Intelligent Control Systems Methods – Neural Networks

Prof Ka C Cheok

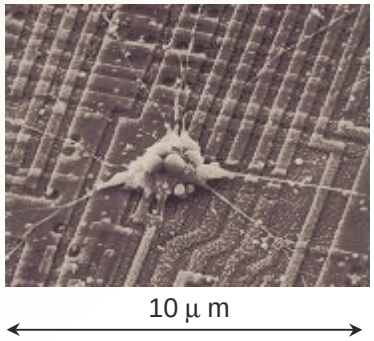
Dept of Electrical and Systems Engineering

Oakland University

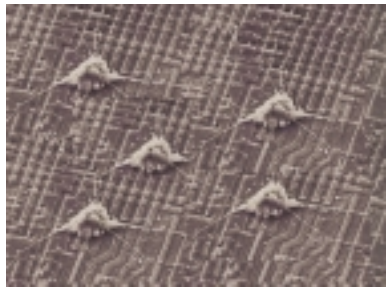
Rochester MI 48309

Summer Technical Workshop Series
NDIA 2nd Annual Intelligent Vehicle Systems Symposium
Grand Traverse Resort & Spa
Travers City, MI
June 3-5, 2002

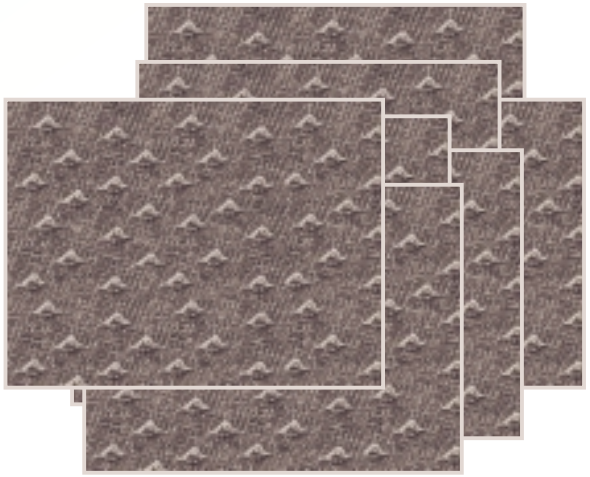
Biological Neural Network



a) An isolated neuron under a microscope with a magnification of about 10^6



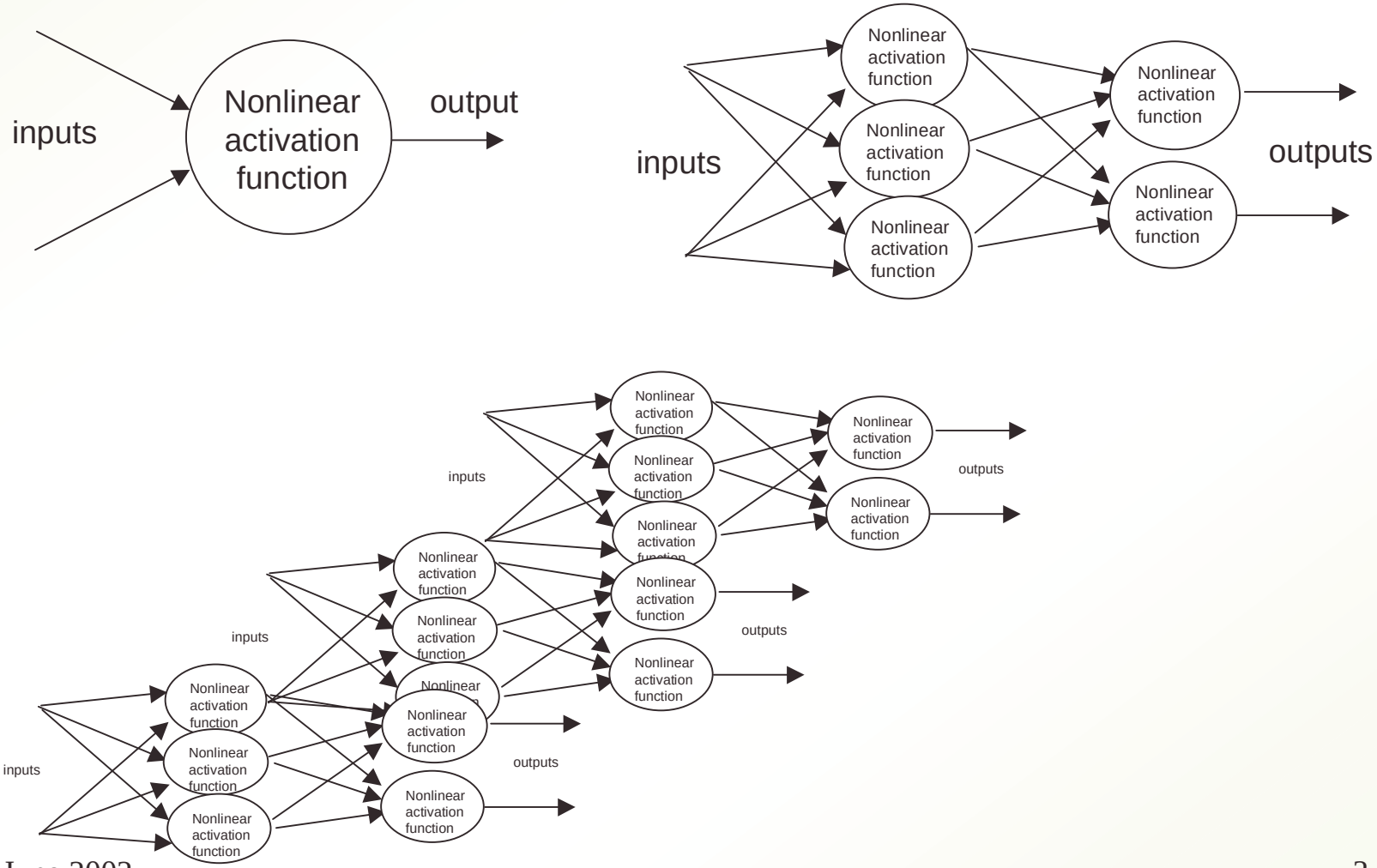
b) Looking at slices of live neurons under microscope, one can observe chemical causing electrical activities among the neurons.



c) A human brain has about a massive network of 10^{11} to 10^{12} neurons, connected in a random-like parallel pattern

Note of interest: Albert Einstein had 10^{13} - 10^{14} neurons. How do we know that? He donated his brain to Princeton Univ after his death so it could be preserved and studied. Somebody did.

Biologically Inspired Neural Network Model



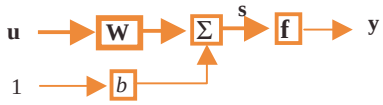
Artificial Neural Networks (ANN)

ANN is a branch of science, engineering and technology involving

- **Observation of biological neural activities**

- **Observation of human behavior and decision activities**

- **Mathematical modeling of observation**

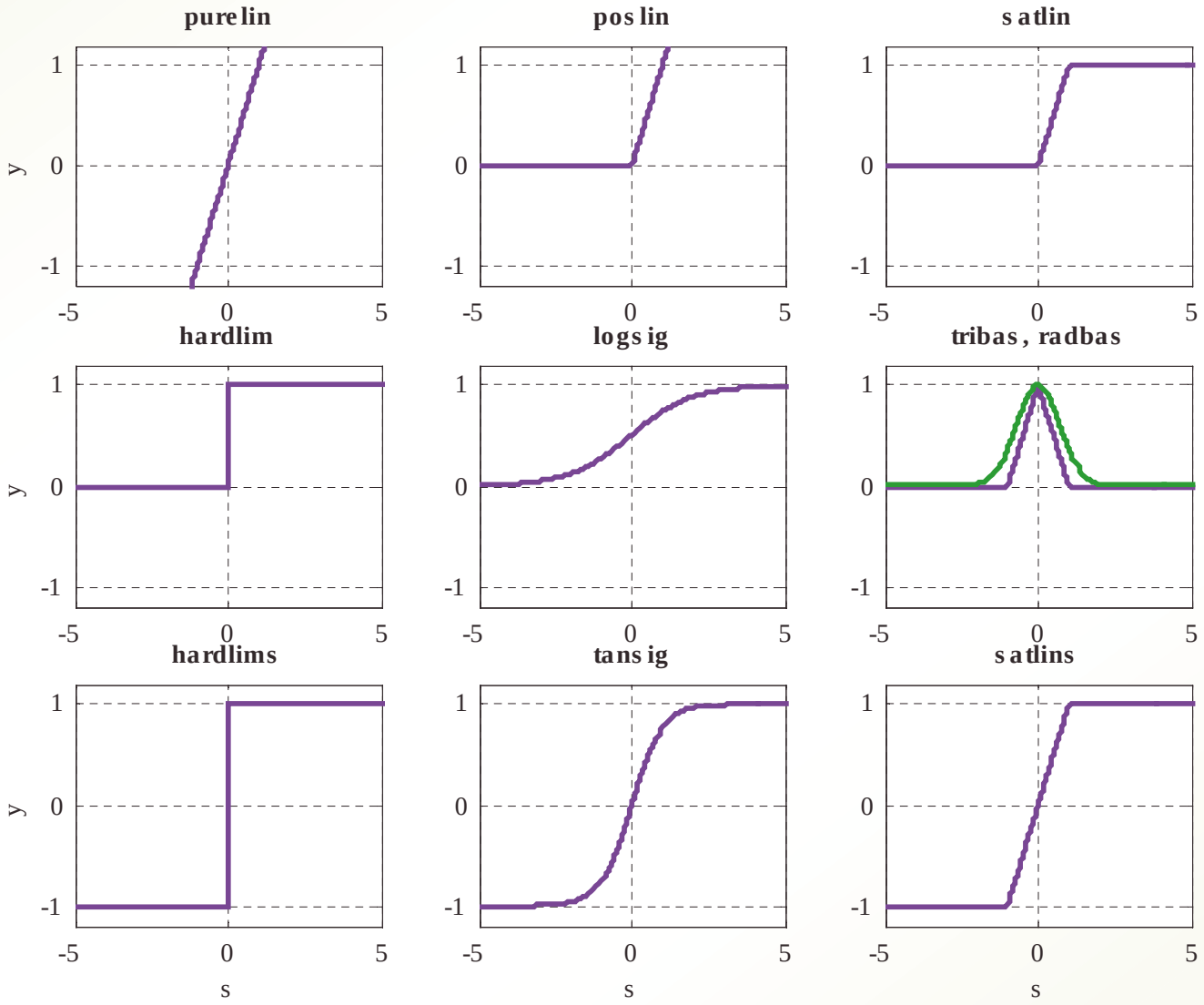


- **Training of model in learning to emulate examples**
- **Self-organization and adaptation of model to optimize performance**
- **Computer hardware and software aspects**
- **Applications to various fields**

Mathematics is a branch of Science.

Science is an endeavor to understand Nature.
Engineering is an endeavor to extend Nature
Technology reflects the success of these endeavors. *KaCC*

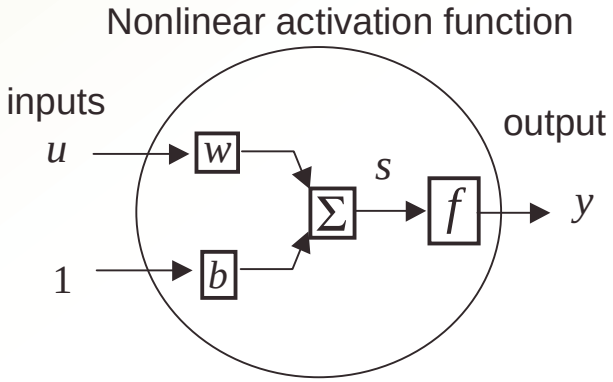
Activation threshold and squashing functions



Matlab function	*Math expression* $y = f(s)$	*Transfer function*
purelin	$y = f(s) = s$	pure linear function
poslin	$y = \begin{cases} s & \text{if } s > 0 \\ 0 & \text{if } s \le 0 \end{cases}$	positive linear function
satlin	$y = \begin{cases} 0 & \text{if } s < 0 \\ s & \text{if } 0 \le s \le s_1 \\ 1 & \text{if } s_1 \le s \end{cases}$	saturation linear function
hard lim	$y = \begin{cases} 1 & \text{if } s > 0 \\ 0 & \text{if } s \le 0 \end{cases}$	hard limits
log *sig*	$y = \frac{1}{1+e^{-s}}$	logarithmic sigmoid
hard lim *s*	$y = \begin{cases} 1 & \text{if } s > 0 \\ -1 & \text{if } s \le 0 \end{cases}$	hard limits symmetric
tan *sig*	$y = \frac{1-e^{-s}}{1+e^{-s}}$	hyperbolic tangent sigmoid
sat lim *s*	$y = \begin{cases} -1 & \text{if } s < -s_1 \\ s & \text{if } -s_1 \le s \le s_1 \\ 1 & \text{if } s_1 \le s \end{cases}$	saturation limits symmetric
tribas	$y = \begin{cases} 0 & \text{if } s < -1 \text{ or } s > 1 \\ s+1 & \text{if } -1 \le s < 0 \\ -s+1 & \text{if } 0 \le s \le 1 \end{cases}$	triangular basis
radbas	$y = e^{-s^2}$	radial basis

