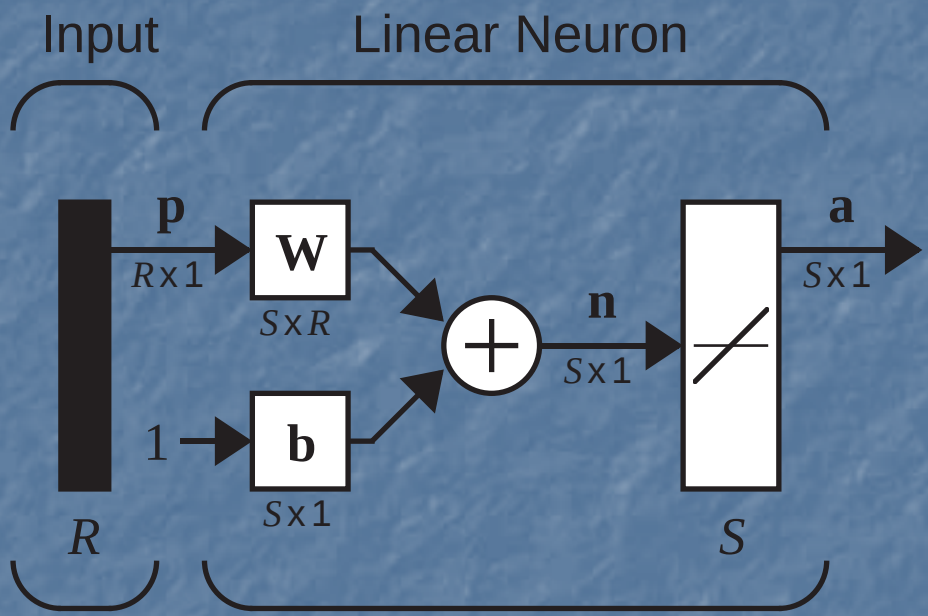
Smoothing: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T - \gamma \mathbf{W}^{old} = (1-\gamma)\mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T$

Delta Rule: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha(\mathbf{t}_q - \mathbf{a}_q)\mathbf{p}_q^T$

Unsupervised: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{a}_q \mathbf{p}_q^T$

Widrow-Hoff Learning
(LMS Algorithm)

ADALINE Network

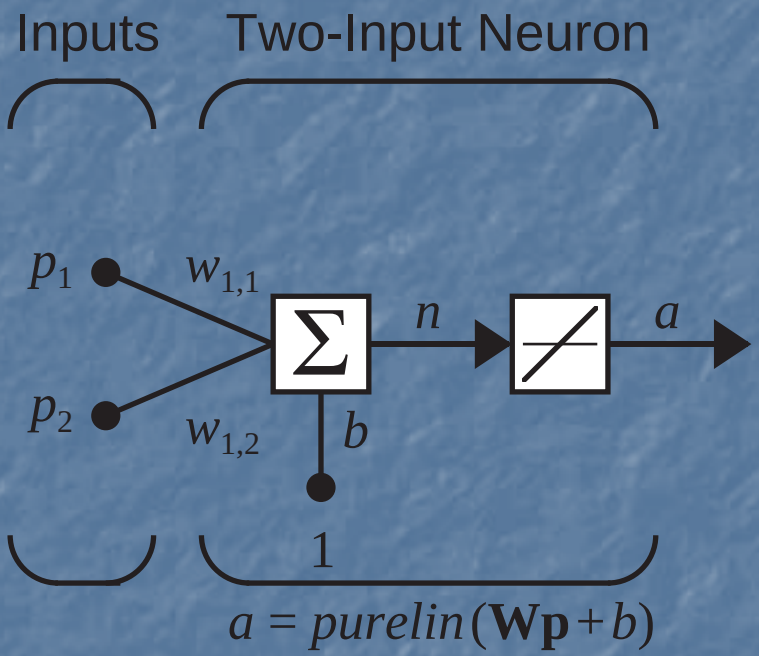


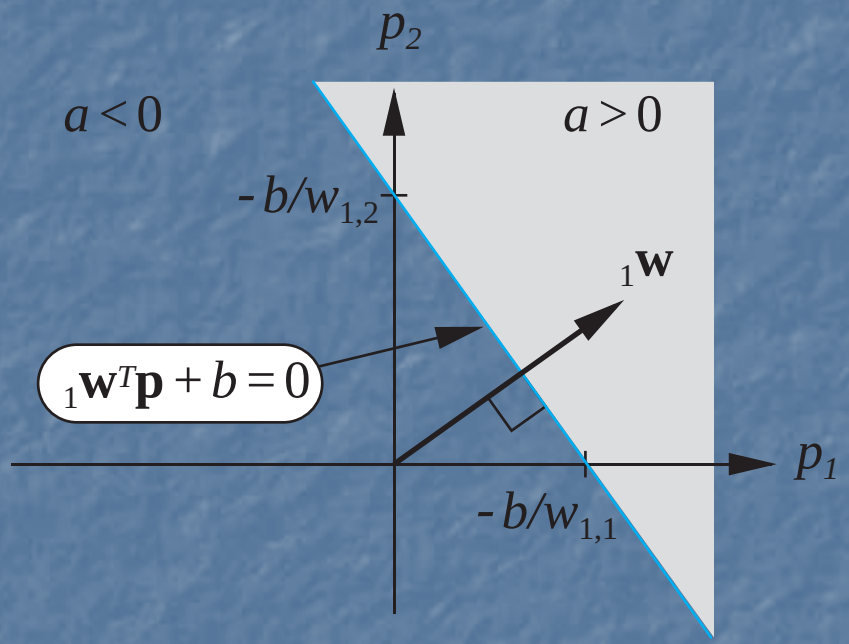
$$\mathbf{a} = \mathbf{purelin}(\mathbf{Wp}+\mathbf{b}) = \mathbf{Wp}+\mathbf{b}$$

$$a_i = purelin(n_i) = purelin({}_i\mathbf{w}^T\mathbf{p}+b_i) = {}_i\mathbf{w}^T\mathbf{p}+b_i$$

$${}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}$$

Two-Input ADALINE





$$a = purelin(n) = purelin({}_1\mathbf{w}^T\mathbf{p}+b) = {}_1\mathbf{w}^T\mathbf{p}+b$$

$$a = {}_1\mathbf{w}^T\mathbf{p}+b = w_{1,1}p_1 + w_{1,2}p_2 + b$$

Mean Square Error

Training Set:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Input: $\mathbf{p}_q$ Target: $\mathbf{t}_q$

Notation:

$$\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix} \qquad \mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$$

$$a = {}_1\mathbf{w}^T\mathbf{p} + b \Rightarrow a = \mathbf{x}^T\mathbf{z}$$

Mean Square Error:

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2]$$

Error Analysis

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2]$$

$$F(\mathbf{x}) = E[t^2 - 2t\mathbf{x}^T\mathbf{z} + \mathbf{x}^T\mathbf{z}\mathbf{z}^T\mathbf{x}]$$

$$F(\mathbf{x}) = E[t^2] - 2\mathbf{x}^T E[t\mathbf{z}] + \mathbf{x}^T E[\mathbf{z}\mathbf{z}^T]\mathbf{x}$$

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}$$

$$c = E[t^2] \qquad \mathbf{h} = E[t\mathbf{z}] \qquad \mathbf{R} = E[\mathbf{z}\mathbf{z}^T]$$

The mean square error for the ADALINE Network is a quadratic function:

$$F(\mathbf{x}) = c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$$

$$\mathbf{d} = -2\mathbf{h} \qquad \mathbf{A} = 2\mathbf{R}$$

Stationary Point

Hessian Matrix:

$$\mathbf{A} = 2\mathbf{R}$$

The correlation matrix **R** must be at least positive semidefinite. If there are any zero eigenvalues, the performance index will either have a weak minimum or else no stationary point, otherwise there will be a unique global minimum $\boldsymbol{x}^*$.

$$\nabla F(\mathbf{x}) = \nabla\left(c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}\right) = \mathbf{d} + \mathbf{A}\mathbf{x} = -2\mathbf{h} + 2\mathbf{R}\mathbf{x}$$

$$-2\mathbf{h} + 2\mathbf{R}\mathbf{x} = \mathbf{0}$$

If R is positive definite:

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$$

Approximate Steepest Descent

Approximate mean square error (one sample):

$$\hat{F}(\mathbf{x}) = (t(k) - a(k))^2 = e^2(k)$$

Approximate (stochastic) gradient:

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k)$$

$$[\nabla e^2(k)]_j = \frac{\partial e^2(k)}{\partial w_{1,j}} = 2e(k)\frac{\partial e(k)}{\partial w_{1,j}} \qquad j = 1, 2, \ldots, R$$

$$[\nabla e^2(k)]_{R+1} = \frac{\partial e^2(k)}{\partial b} = 2e(k)\frac{\partial e(k)}{\partial b}$$

Approximate Gradient Calculation

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial [t(k) - a(k)]}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} [t(k) - ({}_1\mathbf{w}^T \mathbf{p}(k) + b)]$$

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} \left[t(k) - \left(\sum_{i=1}^{R} w_{1,i} p_i(k) + b \right) \right]$$

$$\frac{\partial e(k)}{\partial w_{1,j}} = -p_j(k) \qquad \frac{\partial e(k)}{\partial b} = -1$$

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k) = -2e(k)\mathbf{z}(k)$$

LMS Algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k)$$

$${}_1\mathbf{w}(k+1) = {}_1\mathbf{w}(k) + 2\alpha e(k)\mathbf{p}(k)$$

$$b(k+1) = b(k) + 2\alpha e(k)$$

Multiple-Neuron Case

$$ {}_i\mathbf{w}(k+1) = {}_i\mathbf{w}(k) + 2\alpha e_i(k)\mathbf{p}(k) $$

$$ b_i(k+1) = b_i(k) + 2\alpha e_i(k) $$

Matrix Form:

$$ \mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha \mathbf{e}(k)\mathbf{p}^T(k) $$

$$ \mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha \mathbf{e}(k) $$

Analysis of Convergence

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k)$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha E[e(k)\mathbf{z}(k)]$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{E[t(k)\mathbf{z}(k)] - E[(\mathbf{x}_k^T\mathbf{z}(k))\mathbf{z}(k)]\}$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{E[t_k\mathbf{z}(k)] - E[(\mathbf{z}(k)\mathbf{z}^T(k))\mathbf{x}_k]\}$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{\mathbf{h} - \mathbf{R}E[\mathbf{x}_k]\}$$

$$E[\mathbf{x}_{k+1}] = [\mathbf{I} - 2\alpha\mathbf{R}]E[\mathbf{x}_k] + 2\alpha\mathbf{h}$$

For stability, the eigenvalues of this matrix must fall inside the unit circle.

Conditions for Stability

$$\left|eig([\mathbf{I} - 2\alpha\mathbf{R}])\right| = \left|1 - 2\alpha\lambda_i\right| < 1$$

(where λ_i is an eigenvalue of R)

Since $\lambda_i > 0$, $1 - 2\alpha\lambda_i < 1$.

Therefore the stability condition simplifies to:

$$1 - 2\alpha\lambda_i > -1$$

$$\alpha < 1/\lambda_i \quad \text{for all } i$$

$$0 < \alpha < 1/\lambda_{max}$$

Steady State Response

$$E[\mathbf{x}_{k+1}] = [\mathbf{I} - 2\alpha\mathbf{R}]E[\mathbf{x}_k] + 2\alpha\mathbf{h}$$

If the system is stable, then a steady state condition will be reached.

$$E[\mathbf{x}_{ss}] = [\mathbf{I} - 2\alpha\mathbf{R}]E[\mathbf{x}_{ss}] + 2\alpha\mathbf{h}$$

The solution to this equation is

$$E[\mathbf{x}_{ss}] = \mathbf{R}^{-1}\mathbf{h} = \mathbf{x}^*$$

This is also the strong minimum of the performance index.

Example

Banana $\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\}$ Apple $\left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$

$$\mathbf{R} = E[\mathbf{p}\mathbf{p}^T] = \frac{1}{2}\mathbf{p}_1\mathbf{p}_1^T + \frac{1}{2}\mathbf{p}_2\mathbf{p}_2^T$$

$$\mathbf{R} = \frac{1}{2}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} -1 & 1 & -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

$$\lambda_1 = 1.0, \qquad \lambda_2 = 0.0, \qquad \lambda_3 = 2.0$$

$$\alpha < \frac{1}{\lambda_{max}} = \frac{1}{2.0} = 0.5$$

Iteration One

Banana

$$a(0) = \mathbf{W}(0)\mathbf{p}(0) = \mathbf{W}(0)\mathbf{p}_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = 0$$

$$e(0) = t(0) - a(0) = t_1 - a(0) = -1 - 0 = -1$$

$$\mathbf{W}(1) = \mathbf{W}(0) + 2\alpha e(0)\mathbf{p}^T(0)$$

$$\mathbf{W}(1) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} + 2(0.2)(-1) \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix}$$

Iteration Two

Apple $$a(1) = \mathbf{W}(1)\mathbf{p}(1) = \mathbf{W}(1)\mathbf{p}_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4$$

$$e(1) = t(1) - a(1) = t_2 - a(1) = 1 - (-0.4) = 1.4$$

$$\mathbf{W}(2) = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} + 2(0.2)(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix}$$

Iteration Three

$$a(2) = \mathbf{W}(2)\mathbf{p}(2) = \mathbf{W}(2)\mathbf{p}_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

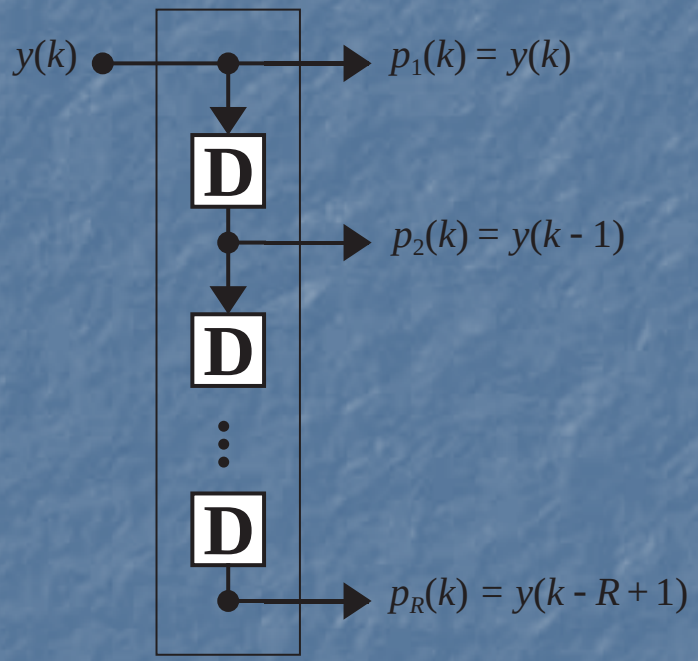
$$e(2) = t(2) - a(2) = t_1 - a(2) = -1 - (-0.64) = -0.36$$

$$\mathbf{W}(3) = \mathbf{W}(2) + 2\alpha e(2)\mathbf{p}^T(2) = \begin{bmatrix} 1.1040 & 0.0160 & -0.0160 \end{bmatrix}$$

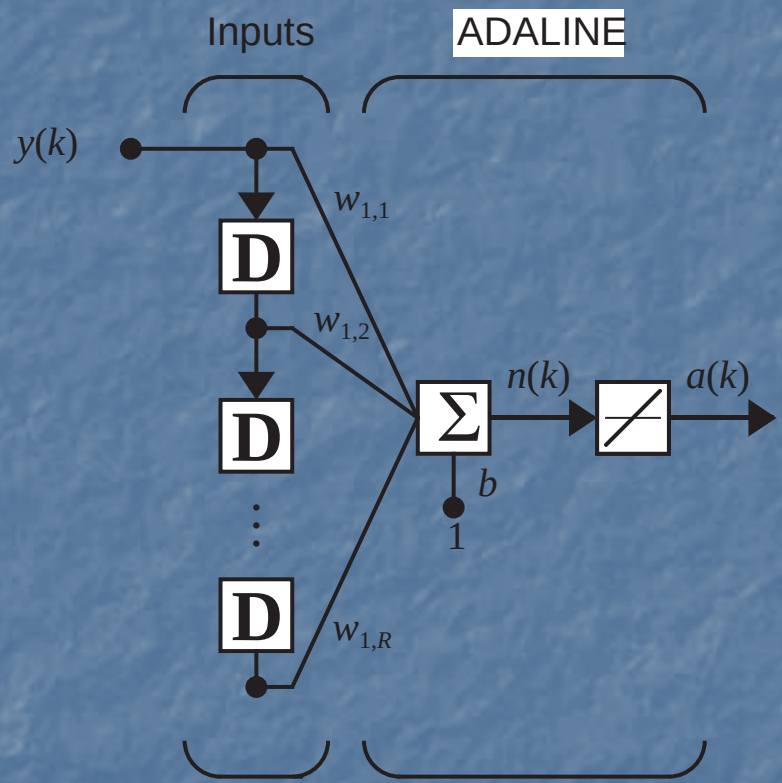
$$\mathbf{W}(\infty) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

Adaptive Filtering

Tapped Delay Line



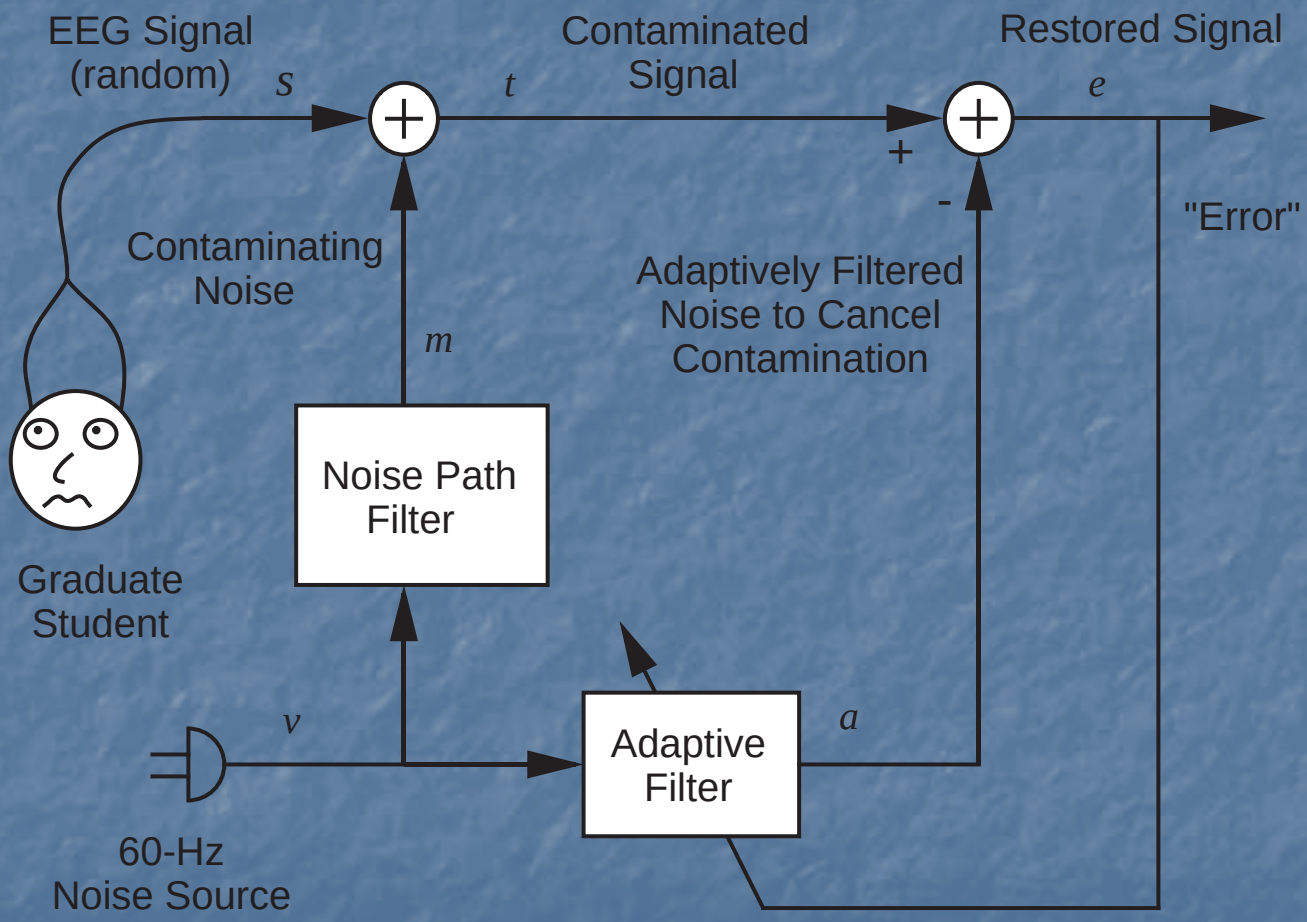
Adaptive Filter



$$a(k) = purelin(\mathbf{W}\mathbf{p}(k) + b)$$

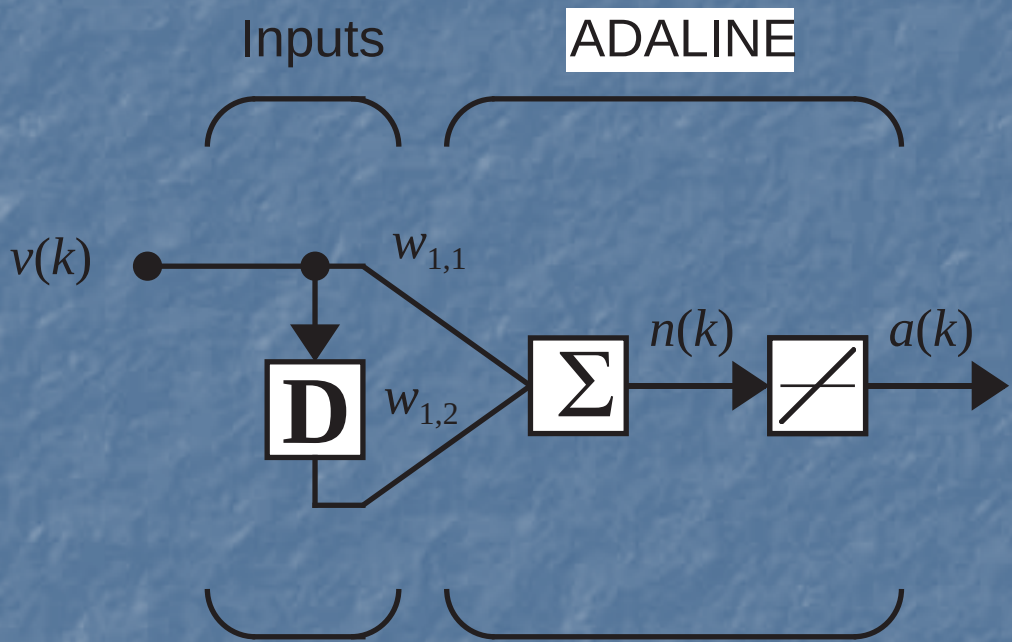
$$a(k) = purelin(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1,i}\, y(k - i + 1) + b$$

Example: Noise Cancellation



Adaptive Filter Adjusts to Minimize Error (and in doing this removes 60-Hz noise from contaminated signal)

Noise Cancellation Adaptive Filter



$$a(k) = w_{1,1}\, v(k) + w_{1,2}\, v(k - 1)$$

Correlation Matrix

$$\mathbf{R} = [\mathbf{z}\mathbf{z}^T] \qquad \mathbf{h} = E[t\mathbf{z}]$$

$$\mathbf{z}(k) = \begin{bmatrix} v(k) \\ v(k-1) \end{bmatrix}$$

$$t(k) = s(k) + m(k)$$

$$\mathbf{R} = \begin{bmatrix} E[v^2(k)] & E[v(k)v(k-1)] \\ E[v(k-1)v(k)] & E[v^2(k-1)] \end{bmatrix}$$

$$\mathbf{h} = \begin{bmatrix} E[(s(k) + m(k))v(k)] \\ E[(s(k) + m(k))v(k-1)] \end{bmatrix}$$

Signals

$$v(k) = 1.2 \sin\left(\frac{2\pi k}{3}\right) \qquad m(k) = 1.2 \sin\left(\frac{2\pi k}{3} - \frac{3\pi}{4}\right)$$

$$E[v^2(k)] = (1.2)^2 \frac{1}{3} \sum_{k=1}^{3} \left(\sin\left(\frac{2\pi k}{3}\right)\right)^2 = (1.2)^2 0.5 = 0.72$$

$$E[v^2(k-1)] = E[v^2(k)] = 0.72$$

$$E[v(k)v(k-1)] = \frac{1}{3} \sum_{k=1}^{3} \left(1.2 \sin\frac{2\pi k}{3}\right)\left(1.2 \sin\frac{2\pi(k-1)}{3}\right)$$

$$= (1.2)^2 0.5 \cos\left(\frac{2\pi}{3}\right) = -0.36$$

$$\mathbf{R} = \begin{bmatrix} 0.72 & -0.36 \\ -0.36 & 0.72 \end{bmatrix}$$

Stationary Point

$$E[(s(k)+m(k))v(k)] = E[s(k)v(k)] + E[m(k)v(k)]$$

$$\Downarrow$$

$$0$$

$$E[m(k)v(k)] = \frac{1}{3}\sum_{k=1}^{3}\left(1.2\ \sin\left(\frac{2\pi k}{3}-\frac{3\pi}{4}\right)\right)\left(1.2\sin\frac{2\pi k}{3}\right) = -0.51$$

$$E[(s(k)+m(k))v(k-1)] = E[s(k)v(k-1)] + E[m(k)v(k-1)]$$

$$\Downarrow$$

$$0$$

$$E[m(k)v(k-1)] = \frac{1}{3}\sum_{k=1}^{3}\left(1.2\ \sin\left(\frac{2\pi k}{3}-\frac{3\pi}{4}\right)\right)\left(1.2\ \sin\frac{2\pi(k-1)}{3}\right) = 0.70$$

$$\mathbf{h} = \begin{bmatrix} E[(s(k)+m(k))v(k)] \\ E[(s(k)+m(k))v(k-1)] \end{bmatrix} \Rightarrow \mathbf{h} = \begin{bmatrix} -0.51 \\ 0.70 \end{bmatrix}$$

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h} = \begin{bmatrix} 0.72 & -0.36 \\ -0.36 & 0.72 \end{bmatrix}^{-1}\begin{bmatrix} -0.51 \\ 0.70 \end{bmatrix} = \begin{bmatrix} -0.30 \\ 0.82 \end{bmatrix}$$

Performance Index

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}$$

$$c = E[t^2(k)] = E[(s(k) + m(k))^2]$$

$$c = E[s^2(k)] + 2E[s(k)m(k)] + E[m^2(k)]$$

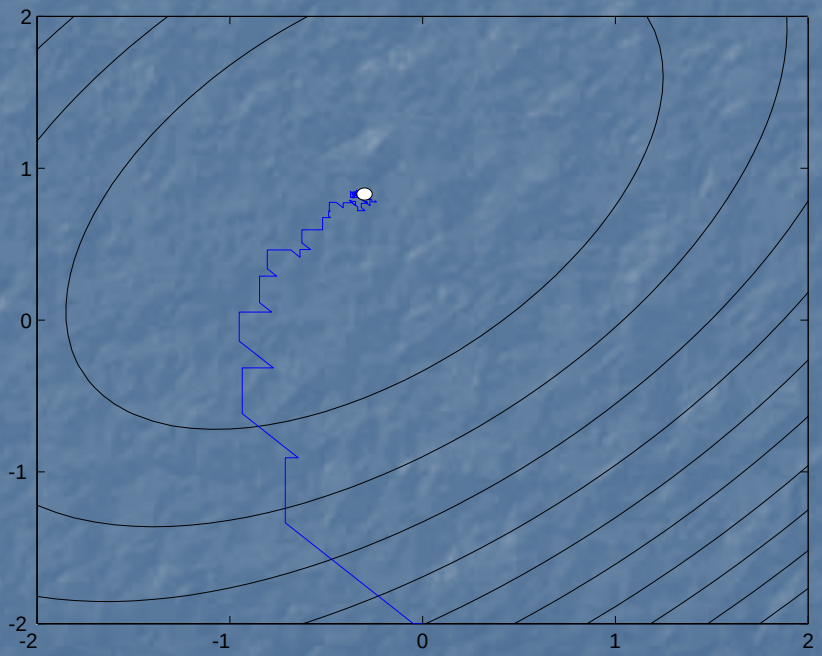
$$E[s^2(k)] = \frac{1}{0.4}\int_{-0.2}^{0.2} s^2 ds = \frac{1}{3(0.4)} s^3\Big|_{-0.2}^{0.2} = 0.0133$$

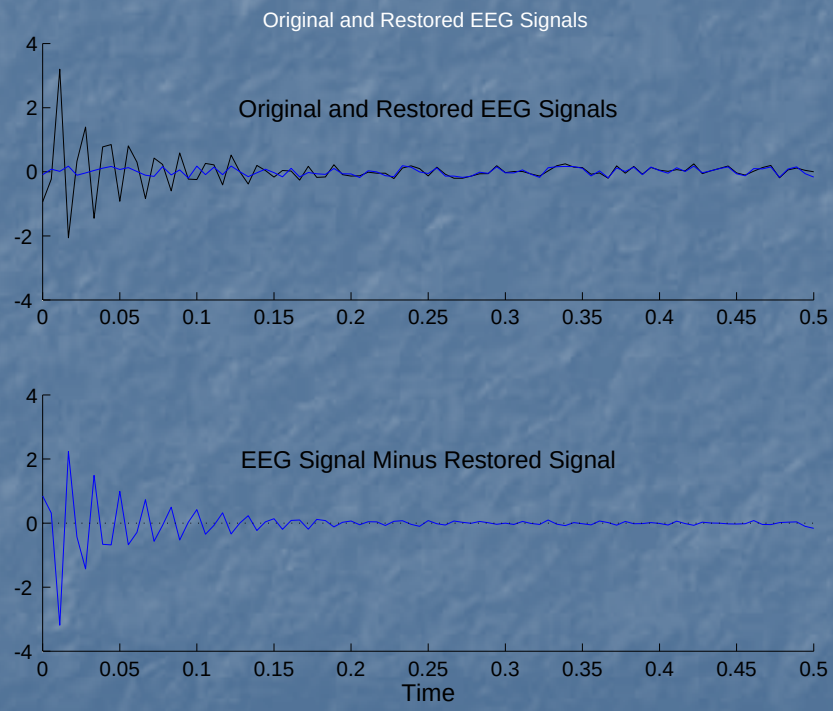
$$E[m^2(k)] = \frac{1}{3}\sum_{k=1}^{3}\left\{1.2\ \sin\left(\frac{2\pi}{3} - \frac{3\pi}{4}\right)\right\}^2 = 0.72$$

$$c = 0.0133 + 0.72 = 0.7333$$

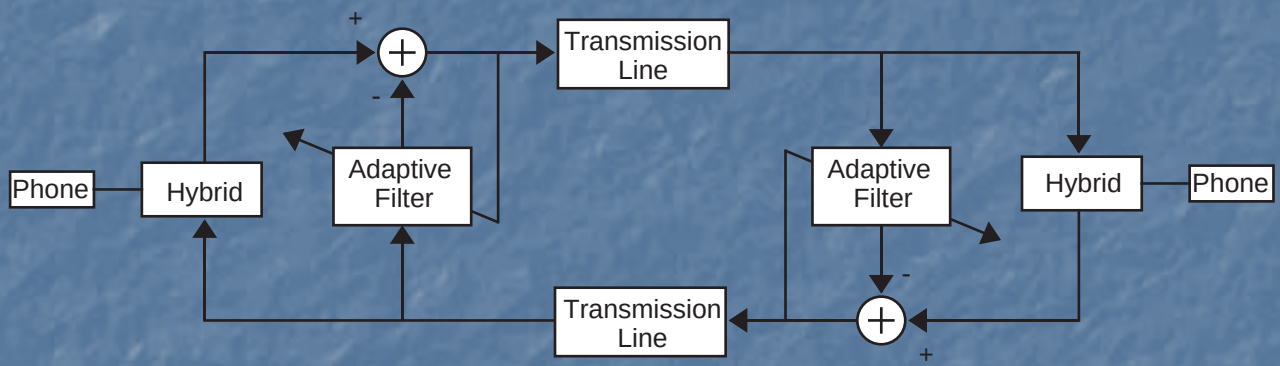
$$F(\mathbf{x}^*) = 0.7333 - 2(0.72) + 0.72 = 0.0133$$

LMS Response



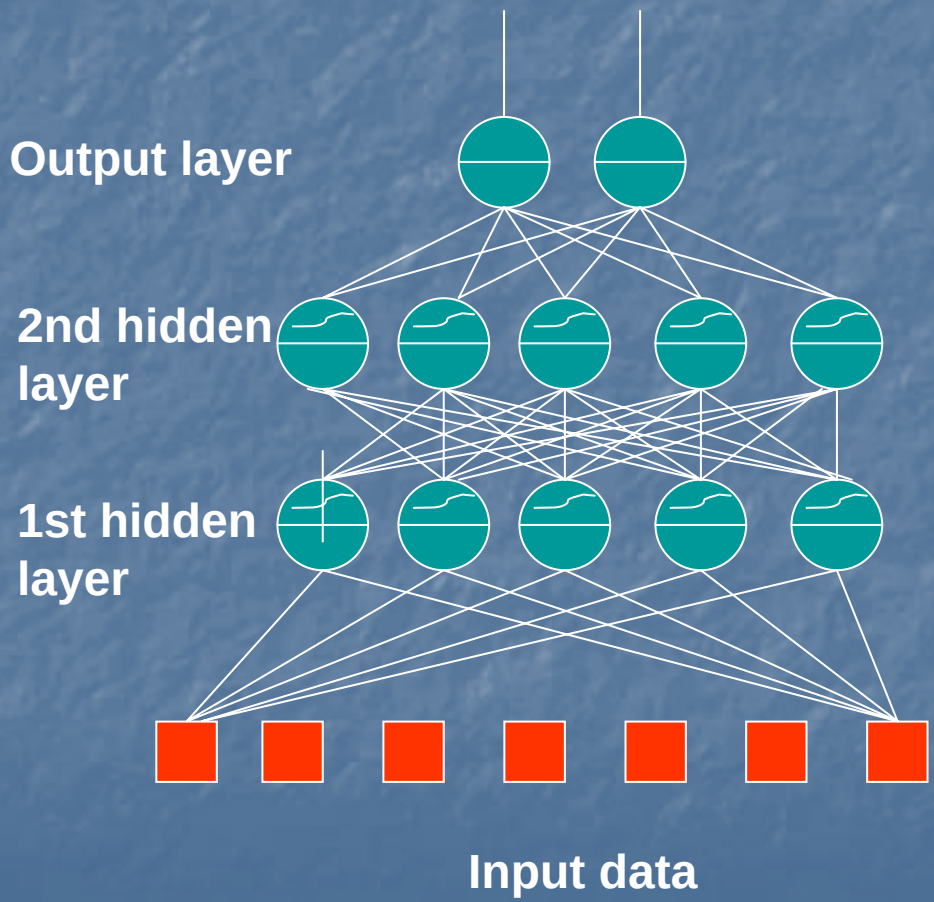


Echo Cancellation



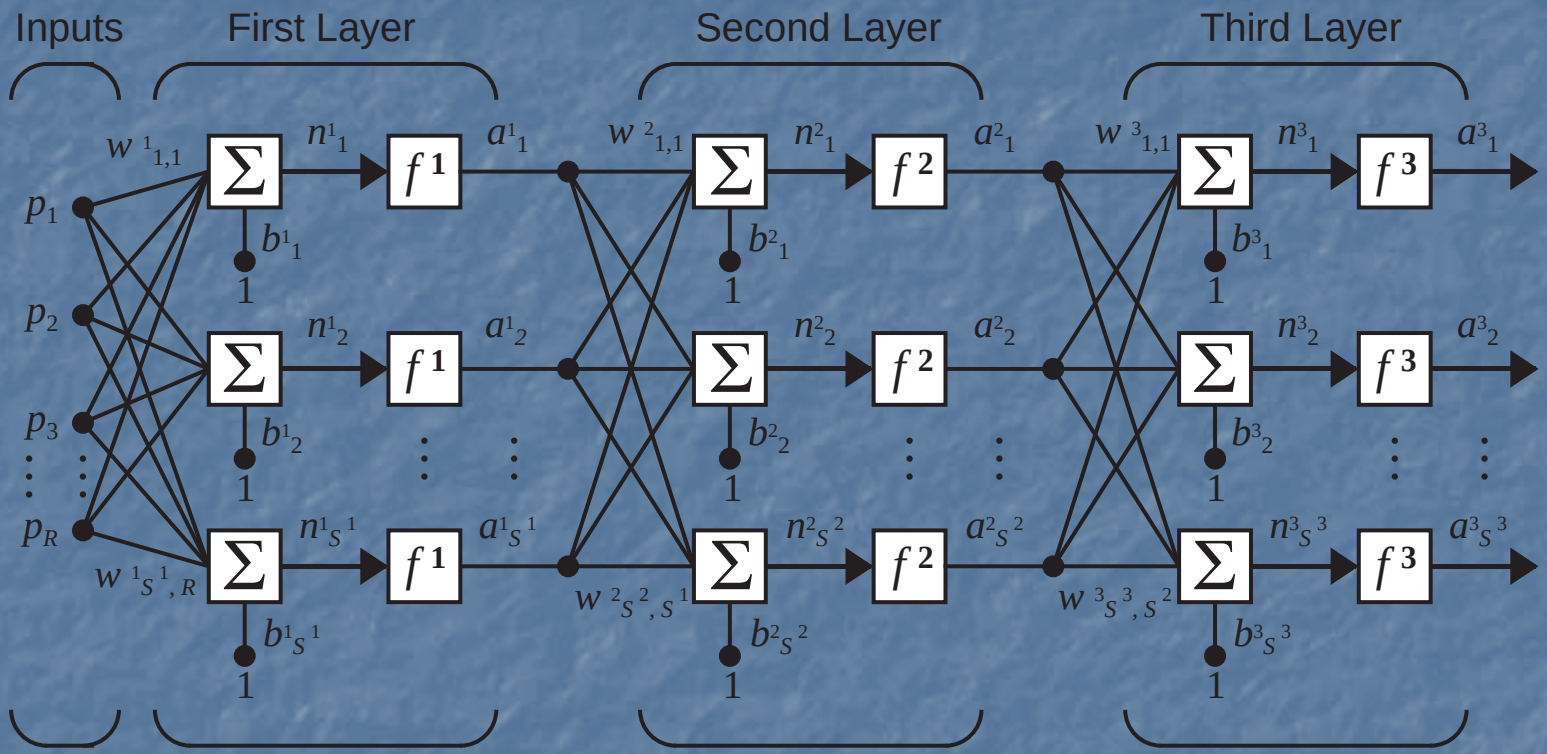
Backpropagation

Multi-Layer Perceptron



- One or more hidden layers
- Sigmoid activations functions

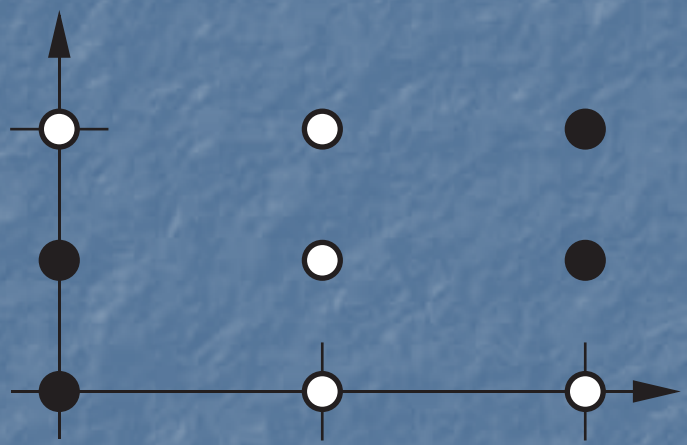
Multilayer Perceptron



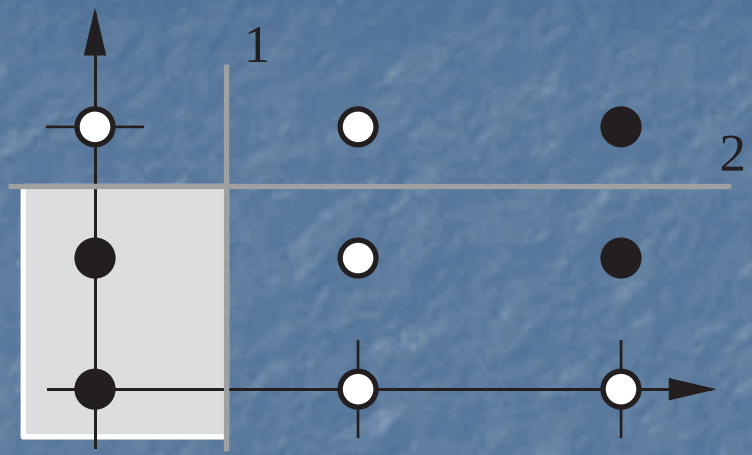
$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

$R - S^1 - S^2 - S^3$ Network

Example



Elementary Decision Boundaries



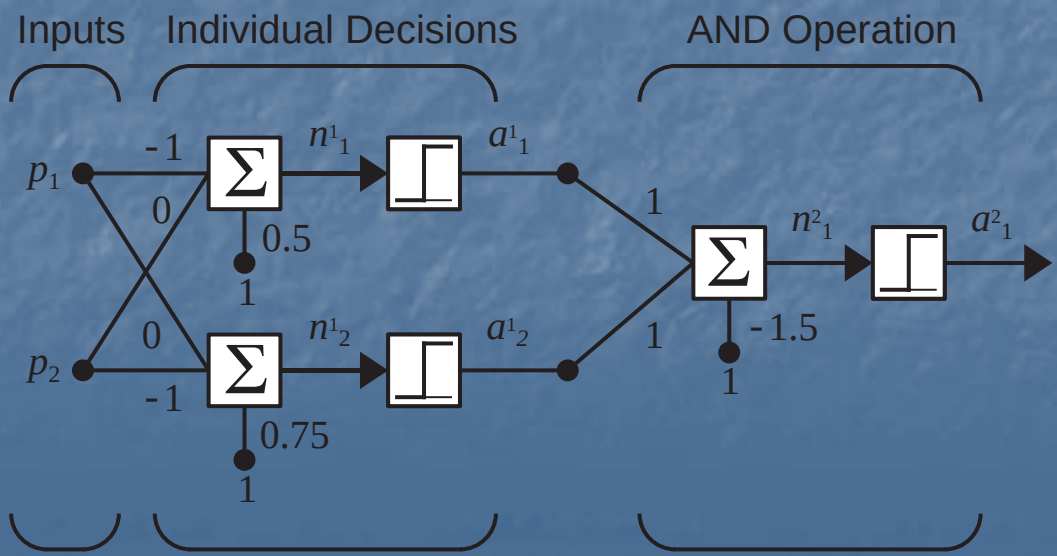
First Boundary:

$$a_1^1 = hardlim\left(\begin{bmatrix}-1 & 0\end{bmatrix}\mathbf{p} + 0.5\right)$$

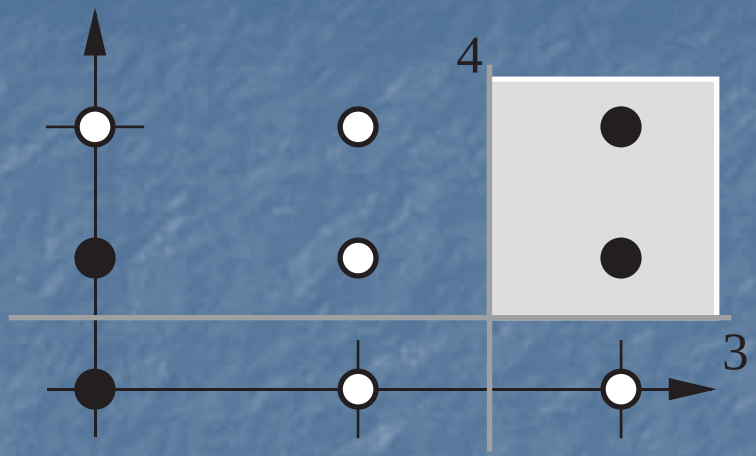
Second Boundary:

$$a_2^1 = hardlim\left(\begin{bmatrix}0 & -1\end{bmatrix}\mathbf{p} + 0.75\right)$$

First Subnetwork



Elementary Decision Boundaries



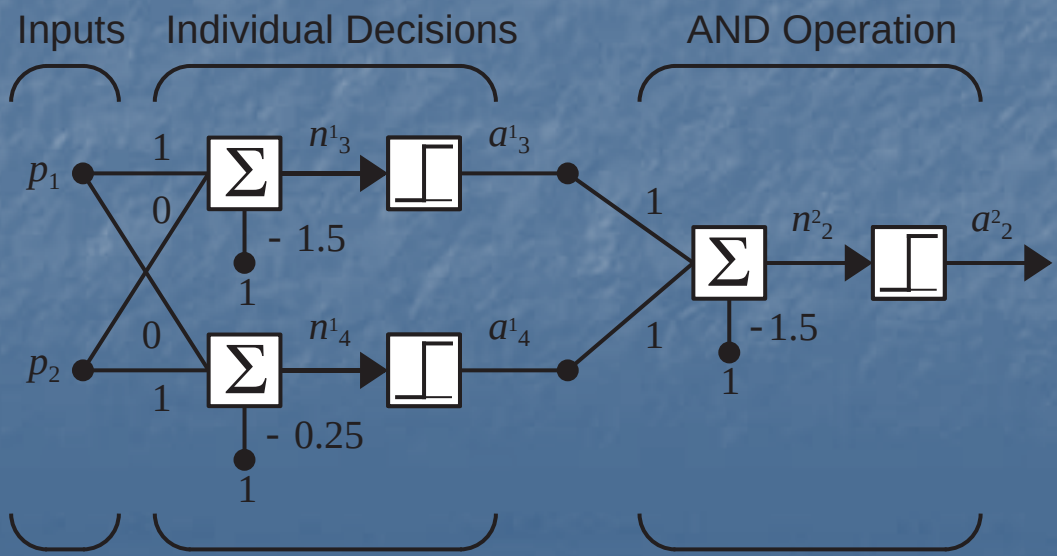
Third Boundary:

$$a_3^1 = hardlim(\begin{bmatrix}1 & 0\end{bmatrix}\mathbf{p} - 1.5)$$

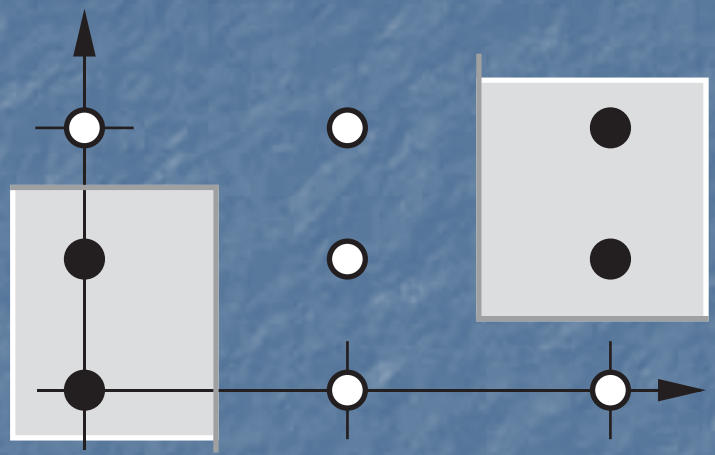
Fourth Boundary:

$$a_4^1 = hardlim(\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{p} - 0.25)$$

Second Subnetwork



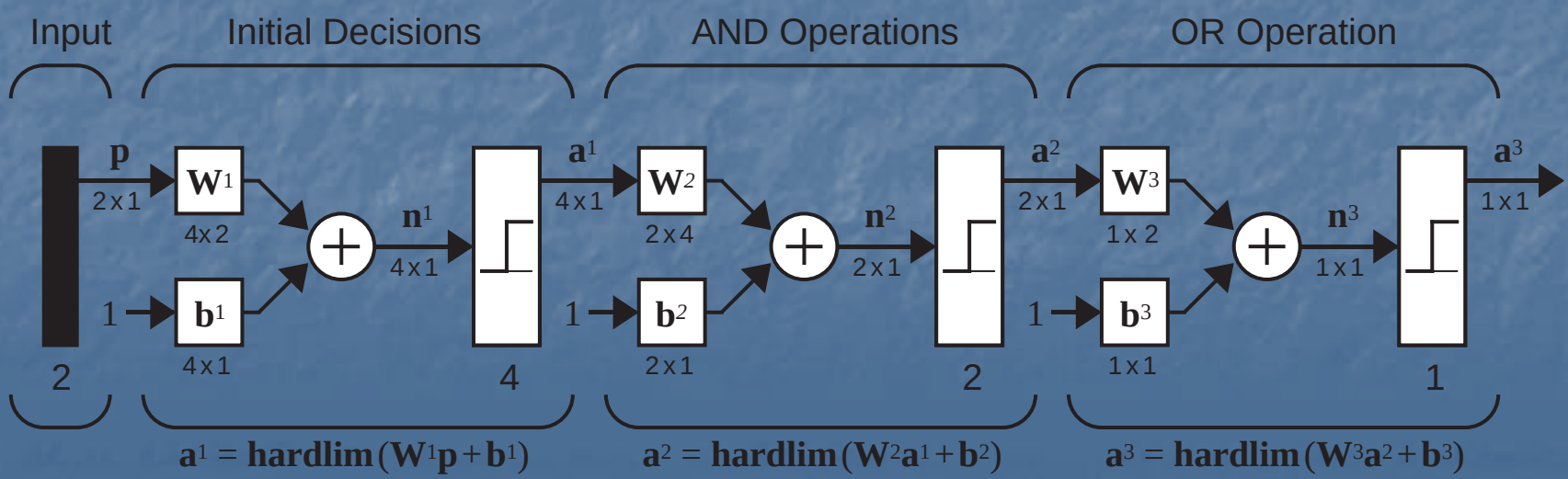
Total Network



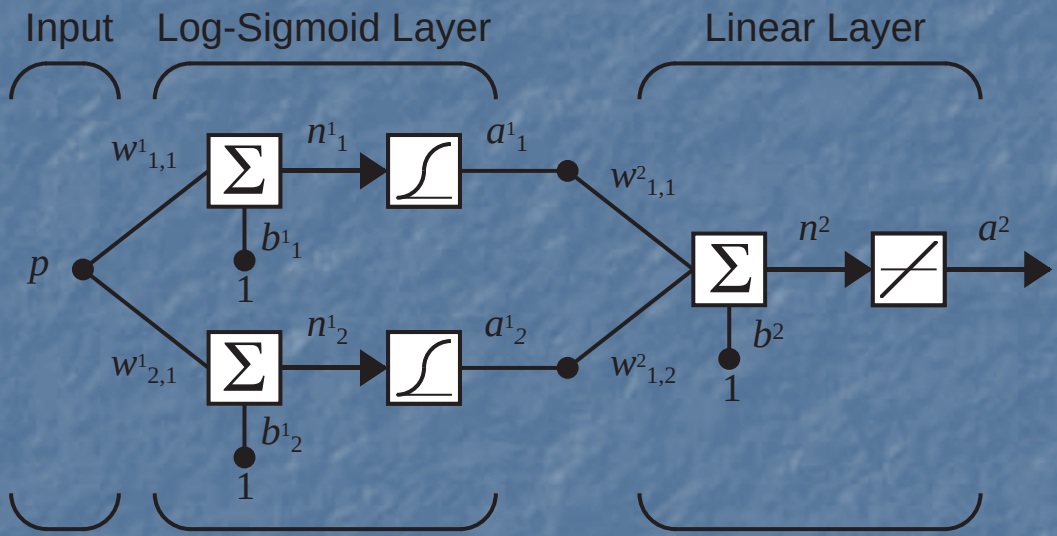
$$\mathbf{W}^1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{b}^1 = \begin{bmatrix} 0.5 \\ 0.75 \\ -1.5 \\ -0.25 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \mathbf{b}^2 = \begin{bmatrix} -1.5 \\ -1.5 \end{bmatrix}$$

$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \mathbf{b}^3 = \begin{bmatrix} -0.5 \end{bmatrix}$$



Function Approximation Example



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

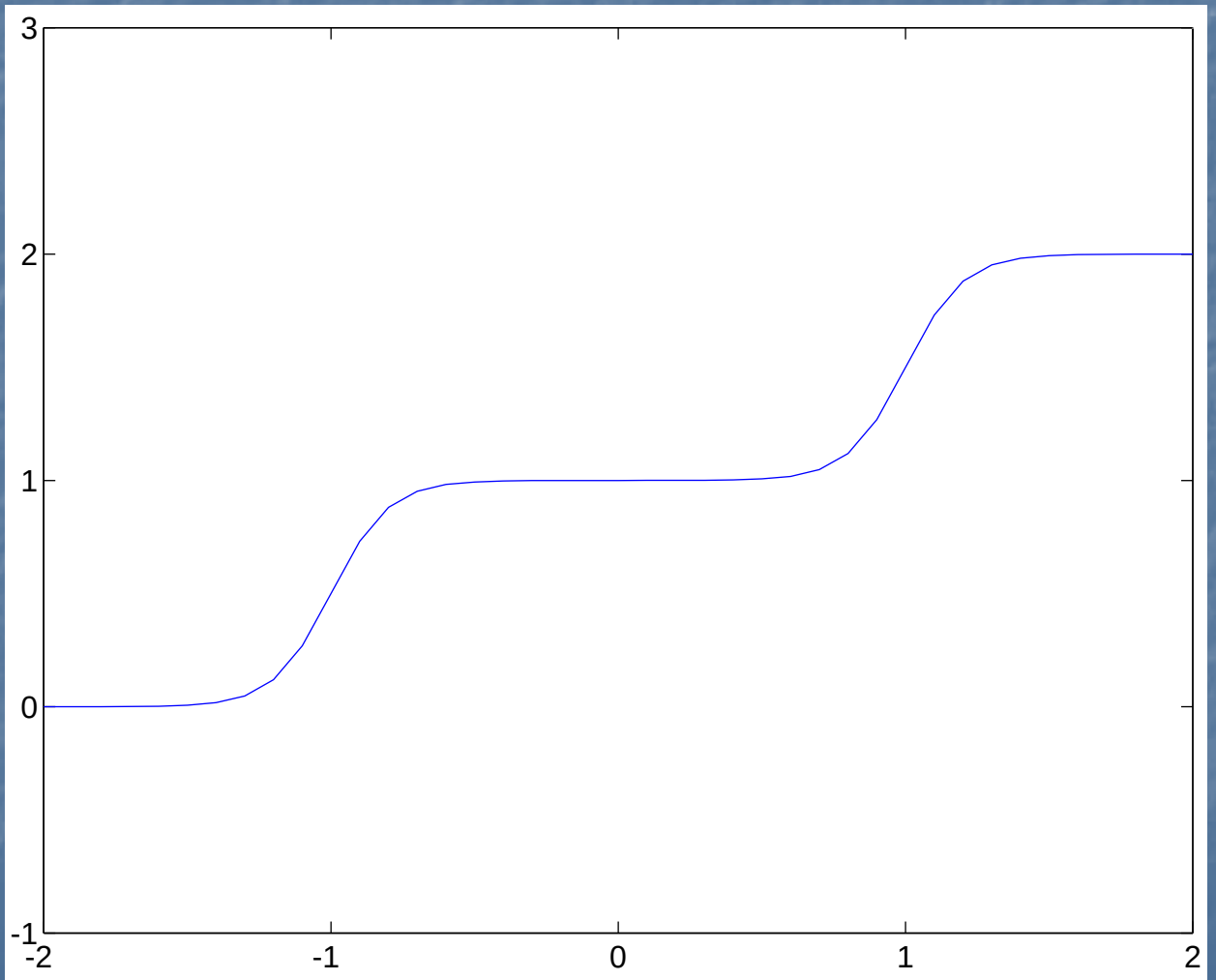
$$f^2(n) = n$$

Nominal Parameter Values

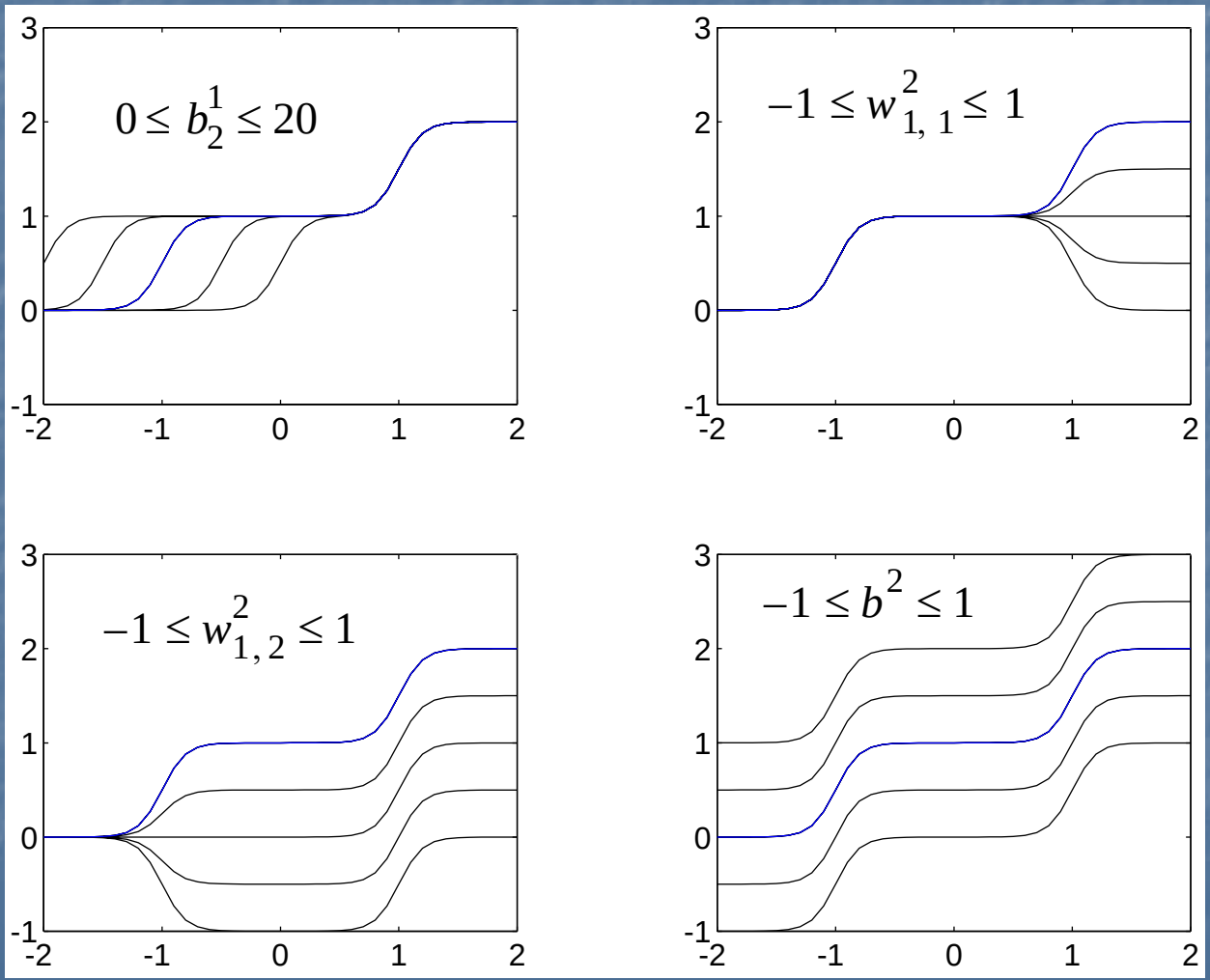
$$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -10 \qquad b^1_2 = 10$$

$$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = 0$$

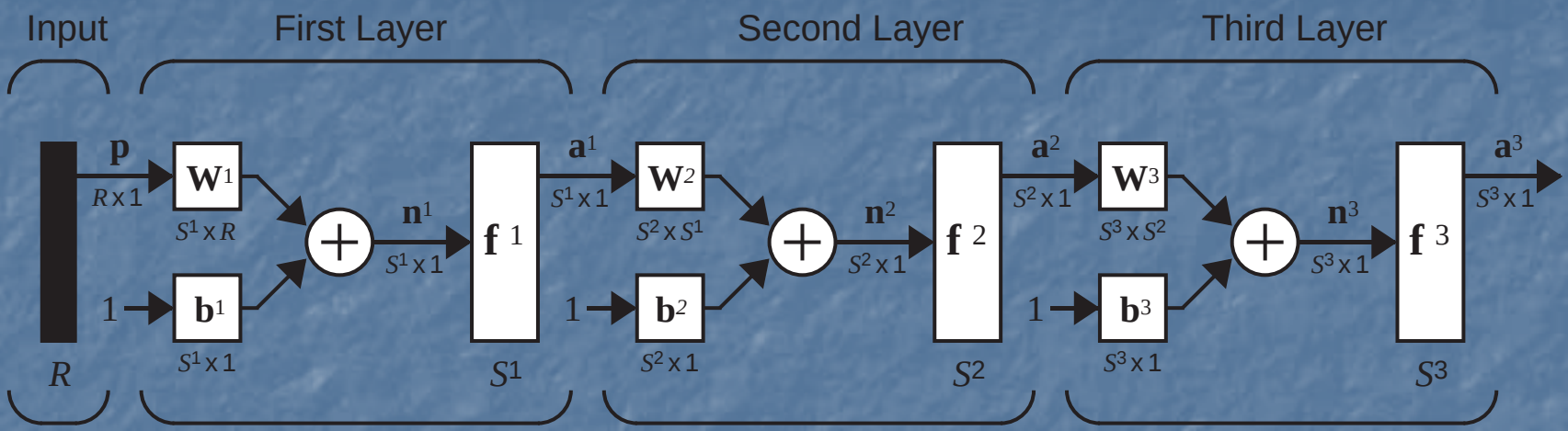
Nominal Response



Parameter Variations



Multilayer Network



$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3)$$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a} = \mathbf{a}^M$$