Feedforward Neural Network (FNN) Model

1-input 1-output single feedforward neuron model



$w = weight$

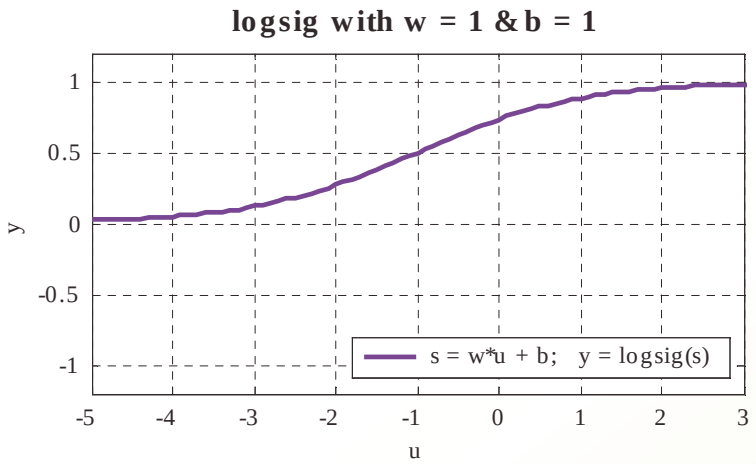
$b = bias$

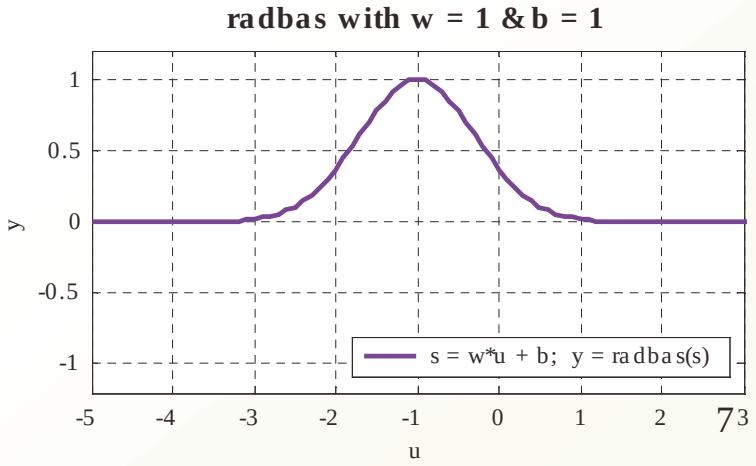
Math model

$$s = wu + b$$

$$y = f(s)$$

Examples of I/O mapping

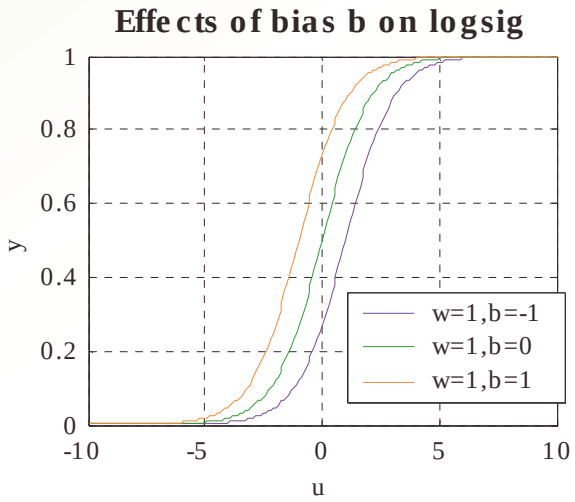


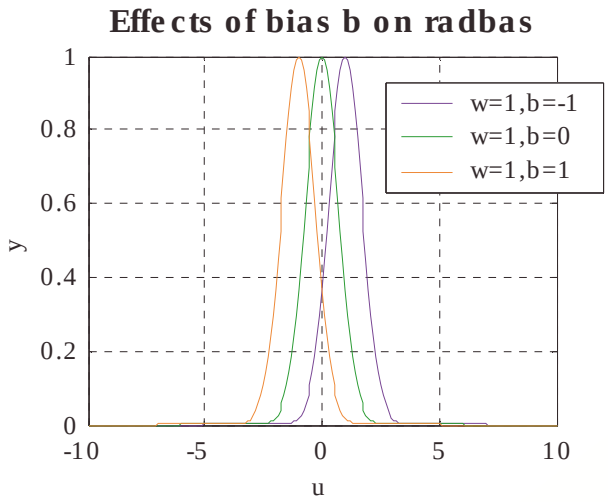


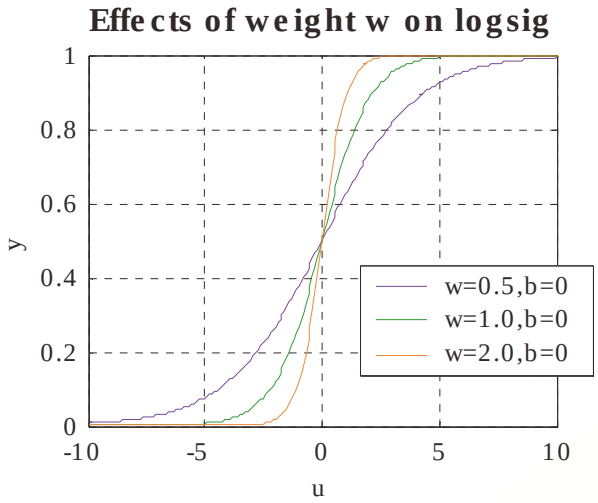
Math model

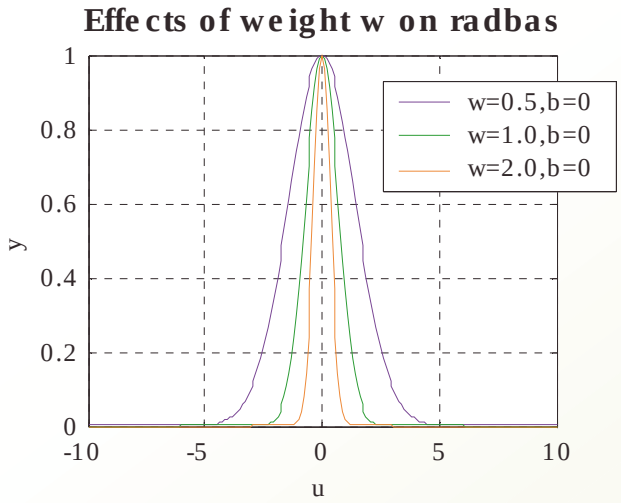
$$s = wu + b$$

$$y = f(s)$$

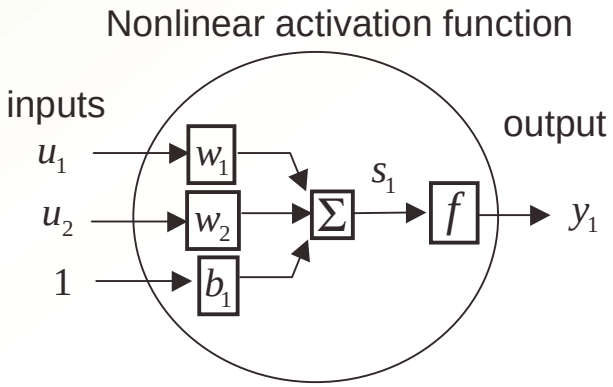








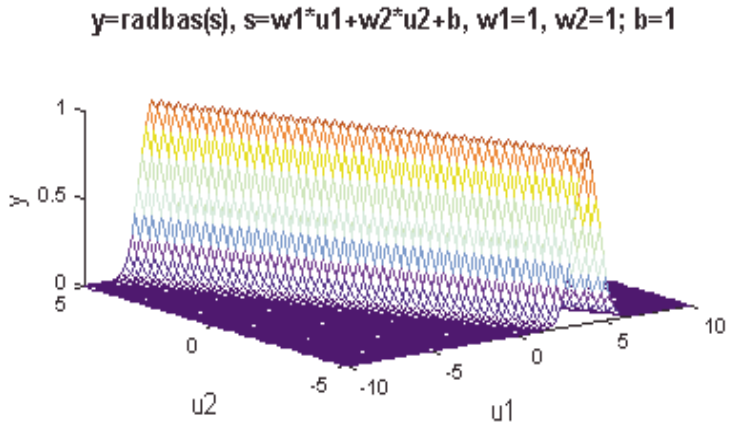
2-input 1-output single feedforward neuron model

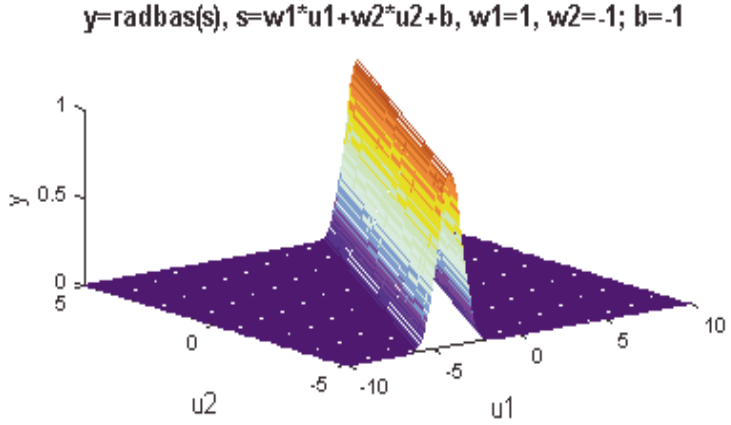


$$s_1 = w_1 u_1 + w_2 u_2 + b_1$$

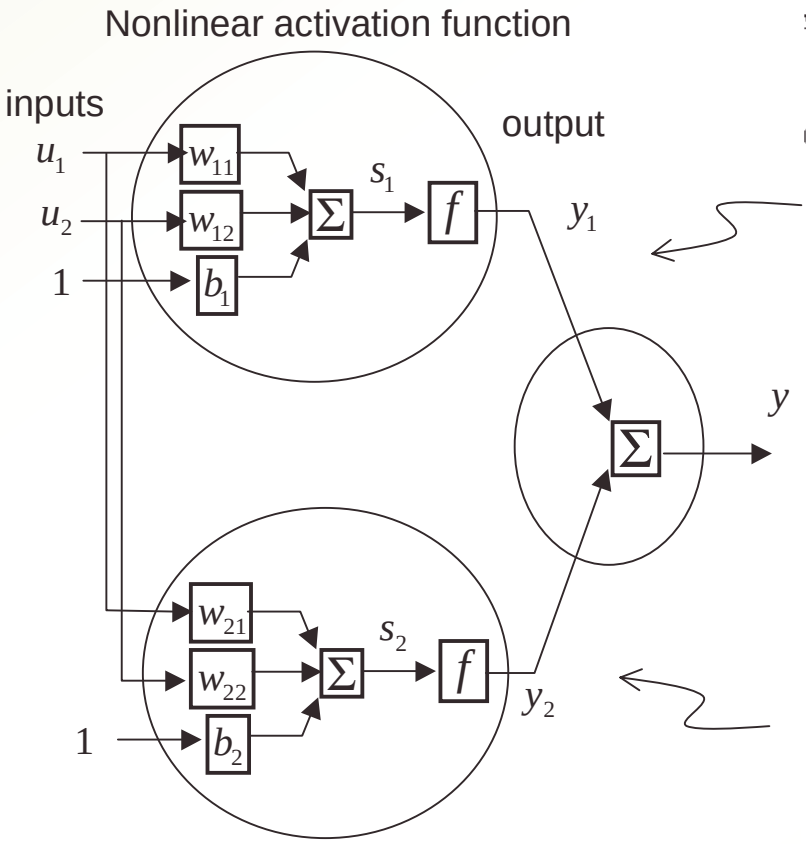
$$y_1 = f(s_1)$$

Examples of I/O mapping

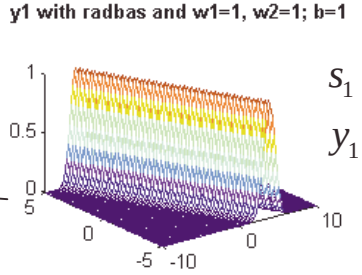




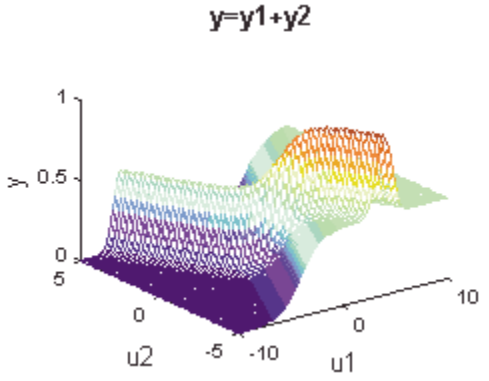
2-2-1 FNN *(2 inputs, 2 hidden neurons, 1 output neuron)*

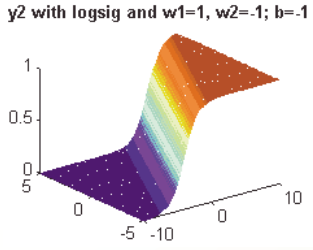


Examples of I/O mapping



$$s_1 = w_{11}u_1 + w_{12}u_2 + b_1$$
$$y_1 = f(s_1)$$





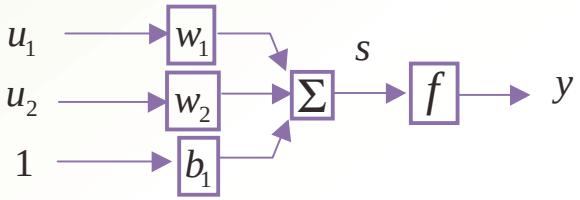
$$s_2 = w_{21}u_1 + w_{22}u_2 + b_2$$
$$y_2 = f(s_2)$$

Note: Feedforward neural networks (FFNN) are capable of mapping various input-output patterns!

More example

Matrix-Vector Models for FNN

Individual Variable Form

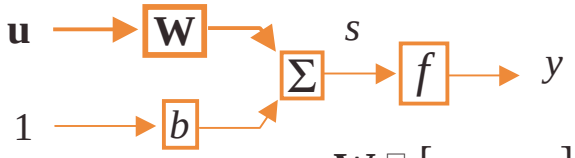


$$s = w_1 u_1 + w_2 u_2 + b$$

$$y = f(s)$$

$$s = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + b$$

Matrix-Vector Form



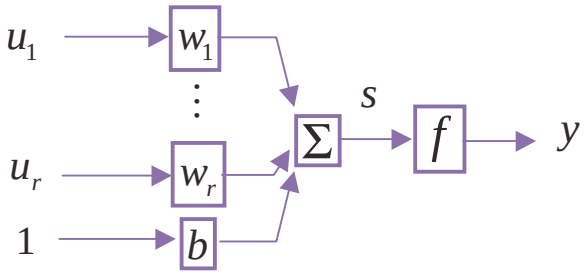
$$\mathbf{W} = \begin{bmatrix} w_1 & w_2 \end{bmatrix}$$

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$s = \mathbf{W}\mathbf{u} + b$$

$$y = f(s)$$

Individual Variable Form

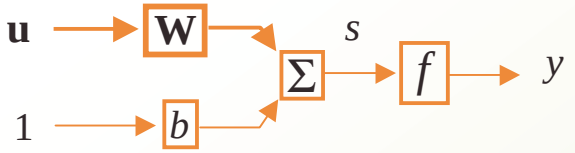


$$s = w_1 u_1 + \cdots + w_r u_r + b$$

$$y = f(s)$$

$$s = \begin{bmatrix} w_1 & \cdots & w_r \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix} + b$$

Matrix-Vector Form



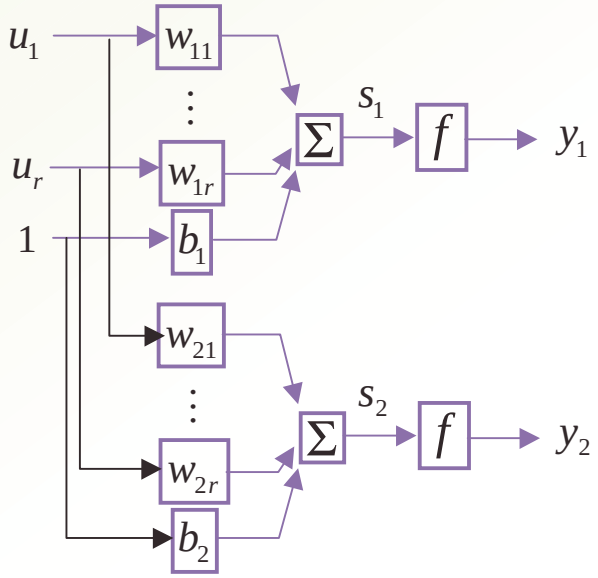
$$\mathbf{W} = \begin{bmatrix} w_1 & \cdots & w_r \end{bmatrix}$$

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}$$

$$s = \mathbf{W}\mathbf{u} + b$$

$$y = f(s)$$

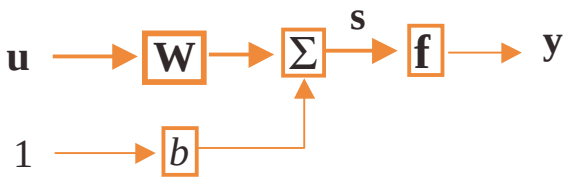
Individual Variable Form



$$s_1 = w_{11}u_1 + \cdots + w_{1r}u_r + b_1$$
$$y_1 = f(s_1)$$

$$s_2 = w_{21}u_1 + \cdots + w_{2r}u_r + b_2$$
$$y_2 = f(s_2)$$

Matrix-Vector Form



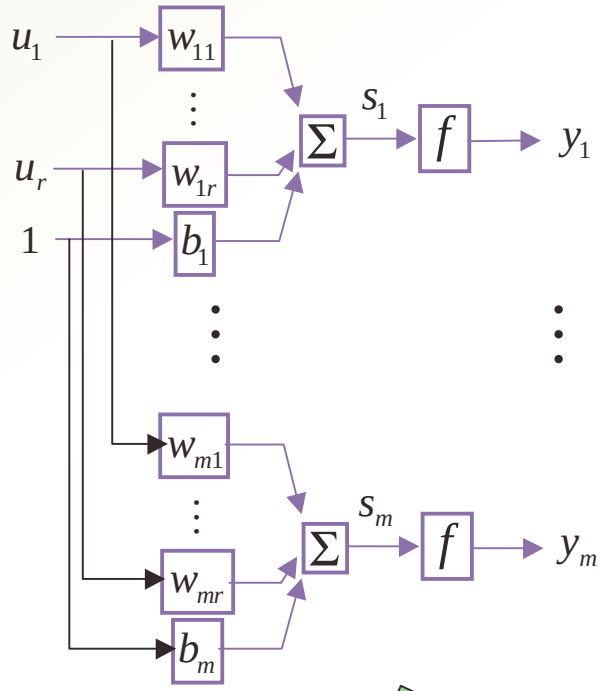
$$\mathbf{s} = \mathbf{W}\mathbf{u} + \mathbf{b}$$
$$\mathbf{y} = \mathbf{f}(\mathbf{s})$$

$$\underbrace{\begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix}}_{\mathbf{s}} = \underbrace{\begin{bmatrix} w_{11} & \cdots & w_{1R} \\ \vdots & & \vdots \\ w_{m1} & \cdots & w_{mr} \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}}_{\mathbf{u}} + \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}}_{\mathbf{b}}$$