Performance Index

Training Set

$$\{\mathbf{p}_1,\mathbf{t}_1\}, \{\mathbf{p}_2,\mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q,\mathbf{t}_Q\}$$

Mean Square Error

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2]$$

Vector Case

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$$

Approximate Mean Square Error (Single Sample)

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k)$$

Approximate Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial\hat{F}}{\partial w^m_{i,j}} \qquad b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial\hat{F}}{\partial b^m_i}$$

Chain Rule

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}$$

Example

$$f(n) = \cos(n) \qquad n = e^{2w} \qquad f(n(w)) = \cos(e^{2w})$$

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (-\sin(n))(2e^{2w}) = (-\sin(e^{2w}))(2e^{2w})$$

Application to Gradient Calculation

$$\frac{\partial \hat{F}}{\partial w_{i,j}^{m}} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^{m}} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m}$$

Gradient Calculation

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$$

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1$$

Sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

Gradient

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = s_i^m$$

Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha s^m_i a^{m-1}_j \qquad b^m_i(k+1) = b^m_i(k) - \alpha s^m_i$$

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n^m_1} \\ \frac{\partial \hat{F}}{\partial n^m_2} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n^m_{S^m}} \end{bmatrix}$$

Next Step: Compute the Sensitivities (Backpropagation)

Jacobian Matrix

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^{m}} & \frac{\partial n_1^{m+1}}{\partial n_2^{m}} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^{m}} \\ \frac{\partial n_2^{m+1}}{\partial n_1^{m}} & \frac{\partial n_2^{m+1}}{\partial n_2^{m}} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^{m}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^{m}} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^{m}} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^{m}} \end{bmatrix}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^{m}} = \frac{\partial \left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^{m} + b_i^{m+1} \right)}{\partial n_j^{m}} = w_{i,j}^{m+1} \frac{\partial a_j^{m}}{\partial n_j^{m}}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^{m}} = w_{i,j}^{m+1} \frac{\partial f^m\left(n_j^m\right)}{\partial n_j^{m}} = w_{i,j}^{m+1} \dot{f}^m\left(n_j^m\right)$$

$$\dot{f}^m\left(n_j^m\right) = \frac{\partial f^m\left(n_i^m\right)}{\partial n_j^{m}}$$

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m\left(\mathbf{n}^m\right)$$

$$\dot{\mathbf{F}}^m\left(n^m\right) = \begin{bmatrix} \dot{f}^m\left(n_1^m\right) & 0 & \cdots & 0 \\ 0 & \dot{f}^m\left(n_2^m\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^m\left(n_{S^m}^m\right) \end{bmatrix}$$

Backpropagation (Sensitivities)

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m\left(\mathbf{n}^m\right)\left(\mathbf{W}^{m+1}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m\left(\mathbf{n}^m\right)\left(\mathbf{W}^{m+1}\right)^T \mathbf{s}^{m+1}$$

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer.

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1$$

Initialization (Last Layer)

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T (\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M} (t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}$$

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M\left(n_i^M\right)}{\partial n_i^M} = \dot{f}^M\left(n_i^M\right)$$

$$s_i^M = -2\left(t_i - a_i\right)\dot{f}^M\left(n_i^M\right)$$

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M\left(\mathbf{n}^M\right)\left(\mathbf{t}-\mathbf{a}\right)$$

Summary

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Backpropagation

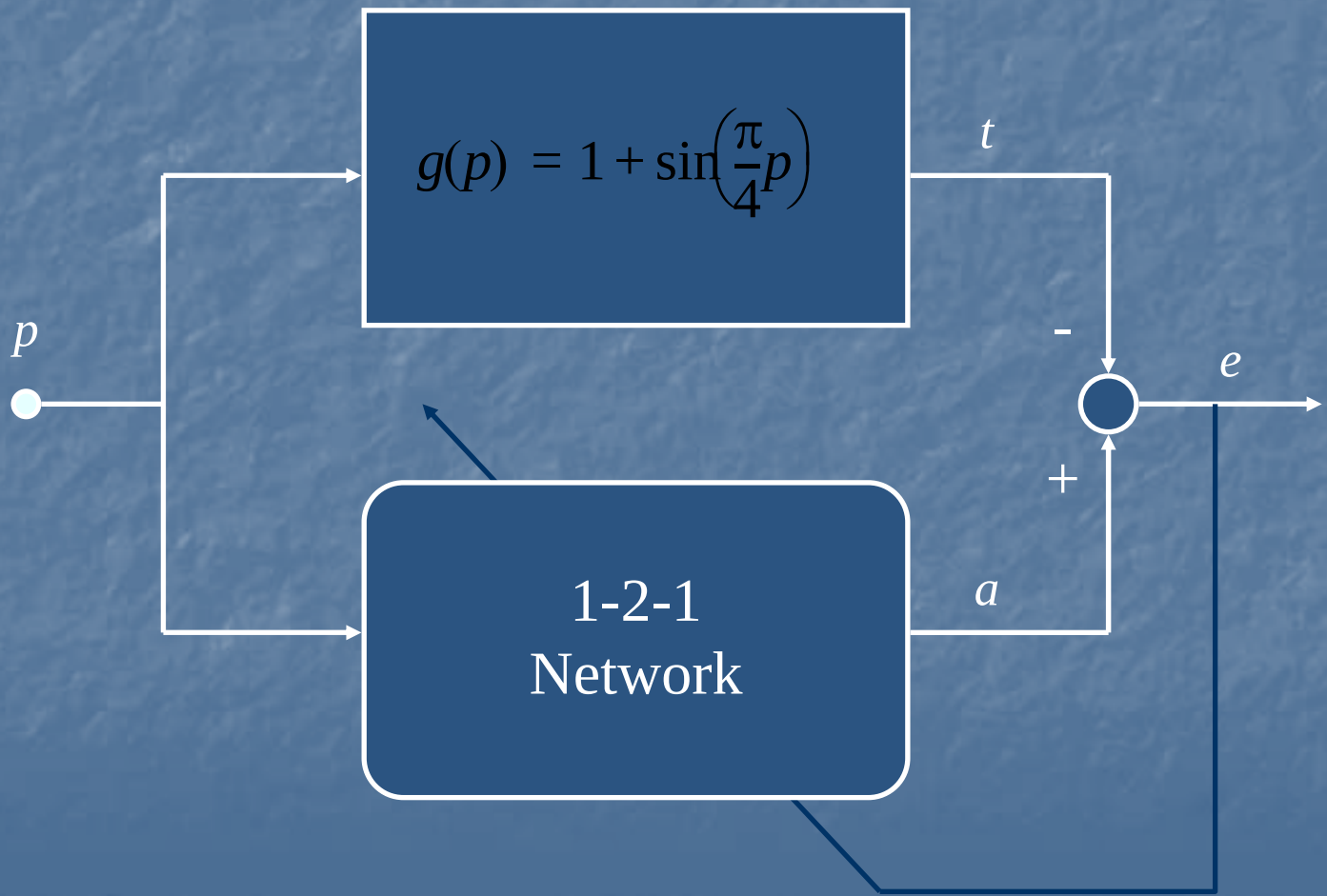
$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \qquad m = M-1,\ldots,2,1$$

Weight Update

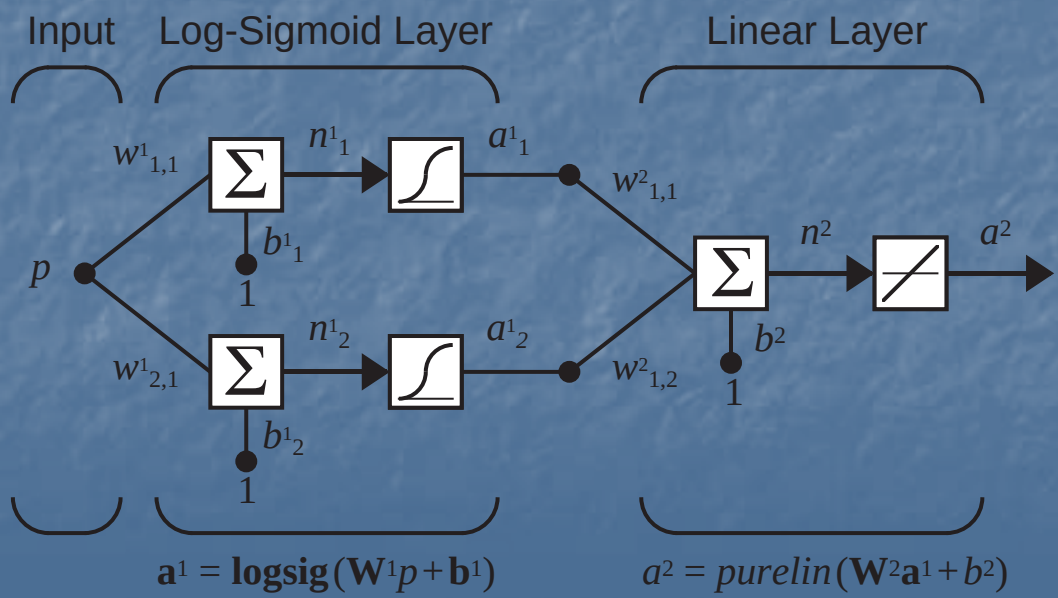
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$$

Example: Function Approximation



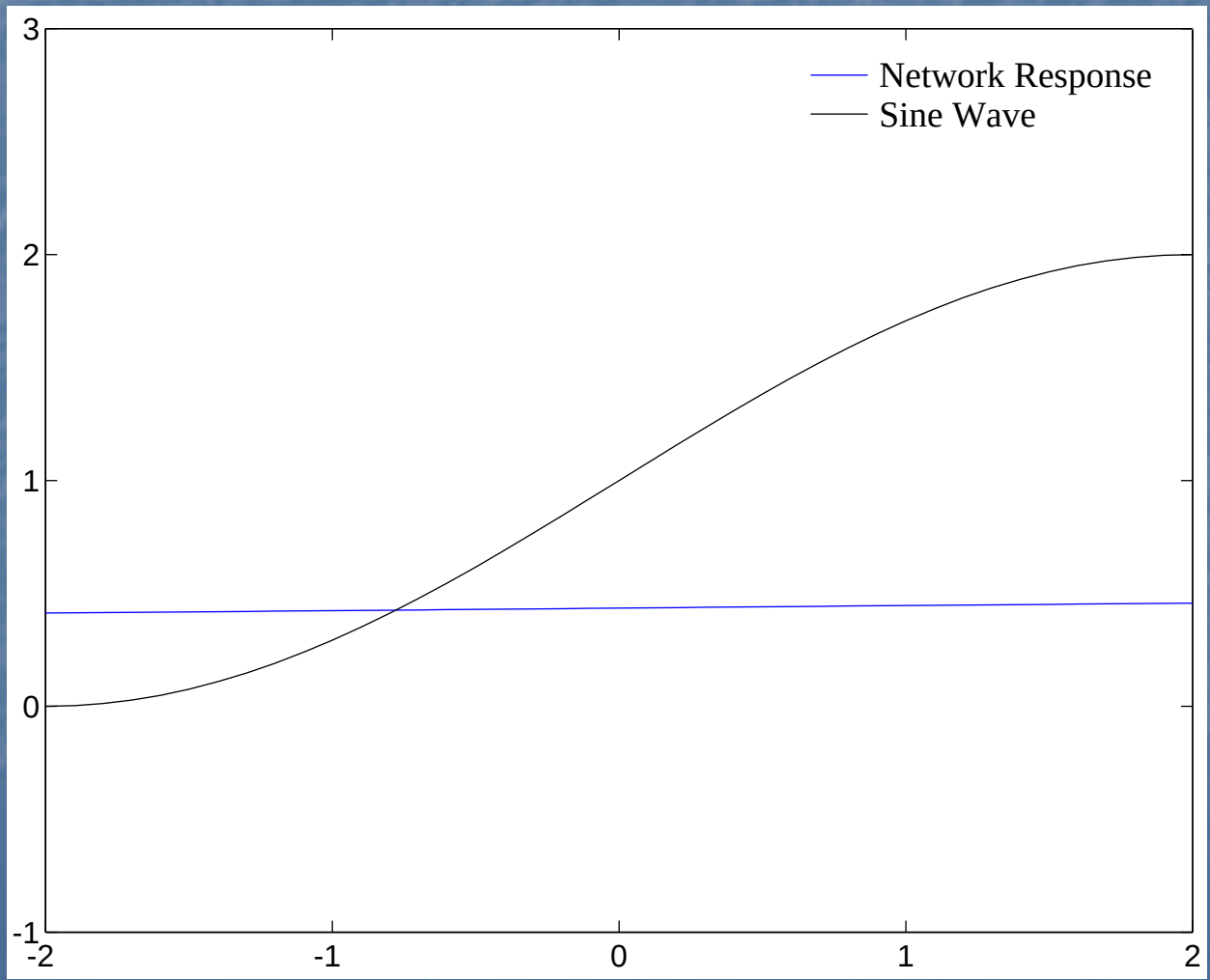
Network


p
1-2-1
Network
a


Input
Log-Sigmoid Layer
Linear Layer
p
1
1
$\mathbf{a}^1 = \mathbf{logsig}(\mathbf{W}^1 p + \mathbf{b}^1)$
$a^2 = purelin(\mathbf{W}^2 \mathbf{a}^1 + b^2)$

Initial Conditions

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} \quad \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \quad \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} \quad \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$



Forward Propagation

$$a^0 = p = 1$$

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{logsig}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}\begin{bmatrix}1\end{bmatrix} + \begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right) = \mathbf{logsig}\left(\begin{bmatrix}-0.75\\-0.54\end{bmatrix}\right)$$

$$\mathbf{a}^1 = \begin{bmatrix}\dfrac{1}{1+e^{0.75}}\\[2ex]\dfrac{1}{1+e^{0.54}}\end{bmatrix} = \begin{bmatrix}0.321\\0.368\end{bmatrix}$$

$$a^2 = \mathrm{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix}0.09 & -0.17\end{bmatrix}\begin{bmatrix}0.321\\0.368\end{bmatrix} + \begin{bmatrix}0.48\end{bmatrix}\right) = \begin{bmatrix}0.446\end{bmatrix}$$

$$e = t - a = \left\{1 + \sin\left(\frac{\pi}{4}p\right)\right\} - a^2 = \left\{1 + \sin\left(\frac{\pi}{4}1\right)\right\} - 0.446 = 1.261$$

Transfer Function Derivatives

$$f^{1}(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^{2}} = \left(1-\frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^{1})(a^{1})$$

$$f^{2}(n) = \frac{d}{dn}(n) = 1$$

Backpropagation

$$\mathbf{s}^2 = -2\dot{\mathbf{F}}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a}) = -2\left[\dot{f}^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522$$

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix}(1-a_1^1)(a_1^1) & 0\\ 0 & (1-a_2^1)(a_2^1)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}(1-0.321)(0.321) & 0\\ 0 & (1-0.368)(0.368)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}0.218 & 0\\ 0 & 0.233\end{bmatrix}\begin{bmatrix}-0.227\\ 0.429\end{bmatrix} = \begin{bmatrix}-0.0495\\ 0.0997\end{bmatrix}$$

Weight Update

$$\alpha = 0.1$$

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2 (\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} \begin{bmatrix} 0.321 & 0.368 \end{bmatrix}$$

$$\mathbf{W}^2(1) = \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix}$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix}$$

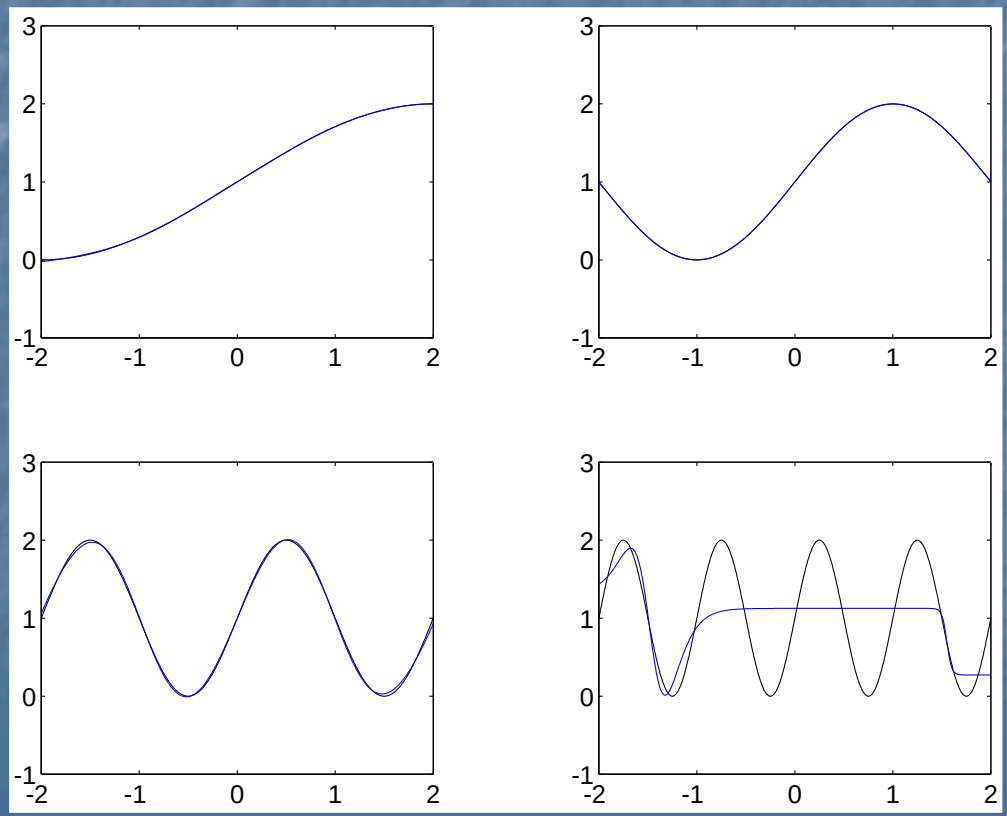
$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}$$

Choice of Architecture

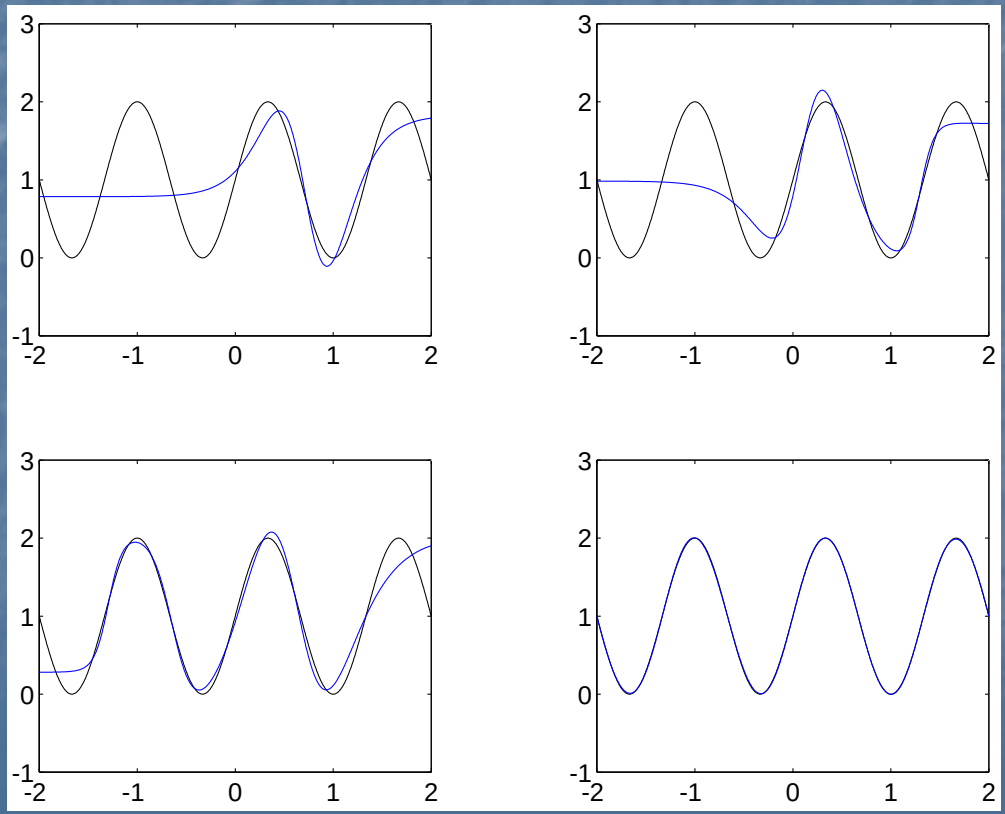
$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right)$$

1-3-1 Network



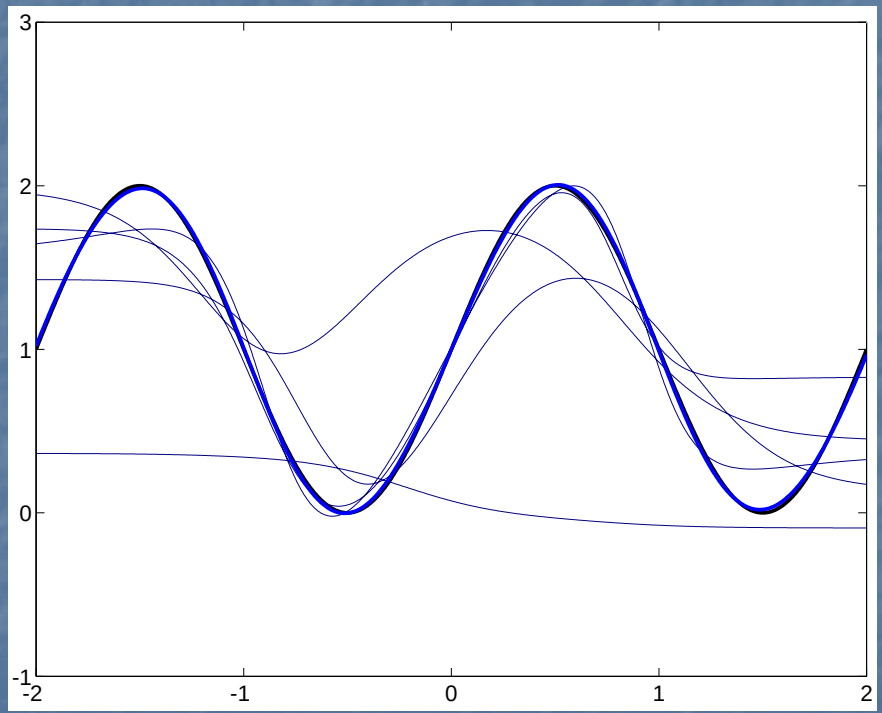
Choice of Network Architecture

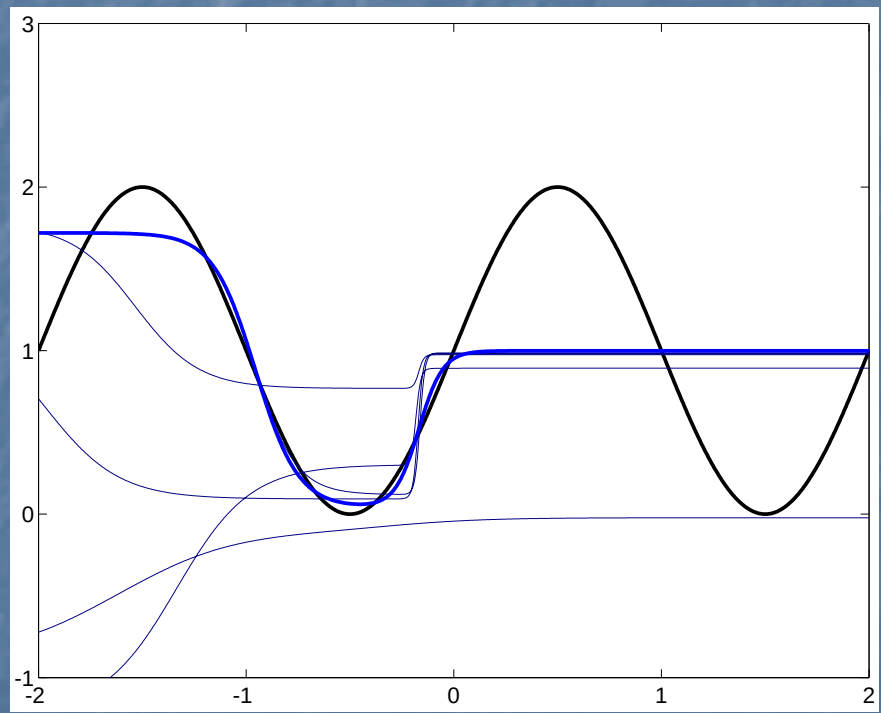
$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right)$$