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} f_1(s_1) \\ \vdots \\ f_m(s_m) \end{bmatrix}}_{\mathbf{f}(\mathbf{s})}$$

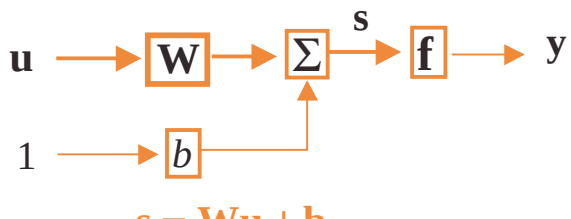
r-input m-output FNN

Individual Variable Form



$$s_1 = w_{11}u_1 + \cdots + w_{1r}u_r + b_1$$
$$y_1 = f(s_1)$$
$$\vdots$$
$$s_m = w_{m1}u_1 + \cdots + w_{mr}u_r + b_m$$
$$y_m = f(s_m)$$

Matrix-Vector Form



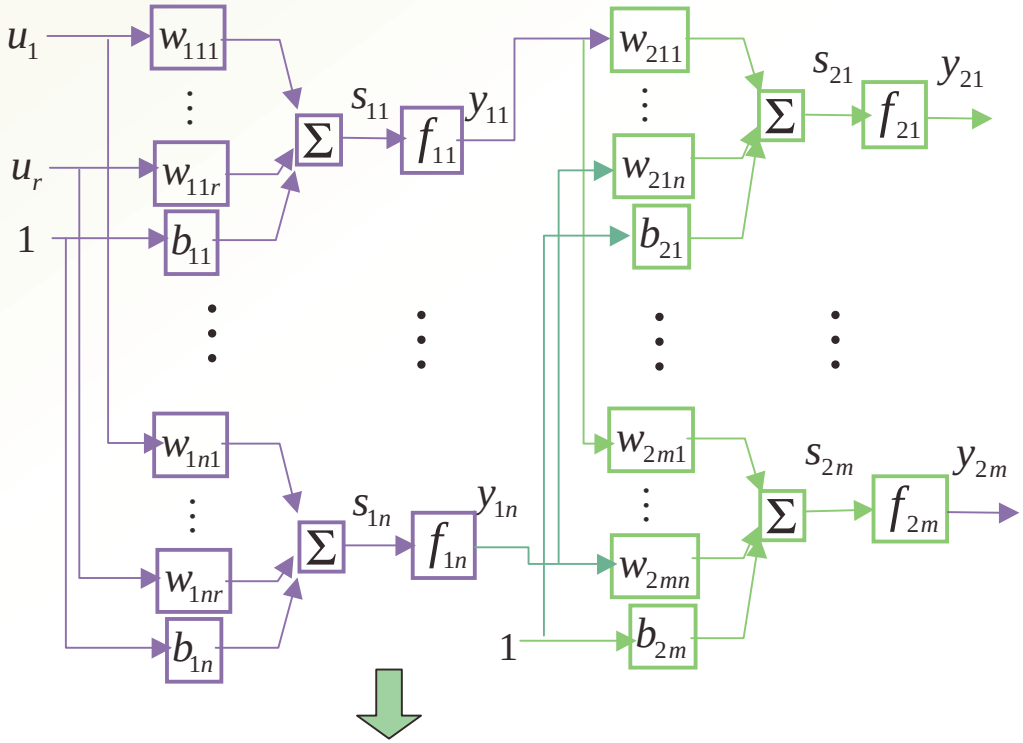
$$\mathbf{s} = \mathbf{W}\mathbf{u} + \mathbf{b}$$
$$\mathbf{y} = \mathbf{f}(\mathbf{s})$$

$$\underbrace{\begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix}}_{\mathbf{s}} = \underbrace{\begin{bmatrix} w_{11} & \cdots & w_{1R} \\ \vdots & & \vdots \\ w_{m1} & \cdots & w_{mr} \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}}_{\mathbf{u}} + \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}}_{\mathbf{b}}$$

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} f_1(s_1) \\ \vdots \\ f_m(s_m) \end{bmatrix}}_{\mathbf{f}(\mathbf{s})}$$

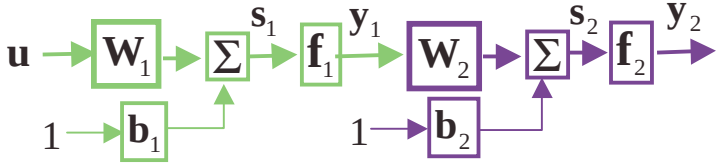
r-n-m FNN *(r-inputs, n-nodes, m-outputs)*

Individual Variable Form



$$s_{11} = w_{111}u_1 + \cdots + w_{11r}u_r + b_{11} \qquad s_{21} = w_{211}y_{11} + \cdots + w_{21n}y_{1n} + b_{21}$$
$$y_{11} = f(s_{11}) \qquad y_{21} = f(s_{21})$$
$$\vdots \qquad \vdots$$
$$s_{1n} = w_{1n1}u_1 + \cdots + w_{1nr}u_r + b_{1n} \qquad s_{2m} = w_{2m1}y_{11} + \cdots + w_{2mn}y_{1n} + b_{2m}$$
$$y_{1n} = f(s_{1n}) \qquad y_{2m} = f(s_{2m})$$

Matrix-Vector Form



$$\mathbf{s}_1 = \mathbf{W}_1\mathbf{u} + \mathbf{b}_1 \qquad \mathbf{s}_2 = \mathbf{W}_2\mathbf{y}_1 + \mathbf{b}_2$$
$$\mathbf{y}_1 = \mathbf{f}(\mathbf{s}_1) \qquad \mathbf{y}_2 = \mathbf{f}(\mathbf{s}_2)$$

$$\underbrace{\begin{bmatrix} s_{11} \\ \vdots \\ s_{1n} \end{bmatrix}}_{\mathbf{s}_1} = \underbrace{\begin{bmatrix} w_{111} & \cdots & w_{11r} \\ \vdots & & \vdots \\ w_{1n1} & \cdots & w_{1nr} \end{bmatrix}}_{\mathbf{W}_1} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}}_{\mathbf{u}} + \underbrace{\begin{bmatrix} b_{11} \\ \vdots \\ b_{1n} \end{bmatrix}}_{\mathbf{b}_1}$$

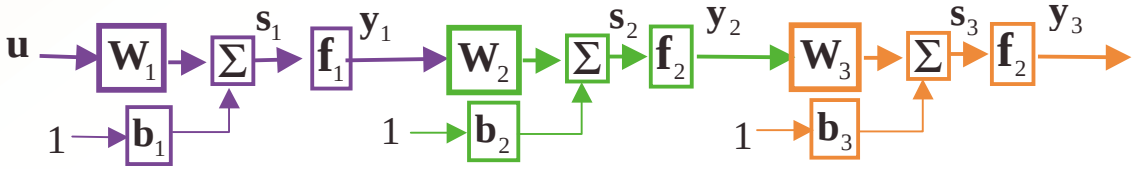
$$\underbrace{\begin{bmatrix} y_{11} \\ \vdots \\ y_{1n} \end{bmatrix}}_{\mathbf{y}_1} = \underbrace{\begin{bmatrix} f_{11}(s_{11}) \\ \vdots \\ f_{1n}(s_{1n}) \end{bmatrix}}_{\mathbf{f}_1(\mathbf{s}_1)}$$

$$\underbrace{\begin{bmatrix} s_{21} \\ \vdots \\ s_{2m} \end{bmatrix}}_{\mathbf{s}_2} = \underbrace{\begin{bmatrix} w_{211} & \cdots & w_{21n} \\ \vdots & & \vdots \\ w_{2m1} & \cdots & w_{2mn} \end{bmatrix}}_{\mathbf{W}_2} \underbrace{\begin{bmatrix} y_{11} \\ \vdots \\ y_{1n} \end{bmatrix}}_{\mathbf{y}_1} + \underbrace{\begin{bmatrix} b_{21} \\ \vdots \\ b_{2m} \end{bmatrix}}_{\mathbf{b}_2}$$

$$\underbrace{\begin{bmatrix} y_{21} \\ \vdots \\ y_{2m} \end{bmatrix}}_{\mathbf{y}_2} = \underbrace{\begin{bmatrix} f_{21}(s_{21}) \\ \vdots \\ f_{2m}(s_{2m}) \end{bmatrix}}_{\mathbf{f}_2(\mathbf{s}_2)}$$

r-n_1–n_2-m FNN *(r-inputs, n_1-nodes, n_1-nodes m-outputs) (3-layers)*

Matrix-Vector Form



$$\mathbf{s}_1 = \mathbf{W}_1\mathbf{u} + \mathbf{b}_1 \qquad \mathbf{s}_2 = \mathbf{W}_2\mathbf{y}_1 + \mathbf{b}_2 \qquad \mathbf{s}_3 = \mathbf{W}_3\mathbf{y}_2 + \mathbf{b}_3$$

$$\mathbf{y}_1 = \mathbf{f}(\mathbf{s}_1) \qquad \mathbf{y}_2 = \mathbf{f}(\mathbf{s}_2) \qquad \mathbf{y}_3 = \mathbf{f}(\mathbf{s}_3)$$

$$\underbrace{\begin{bmatrix} s_{11} \\ \vdots \\ s_{1n} \end{bmatrix}}_{\mathbf{s}_1} = \underbrace{\begin{bmatrix} w_{111} & \cdots & w_{11r} \\ \vdots & & \vdots \\ w_{1n1} & \cdots & w_{1nr} \end{bmatrix}}_{\mathbf{W}_1} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}}_{\mathbf{u}} + \underbrace{\begin{bmatrix} b_{11} \\ \vdots \\ b_{1n} \end{bmatrix}}_{\mathbf{b}_1}$$

$$\underbrace{\begin{bmatrix} y_{11} \\ \vdots \\ y_{1n} \end{bmatrix}}_{\mathbf{y}_1} = \underbrace{\begin{bmatrix} f_{11}(s_{11}) \\ \vdots \\ f_{1n}(s_{1n}) \end{bmatrix}}_{\mathbf{f}_1(\mathbf{s}_1)}$$

$$\underbrace{\begin{bmatrix} s_{21} \\ \vdots \\ s_{2n_2} \end{bmatrix}}_{\mathbf{s}_2} = \underbrace{\begin{bmatrix} w_{211} & \cdots & w_{21n_1} \\ \vdots & & \vdots \\ w_{2n_21} & \cdots & w_{2n_2n_1} \end{bmatrix}}_{\mathbf{W}_2} \underbrace{\begin{bmatrix} y_{11} \\ \vdots \\ y_{1n_1} \end{bmatrix}}_{\mathbf{y}_1} + \underbrace{\begin{bmatrix} b_{21} \\ \vdots \\ b_{2n_2} \end{bmatrix}}_{\mathbf{b}_2}$$

$$\underbrace{\begin{bmatrix} y_{21} \\ \vdots \\ y_{2n_2} \end{bmatrix}}_{\mathbf{y}_2} = \underbrace{\begin{bmatrix} f_{21}(s_{21}) \\ \vdots \\ f_{2n_2}(s_{2n_2}) \end{bmatrix}}_{\mathbf{f}_2(\mathbf{s}_2)}$$

$$\underbrace{\begin{bmatrix} s_{31} \\ \vdots \\ s_{3m} \end{bmatrix}}_{\mathbf{s}_3} = \underbrace{\begin{bmatrix} w_{311} & \cdots & w_{31n_2} \\ \vdots & & \vdots \\ w_{3m1} & \cdots & w_{3mn_2} \end{bmatrix}}_{\mathbf{W}_3} \underbrace{\begin{bmatrix} y_{21} \\ \vdots \\ y_{2n_2} \end{bmatrix}}_{\mathbf{y}_2} + \underbrace{\begin{bmatrix} b_{31} \\ \vdots \\ b_{3m} \end{bmatrix}}_{\mathbf{b}_3}$$

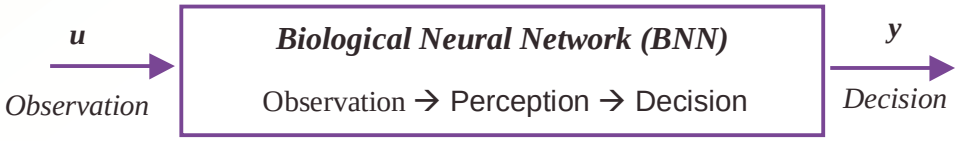
$$\underbrace{\begin{bmatrix} y_{31} \\ \vdots \\ y_{3m} \end{bmatrix}}_{\mathbf{y}_3} = \underbrace{\begin{bmatrix} f_{31}(s_{31}) \\ \vdots \\ f_{3m}(s_{3m}) \end{bmatrix}}_{\mathbf{f}_3(\mathbf{s}_3)}$$

FNN Can Emulate Examples

Human learns to imitate actions of others.

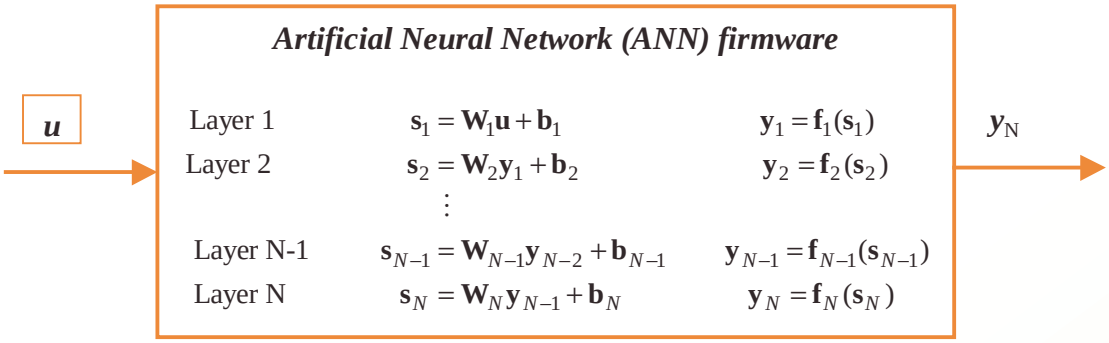
Biological neural networks are responsible for decision making capability of a person.
Artificial neural networks can be programmed to imitate the perception involved in the decision.

For example



Note: In general, we need not restrict an ANN to emulate only a BNN

may be replaced by



Note: In practice, we'd need to specify
The number of layers of neural network

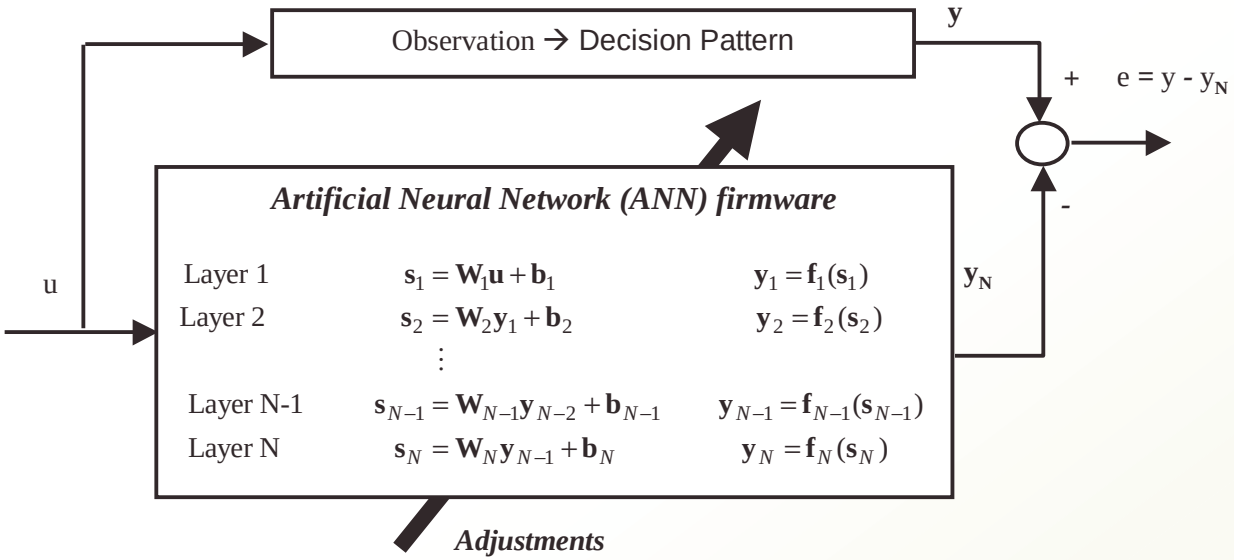
- The number of neurons in each layer
- The type of activation functions
- The weights and biases for these

Training FNN to Learn from Examples

Acquire several (if possible) sets of observation (input u) → decision (output) pattern representing the perception we would like to retain.