Convergence

$$g(p) = 1 + \sin(\pi p)$$

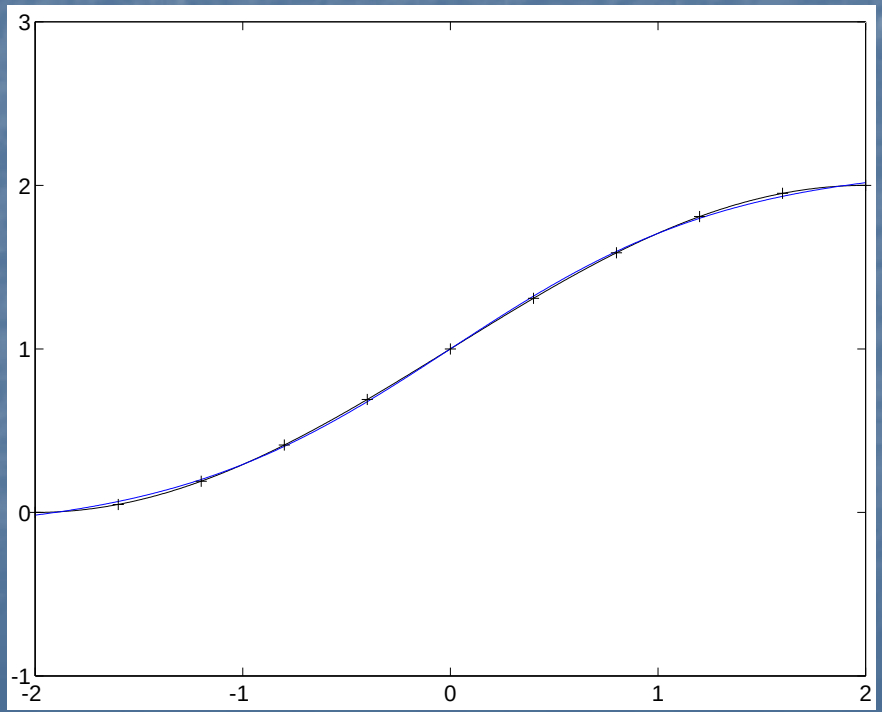


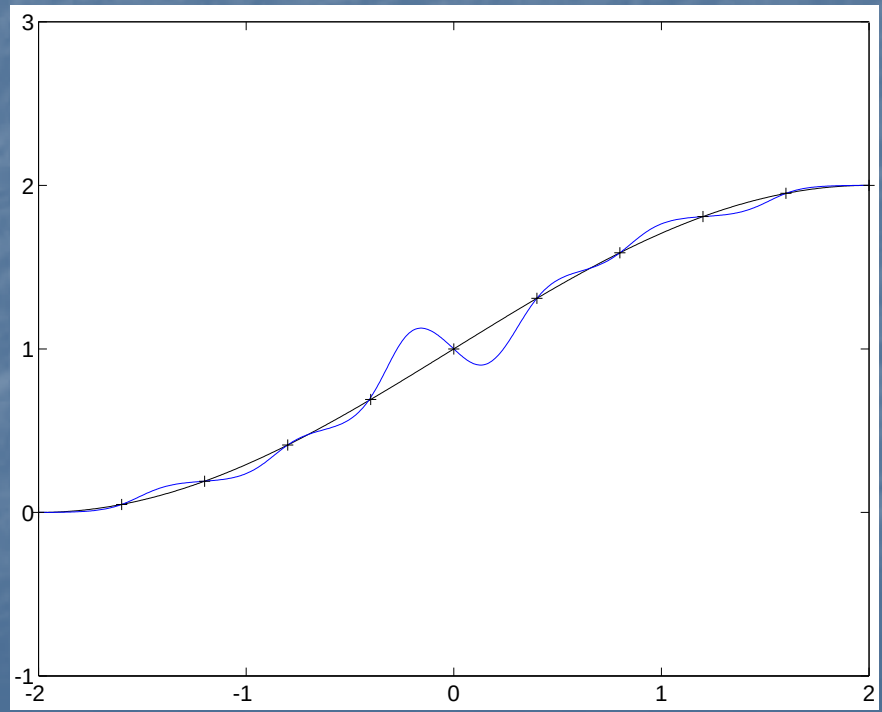


Generalization

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right) \qquad p = -2, -1.6, -1.2, \ldots, 1.6, 2$$





Fuzzy Logic

Introduction

Example of Application areas of Fuzzy Logic:

- Fuzzy Control
  - Subway trains
  - Cement kilns
  - Washing Machines
  - Fridges

Fuzzy Sets

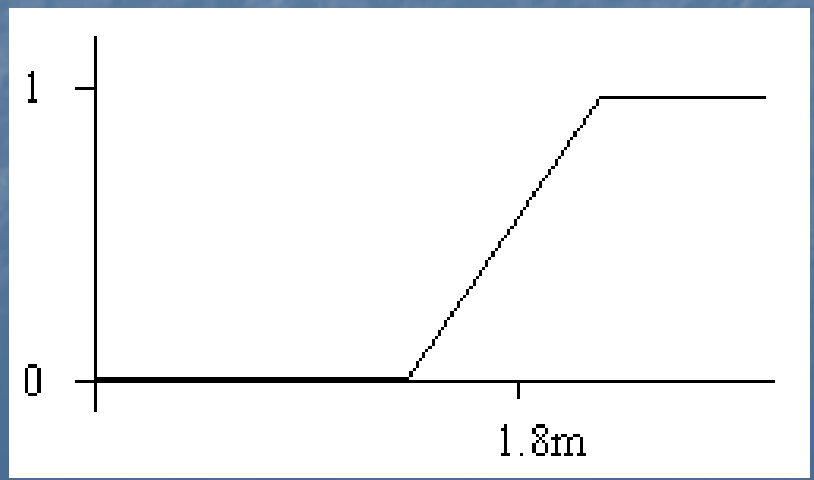
- Extension of Classical Sets
- Not just a membership value of in the set and out the set, 1 and 0
  - *but partial membership value, between* 1 *and* 0

Example: Height

- Tall people: say taller than or equal to 1.8m
  - 1.8m , 2m, 3m etc member of this set
  - 1.0 m, 1.5m or even 1.79999m not a member
- Real systems have measurement uncertainty
  - so near the border lines, many misclassifications

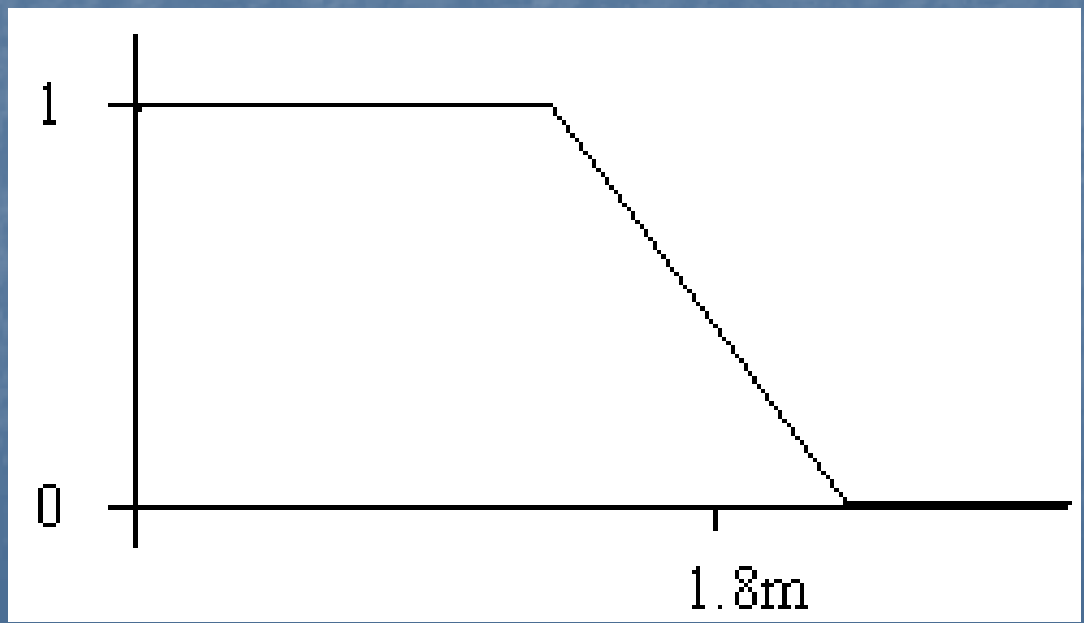
Member Functions

- Membership function
  - better than listing membership values
- e.g. Tall(x) = {1 if x >= 1.9m ,
  0 if x <= 1.7m
  else (x - 1.7) / 0.2 }



Example: Fuzzy Short

- Short(x) = {0 if x >= 1.9m ,
  1 if x <= 1.7m
  else (1.9 - x) / 0.2 }

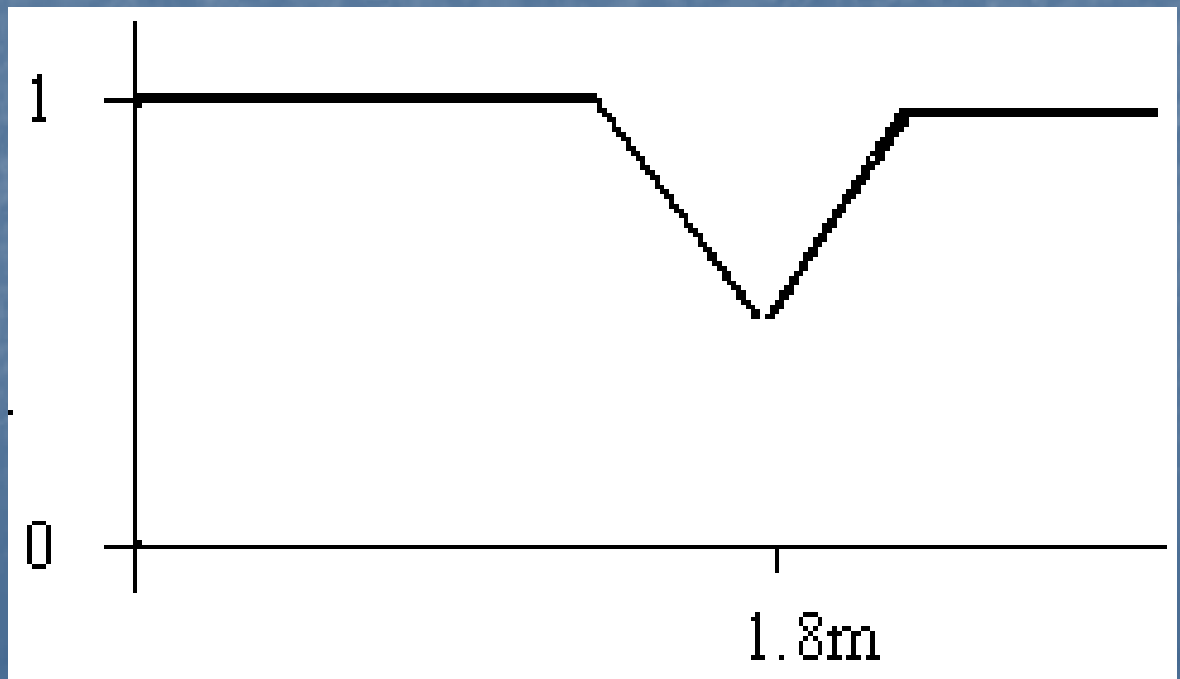


Fuzzy Set Operators

- Fuzzy Set:
  - Union
  - Intersection
  - Complement
- Many possible definitions
  - we introduce one possibility

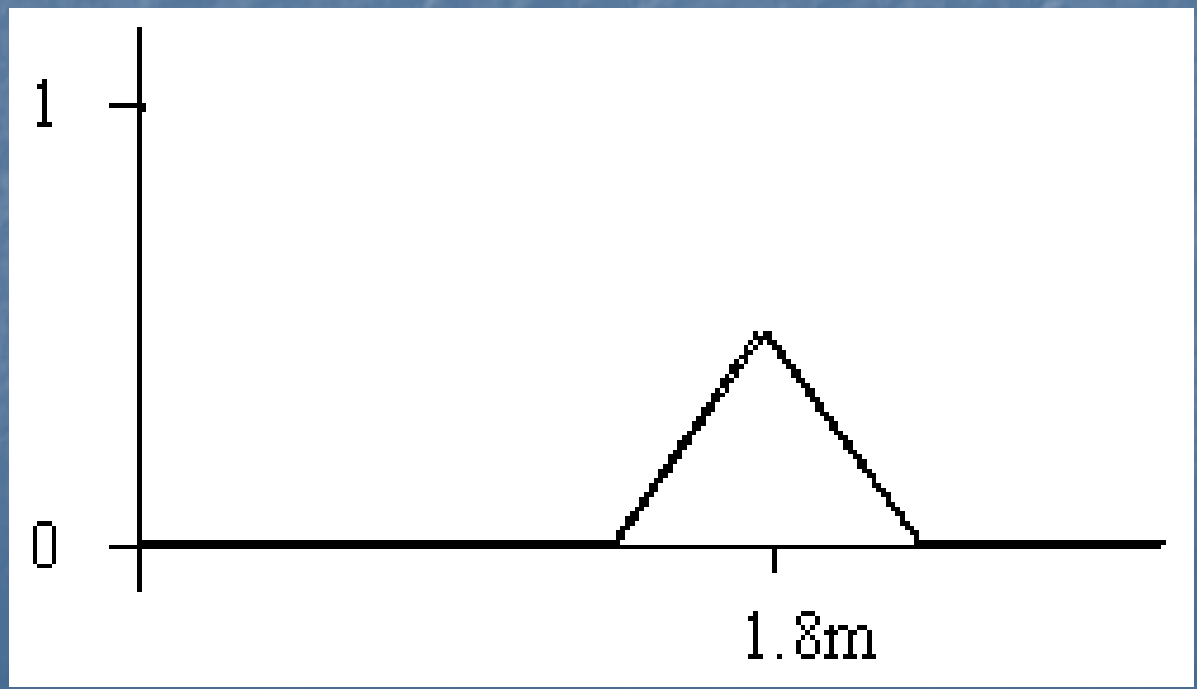
Fuzzy Set Union

- Union (fA(x) and fB(x)) =
  max (fA(x) , fB(x))
- Union (Tall(x) and Short(x))



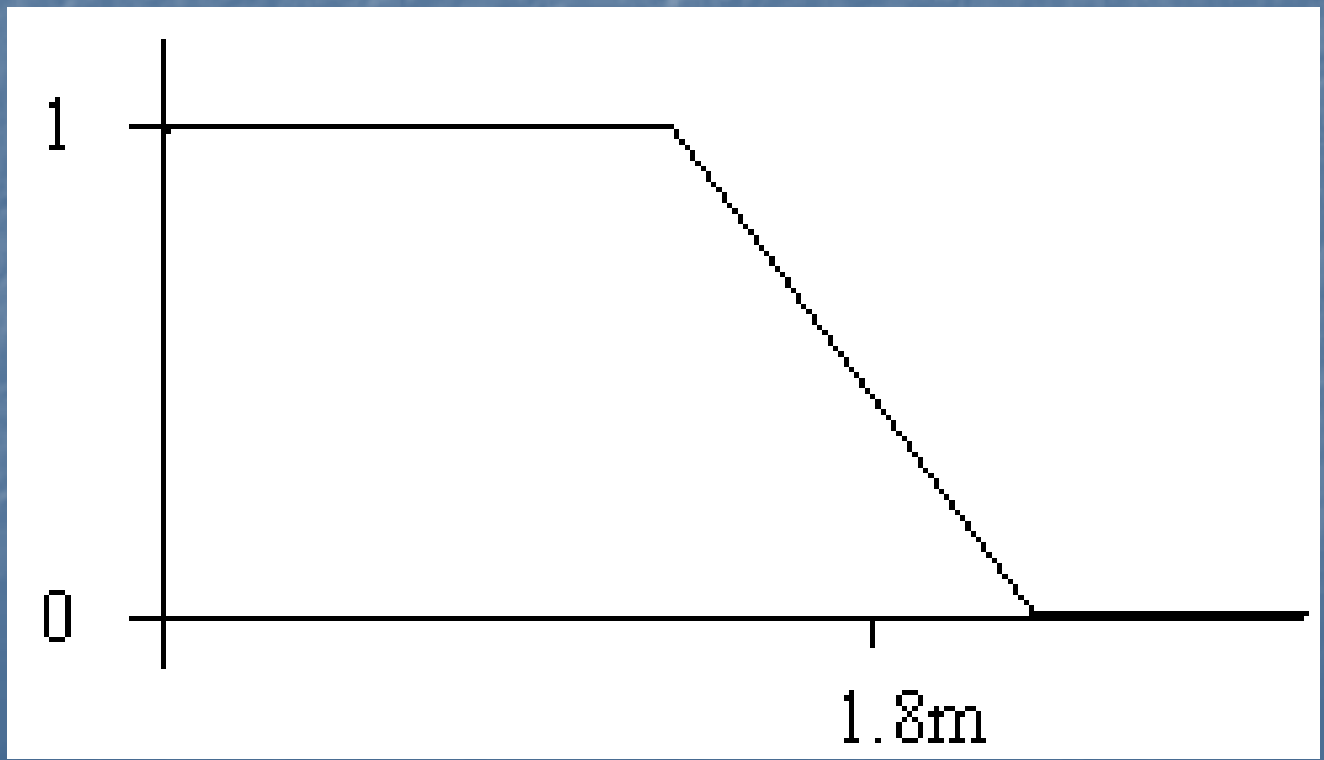
Fuzzy Set Intersection

- Intersection (fA(x) and fB(x)) = min (fA(x), fB(x))
- Intersection (Tall(x) and Short(x))



Fuzzy Set Complement

- Complement(fA(x)) = 1 - fA(x)
  - Not (Tall(x))



Fuzzy Logic Operators

- Fuzzy Logic:
  - **NOT (A) = 1 - A**
  - **A AND B = min (A, B)**
  - **A OR B = max (A, B)**

Fuzzy Logic NOT

A	NOT A
0	1
0.25	0.75
0.5	0.5
0.75	0.25
1	0

Fuzzy Logic AND

A AND B B A	0	0.25	0.5	0.75	1.0
0	0	0	0	0	0
0.25	0	0.25	0.25	0.25	0.25
0.5	0	0.25	0.5	0.5	0.5
0.75	0	0.25	0.5	0.75	0.75
1	0	0.25	0.5	0.75	1

Fuzzy Logic OR

A OR B — B / A	0	0.25	0.5	0.75	1.0
0	0	0.25	0.5	0.75	1.0
0.25	0.25	0.25	0.5	0.75	1.0
0.5	0.5	0.5	0.5	0.75	1.0
0.75	0.75	0.75	0.75	0.75	1.0
1	1.0	1.0	1.0	1.0	1.0

Fuzzy Sets

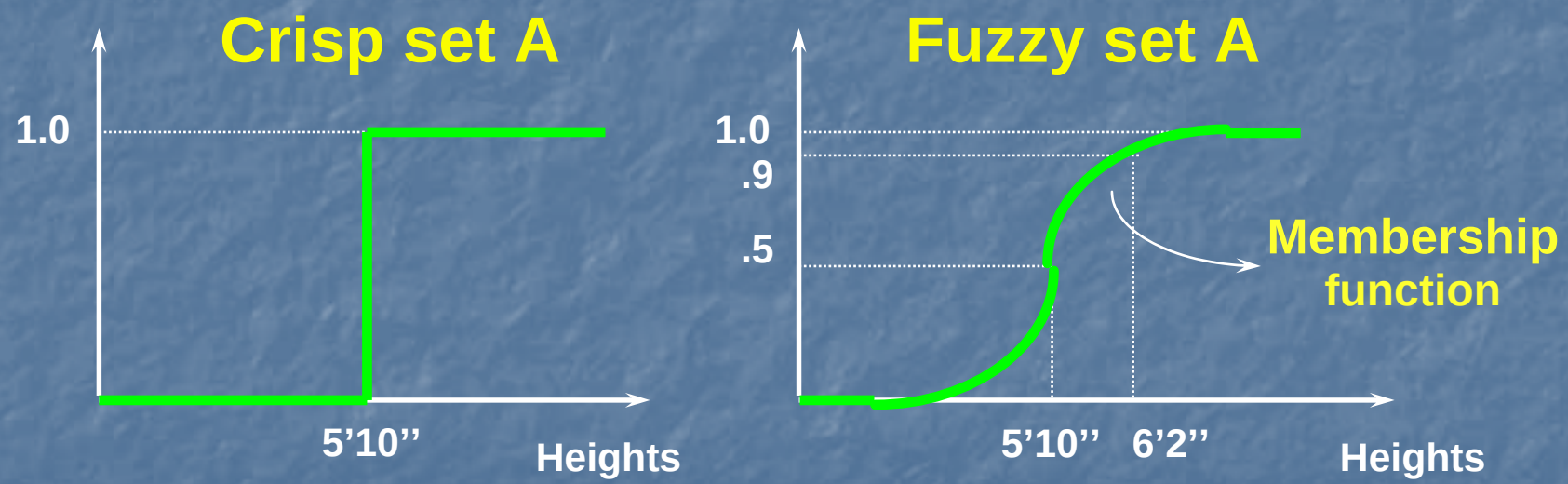
Fuzzy Sets

- Introduction
- Basic definitions and terminology
- Set-theoretic operations
- MF formulation and parameterization
  - MFs of one and two dimensions
  - Derivatives of parameterized MFs
- More on fuzzy union, intersection, and complement
  - Fuzzy complement
  - Fuzzy intersection and union
  - Parameterized T-norm and T-conorm

Fuzzy Sets

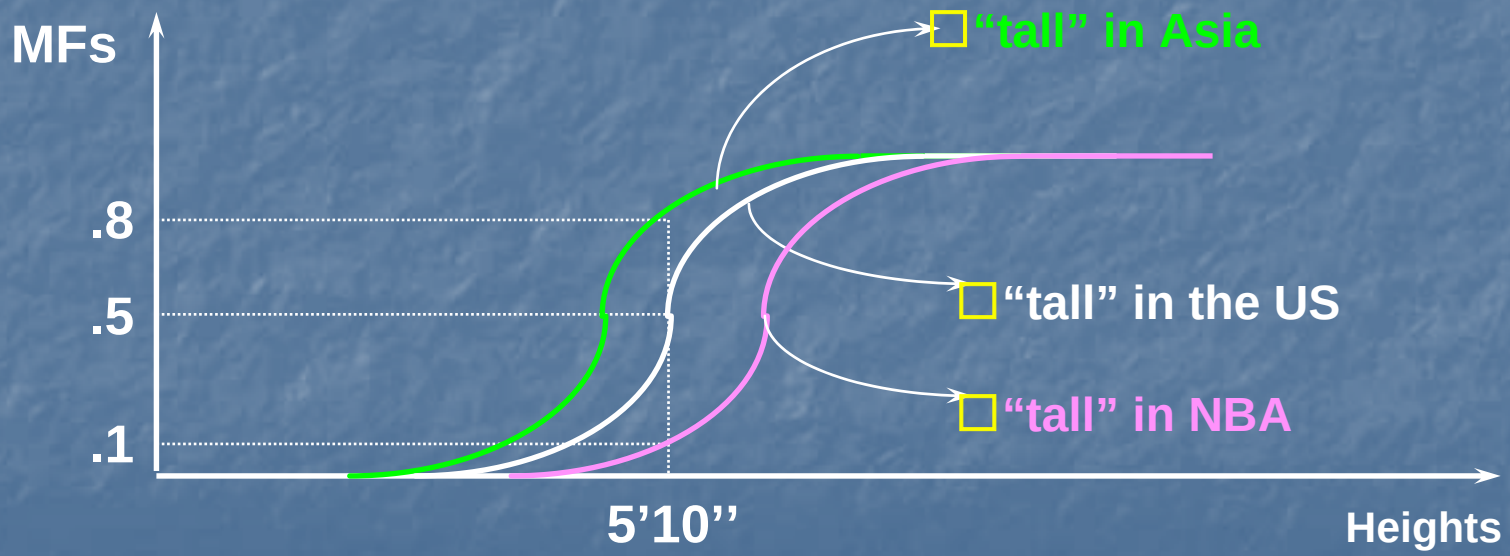
- Sets with fuzzy boundaries

A = Set of tall people



Membership Functions (MFs)

- Characteristics of MFs:
  - Subjective measures
  - Not probability functions



Fuzzy Sets

- Formal definition:
  - A fuzzy set A in X is expressed as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) | x \in X\}$$

Fuzzy set

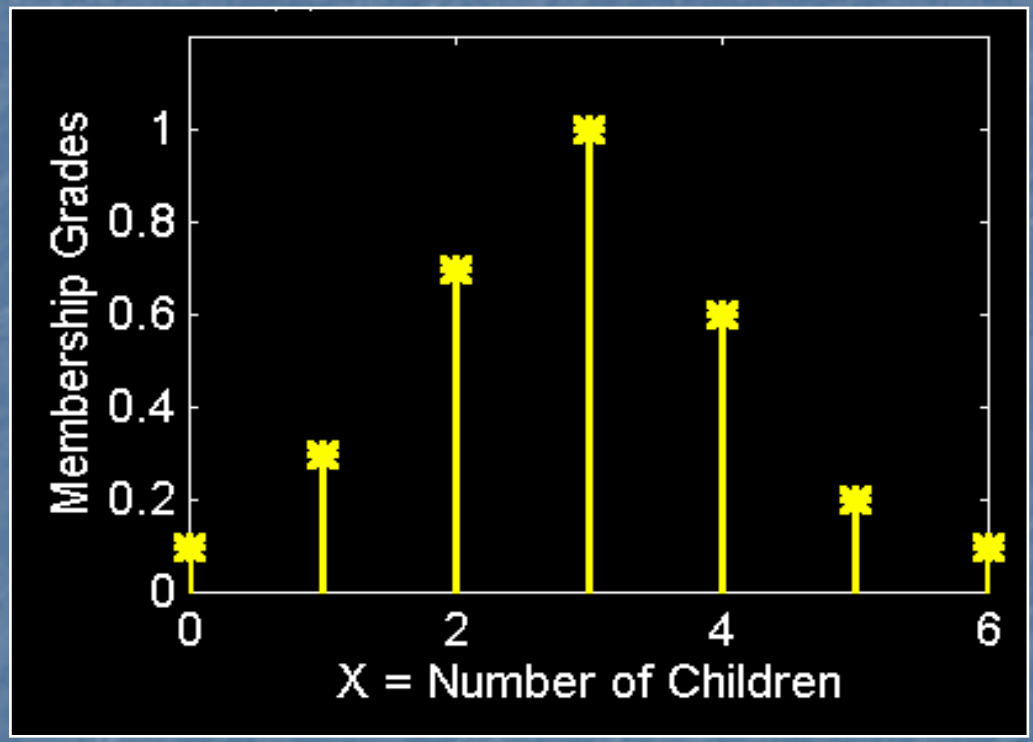
Membership function (MF)

Universe or universe of discourse

A fuzzy set is totally characterized by a membership function (MF).

Fuzzy Sets with Discrete Universes

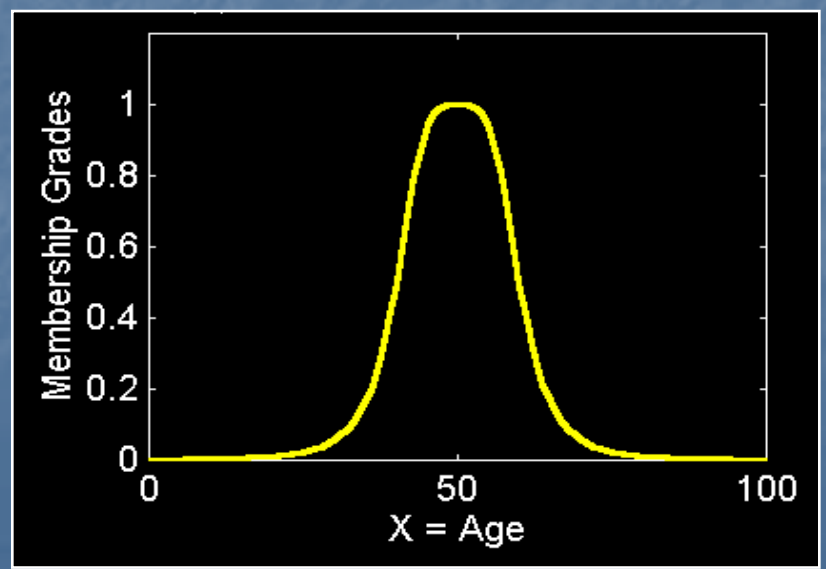
- Fuzzy set C = “desirable city to live in”
  - X = {SF, Boston, LA} (discrete and nonordered)
  - C = {(SF, 0.9), (Boston, 0.8), (LA, 0.6)}
- Fuzzy set A = “sensible number of children”
  - X = {0, 1, 2, 3, 4, 5, 6} (discrete universe)
  - A = {(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)}


Membership Grades
1
0.8
0.6
0.4
0.2
0
0
2
4
6
X = Number of Children

Fuzzy Sets with Cont. Universes

- Fuzzy set B = “about 50 years old”
  - X = Set of positive real numbers (continuous)
  - B = {(x, mB(x)) | x in X}

$$\mu_B(x) = \frac{1}{1 + \left(\frac{x - 50}{10} \right)^2}$$



Alternative Notation

- A fuzzy set A can be alternatively denoted as follows:

X is discrete

$$A = \sum_{x_i \in X} \mu_A(x_i) / x_i$$

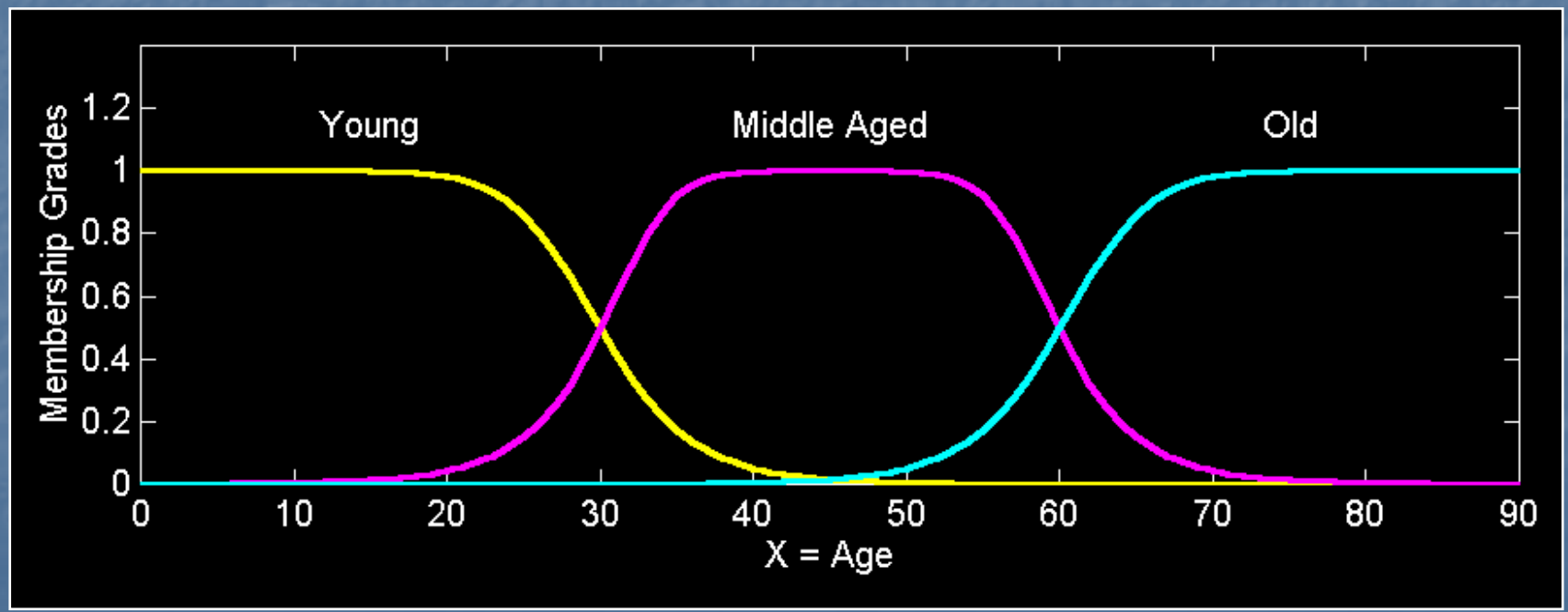
X is continuous

$$A = \int_X \mu_A(x) / x$$

Note that Σ and integral signs stand for the union of membership grades; "/" stands for a marker and does not imply division.

Fuzzy Partition

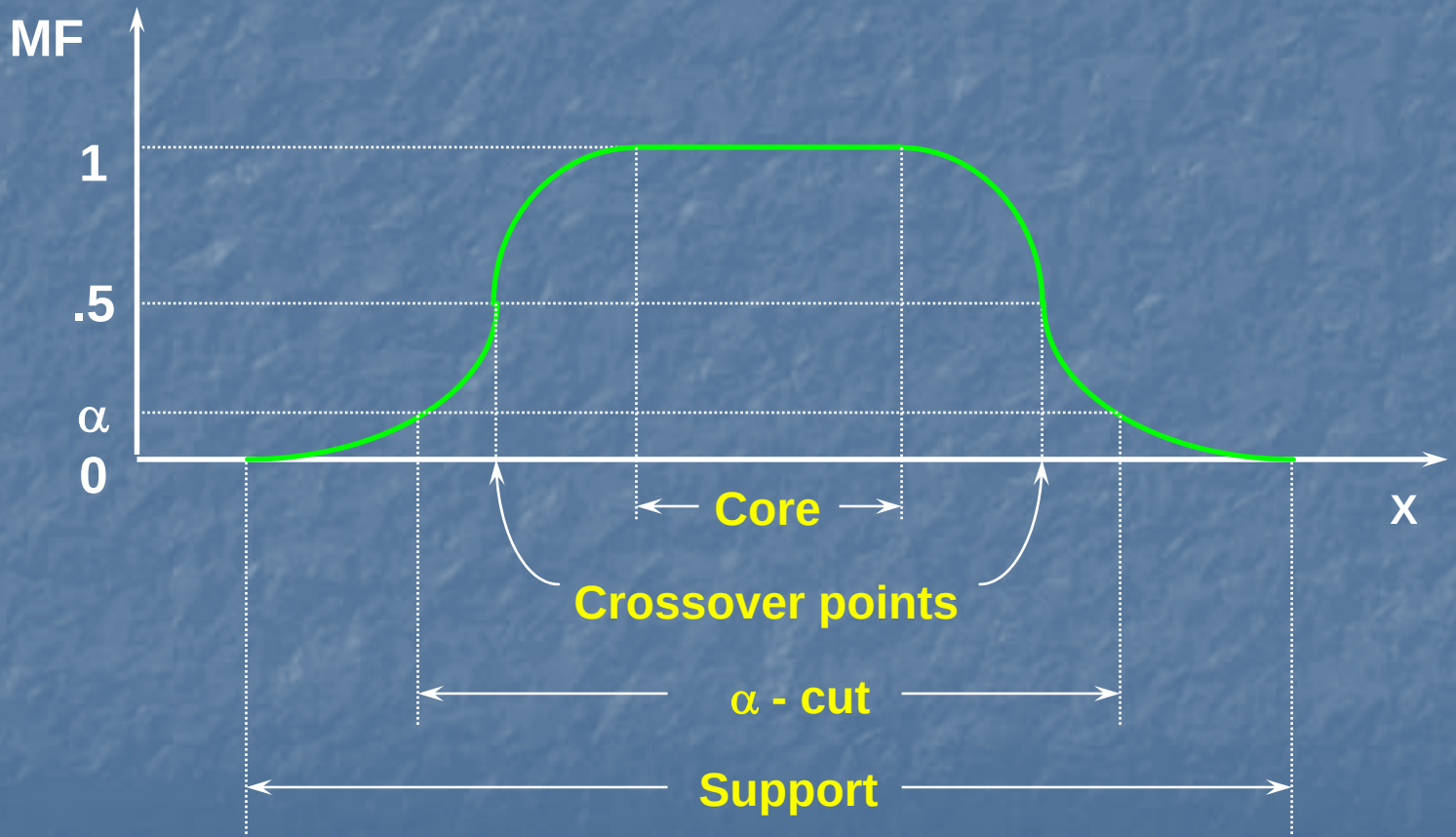
- Fuzzy partitions formed by the linguistic values “young”, “middle aged”, and “old”:



More Definitions

- Support
- Core
- Normality
- Crossover points
- Fuzzy singleton
- a-cut, strong a-cut
- Convexity
- Fuzzy numbers
- Bandwidth
- Symmetricity
- Open left or right, closed

MF Terminology

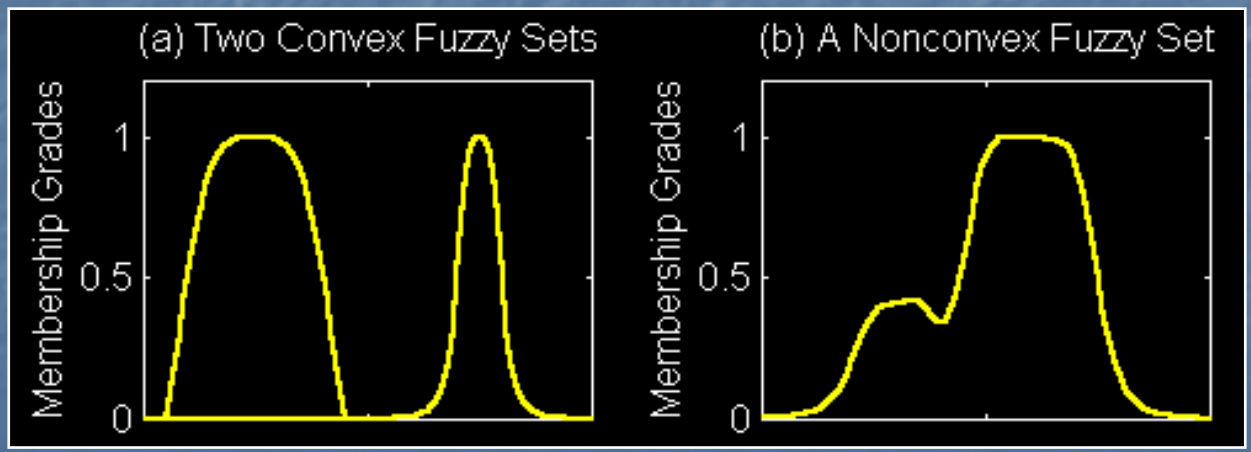


Convexity of Fuzzy Sets

- A fuzzy set *A* is convex if for any l in [0, 1]

$$\mu_A(\lambda x_1 + (1-\lambda)x_2) \geq \min(\mu_A(x_1), \mu_A(x_2))$$

Alternatively, *A* is convex is all its α-cuts are convex.



Set-Theoretic Operations

- Subset:

$$A \subseteq B \Leftrightarrow \mu_A \leq \mu_B$$

- Complement:

$$\overline{A} = X - A \Leftrightarrow \mu_{\overline{A}}(x) = 1 - \mu_A(x)$$

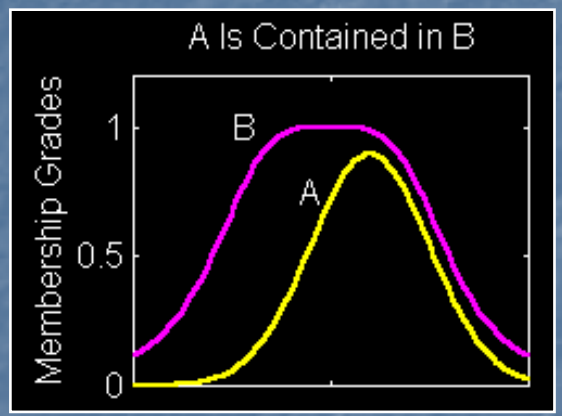
- Union:

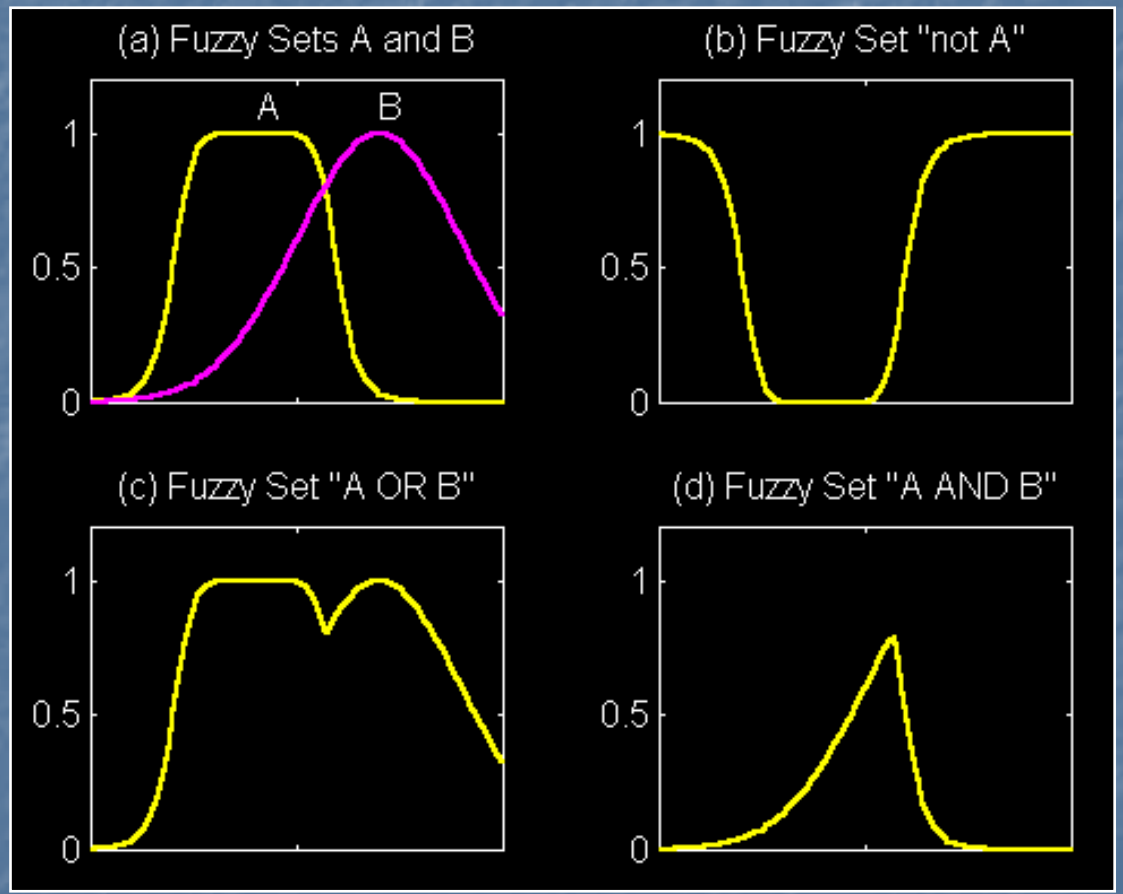
$$C = A \cup B \Leftrightarrow \mu_c(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

- Intersection:

$$C = A \cap B \Leftrightarrow \mu_c(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

Set-Theoretic Operations





MF Formulation

- **Triangular MF:**

$$trimf(x;a,b,c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right)$$

Trapezoidal MF:

$$trapmf(x;a,b,c,d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right)$$

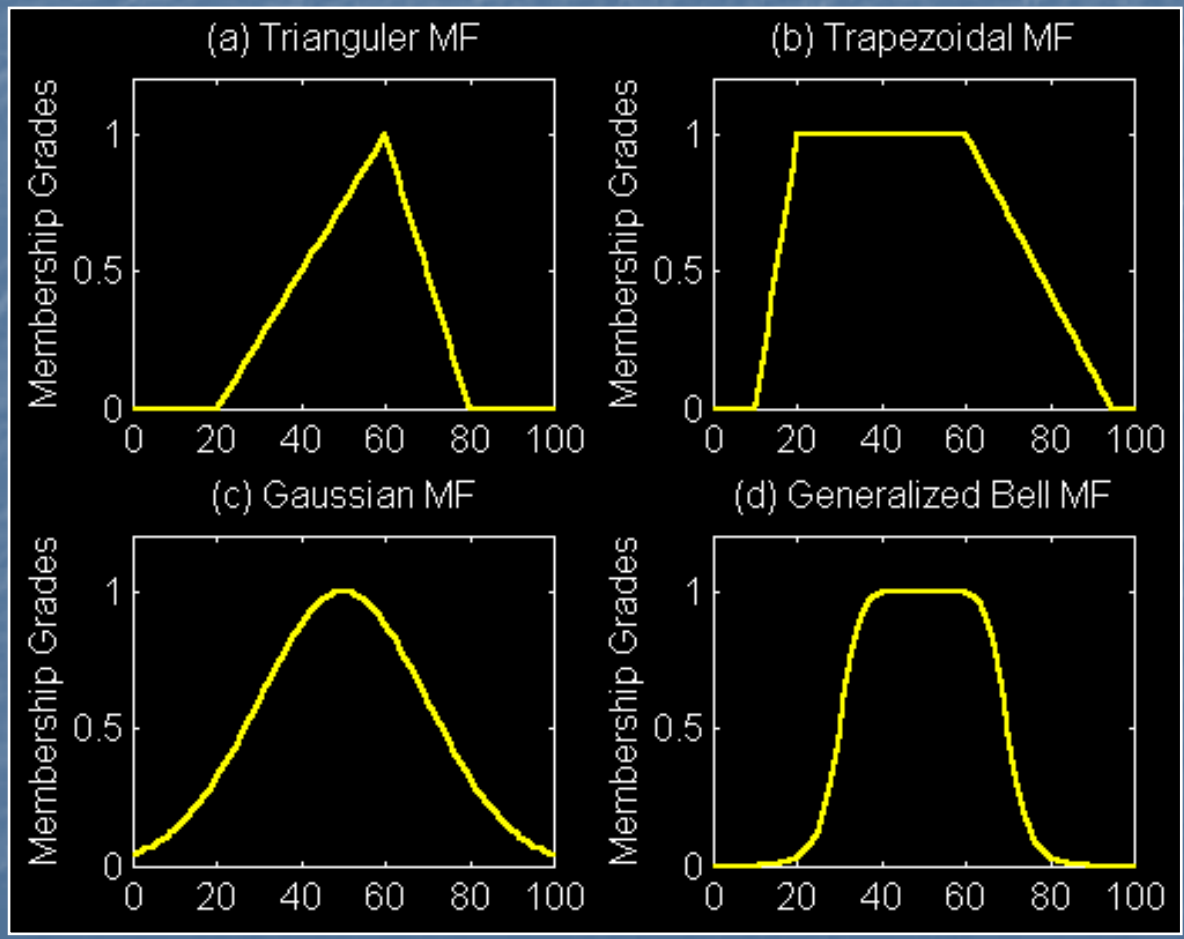
Gaussian MF:

$$gaussmf(x;a,b,c) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

Generalized bell MF:

$$gbellmf(x;a,b,c) = \frac{1}{1+\left|\frac{x-c}{b}\right|^{2b}}$$

MF Formulation



MF Formulation

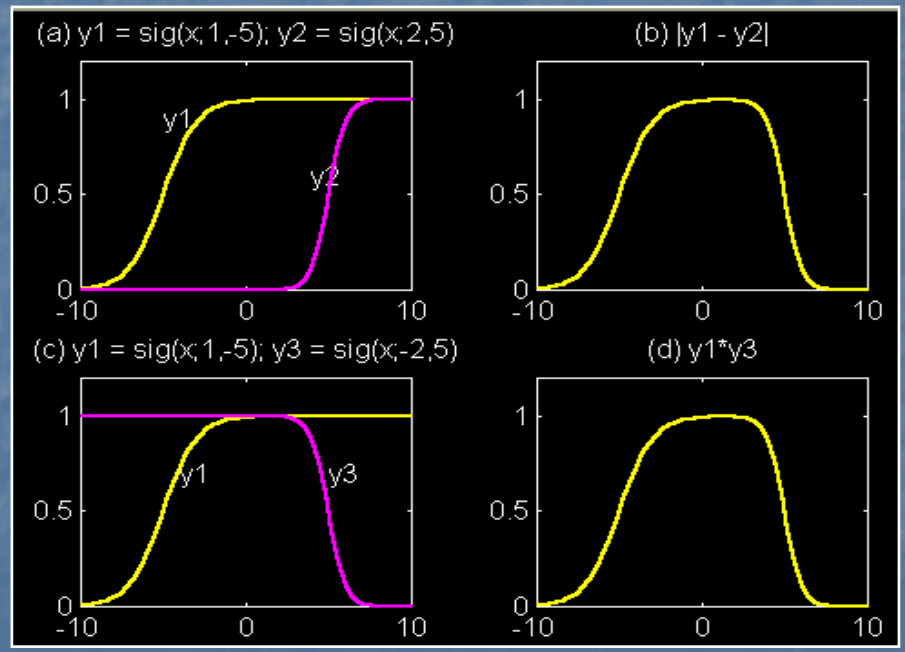
- **Sigmoidal MF:**

$$sigmf\ (x;a,b,c) = \frac{1}{1+e^{-a(x-c)}}$$

Extensions:

Abs. difference of two sig. MF

Product of two sig. MF



MF Formulation

- **L-R MF:**

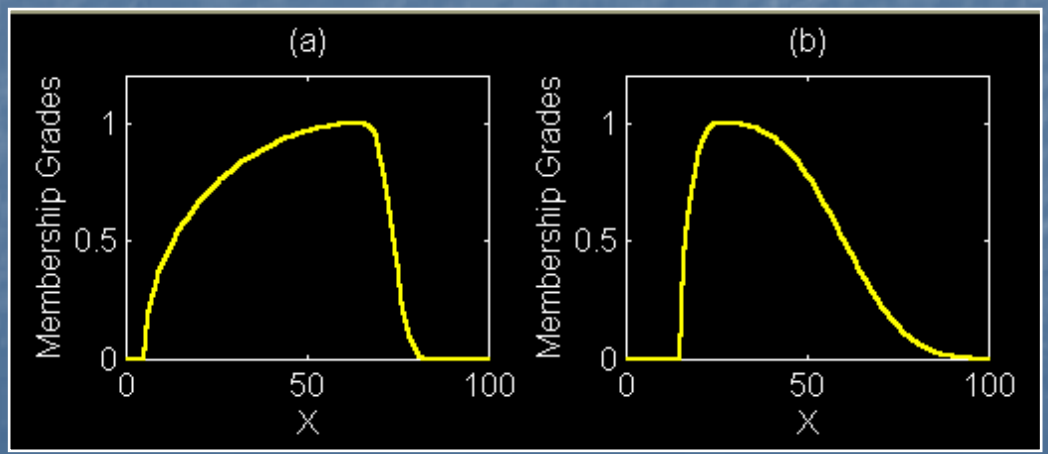
$$LR(x;c,\alpha,\beta) = \begin{cases} F_L\left(\frac{c-x}{\alpha}\right), x < c \\ F_R\left(\frac{x-c}{\beta}\right), x \geq c \end{cases}$$

Example:

$$F_L(x) = \sqrt{\max(0, 1-x^2)}$$

$$F_R(x) = \exp(-|x|^3)$$

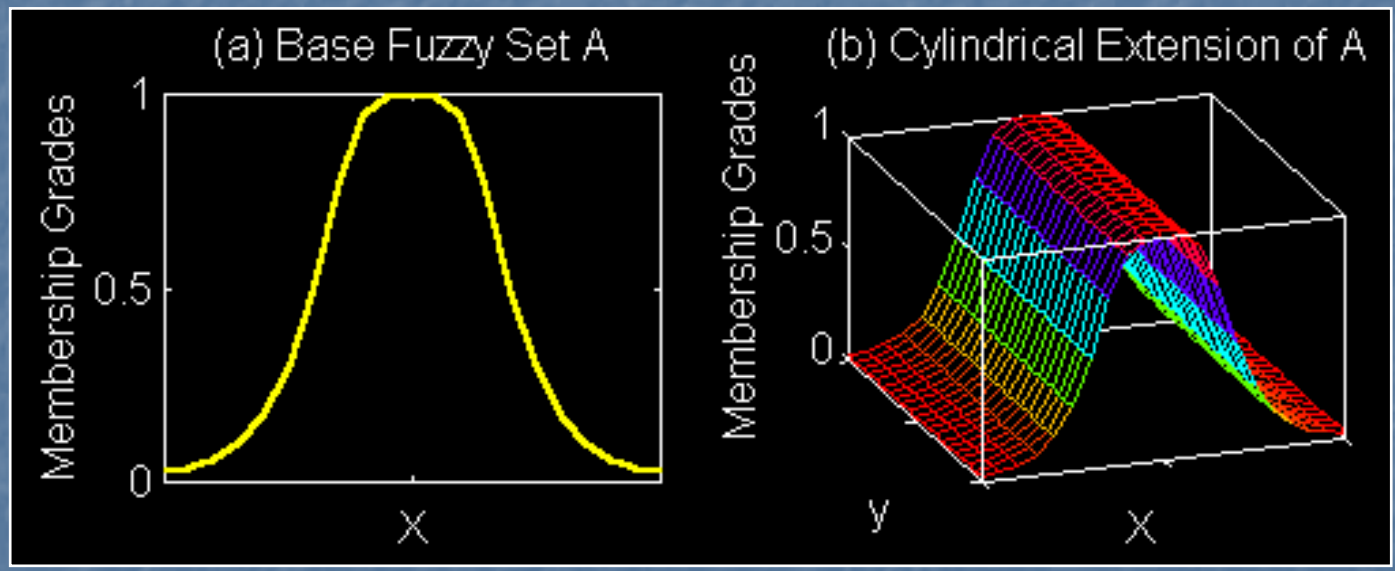
c=65
a=60
b=10



c=25
a=10
b=40

Cylindrical Extension

Base set A **Cylindrical Ext. of A**

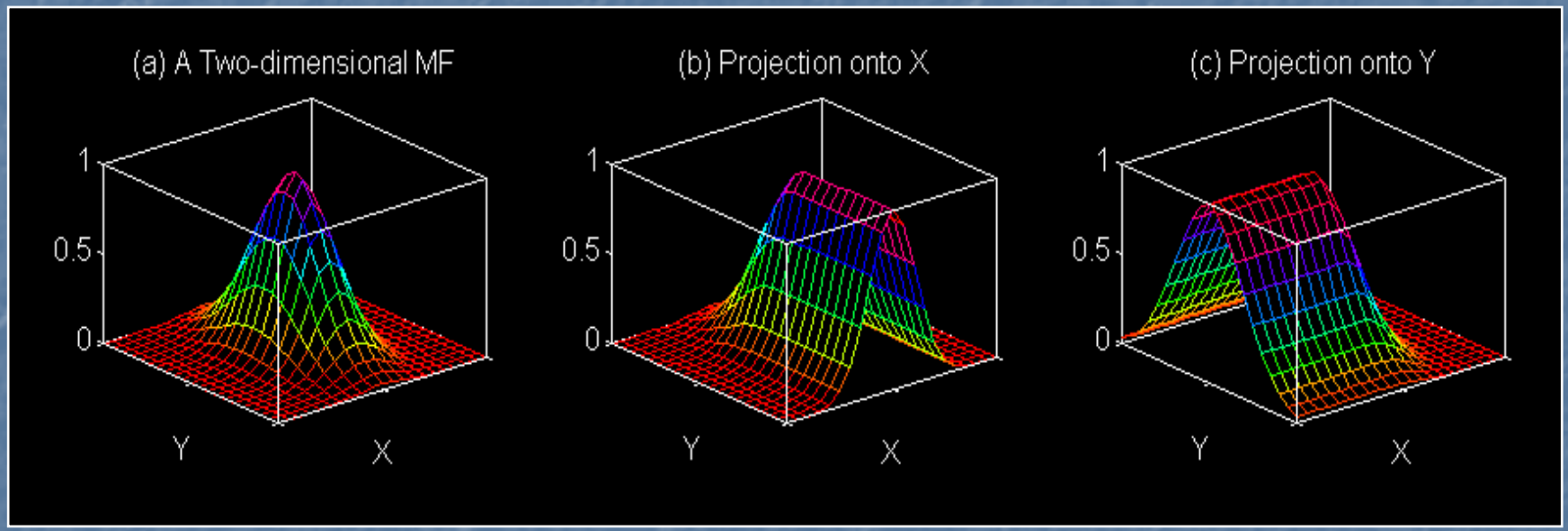


2D MF Projection

Two-dimensional MF

Projection onto X

Projection onto Y

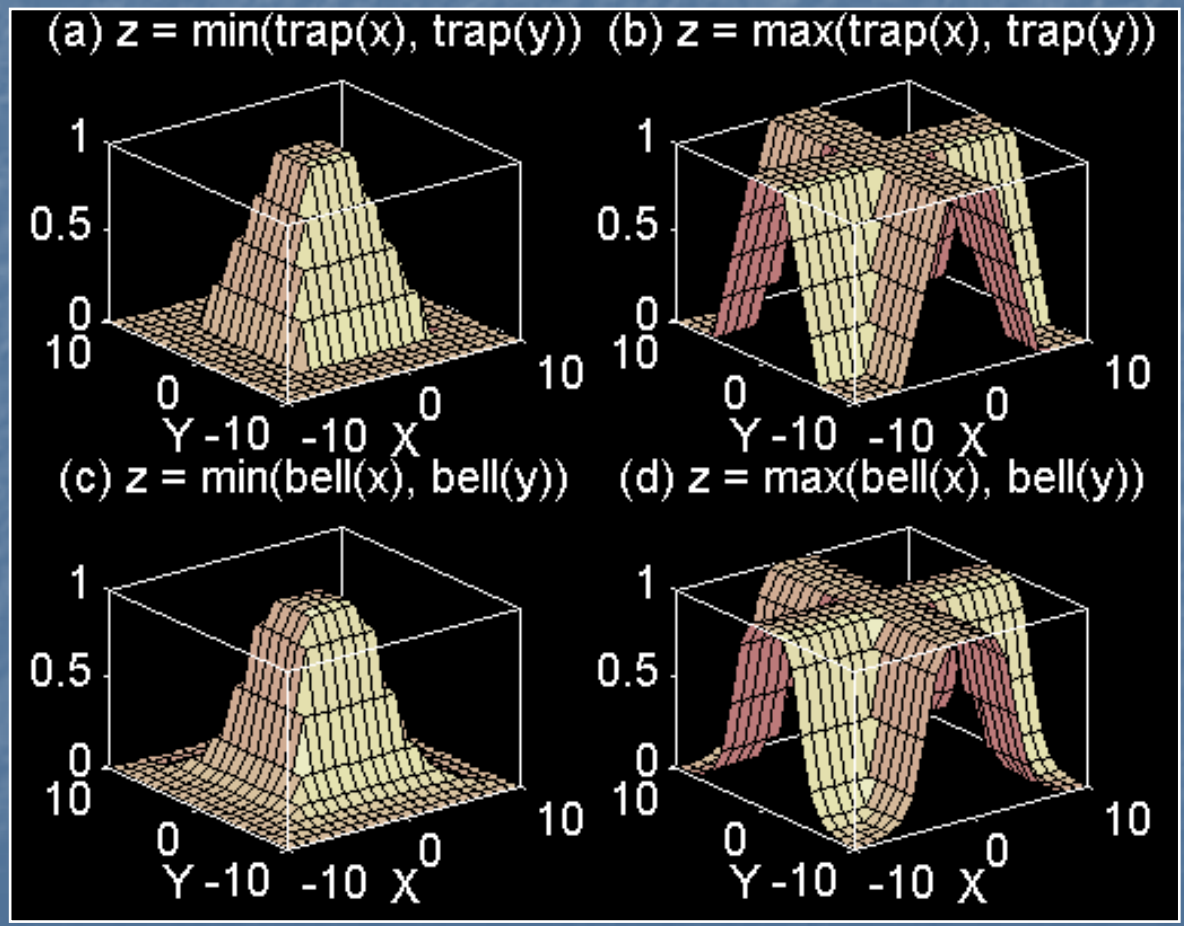


$$\mu_R(x, y)$$

$$\mu_A(x) = \max_y \mu_R(x, y)$$

$$\mu_B(y) = \max_x \mu_R(x, y)$$

2D MFs


(a) z = min(trap(x), trap(y))
(b) z = max(trap(x), trap(y))
(c) z = min(bell(x), bell(y))
(d) z = max(bell(x), bell(y))
1
0.5
0
10
0
Y -10
-10
X
0
10