Specify the configuration for the FNN (E.g., r-n_1- n_2-m layers and type of activation functions). Guess the weights and biases, and compare the FNN outputs to the Training Patterns, as shown in the figure below)

Tune FNN by adjusting the weights and biases so that its output y_N matches that of the pattern ***y***. The goal then is to make the error $\boldsymbol{e = y - y_N}$, as small as possible when the FNN is presented with observation ***u***.



Optimization of Compared Outcome

The observation u and decision y in a multi-input multi-output pattern are vectors

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_{n_u} \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{n_y} \end{bmatrix}$$

There will be many instances/samples of the patterns:

$$\mathbf{U} = [\mathbf{u}(1) \quad \mathbf{u}(2) \quad \cdots \quad \mathbf{u}(K)]$$
$$\mathbf{Y} = [\mathbf{y}(1) \quad \mathbf{y}(2) \quad \cdots \quad \mathbf{y}(K)]$$

Similarly, the ANN output can be expressed as

$$\mathbf{Y}_N = [\mathbf{y}_N(1) \quad \mathbf{y}_N(2) \quad \cdots \quad \mathbf{y}_N(K)]$$

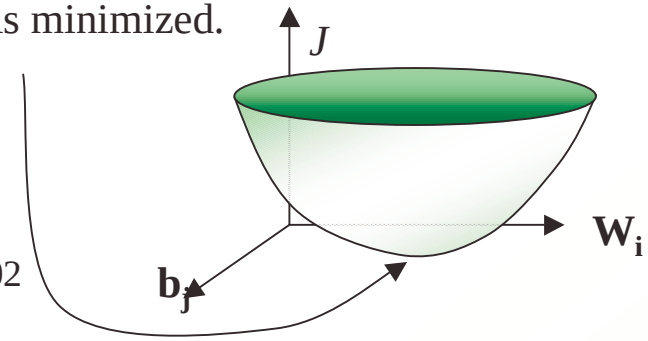
The error vectors (compared outcome) can then be represented as

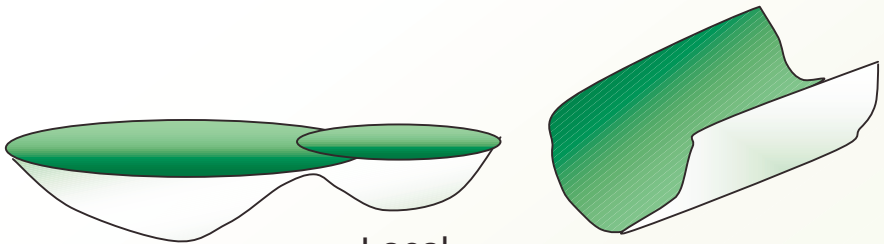
$$\mathbf{E} = [\mathbf{y}(1) - \mathbf{y}_N(1) \quad \mathbf{y}(2) - \mathbf{y}_N(2) \quad \cdots \quad \mathbf{y}(K) - \mathbf{y}_N(k)]$$
$$= [\mathbf{e}(1) \quad \mathbf{e}(2) \quad \cdots \quad \mathbf{e}(K)]$$

A typical cost function for judging the goodness of fit in the emulation is given by

$$J = \frac{1}{2}[\mathbf{e}'(1)\mathbf{e}(1) + \mathbf{e}'(2)\mathbf{e}(2) + \cdots + \mathbf{e}'(K)\mathbf{e}(K)]$$
$$= \frac{1}{2}\sum_{k=1}^{K} \mathbf{e}'(k)\mathbf{e}(k)$$

Find weights $\mathbf{W_i}$ and $\mathbf{b_j}$ such that the cost function is minimized.





Optimization Techniques

Given a cost function, there are several ways to find an optimum solution. Optimization techniques can be categorized onto the following approaches

Calculus Gradient techniques which adjust a parameter p based on sensitivity ($\Delta J/\Delta p$ = variation of cost function over variation of parameter). Examples: The Delta Rule, Hill Climbing, Back-Propagation, etc.

AI Heuristics techniques which expand a branch in tree search based on evaluation of the cost function. Examples: Breadth first, depth first, A* algorithm, etc.

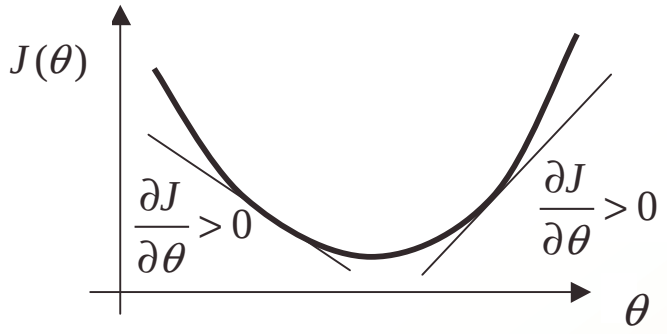
Evolution type techniques which use a population of parameters to evolve into better and better generations of parameters. Examples: Evolution algorithm, Genetic Algorithm, Genetic Programming

Delta Rule & Back-Propagation

The Delta Rule

Suppose we have a cost function $J(\theta)$ of that has a minimum as shown in the figure on the right. The Delta rule for searching for the minimum is to

- Step backward if the gradient is uphill
- Step forward if the gradient is downhill
- Stop if the gradient is close to being flat



θ may be a vector
Although drawn here as a scalar

. We can mathematically describe the rule as

Change in parameter value $\Delta\theta = -\gamma \frac{\partial J}{\partial \theta}$

Parameter update $\theta_{new} = \theta_{old} + \Delta\theta$

γ = factor that determines search step size

Back-propagation is a well-known gradient search technique for training FNN that is based on the Delta Rule

Training FNN using Back-Propagation

Back-Propagation for a 1-1 Neuron

Input-output pattern to be emulated: $\{u(1), u(2), \cdots, u(K)\}$ $\{y(1), y(2), \cdots, y(K)\}$

Math model of neuron: $s_1 = wu + b$ $y_1 = f(s_1)$

Input-output generated by neuron model: $\{u(1), u(2), \cdots, u(K)\}$ $\{y_1(1), y_1(2), \cdots, y_1(K)\}$

Cost function to be minimized:

$$J = \frac{1}{2}\left[\left(\mathbf{y}(1) - \mathbf{y}_1(1)\right)^2 + \left(\mathbf{y}(2) - \mathbf{y}_1(2)\right)^2 + \cdots + \left(\mathbf{y}(K) - \mathbf{y}_1(K)\right)^2\right]$$

Back-Propagation update is given by:

$$w = w + \Delta w \qquad b = b + \Delta b$$

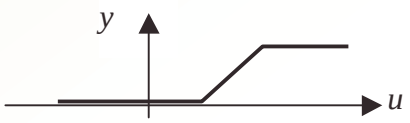
$$\Delta w = -\gamma_w \frac{\partial J}{\partial w} \qquad \Delta b = -\gamma_b \frac{\partial J}{\partial b}$$

$$\frac{\partial J}{\partial w} = \left(\frac{\partial J}{\partial s_1}\right)\left(\frac{\partial s_1}{\partial w}\right) = \left(\frac{\partial J}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial s_1}\right)\left(\frac{\partial s_1}{\partial w}\right) = \left((y - y_1)(-1)\right)\left(g(s_1)\right)(\mathrm{u}) \qquad g(s_1) = \frac{\partial y_1}{\partial s_1} = \frac{\partial f(s_1)}{\partial s_1}$$

$$\frac{\partial J}{\partial b} = \left(\frac{\partial J}{\partial s_1}\right)\left(\frac{\partial s_1}{\partial b}\right) = \left(\frac{\partial J}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial s_1}\right)\left(\frac{\partial s_1}{\partial b}\right) = \left((y - y_1)(-1)\right)\left(g(s_1)\right)(1)$$

Example: Back-Propagation Training for a 1-1 Neuron

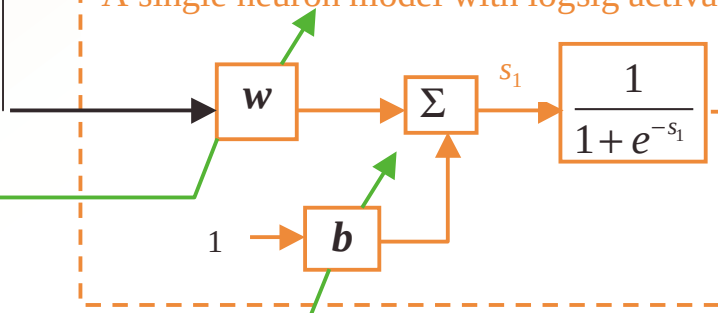
A pattern or phenomenon that we can observe. In this example, it so happens that the pattern behaves like this. (y is normalized)



u

y

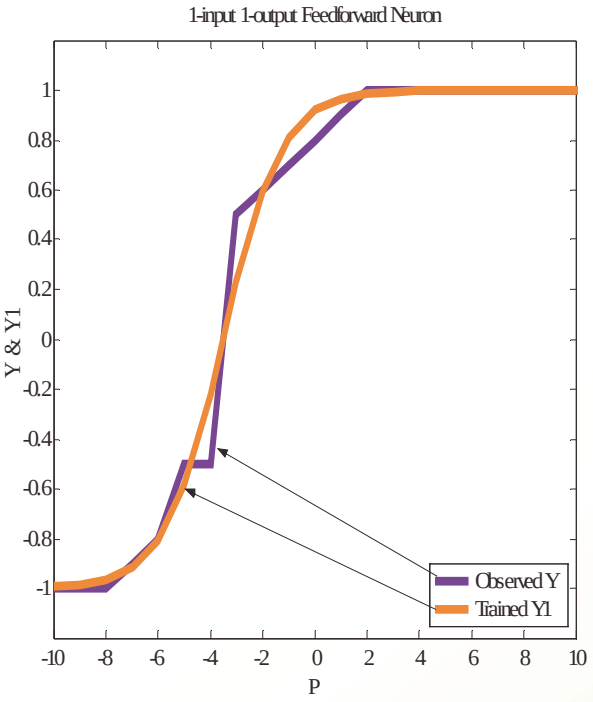
A single neuron model with logsig activation

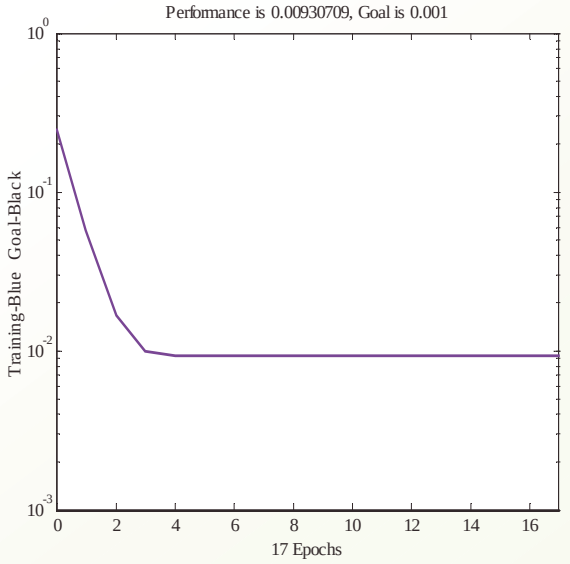


Back-propagation algorithm

$$w_{new} = w_{old} + \Delta w \quad \Leftarrow \quad \Delta w = -\gamma_1 \frac{\partial J}{\partial w} \quad \Leftarrow \quad \frac{\partial J}{\partial w} = (e)(-1)(1-y_1)\,y_1 u$$

$$b_{new} = b_{old} + \Delta b \quad \Leftarrow \quad \Delta b = -\gamma_2 \frac{\partial J}{\partial b} \quad \Leftarrow \quad \frac{\partial J}{\partial b} = (e)(-1)(1-y_1)\,y_1$$





June 2002

Back-Propagation Training for a 2-1 Neuron

A pattern or phenomenon that we can observe. In this example, it so happens that the pattern behaves like this. (y is normalized)

$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$

y

y_1

$\mathbf{W}$

Σ

f

1

b

$\mathbf{W} = [w_1 \quad w_2]$

$s = \mathbf{W}\mathbf{u} + b$

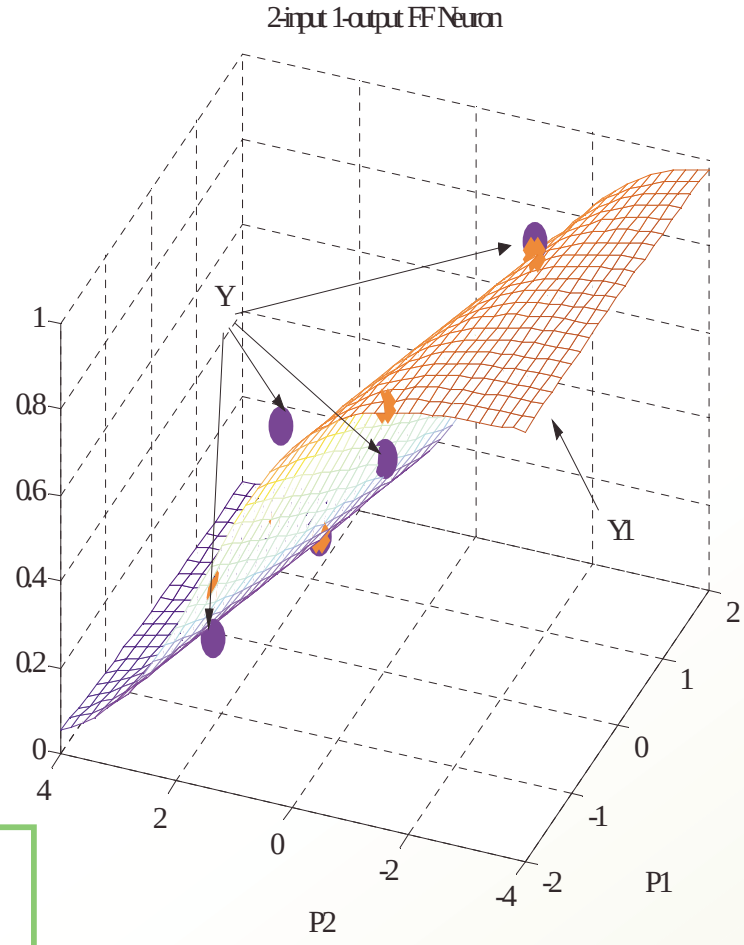
$y = f(s)$

$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$

$$w_{1new} = w_{1old} + \Delta w_1 \Leftarrow \Delta w_1 = -\gamma_1 \frac{\partial J}{\partial w_1} \Leftarrow \frac{\partial J}{\partial w_1} = (e)(-1)(1-y_1)y_1 u_1$$

$$w_{2new} = w_{2old} + \Delta w_2 \Leftarrow \Delta w_2 = -\gamma_1 \frac{\partial J}{\partial w_2} \Leftarrow \frac{\partial J}{\partial w_2} = (e)(-1)(1-y_1)y_1 u_2$$