Fuzzy Complement

- General requirements:
  - Boundary: N(0)=1 and N(1) = 0
  - Monotonicity: N(a) > N(b) if a < b
  - Involution: N(N(a)) = a
- Two types of fuzzy complements:
  - Sugeno's complement:

$$N_s(a) = \frac{1-a}{1+sa}$$

  - Yager's complement:

$$N_w(a) = (1-a^w)^{1/w}$$

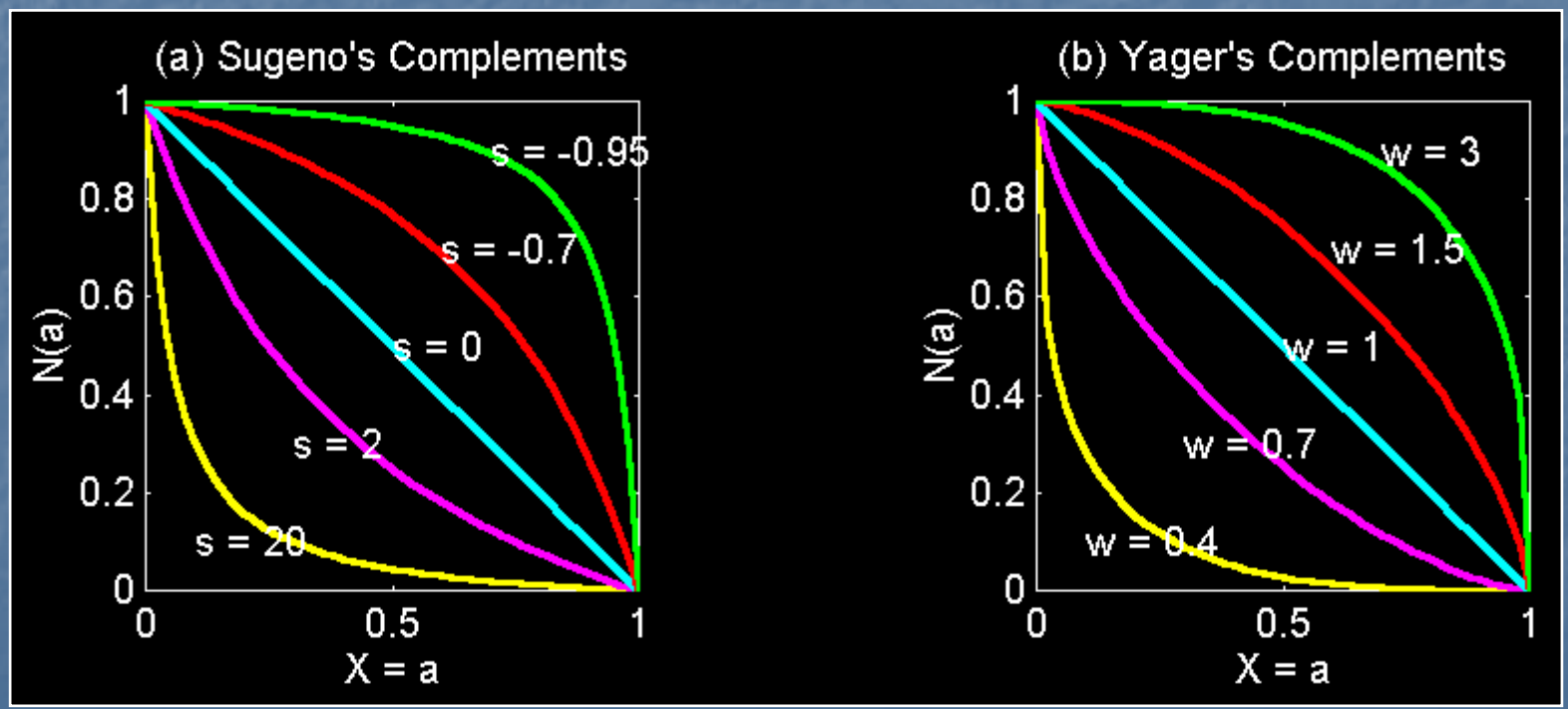
Fuzzy Complement

Sugeno's complement:

$$N_s(a) = \frac{1-a}{1+sa}$$

Yager's complement:

$$N_w(a) = (1-a^w)^{1/w}$$



Fuzzy Intersection: T-norm

- Basic requirements:
  - Boundary: $T(0, 0) = 0$, $T(a, 1) = T(1, a) = a$
  - Monotonicity: $T(a, b) < T(c, d)$ if $a < c$ and $b < d$
  - Commutativity: $T(a, b) = T(b, a)$
  - Associativity: $T(a, T(b, c)) = T(T(a, b), c)$
- Four examples:
  - Minimum: $Tm(a, b)$
  - Algebraic product: $Ta(a, b)$
  - Bounded product: $Tb(a, b)$
  - Drastic product: $Td(a, b)$

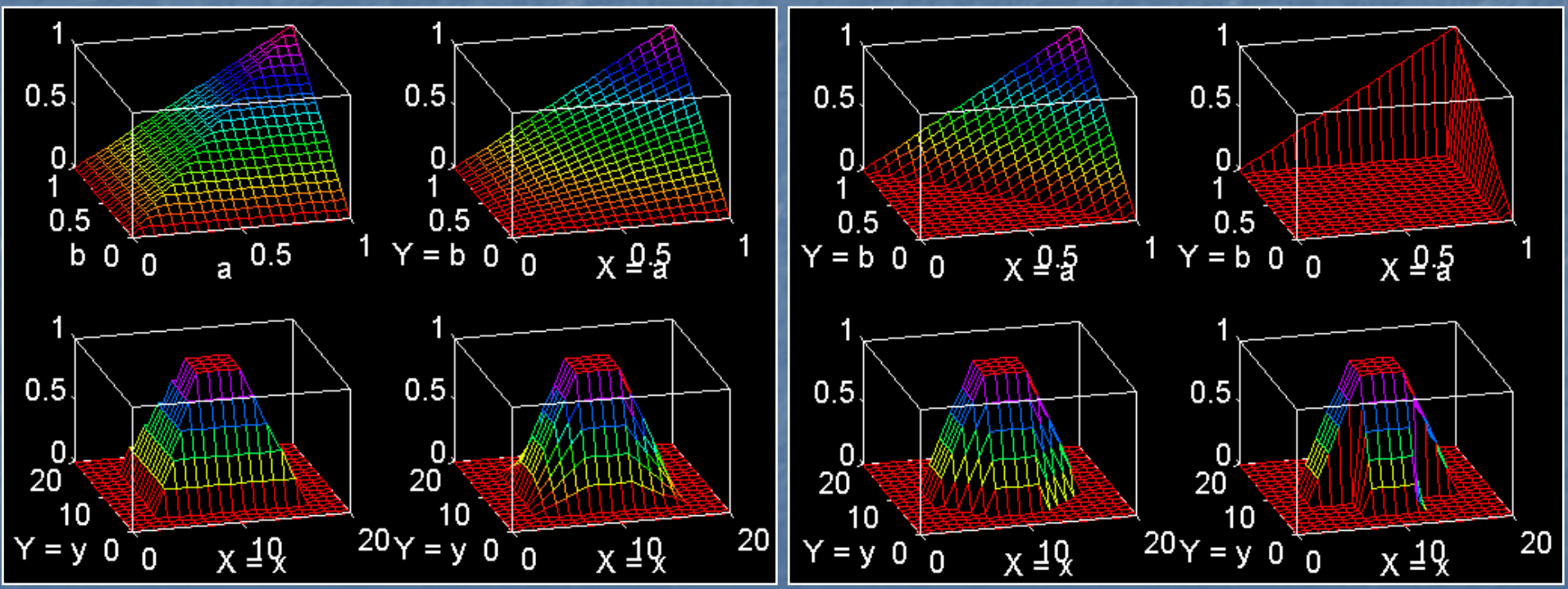
T-norm Operator

Minimum: $T_m(a, b)$

Algebraic product: $T_a(a, b)$

Bounded product: $T_b(a, b)$

Drastic product: $T_d(a, b)$



Fuzzy Union: T-conorm or S-norm

- Basic requirements:
  - Boundary: $S(1, 1) = 1$, $S(a, 0) = S(0, a) = a$
  - Monotonicity: $S(a, b) < S(c, d)$ if $a < c$ and $b < d$
  - Commutativity: $S(a, b) = S(b, a)$
  - Associativity: $S(a, S(b, c)) = S(S(a, b), c)$
- Four examples:
  - Maximum: $Sm(a, b)$
  - Algebraic sum: $Sa(a, b)$
  - Bounded sum: $Sb(a, b)$
  - Drastic sum: $Sd(a, b)$

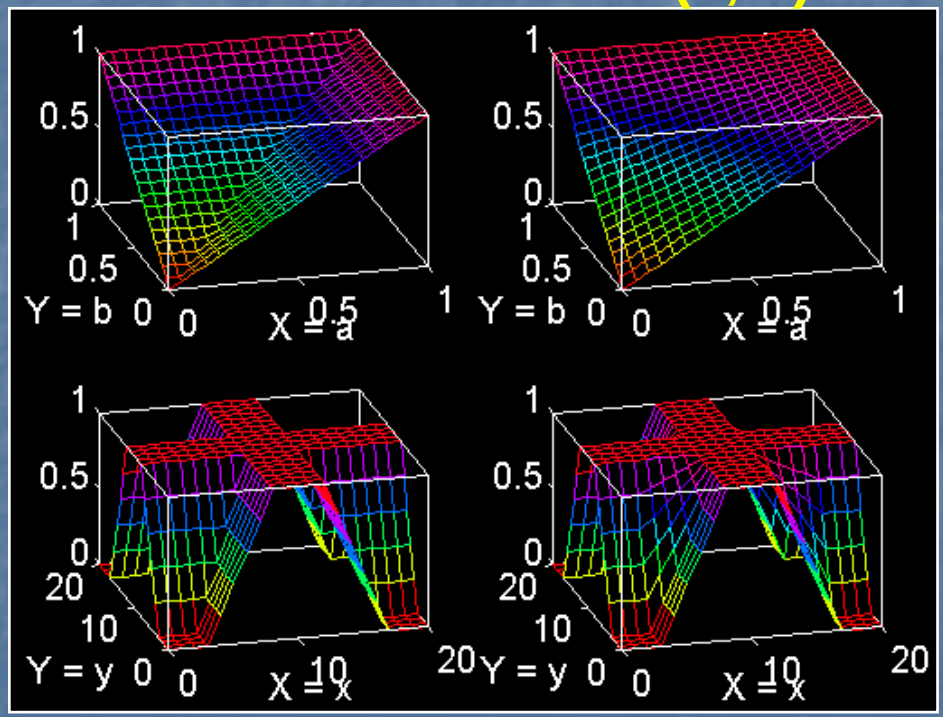
T-conorm or S-norm

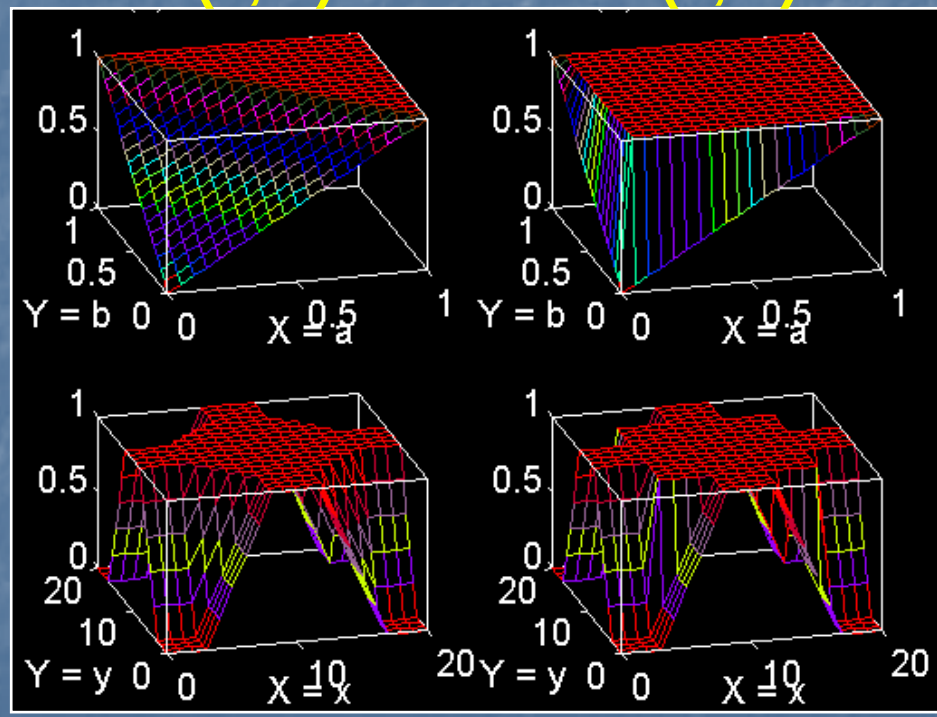
Maximum: $S_m(a, b)$

Algebraic sum: $S_a(a, b)$

Bounded sum: $S_b(a, b)$

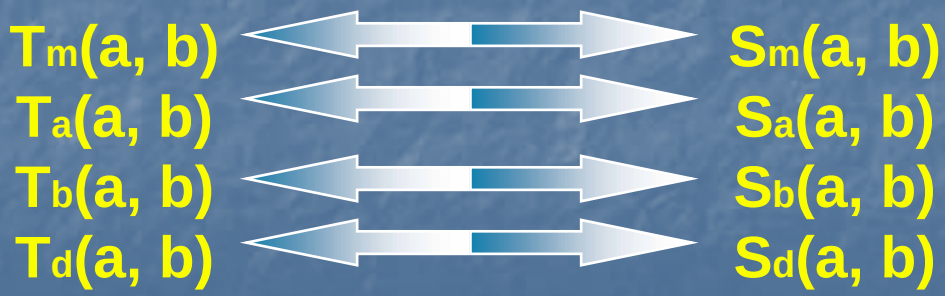
Drastic sum: $S_d(a, b)$





Generalized DeMorgan's Law

- T-norms and T-conorms are ***duals*** which support the generalization of DeMorgan's law:
  - T(a, b) = N(S(N(a), N(b)))
  - S(a, b) = N(T(N(a), N(b)))



Parameterized T-norm and S-norm

- Parameterized T-norms and dual T-conorms have been proposed by several researchers:
  - Yager
  - Schweizer and Sklar
  - Dubois and Prade
  - Hamacher
  - Frank
  - Sugeno
  - Dombi

Fuzzy Rules
&
Fuzzy Reasoning

Outline

- Extension principle
- Fuzzy relations
- Fuzzy *if* - *then* rules
- Compositional rule of inference
- Fuzzy reasoning

Extension Principle

A is a fuzzy set on *X* :

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \cdots + \mu_A(x_n)/x_n$$

The image of *A* under *f( )* is a fuzzy set *B:*

$$B = \mu_B(x_1)/y_1 + \mu_B(x_2)/y_2 + \cdots + \mu_B(x_n)/y_n$$

where $y_i = f(x_i)$, $i = 1$ to n.

If *f( )* is a many-to-one mapping, then

$$\mu_B(y) = \max_{x = f^{-1}(y)} \mu_A(x)$$

Fuzzy Relations

- A fuzzy relation *R* is a 2D MF:

$$R = \{((x, y), \mu_R(x, y)) | (x, y) \in X \times Y\}$$

- Examples:
  - x is close to y (x and y are numbers)
  - x depends on y (x and y are events)
  - x and y look alike (x, and y are persons or objects)
  - If x is large, then y is small (x is an observed reading and Y is a corresponding action)

Max-Min Composition

- The max-min composition of two fuzzy relations *R1* (defined on *X* and *Y*) and *R2* (defined on *Y* and *Z*) is

$$\mu_{R_1 \circ R_2}(x,z) = \bigvee_y [\mu_{R_1}(x,y) \wedge \mu_{R_2}(y,z)]$$

- Properties:
  - Associativity: $R \circ (S \circ T) = (R \circ S) \circ T$
  - Distributivity over union:

$$R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$$

  - Week distributivity over intersection:

$$R \circ (S \cap T) \subseteq (R \circ S) \cap (R \circ T)$$

  - Monotonicity:

$$S \subseteq T \Rightarrow (R \circ S) \subseteq (R \circ T)$$

Max-Star Composition

- Max-product composition:

$$\mu_{R_1 \circ R_2}(x,z) = \bigvee_y [\mu_{R_1}(x,y)\mu_{R_2}(y,z)]$$

- In general, we have max-* composition:

$$\mu_{R_1 \circ R_2}(x,z) = \bigvee_y [\mu_{R_1}(x,y) * \mu_{R_2}(y,z)]$$

- where * is a T-norm operator.

Linguistic Variables

- A numerical variables takes numerical values:

  Age = 65

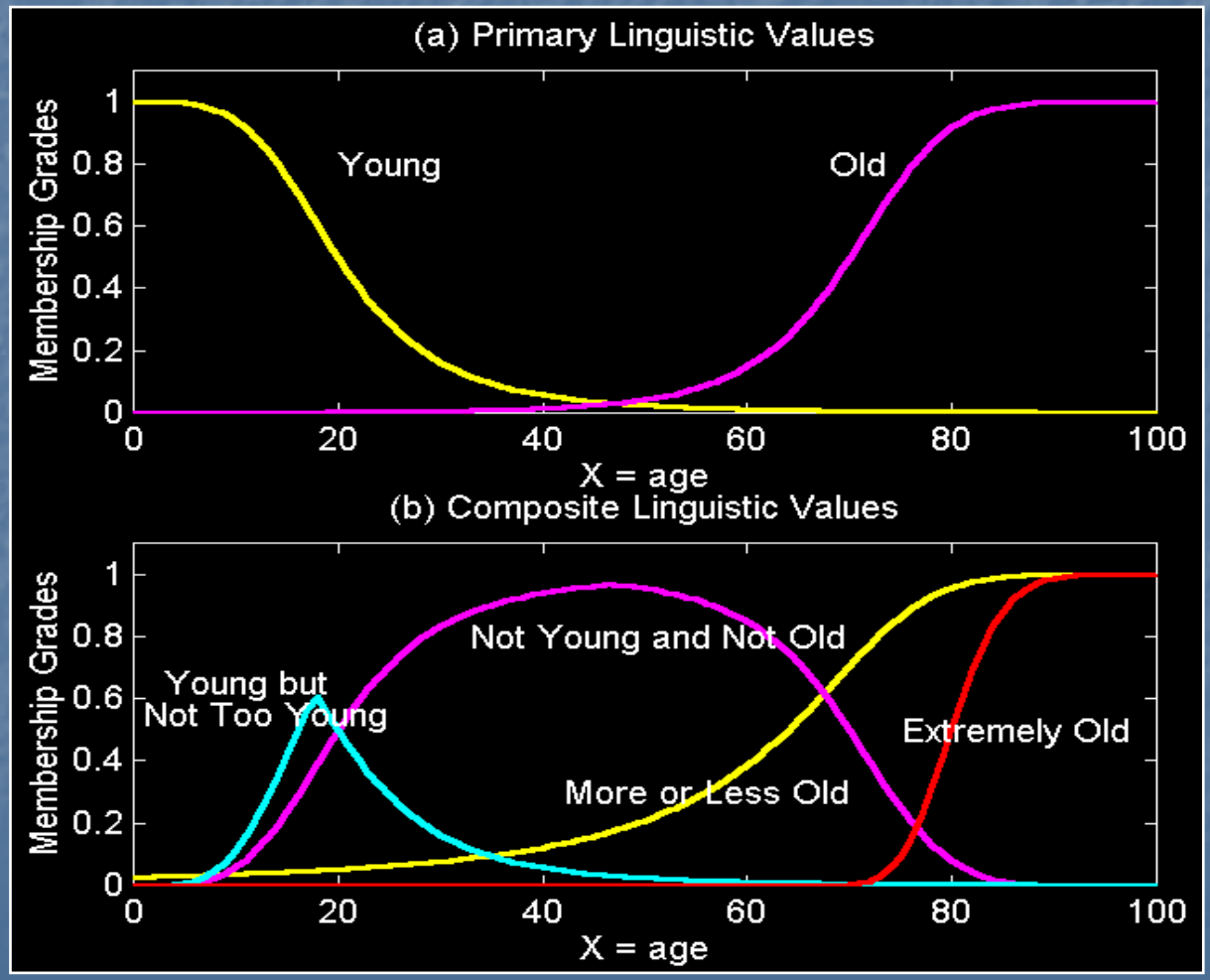
- A linguistic variables takes linguistic values:

  Age is old

- A linguistic values is a fuzzy set.
- All linguistic values form a term set:

  T(age) = {young, not young, very young, ...
  middle aged, not middle aged, ...
  old, not old, very old, more or less old, ...
  not very yound and not very old, ...}

Linguistic Values (Terms)



Operations on Linguistic Values

Concentration:

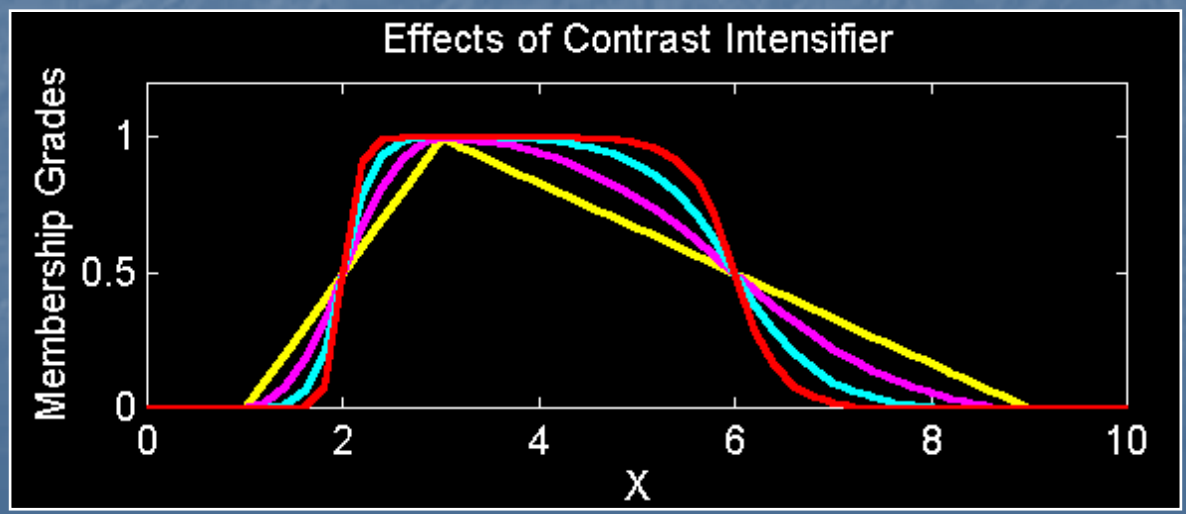
$$CON(A) = A^2$$

Dilation:

$$DIL(A) = A^{0.5}$$

Contrast intensification:

$$INT(A) = \begin{cases} 2A^2, & 0 \le \mu_A(x) \le 0.5 \\ \neg 2(\neg A)^2, & 0.5 \le \mu_A(x) \le 1 \end{cases}$$



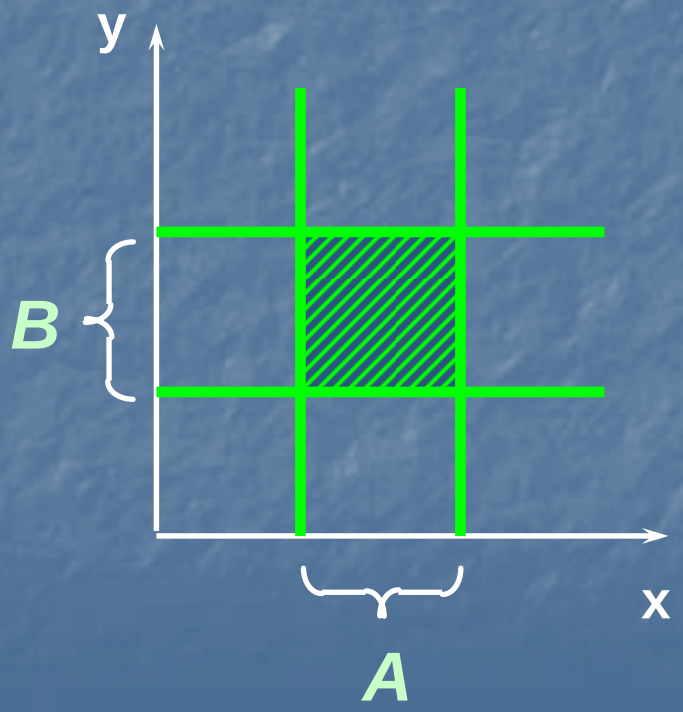
Fuzzy If-Then Rules

- General format:
  - If x is A then y is B
- Examples:
  - If pressure is high, then volume is small.
  - If the road is slippery, then driving is dangerous.
  - If a tomato is red, then it is ripe.
  - If the speed is high, then apply the brake a little.

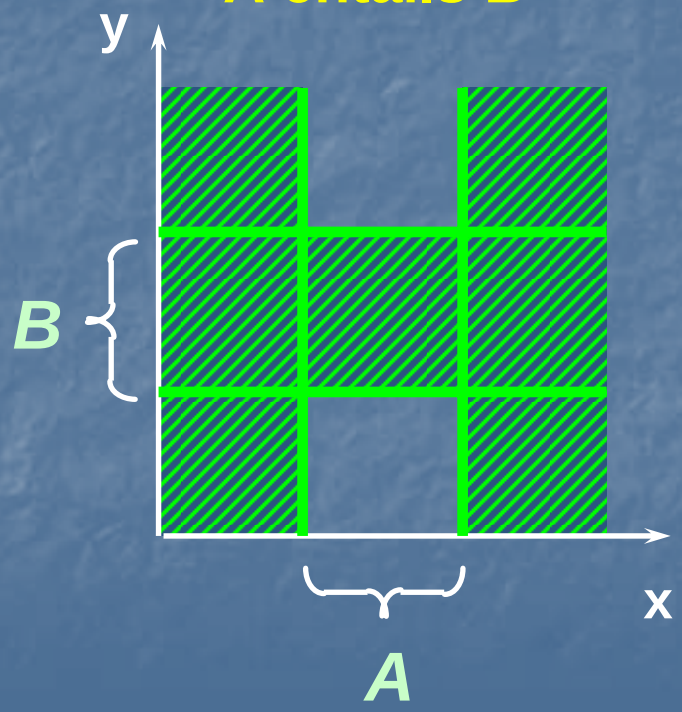
Fuzzy If-Then Rules

Two ways to interpret “If x is A then y is B”:

A coupled with B



A entails B



Fuzzy If-Then Rules

- Two ways to interpret “If x is A then y is B”:
  - A coupled with B: (*A and B*)

$$R = A \rightarrow B = A \times B = \int \mu_A(x) \tilde{*} \mu_B(y) | (x, y)$$

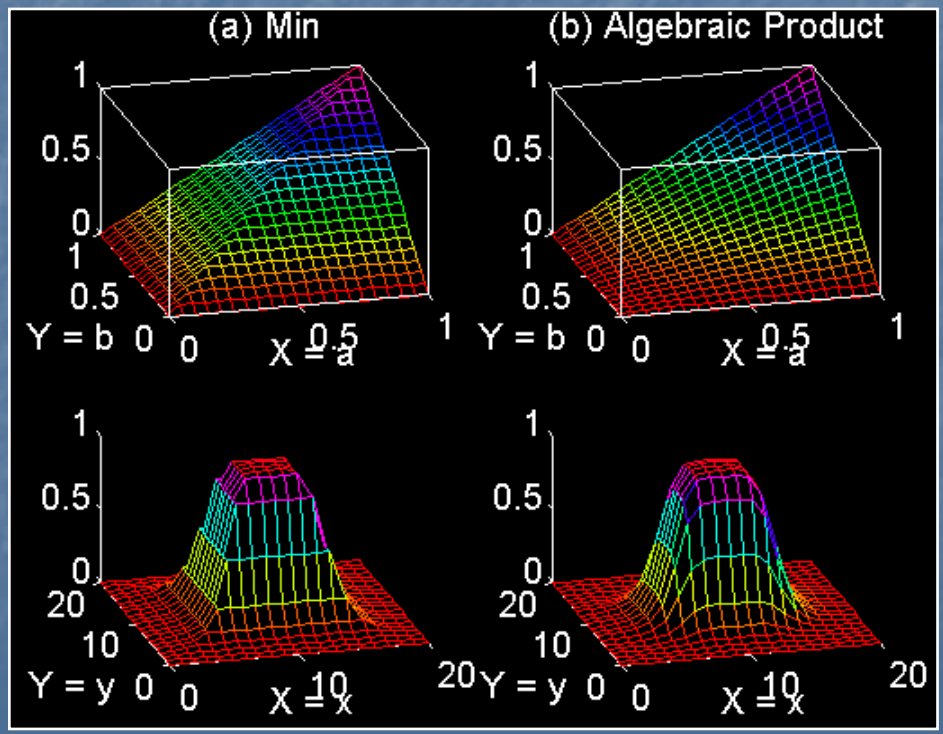
  - A entails B: (*not A or B*)
    - Material implication
    - Propositional calculus
    - Extended propositional calculus
    - Generalization of modus ponens

Fuzzy If-Then Rules

- Fuzzy implication function:

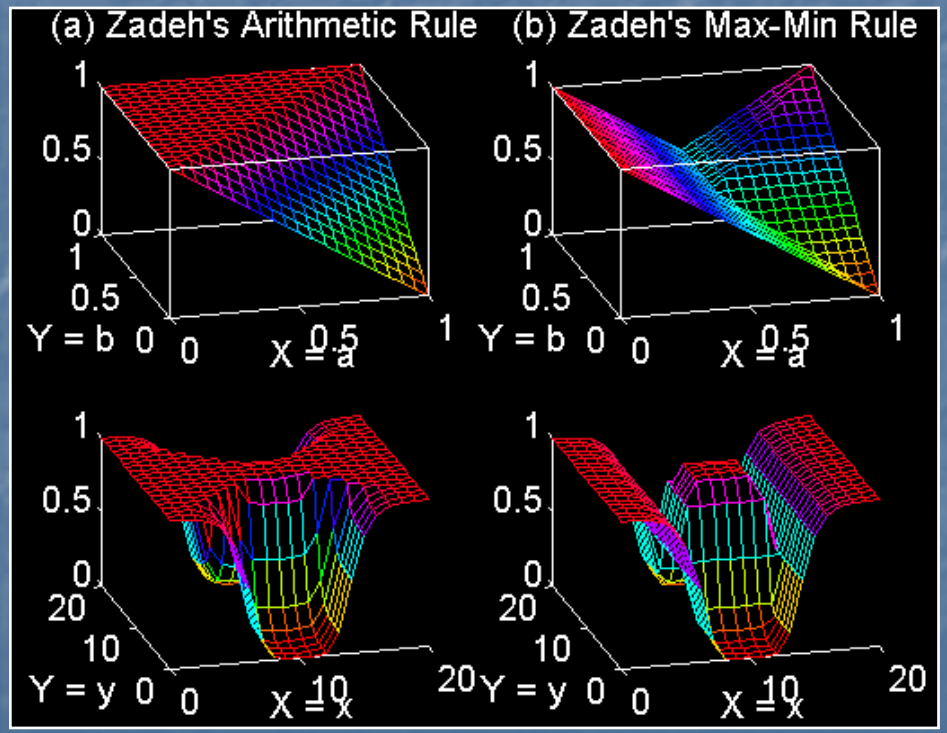
$$\mu_R(x,y) = f(\mu_A(x), \mu_B(y)) = f(a,b)$$

A coupled with B



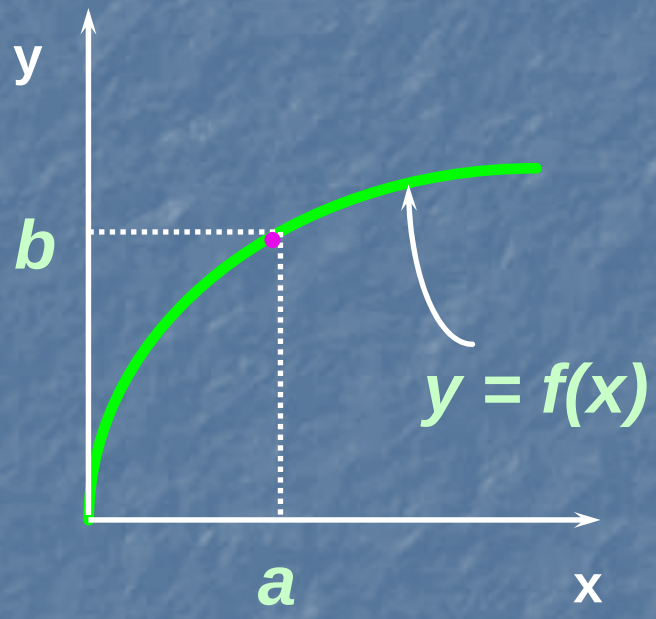
Fuzzy If-Then Rules

A entails B



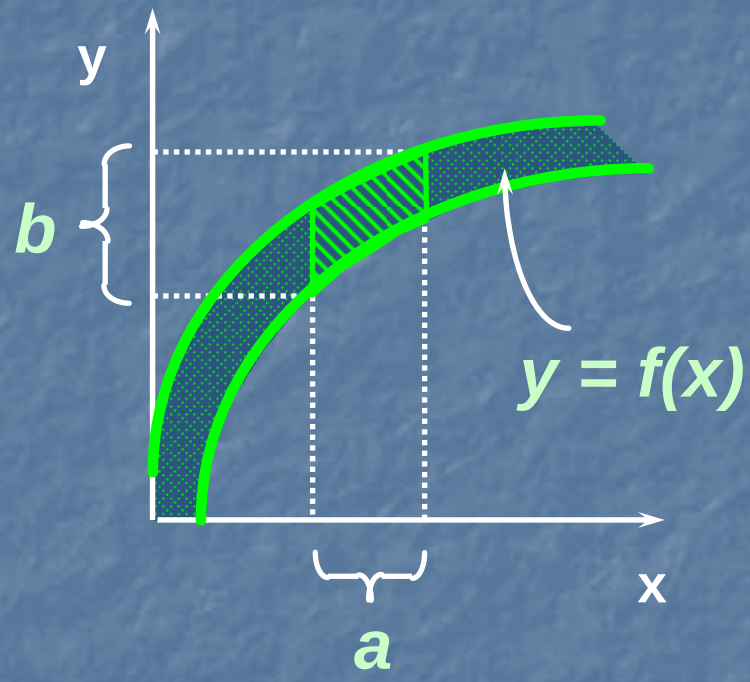
Compositional Rule of Inference

- Derivation of $y = b$ from $x = a$ and $y = f(x)$:



a and b: points

$y = f(x)$: a curve

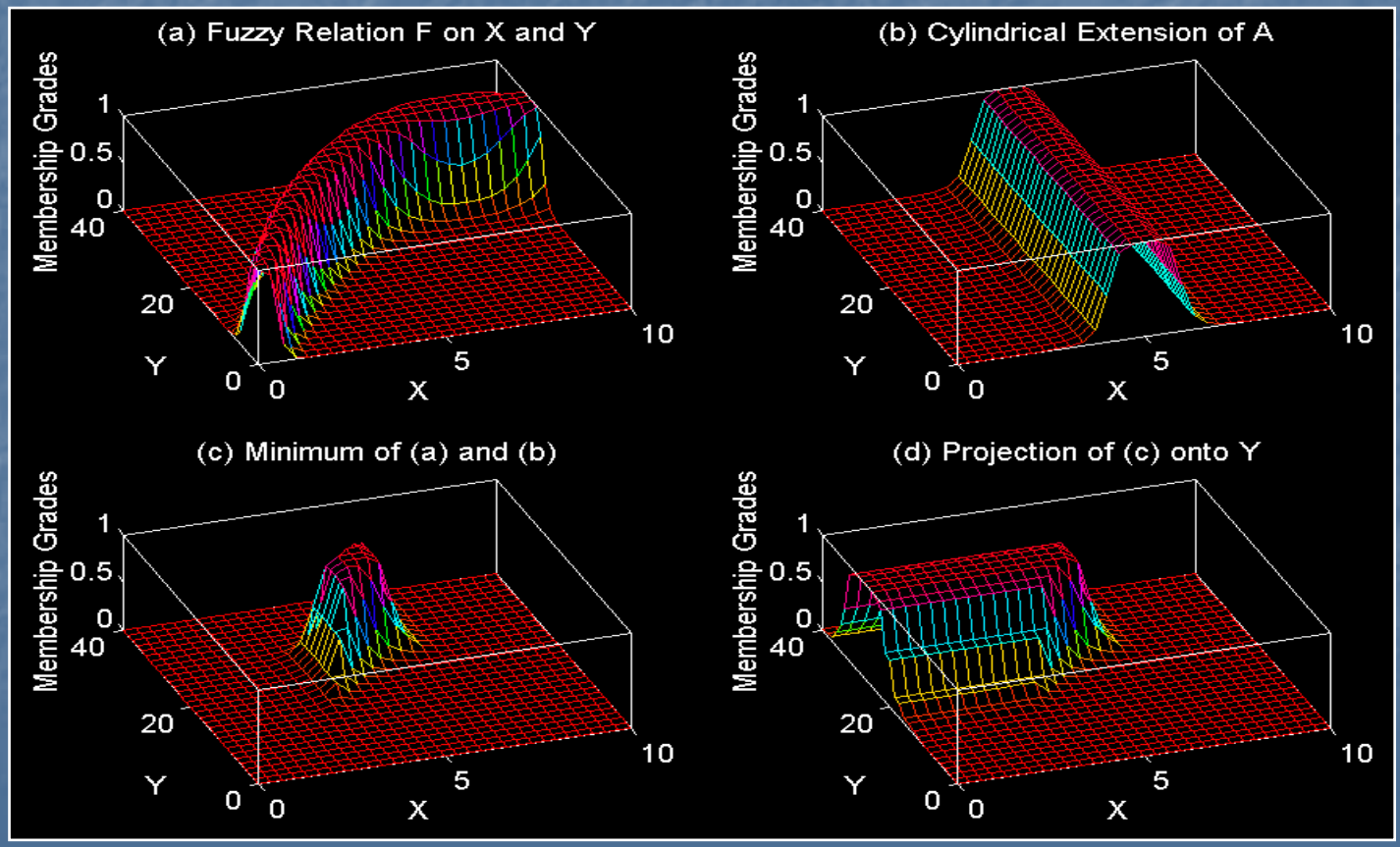


a and b: intervals

$y = f(x)$: an interval-valued function

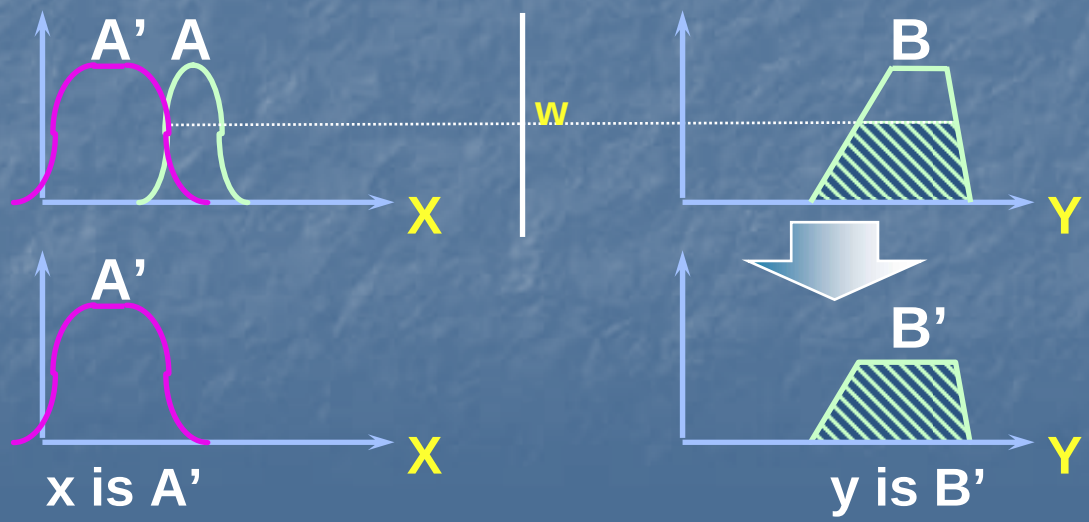
Compositional Rule of Inference

- *a* is a fuzzy set and *y = f(x)* is a fuzzy relation:



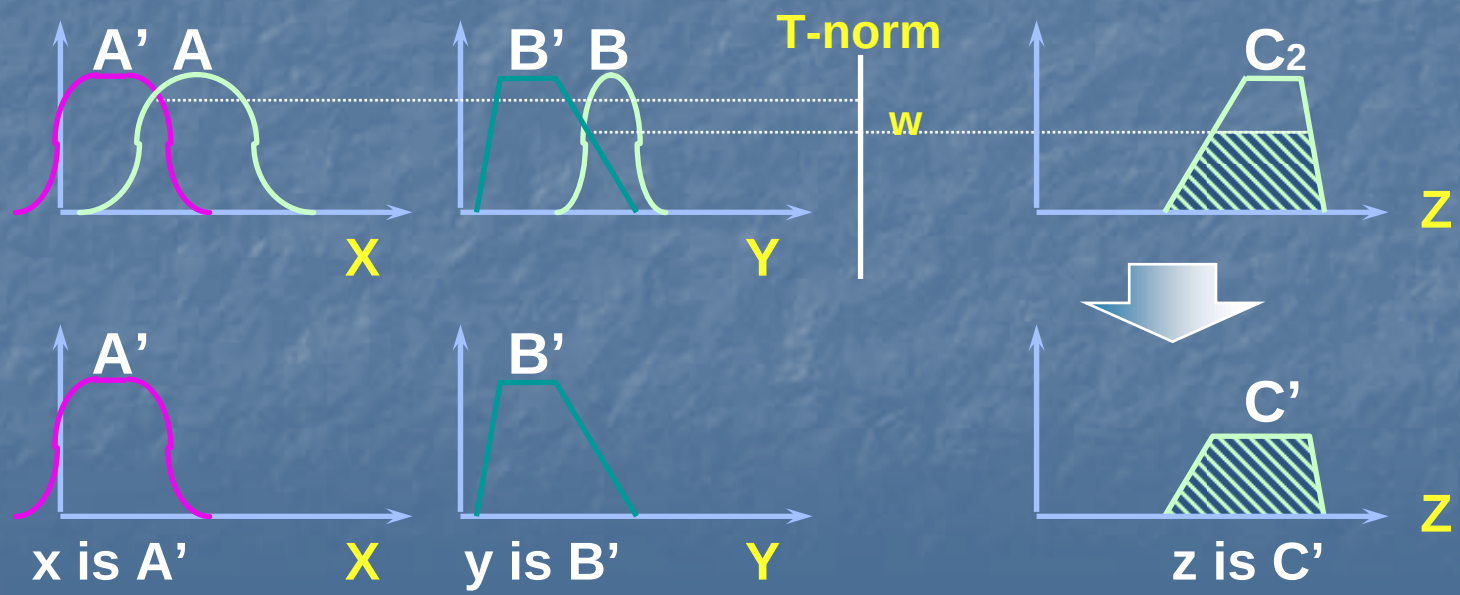
Fuzzy Reasoning

- Single rule with single antecedent
  - Rule: if x is A then y is B
  - Fact: x is A’
  - Conclusion: y is B’
- Graphic Representation:



Fuzzy Reasoning

- Single rule with multiple antecedent
  - Rule: if x is A and y is B then z is C
  - Fact: x is A’ and y is B’
  - Conclusion: z is C’
- Graphic Representation:

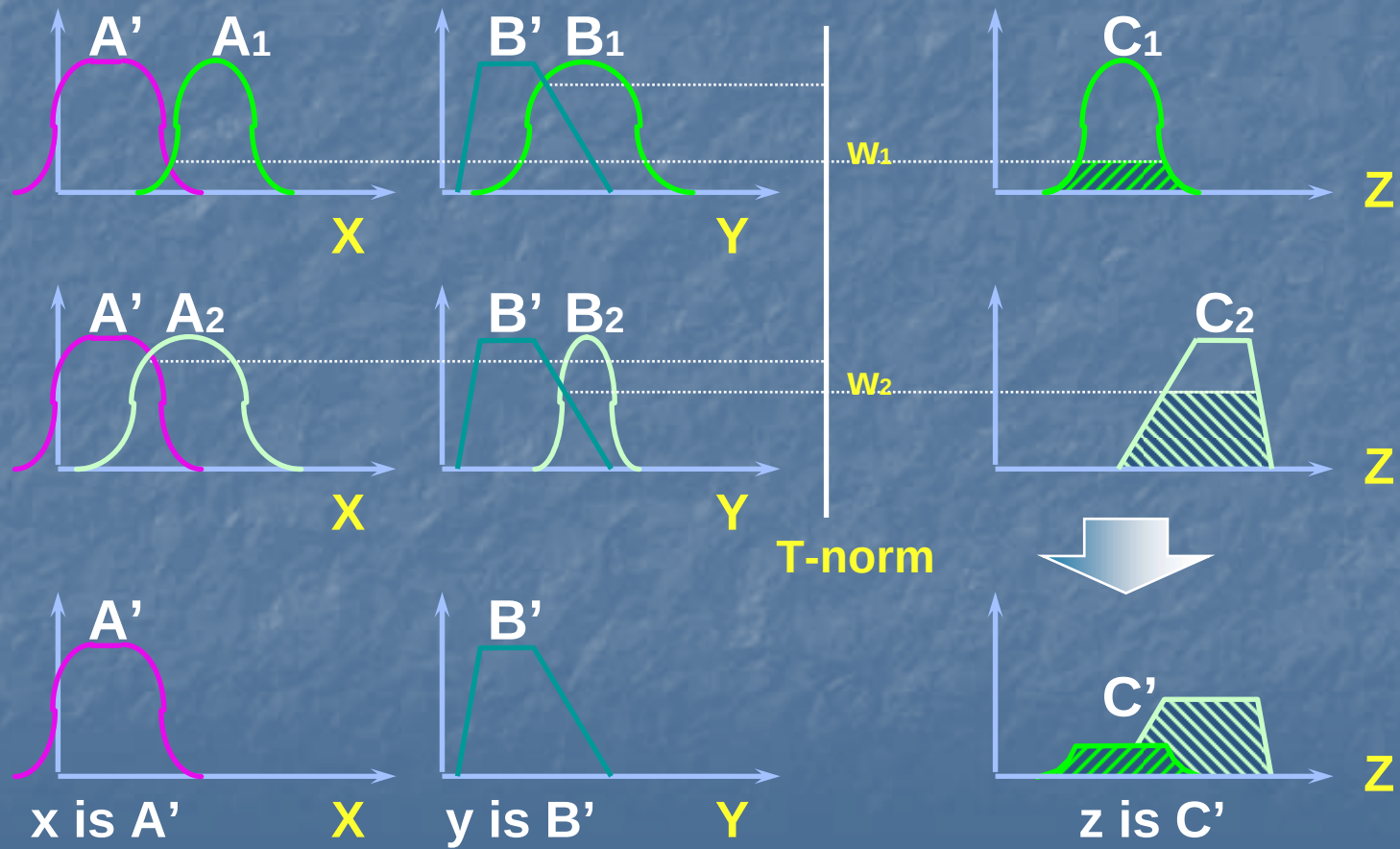


Fuzzy Reasoning

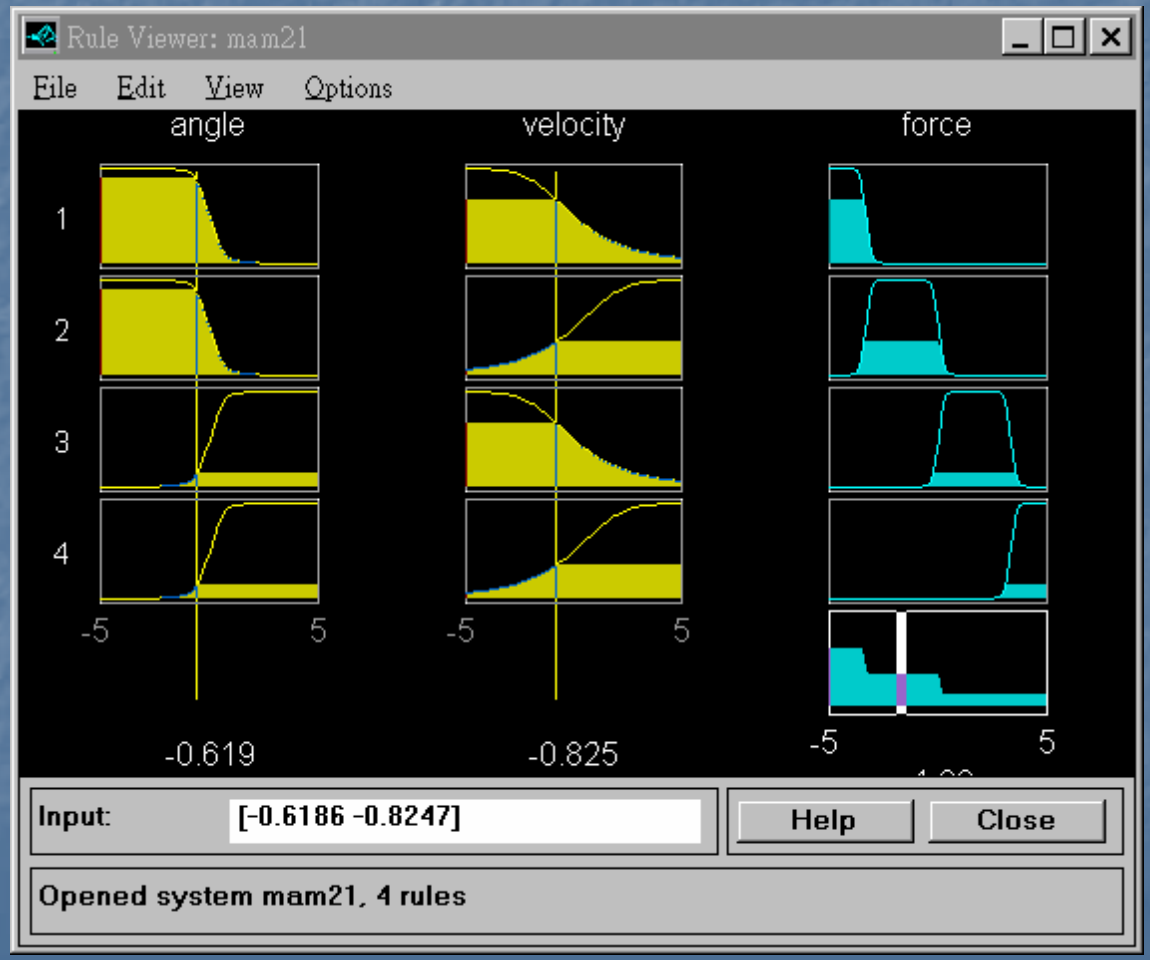
- Multiple rules with multiple antecedent
  - Rule 1: if x is A1 and y is B1 then z is C1
  - Rule 2: if x is A2 and y is B2 then z is C2
  - Fact: x is A’ and y is B’
  - Conclusion: z is C’
- Graphic Representation: (next slide)

Fuzzy Reasoning

- Graphics representation:



Fuzzy Reasoning: MATLAB Demo



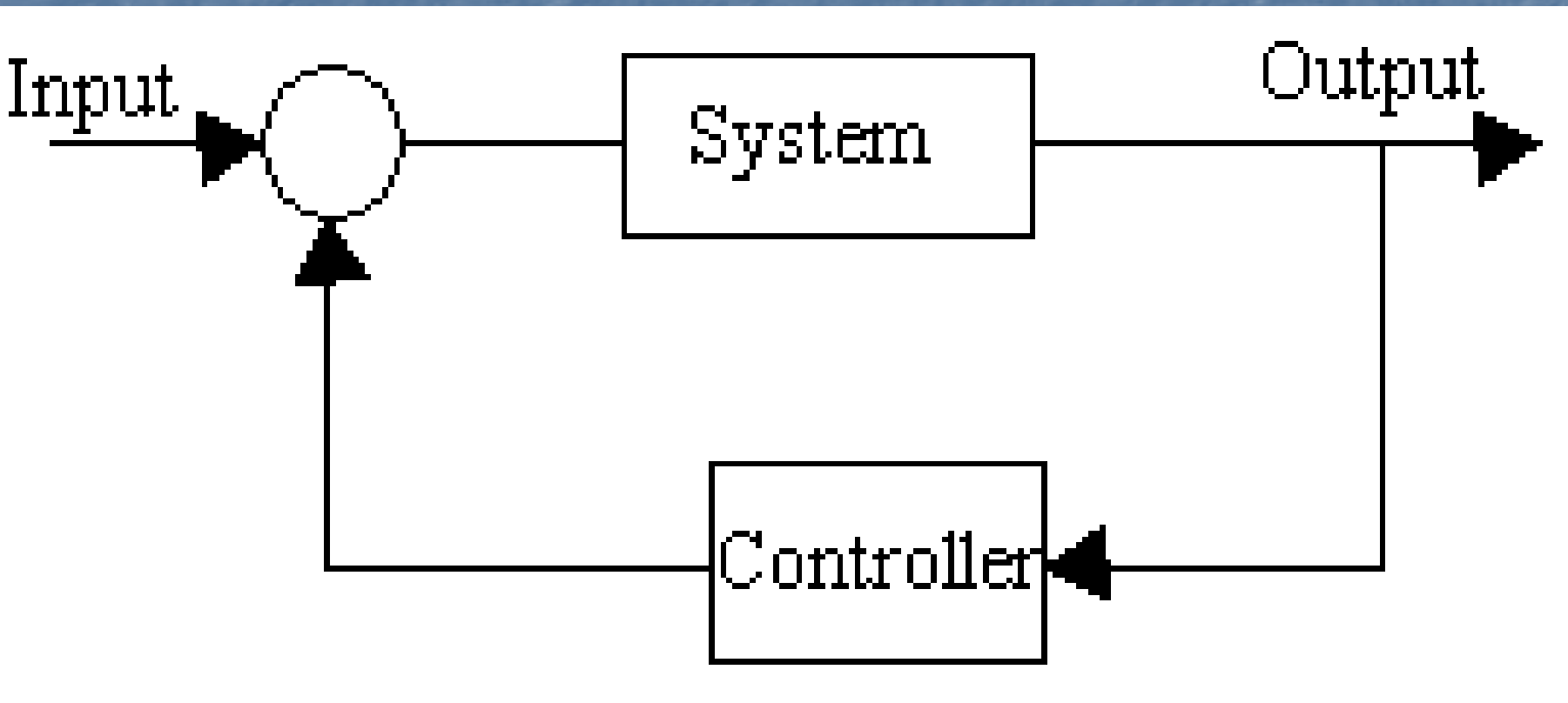
Other Variants

- Some terminology:
  - Degrees of compatibility (match)
  - Firing strength
  - Qualified (induced) MFs
  - Overall output MF

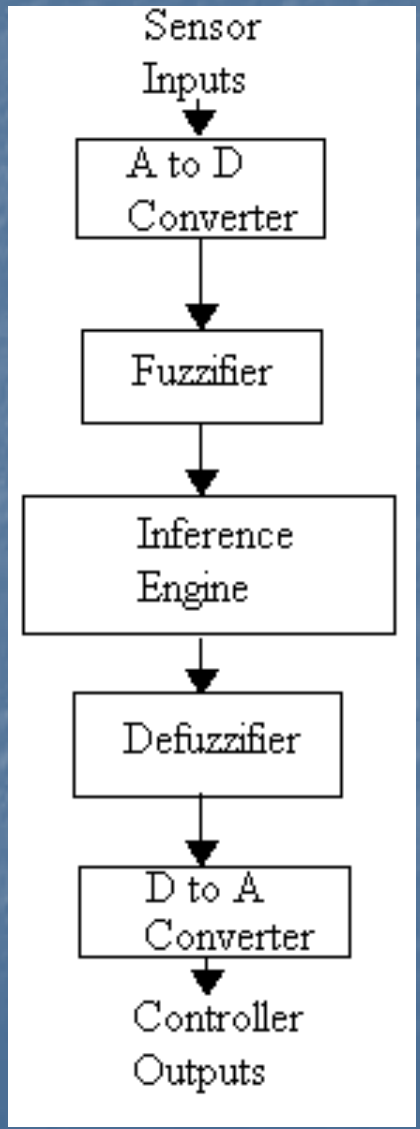
Fuzzy Applications

Fuzzy Controllers

Used to control a physical system



Structure of a Fuzzy Controller



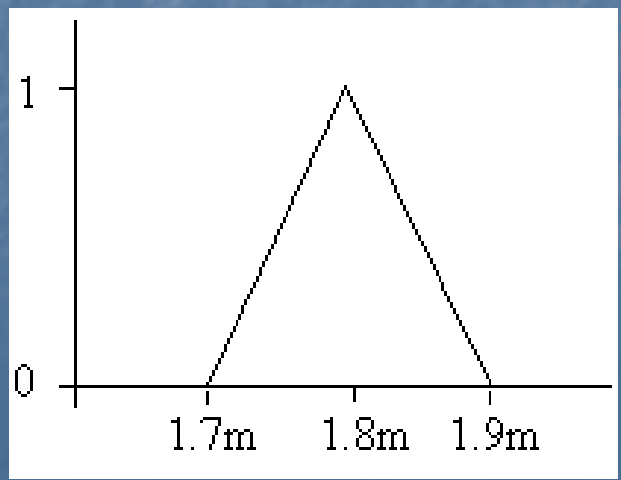
Fuzzification

- Conversion of real input to fuzzy set values
- e.g. Medium (x) = {

  0 if x >= 1.90 or x < 1.70,

  (1.90 - x)/0.1 if x >= 1.80 and x < 1.90,

  (x- 1.70)/0.1 if x >= 1.70 and x < 1.80 }



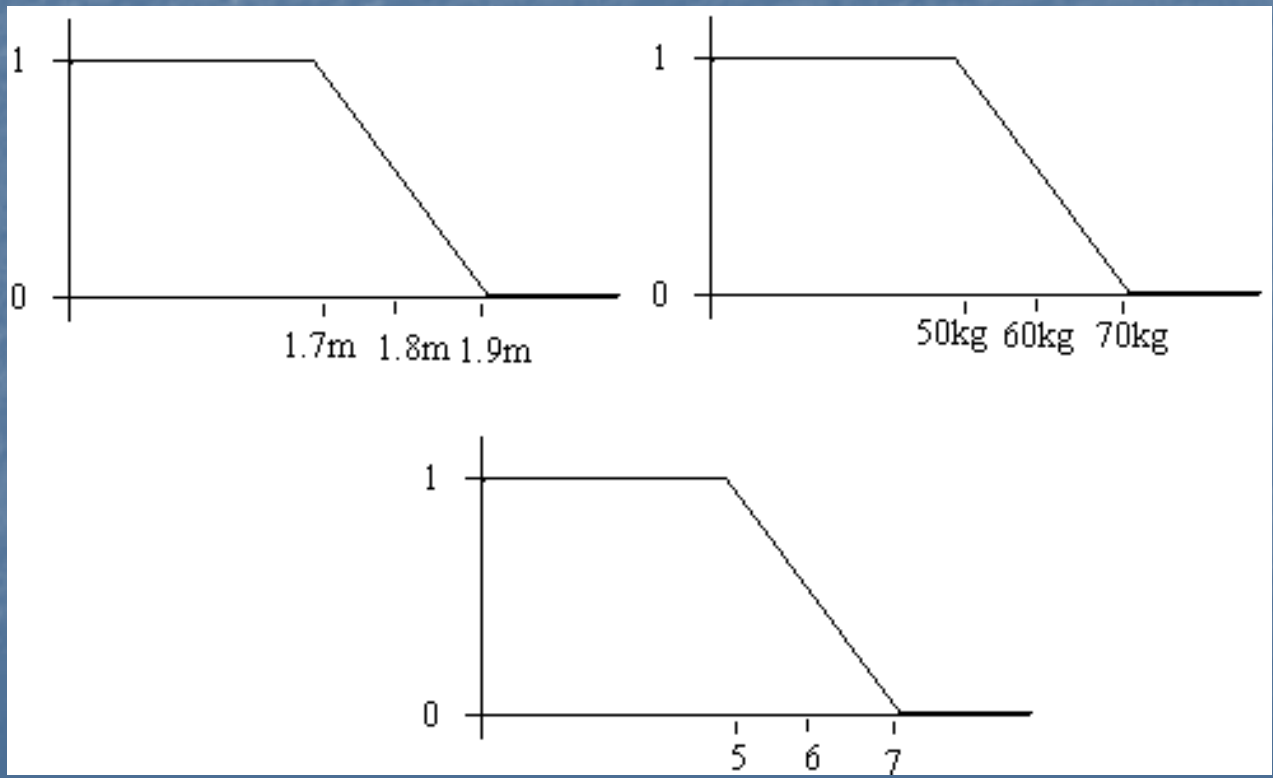
Inference Engine

- **Fuzzy rules**
  - based on fuzzy premises and fuzzy consequences

- **e.g.**
  - If height is Short and weight is Light then feet are Small
  - Short(height) AND Light(weight) => Small(feet)