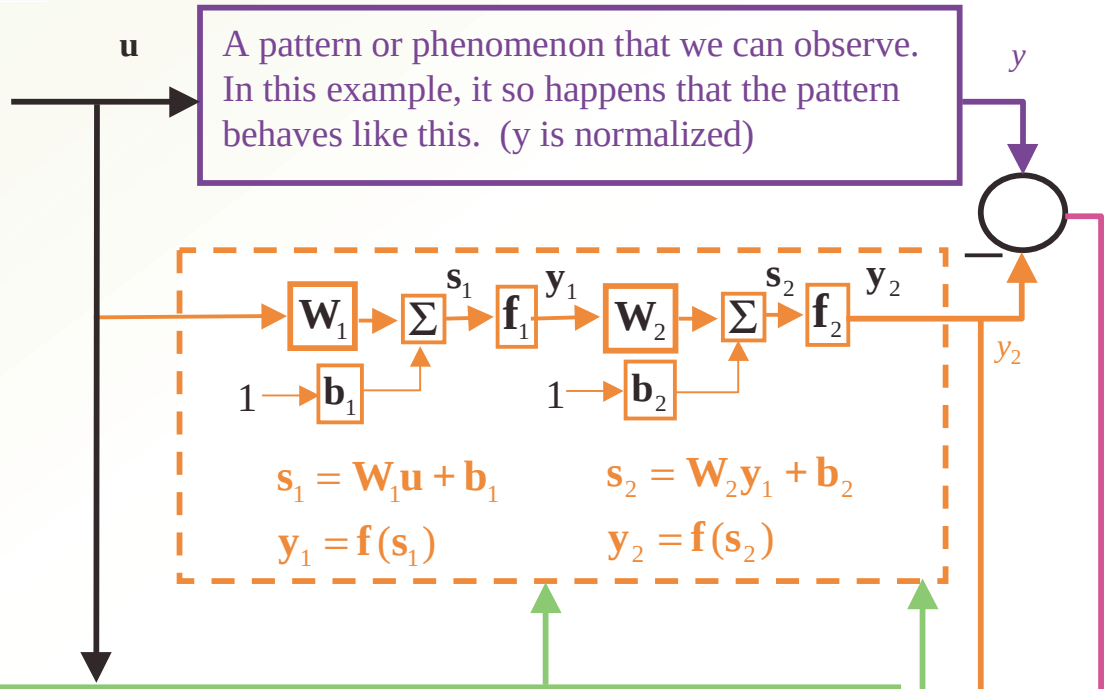
$$b_{new} = b_{old} + \Delta b \Leftarrow \Delta b = -\gamma_2 \frac{\partial J}{\partial b} \Leftarrow \frac{\partial J}{\partial b} = (e)(-1)(1-y_1)y_1$$

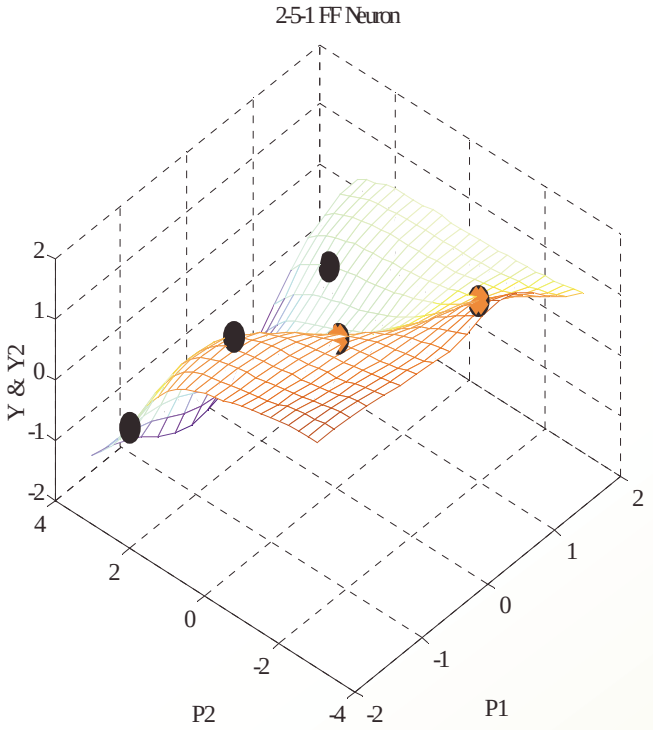


1 layer NN may not fit all data points

Back-Propagation Training for a Two-Layer r-n-m Neural Network

A pattern or phenomenon that we can observe. In this example, it so happens that the pattern behaves like this. (y is normalized)





2 layer NN fits data points better

$$\left[\frac{\partial J}{\partial \mathbf{s}_1}\right] = \left[\frac{\partial J}{\partial s_{11}} \quad \frac{\partial J}{\partial s_{12}}\right] = \left[(g_{11}(s_{11}))(w_{211}) \quad (g_{12}(s_{12}))(w_{212})\right]\left[\frac{\partial J}{\partial s_2}\right]$$

$$\mathbf{W}_1 = \mathbf{W}_1 + \Delta\mathbf{W}_1, \quad \Delta\mathbf{W}_1 = -\gamma_{\mathbf{W}_1}\left[\frac{\partial J}{\partial \mathbf{W}_1}\right]', \quad \left[\frac{\partial J}{\partial \mathbf{W}_1}\right] = [u \quad u]\begin{bmatrix}\frac{\partial J}{\partial s_{11}} & 0\\ 0 & \frac{\partial J}{\partial s_{12}}\end{bmatrix}$$

$$\mathbf{b}_1 = \mathbf{b}_1 + \Delta\mathbf{b}_1, \quad \Delta\mathbf{b}_1 = -\gamma_{\mathbf{b}_1}\left[\frac{\partial J}{\partial \mathbf{b}_1}\right]', \quad \left[\frac{\partial J}{\partial \mathbf{b}_1}\right] = [1 \quad 1]\begin{bmatrix}\frac{\partial J}{\partial s_{11}} & 0\\ 0 & \frac{\partial J}{\partial s_{12}}\end{bmatrix}$$

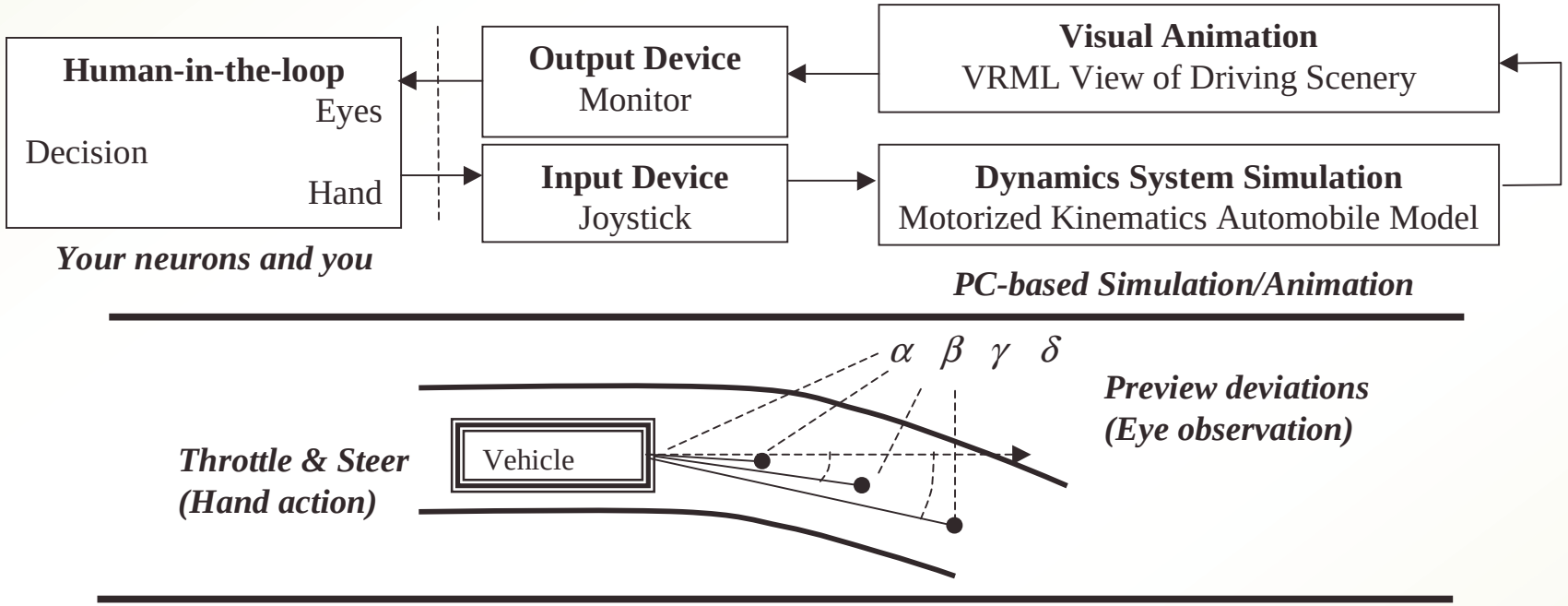
$$\left[\frac{\partial J}{\partial s_2}\right] = \left((y - y_2)(-1)\right)\left(g_2(s_2)\right)$$

$$\mathbf{W}_2 = \mathbf{W}_2 + \Delta\mathbf{W}_2, \quad \Delta\mathbf{W}_2 = -\gamma_{\mathbf{W}_2}\left[\frac{\partial J}{\partial \mathbf{W}_2}\right]', \quad \left[\frac{\partial J}{\partial \mathbf{W}_2}\right] = (\mathbf{y}_1)\left[\frac{\partial J}{\partial s_2}\right]$$

$$b_2 = b_2 + \Delta b_2, \quad \Delta b_2 = -\gamma_{b_2}\left[\frac{\partial J}{\partial b_2}\right]', \quad \left[\frac{\partial J}{\partial b_2}\right] = (1)\left[\frac{\partial J}{\partial s_2}\right]$$

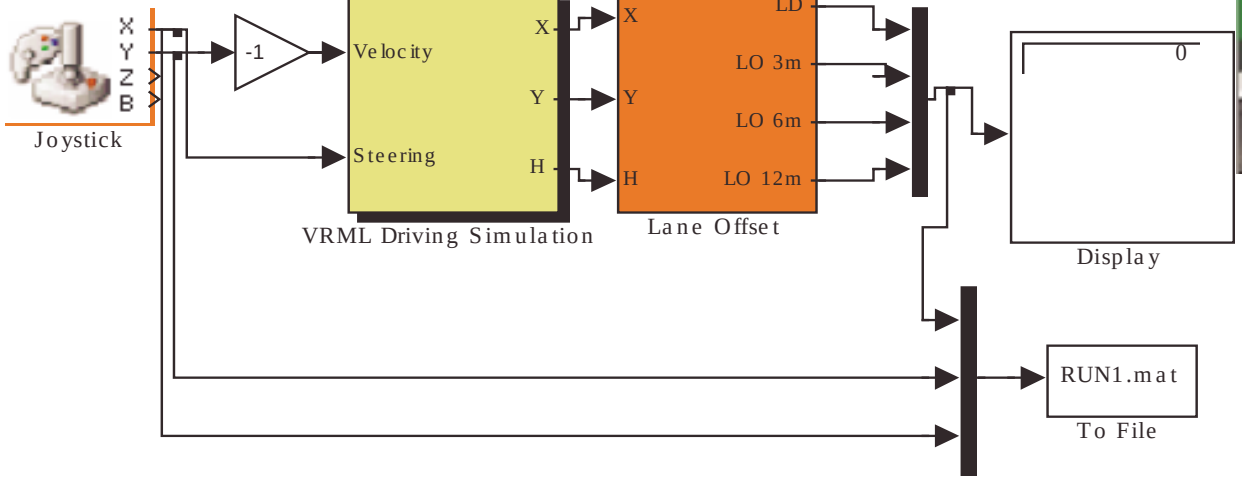
Example Application of FNN

Use an FNN to mimic an operator eye-hand coordination

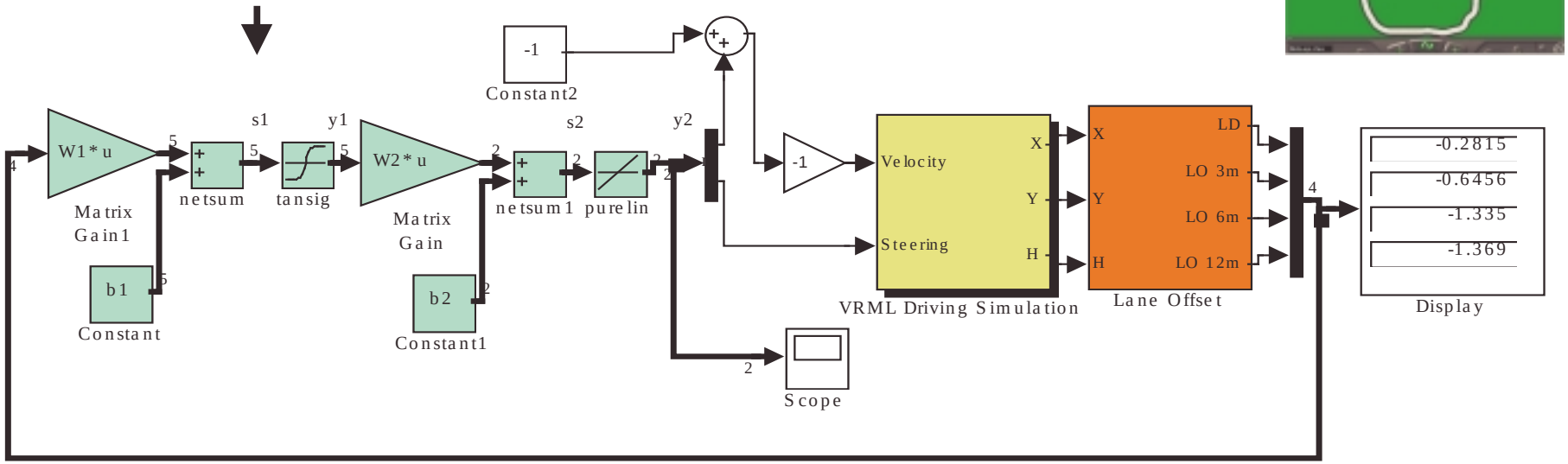


Virtual Simulation of Lane Keeping

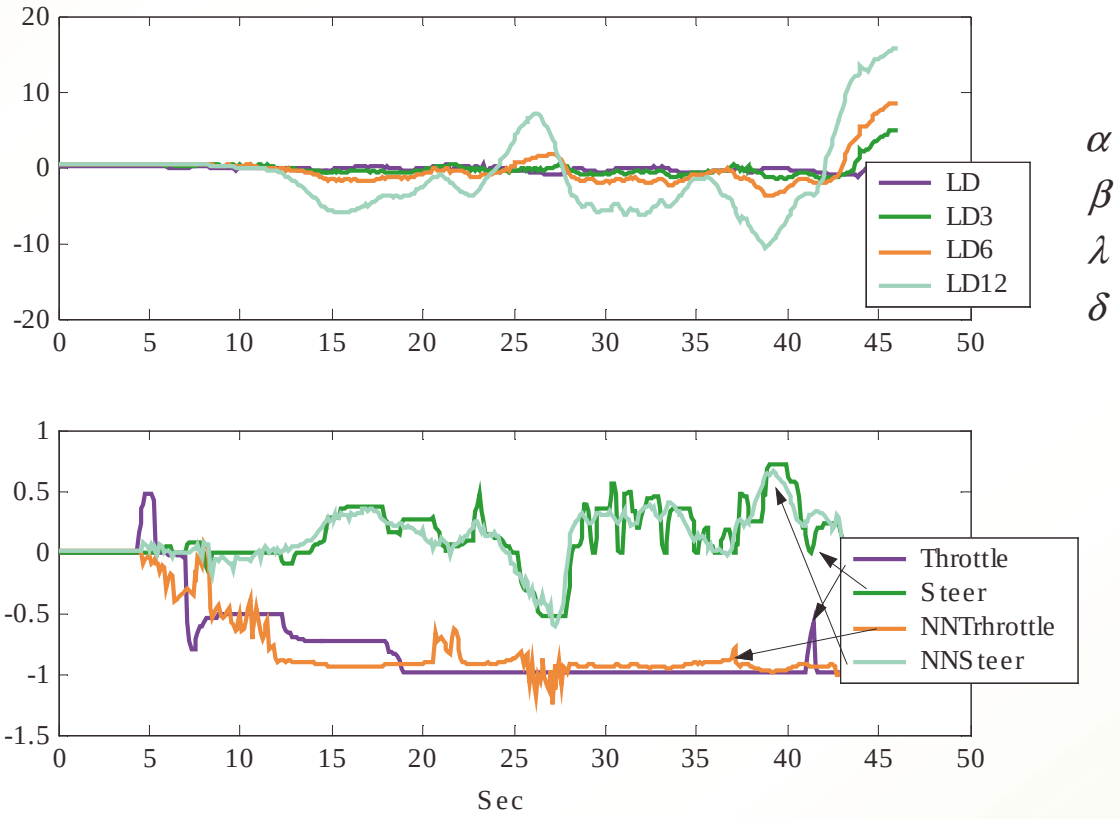
Human outputs (Throttle, Steer)