Fuzzification & Inference Example

- If height is 1.7 m and weight is 55 kg
  - what is the value of Size (feet)

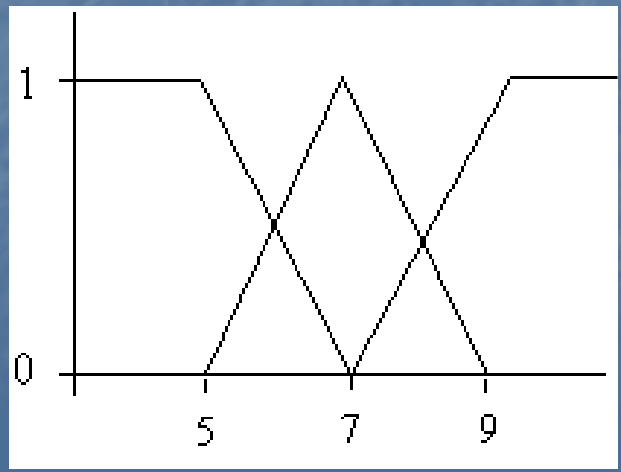


Defuzzification

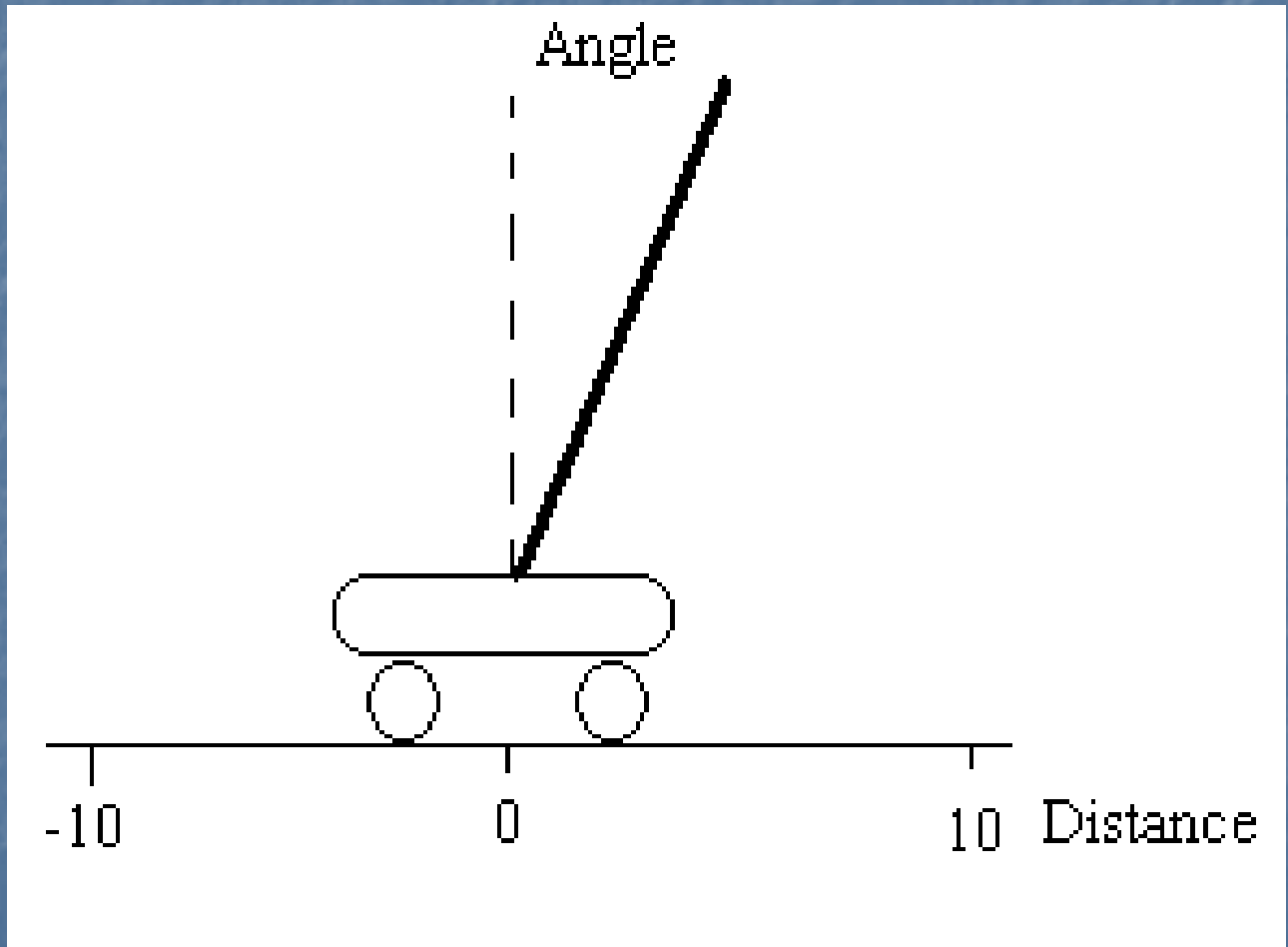
- Rule base has many rules
  - so some of the output fuzzy sets will have membership value > 0
  - Defuzzify to get a real value from the fuzzy outputs
    - ***One approach is to use a centre of gravity method***

Defuzzification Example

- Imagine we have output fuzzy set values:
  - Small membership value = 0.5
  - Medium membership value = 0.25
  - Large membership value = 0.0
- What is the deffuzzified value ?

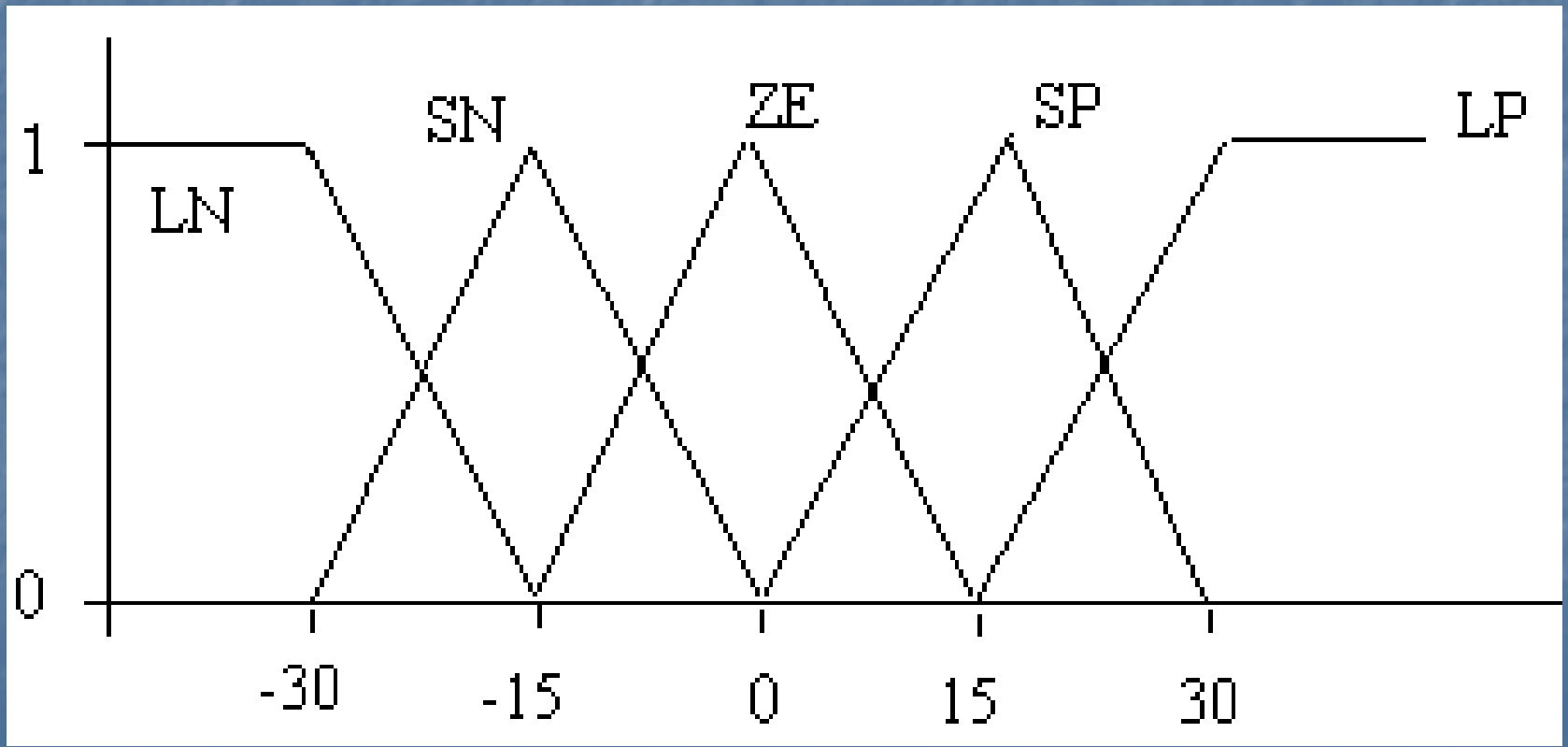


Fuzzy Control Example



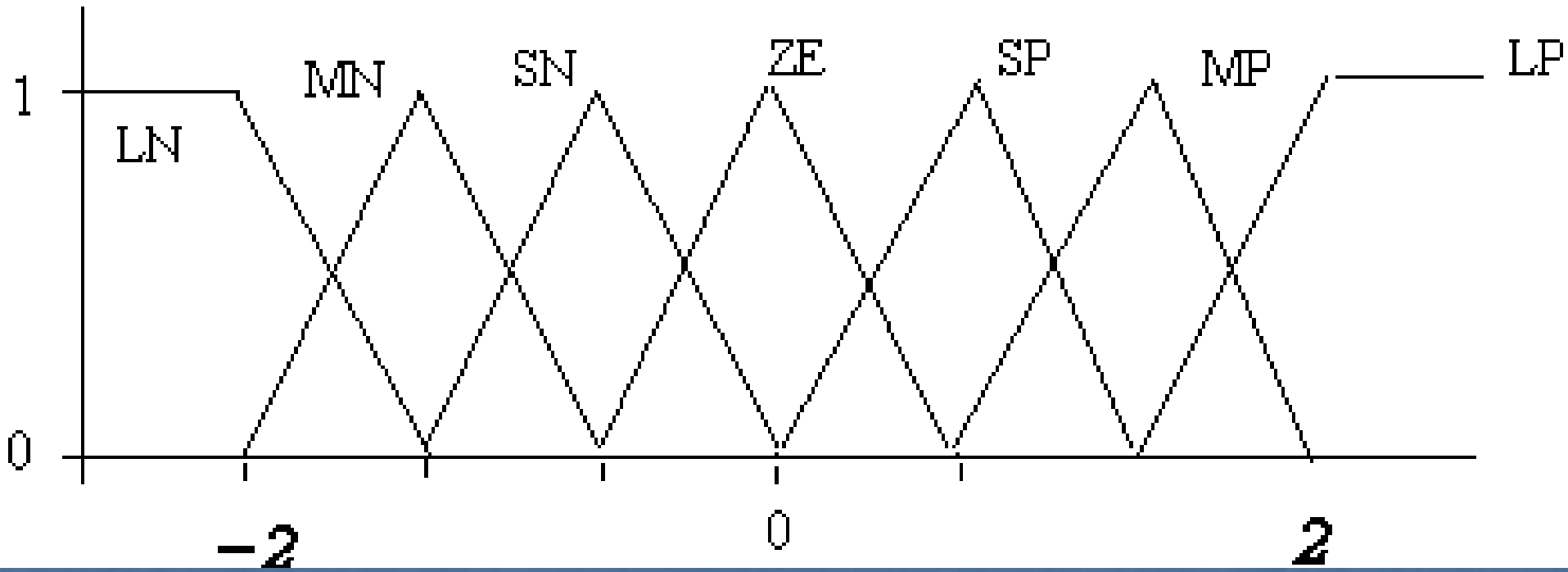
Input Fuzzy Sets

- Angle: -30 to 30 degrees



Output Fuzzy Sets

Car velocity: -2.0 to 2.0 meters per second



Fuzzy Rules

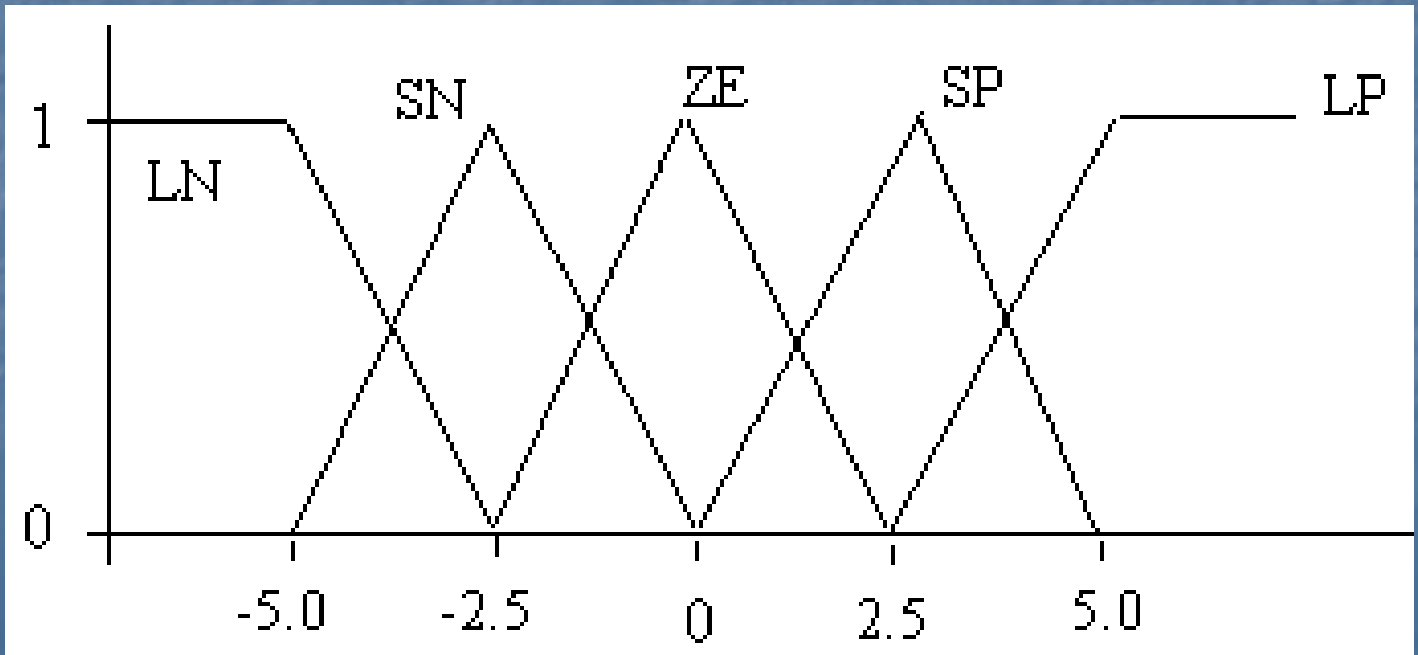
- If Angle is Zero then output ?
- If Angle is SP then output ?
- If Angle is SN then output ?
- If Angle is LP then output ?
- If Angle is LN then output ?

Fuzzy Rule Table

Angle	Output Velocity
LN	MP
SN	SP
ZE	ZE
SP	SN
LP	MN

Extended System

- Make use of additional information
  - angular velocity: -5.0 to 5.0 degrees/ second
- Gives better control



New Fuzzy Rules

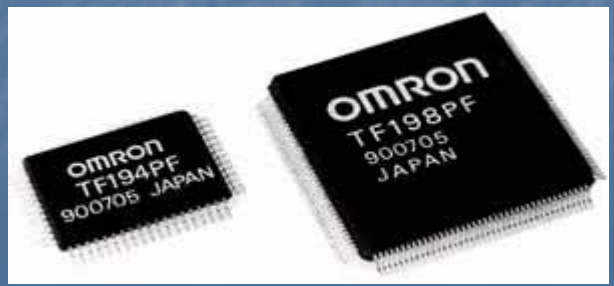
- Make use of old Fuzzy rules for angular velocity Zero
- If Angle is Zero and **Angular vel is Zero**
  then output Zero velocity
- If Angle is SP and **Angular vel is Zero**
  then output SN velocity
- If Angle is SN and **Angular vel is Zero**
  then output SP velocity

Table format

AngleVel / Angle	LN	SN	ZE	SP	LP
LN			MP		
SN			SP		
ZE			ZE		
SP			SN		
LP			MN		

Digital Fuzzy Processor

- Omron was the first to launch a controller employing fuzzy logic for improved control and tuning
- Production of the world's fastest **digital fuzzy processor (DFP)** in 1990.
- Reasoning speed of 10 MFLIPS (1 Million Fuzzy Logic Inferences per second)



Applications of Fuzzy Logic to Traffic Signal Control

Input Variables for Fuzzy Logic Traffic Signal Controller:

Maximum Queue Length (in metres): the distance in metres from the stop-line over which vehicles have queued

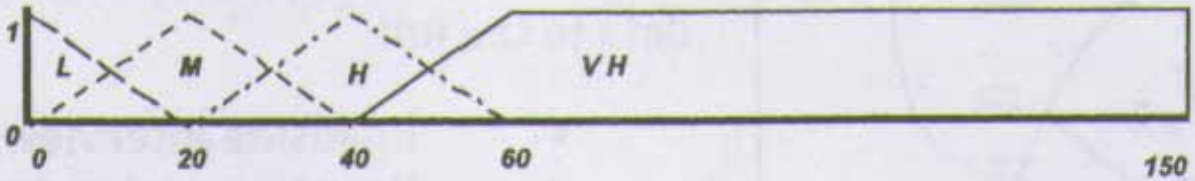
Average Occupancy Rate (%): percentage of time that the detection area was occupied by one or more vehicles.

Output Variable for the Controller:

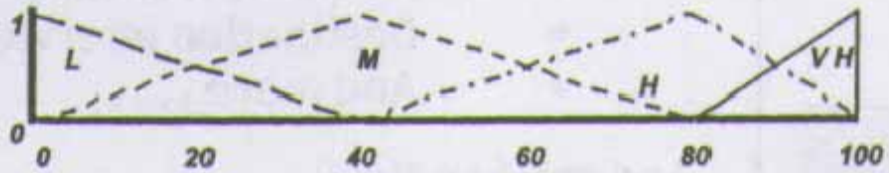
Weight [0, 100]: the degree of green traffic signal requirement

Improvement

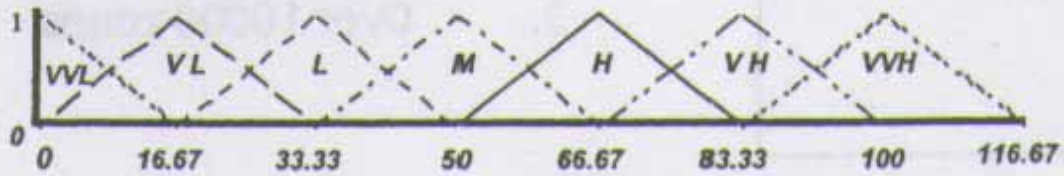
(up to) 25% in average travel time



(a) Input Fuzzy Variable 1: Maximum Queue Length (metres)



(b) Input Fuzzy Variable 2: Average Occupancy Rate (%)



(c) Output Fuzzy Variable: Weight

# of rules : 16

Rule Table/Matrix for Traffic Signal Control

Maximum Queue Length

Average Occupancy Rate	L	M	H	VH
L	VVL	L	M	H
M	VL	L	H	VH
H	L	M	H	VVH
VH	M	H	VH	VVH

L: Low, VL: Very Low, VVL: Very Very Low,

M: Medium, H: High, VH: Very High, VVH: Very Very High

More fuzzy logic applications

use *fuzdemos* MATLAB function

- Ball Juggler
- Water Tank (Water Level Control)
- Controlling cart and pole
- Controlling ball and beam
- Backing truck

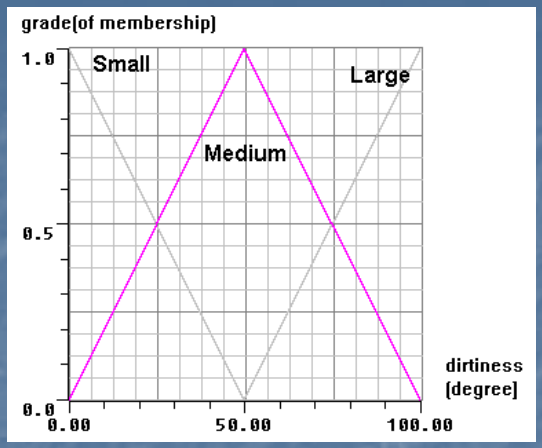
Live Demo

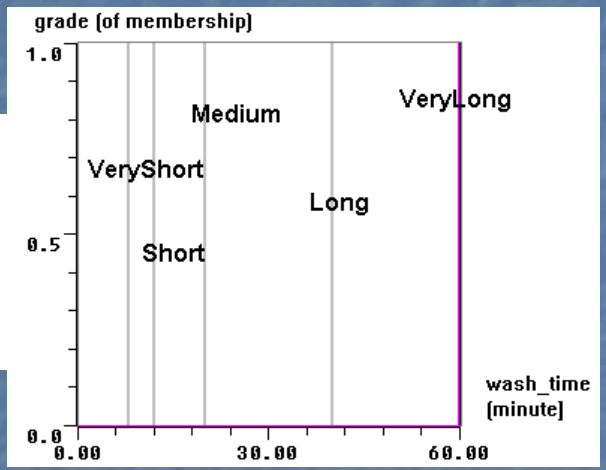
MATLAB

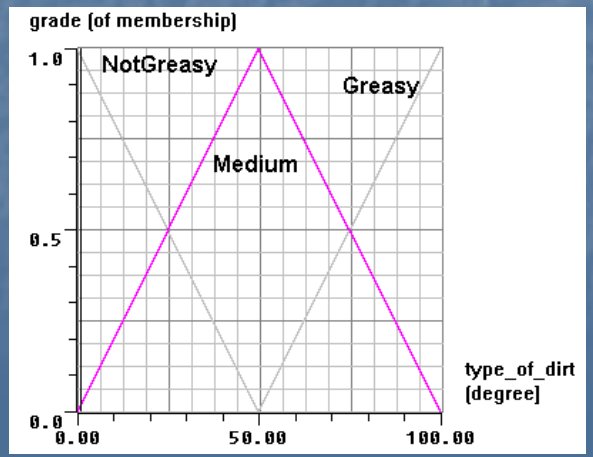
More fuzzy logic applications

Fuzzy Washing Machine
Camera Autofocus
Servo Motor Force Control
Glass Melting Furnace Control
Air Conditioner Control
Reactor Control
CAR Automatic Transmissions
Disc Drive Spindle Servos
Fuzzy Automated Manufacturing
Two-Stage Inverted Pendulum

Fuzzy Washing Machine







Fuzzy Controller for Washing Machine

Fuzzy Logic applications in Bio-Health Informatics

- Intelligent Detection of abnormal neonatal cerebral heomodynamics
- Bad cell prognosis
- Gene identification and gene network modelling

Intelligent Detection of abnormal neonatal cerebral heomodynamics

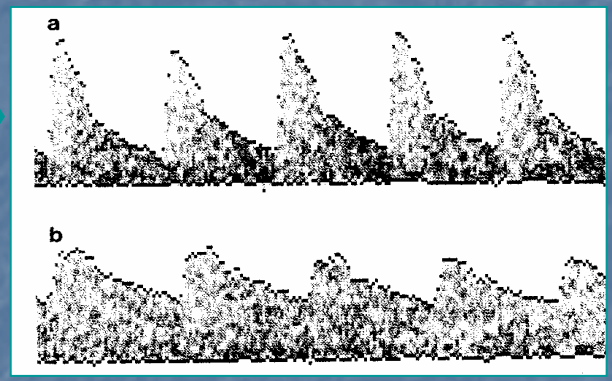
DOPPLER ULTRASOUND UNIT

SIGNAL PROCESSING UNIT

FEATURE EXTRACTION UNIT

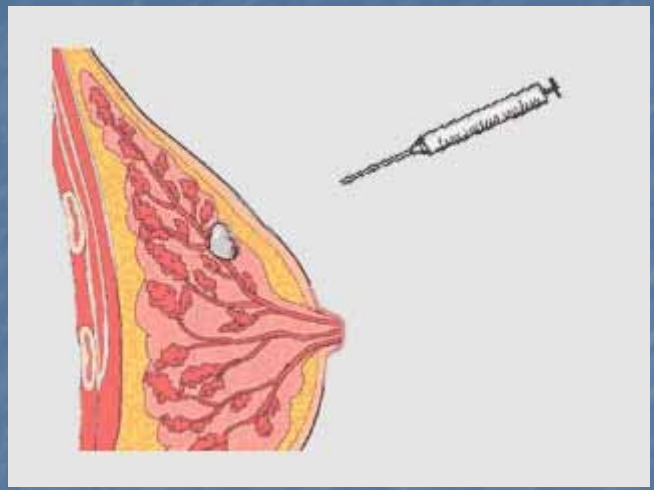
Neuro - Fuzzy System for Detection

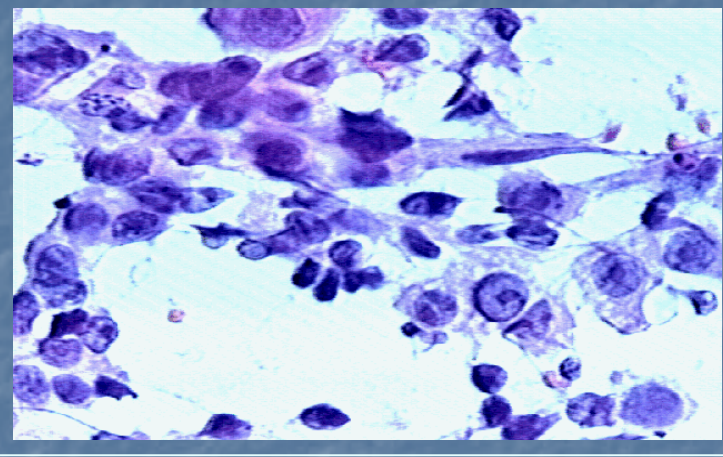
Normal baby



Baby with severe birth asphyxia

Bad Cells prognosis





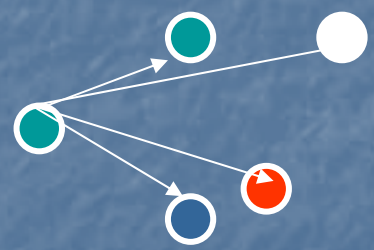
Neuro - Fuzzy Rule Based System

Decision on Prognosis

Gene identification and gene network modelling

Fuzzy System

List of genes associated with diseases/condition



Gene networks