Trained FNN outputs (NNThrottle, NNSteer) replaces human operator



Comparison of Trained FNN outputs (NNThrottle, NNSteer) to Human outputs (Throttle, Steer)



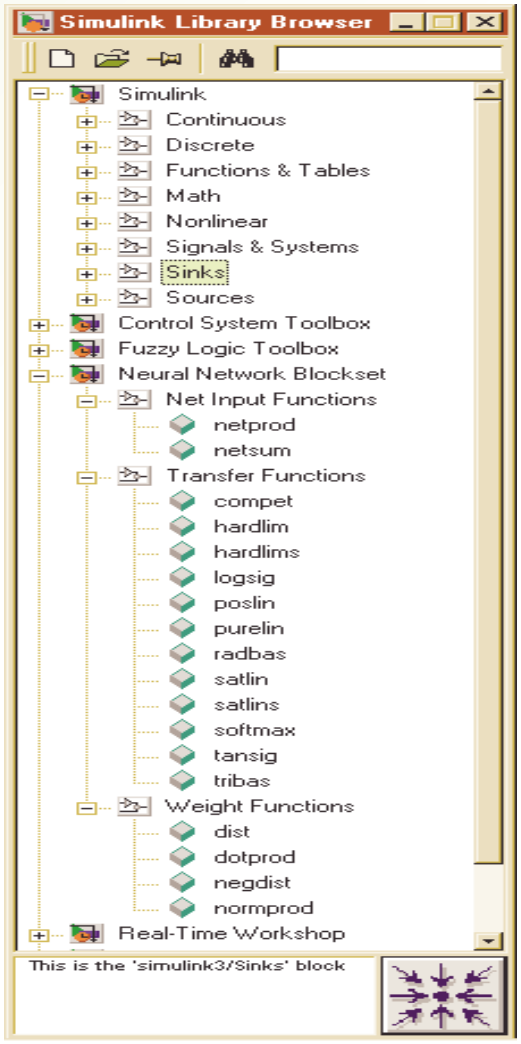
Neural networks learn and mimic I/O behavior of a pattern.

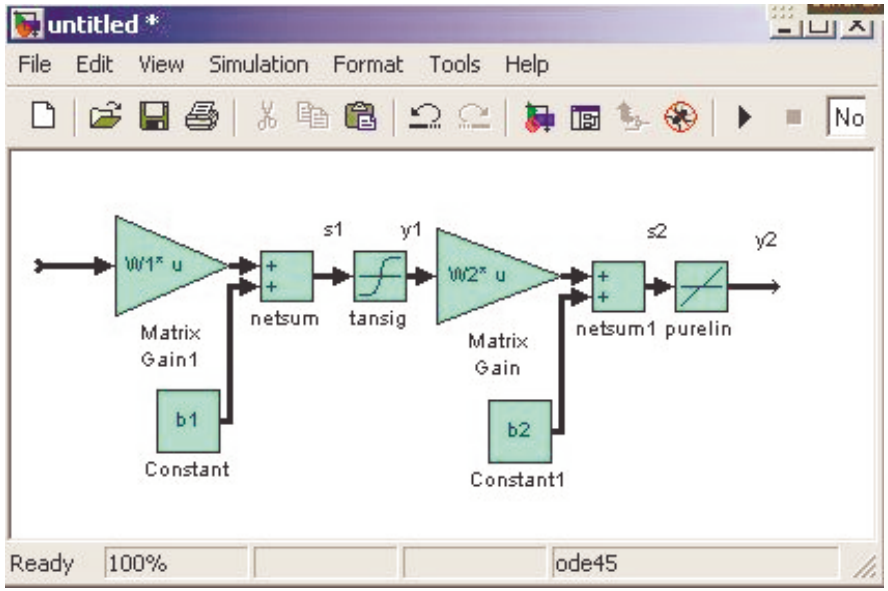
Matlab Neural Networks Toolbox

This example shows how Matlab NN Toolbox was used to train the Lane Keeping FNN

```
load Run1
figure(1);
subplot(2,1,1); plot(LD4XY(1,:),LD4XY(2:5,:),'linewidth',2);   legend('LD','LD3','LD6','LD12')
subplot(2,1,2); plot(LD4XY(1,:),LD4XY(6:7,:),'linewidth',2);   legend('Throttle','Steer');
P = LD4XY(2:5,:);        T = LD4XY(6:7,:);

% Training and Simulation of a 4-n1-2 FFNN
% We'd like to train the following FFNN
%   s11 = [w111 w112 w113 w114]*[p1 p2 p3 p4]' + [b11]
%   s12 = [w121 w122 w123 w124]*[p1 p2 p3 p4]' + [b12]
%   s13 = [w131 w132 w133 w134]*[p1 p2 p3 p4]' + [b13]
%   s14 = [w141 w142 w143 w144]*[p1 p2 p3 p4]' + [b14]
%   s15 = [w151 w152 w153 w154]*[p1 p2 p3 p4]' + [b15]
%   y11 = tansig(s11)
%   y12 = tansig(s12)
%   y13 = tansig(s13)
%   y14 = tansig(s14)
%   y15 = tansig(s15)
%   y21 = [w211 w212 w213 w214 w215]*[y11 y12 y13 y14 y15]' + [b21]
%   y22 = [w221 w222 w223 w224 w225]*[y11 y12 y13 y14 y15]' + [b22]
%  so that y2 duplicates T.

net = newff([-5 5; -5 5; -5 5; -5 5],[5 2],{'tansig' 'purelin'});   % 4-5-2 FNN with input ranges

net.trainParam.epochs = 100;          %  Train for this no. of epochs
net = train(net,P,T);
YTrained = sim(net,P);

figure(1); subplot(2,1,2); plot(LD4XY(1,:),[LD4XY(6:7,:); YTrained],'linewidth',2);
legend('Throttle','Steer','NNTrhrottle','NNSteer');

W1 = net.IW{1,1},   b1 = net.b{1,1},  W2 = net.LW{2,1},  b2 = net.b{2,1}
```

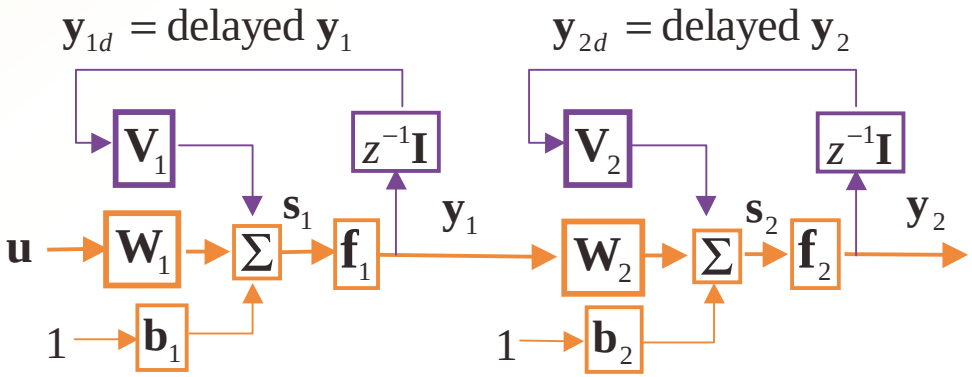
Simulink Neural Networks Blockset





Matlab Simulink provides NN Blocksets for simulating NN using block diagrams

Recursive Neural Network (RNN)



$$s_1 = W_1 u + b_1 + V_1 y_{1d} \qquad s_2 = W_2 y_1 + b_2 + V_2 y_{2d}$$

$$y_1 = f(s_1) \qquad y_2 = f(s_2)$$

Neural networks learn and mimic I/O behavior of a pattern which depends on output past values

Intelligent Control Systems Methods – Fuzzy Logic

Prof Ka C Cheok

Dept of Electrical and Systems Engineering

Oakland University

Rochester MI 48309

Summer Technical Workshop Series
NDIA 2nd Annual Intelligent Vehicle Systems Symposium
Grand Traverse Resort & Spa
Travers City, MI
June 3-5, 2002

Historical notes:

1920-30. Heisenberg & Max Black (mathematician) introduced principle of uncertainty

1930's Probability theory

- Lofti Zadeh introduced fuzzy set theory & fuzzy logic.

Many claimed that fuzzy logic theory which represents "possibility theory" resembled "probability theory" even though they are mathematically and conceptually different.

1970's. Several successful commercial application of FL in Japan made the world aware of its potential. Accelerating & decelerating a subway train; camera adjustments, consumer appliances, etc.

Precision & Non-Precision

Precise Statement

Mathematics is a precise language for describing scientific, engineering and business principles.

E.g. You will earn 3.4% A.P.R as interest for your saving account. The interest dividend will be computed and distributed at the end of each month. Bank statements?

E.g. Force-Acceleration-Speed-Displacement calculus

$$ma = -bv - kd + f$$

$$v(t) = \int_0^t a(\tau)\,d\tau + v(0)$$

$$d(t) = \int_0^t v(\tau)\,d\tau + d(0)$$

Non-Precise Statement

Adjectives and adverbs are non-precise words for describing fuzzy meaning ideas.

"You look nice."
Better still, "You look like shit. What's your secret?"
"Did you like it?" "Er, interesting."
Few, several, many, plentiful, lots

Fuzzy Variables, Values and Membership

Rating a movie from 0 to 10… *Spiderman* is a *nine* out of 10. Or simply *marvelous.*

"He's *twenty five* years old," means he could be between 25.000 and 25.999 years of age.

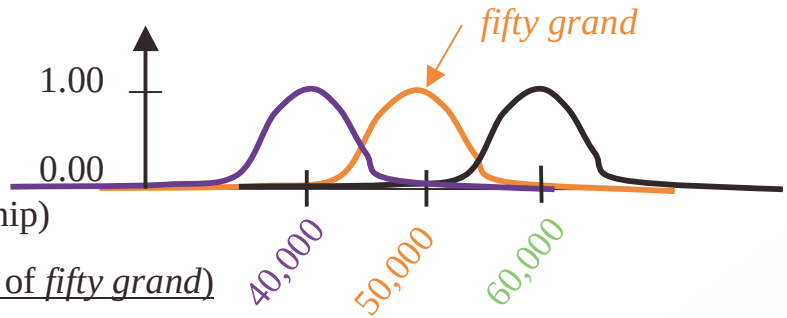
Human are non-precise in thinking and speaking:

E.g., I would like to spend *fifty grand* dollars on a car.

$50,000 is really close to the verbal *fifty grand* . (100% membership)

$55,000 is on the high side of *fifty grand* . (Say 50% membership of *fifty grand*)

$42,000 is on the low side of *fifty grand* . (Say 25% membership)



Precise Variables and Values

A computer, on the other hand, can operate with high precision

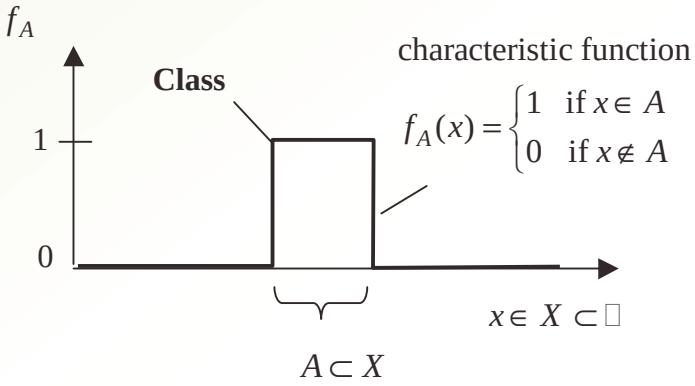
E.g. a double precision floating number can deal with any number in the range

Variable = +/-y.yyyyyyyyyyyyyyy x $10^{+/-308}$

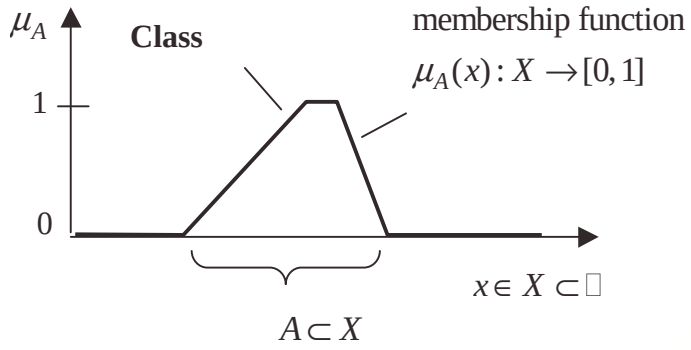
E.g., ◘ = 3.141592653589793… is a magical number

Numbering system would be simpler and more natural if we have only 2 or 4 or 8 or 2^n fingers. Octopi got the number systems right.

Boolean Sets versus Fuzzy Sets



Classical Boolean Set
(crisp membership value, either 0 or 1)

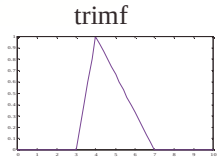


Fuzzy Set
(soft membership values varying between 0 or 1)

Shapes of Membership Functions

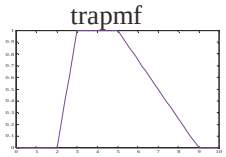
Triangular

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



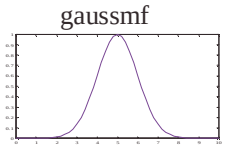
Trapezoidal

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

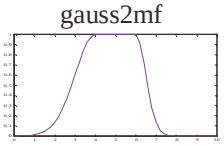


Gaussian standard bell shape

$$\mu = e^{\frac{-(x-c)^2}{2\sigma^2}}$$

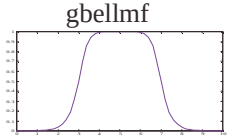


Gaussian with two different sides



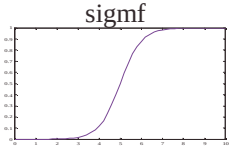
Generalized Gaussian bell shape

$$\mu = \frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}}$$



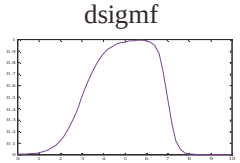
Sigmoid

$$\mu = \frac{1}{1+e^{-a(x-c)}}$$



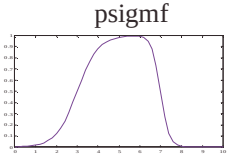
Difference of 2 sigmoids

$$\mu = \frac{1}{1+e^{-a_1(x-c_1)}} - \frac{1}{1+e^{-a_2(x-c_2)}}$$

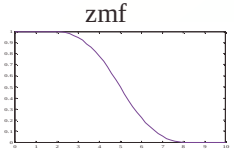


Product of 2 sigmoids

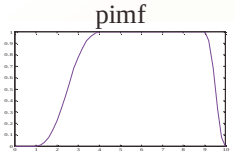
$$\mu = \frac{1}{1+e^{-a_1(x-c_1)}} \times \frac{1}{1+e^{-a_2(x-c_2)}}$$



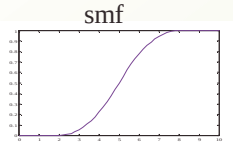
Z-shape spline



π-shape spline



S-shape spline



ANY OTHER CONVEX TYPE SHAPE WILL ALSO DO!!!

If-Then Rules

"If-Then" rules are the most commonly used fuzzy logic statements. They can be used to represent knowledge.

A single antecedent-single consequence "If-Then" rules has the form:

Rule1. **If *x* is *A1* then *y* is *B1***
Rule2. **If *x* is *A2* then *y* is *B2***
Rule3. **If *x* is *A3* then *y* is *B3***

Antecedent Consequent

If *Road* is *Left* then *Steer* to *Left*
If *Road* is *Middle* then *Steer* to *Middle*
If *Road* is *Right* then *Steer* to *Right*

Compounded-antecedent-single consequence rules take the form

Rule1. **If *x* is *A1* and/or *y* is *B1* then *z* is *C1***
Rule2. **If *x* is *A2* and/or *y* is *B2* then *z* is *C2***
Rule3. **If *x* is *A3* and/or *y* is *B3* then *z* is *C3***

Antecedent Consequent

If *Road* is *Left* or *Obstacle* is *Right* then *Steer* to *Left*
If *Road* is *Left* and *Obstacle* is *Left* then *Steer* to *Middle*
And so on…

Fuzzy Inference Systems (FIS)

Knowledge & Data Base

Rule1. **If *x* is *A1* and/or *y* is *B1* then *z* is *C1***
Rule2. **If *x* is *A2* and/or *y* is *B2* then *z* is *C2***
Rule3. **If *x* is *A3* and/or *y* is *B3* then *z* is *C3***

Fuzzification
Convert crisp value into fuzzy association

Crisp numerical input value

Sensors

Defuzzification
Convert fuzzy set into a crisp value

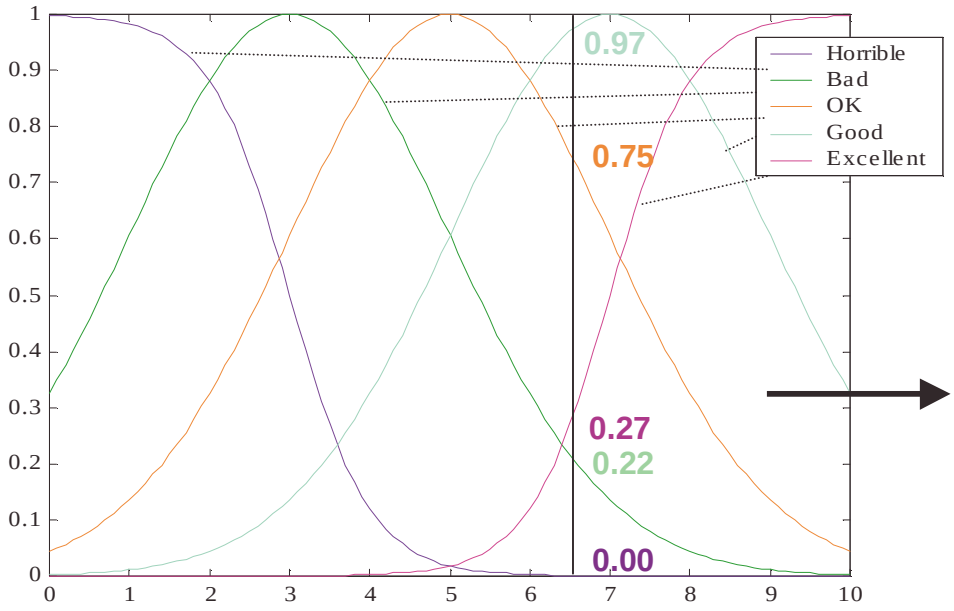
Crisp numerial output value

Actuators

FUZZIFICATION

Associate a crisp numerical input value to fuzzy values with degree of membership.

Example: Sometimes we rate a movie on the scale of 0 to 10. Let's say we just want to use a fuzzy labels like `'Horrible'`, `'Bad'`, `'OK'`, `'Good'`, `'Excellent'`, to describe the movie. Suppose that the membership function for these fuzzy labels/values are as shown below:



We associate a **crisp value 6.50** to

0% Horrible
22% Bad
75% OK
97% Good
27% Excellent

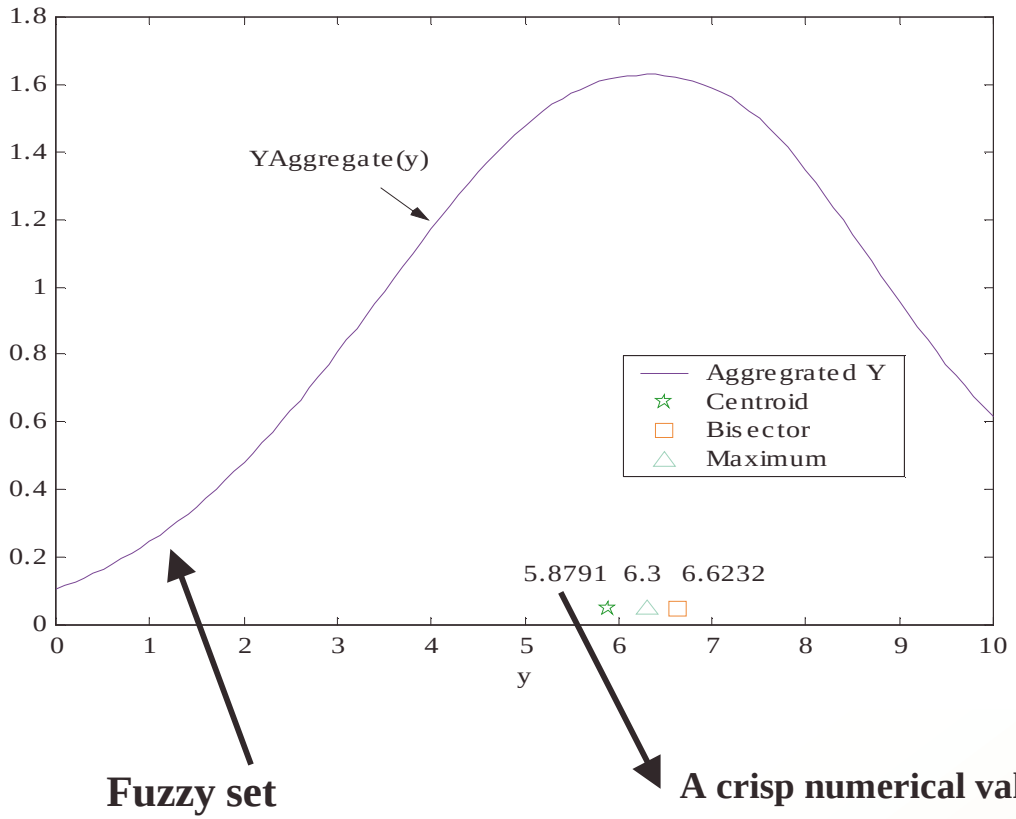
A crisp numerical value

Fuzzy values with degree of membership

DEFUZZIFICATION

Convert fuzzy sets into a crisp numerical output value.

Example: Suppose we have the fuzzy set shown below:



There are many ways to do this:

Centroid.

$$y_{Centroid} = \frac{\int_0^{10} y \times y_{Aggregrate}(y)dy}{\int_0^{10} y_{Aggregrate}(y)dy} = \frac{\left\| y \square \ y_{Aggregrate} \right\|_1}{\left\| y_{Aggregrate} \right\|_1}$$

Bisector/Median.

$$y_{Bisector} = median\left(y \times y_{Aggregrate}(y)\right)$$

Maximum

$$\left[y_{AggMax}, i_{Max} \right] = max\left(y_{Aggregrate}(y) \right)$$

$$y_{Max} = y(i_{Max})$$

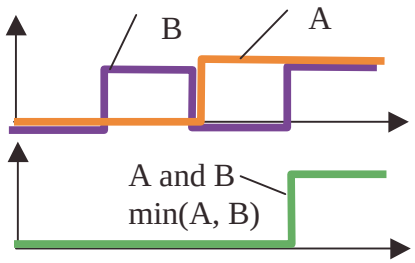
LOGICAL OPERATIONS

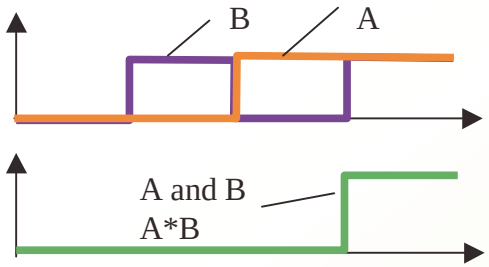
An *AND* operation can be treated as equivalent to a *MIN or PRODUCT* operation

A	B	A and B	min(A, B)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

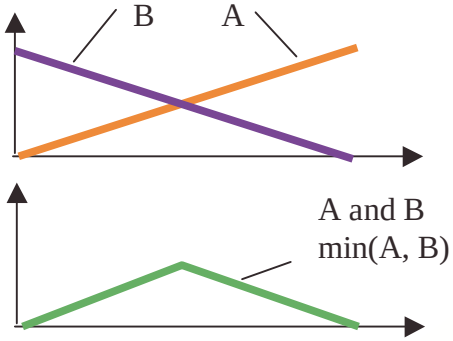
A	B	A and B	A*B
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

***Boolean AND* operation**

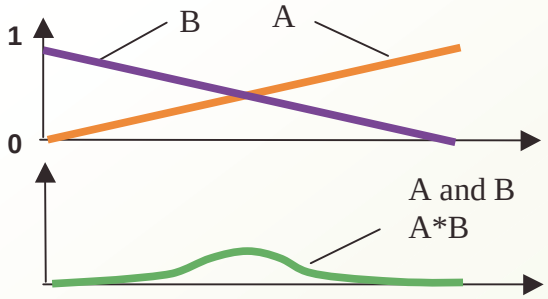




***Fuzzy AND* operation**



Note: 0.5 is the largest value



Note: 0.25 is the largest value

An *OR* operation can be treated as equivalent to a *MAX or PROBOR* operation

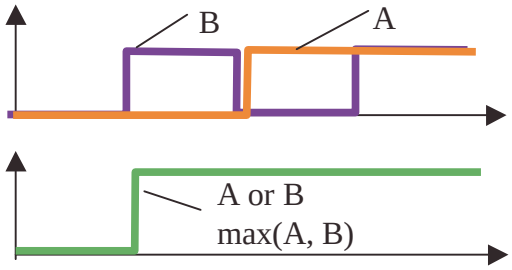
Max operation

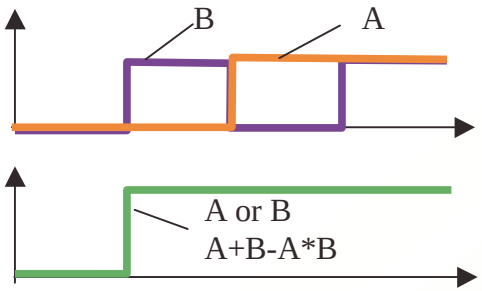
A	B	A or B	max(A, B)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Probabilistic OR
Probor operation

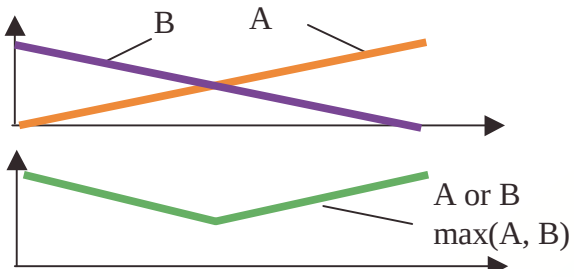
A	B	A or B	A+B-A*B
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Boolean OR **operation**

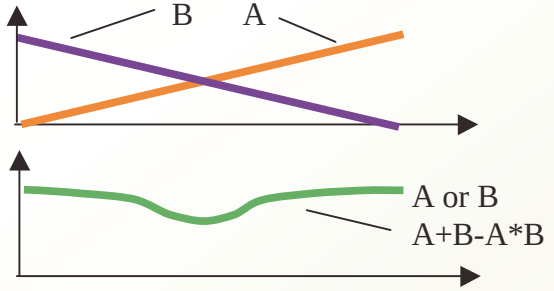




Fuzzy OR **operation**



Note: 0.5 is the smallest value



Note: 0.75 is the smallest value

Mamdani Style FUZZY LOGIC
Example

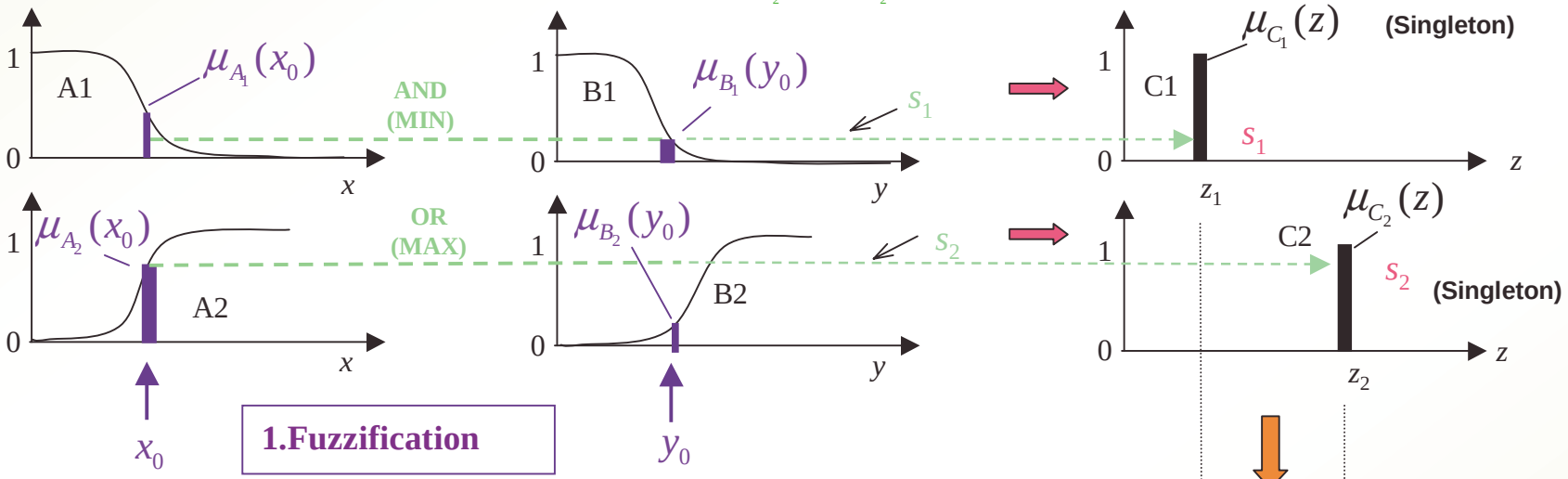
Rule1. **If x is A1 and y is B1 then z is C1.**
Rule2. **If x is A2 or y is B2 then z is C2.**

2. Fuzzy operation

$s_1 = \min(\mu_{A_1}(x_0), \mu_{B_1}(y_0))$

$s_2 = \max(\mu_{A_2}(x_0), \mu_{B_2}(y_0))$

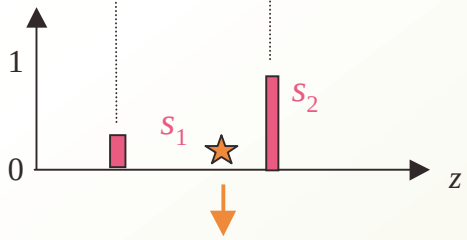
3. Implication



1.Fuzzification

4. Aggregation

$\mu_{C_{Aggregrate}}(z) = s_1 @ z_1 + s_2 @ z_2$



5. Defuzzification

$z_0 = \dfrac{s_1 z_1 + s_2 z_2}{s_1 + s_2}$

The Math for the Fuzzy Logic

$$s_1 = \min(\mu_{A_1}(x_0), \mu_{B_1}(y_0))$$

$$s_2 = \max(\mu_{A_2}(x_0), \mu_{B_2}(y_0))$$

$$z_0 = \frac{s_1 z_1 + s_2 z_2}{s_1 + s_2}$$

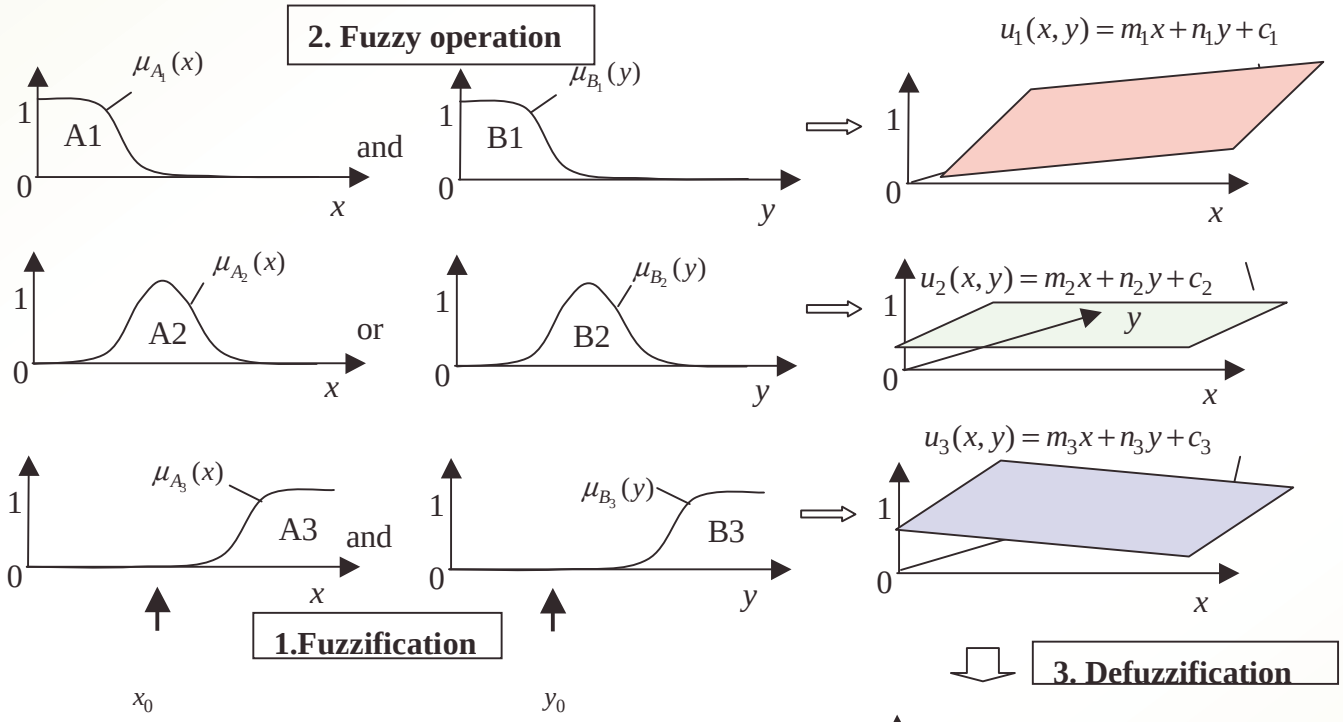
Surprisingly simple!!!

Sugeno Style FUZZY LOGIC
Example

Rule1. **If x is A1 and y is B1 then u is** $u_1 = m_1 x + n_1 y + c_1$

Rule2. **If x is A2 or y is B2 then u is** $u_2 = m_2 x + n_2 y + c_2$

Rule3. **If x is A3 and y is B3 then u is** $u_3 = m_3 x + n_3 y + c_3$



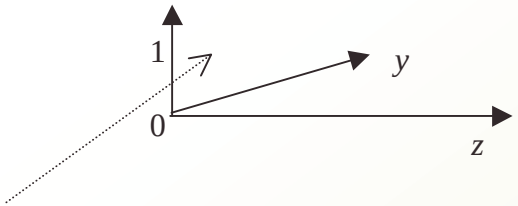
The Math for the Fuzzy Logic

$$s_1 = \mu_{A_1}(x)\mu_{B_1}(y)$$

$$s_2 = \mu_{A_2}(x) + \mu_{B_2}(y) - \mu_{A_2}(x)\mu_{B_2}(y)$$

$$s_3 = \mu_{A_3}(x)\mu_{B_3}(y)$$

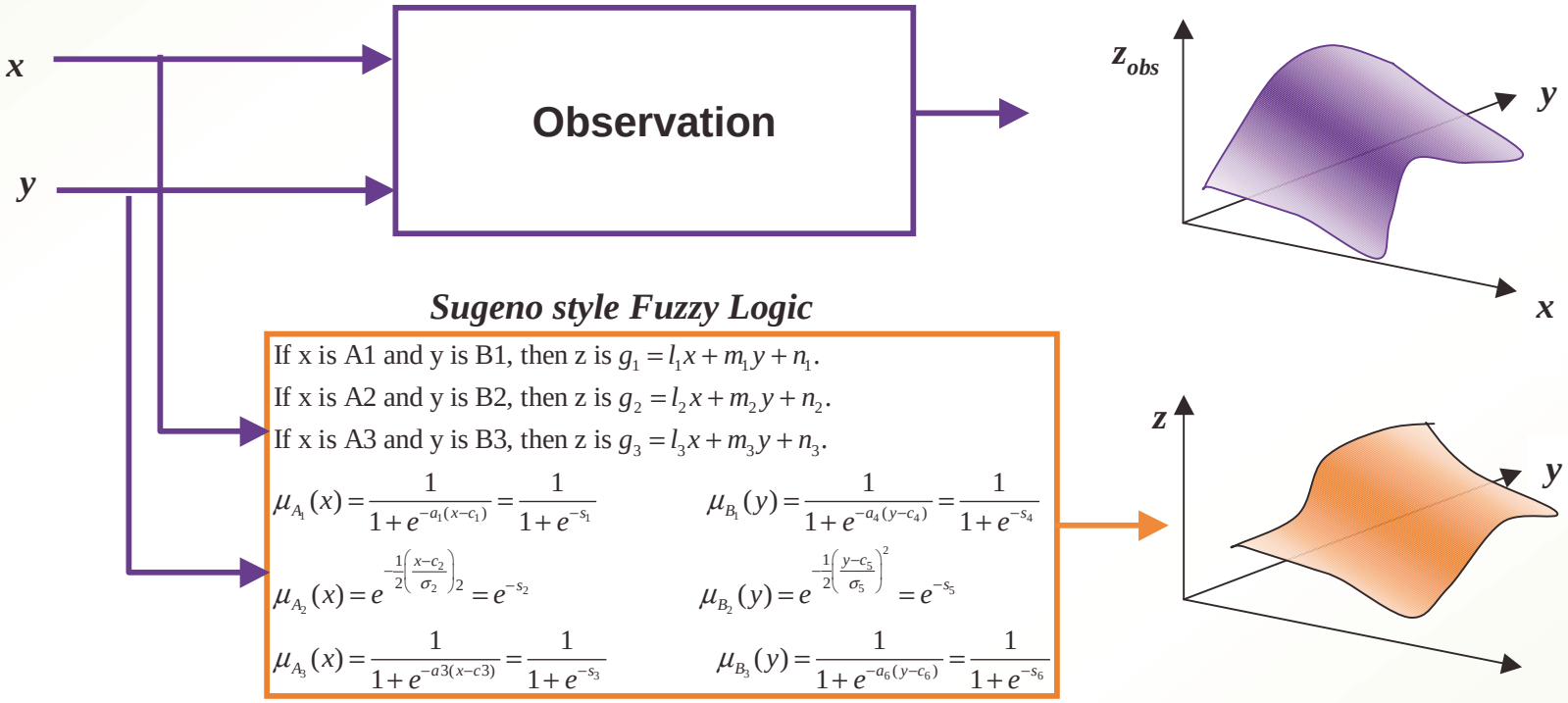
$$u(x,y) = \frac{s_1 u_1 + s_2 u_2 + s_3 u_3}{s_1 + s_2 + s_3}$$



Surprisingly simple!!!

ANFIS (Adaptive Network Fuzzy Inference System)

ANFIS is a Sugeno-style Fuzzy Logic that learns to emulate an I/O pattern (similar to a neural network).



ANFIS *will automatically tune the parameters*

a_1, σ_2, a_3, a_4, σ_5, a_6, c_1, c_2, c_3, c_4, c_5 *and* c_6,, *and* l_1, m_1, n_1, l_2, m_2, n_2, l_3, m_3, n_3 *so that the fuzzy logic output* **z** *matches up with the observed* $\mathbf{z}_{obs.}$

Sugeno fuzzy logic

$$s_1 = a_1(x - c_1) \qquad s_4 = a_4(y - c_4) \qquad w_1 = \mu_{A_1}(x)\mu_{B_1}(y) = \frac{1}{1+e^{-s_1}} \cdot \frac{1}{1+e^{-s_4}}$$

$$s_2 = \frac{1}{2}\left(\frac{(x - c_2)}{\sigma_2}\right)^2 \qquad s_5 = \frac{1}{2}\left(\frac{(y - c_5)}{\sigma_5}\right)^2 \qquad w_2 = \mu_{A_2}(x)\mu_{B_2}(y) = e^{-s_2} \cdot e^{-s_5}$$

$$s_3 = a_3(x - c_3) \qquad s_6 = a_6(y - c_6) \qquad w_3 = \mu_{A_3}(x)\mu_{B_3}(y) = \frac{1}{1+e^{-s_3}} \cdot \frac{1}{1+e^{-s_6}}$$

$$g_1 = g_1(x, y) = l_1 x + m_1 y + n_1$$
$$g_2 = g_2(x, y) = l_2 x + m_2 y + n_2$$
$$g_3 = g_3(x, y) = l_3 x + m_3 y + n_3$$
$$z = \frac{w_1 g_1 + w_2 g_2 + w_3 g_3}{w_1 + w_2 + w_3}$$

ANFIS scheme

Cost function to be minimized $\qquad J = \frac{1}{2}\sum_{k=1}^{N}(R_k - Z_k)^2$

1. Initialize the coefficients a_1, s_2, a_3, a_4, s_5, a_6, c_1, c_2, c_3, c_4, c_5 and c_6, to some estimated values.

2. Calculate w_1, w_2 & w_3 for each set of input data, Xi and Yi.

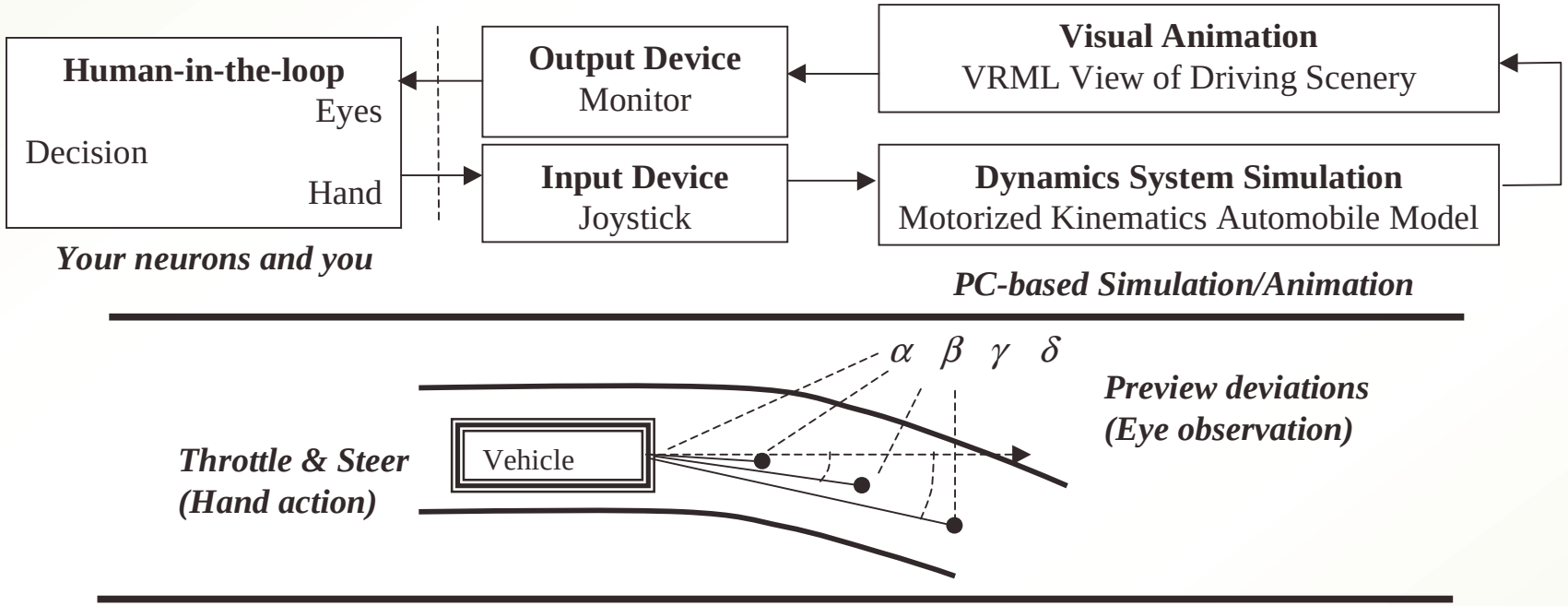
3. Apply least square estimation technique to calculate l_1, m_1, n_1, l_2, m_2, n_2, l_3, m_3, n_3

4. Apply gradient search technique to update a_1, s_2, a_3, a_4, s_5, a_6, c_1, c_2, c_3, c_4, c_5 and c_6

5. Repeat from Step 2, until satisfactory

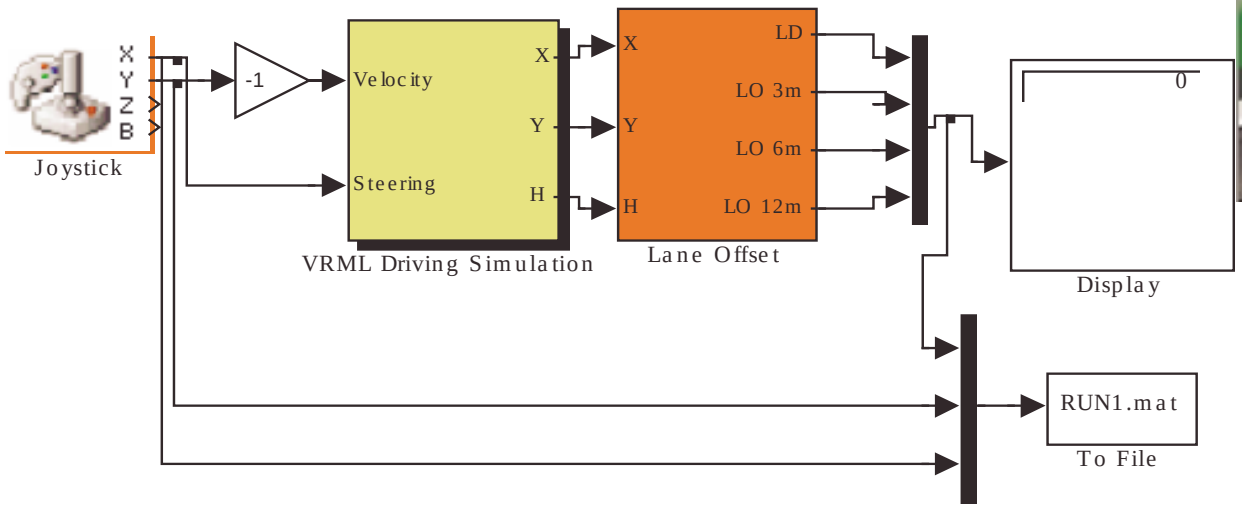
Example Application of ANFIS

Use an ANFIS to mimic an operator eye-hand coordination



Virtual Simulation of Lane Keeping

Human outputs (Throttle, Steer)



Trained ANFIS outputs (FLThrottle, FLSteer) replaces human operator