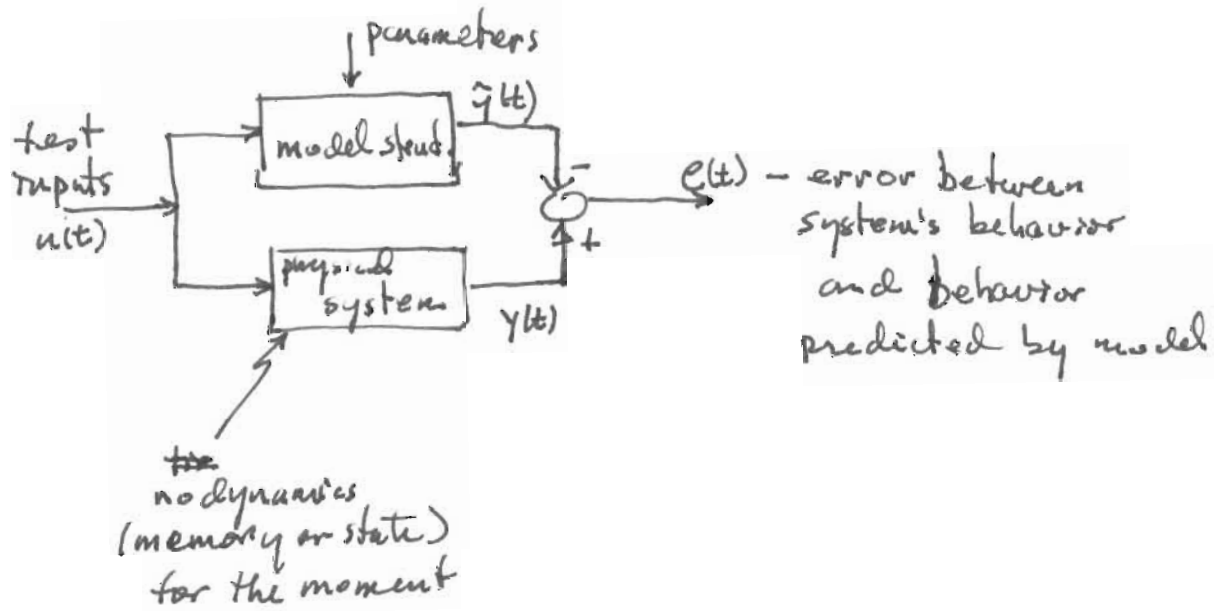


# Neural Network Models

Both NNs and fuzzy logic rule sets are fundamentally function approximation methods.

Given



We want to devise test signals, collect input/output data, and adjust the model's parameters to achieve low error.

The parameters of a NN are weights, or multipliers, applied to each input signal (and possibly internal signals).

The parameters of a fuzzy logic model are the fuzzy sets defining a signal's membership in each category (e.g., "HOT", "COLD", "WARM").

In both cases, a model's structure is first chosen (fuzzy logic rules or NN layers and codings). Parameter values are then identified (and iteration may be necessary).

We will concentrate upon NN models first. NNs are able to emulate observed process behavior through training with examples (input/output measurements).

This lecture is just an introduction. Our goal is to provide some of the information necessary to determine when NNs might be appropriate for a given application.

## Outline

- Overview
- NN Fundamentals
- The Training Problem (Identification of Parameters)
- Process Modeling
- Software

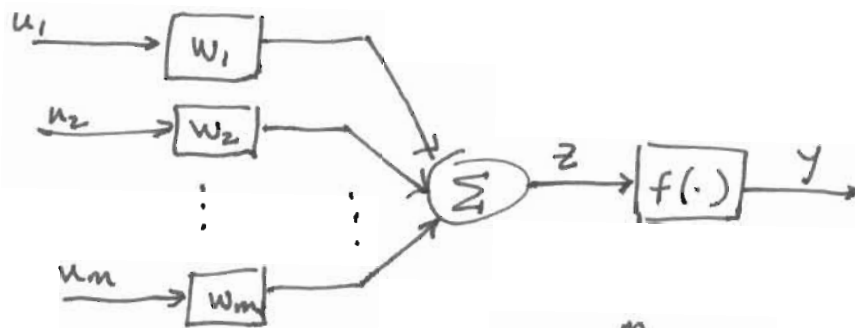
## What is a Neural Network?

- An interconnected network of small and simple nonlinear elements.
- These nonlinear elements usually form a weighted sum of their inputs and map this through a nonlinear "threshold function".
- Input signals are usually encoded in a discrete form (such as binary), with each bit becoming a separate input signal line.
- The result is an implementation of a memoryless function.
- The NN design process involves choosing a structure and determining weights.

## Why is the Technology Useful?

- Often, a function or process is understood only through observation of its behavior.
- A NN can provide a working model of the process constructed only from examples of its operation.
- The NN provides a reasonably smooth interpolation of behavior between observed examples.

## An artificial neuron



$$z = \sum_{i=1}^m w_i u_i$$

$$y = f(z)$$

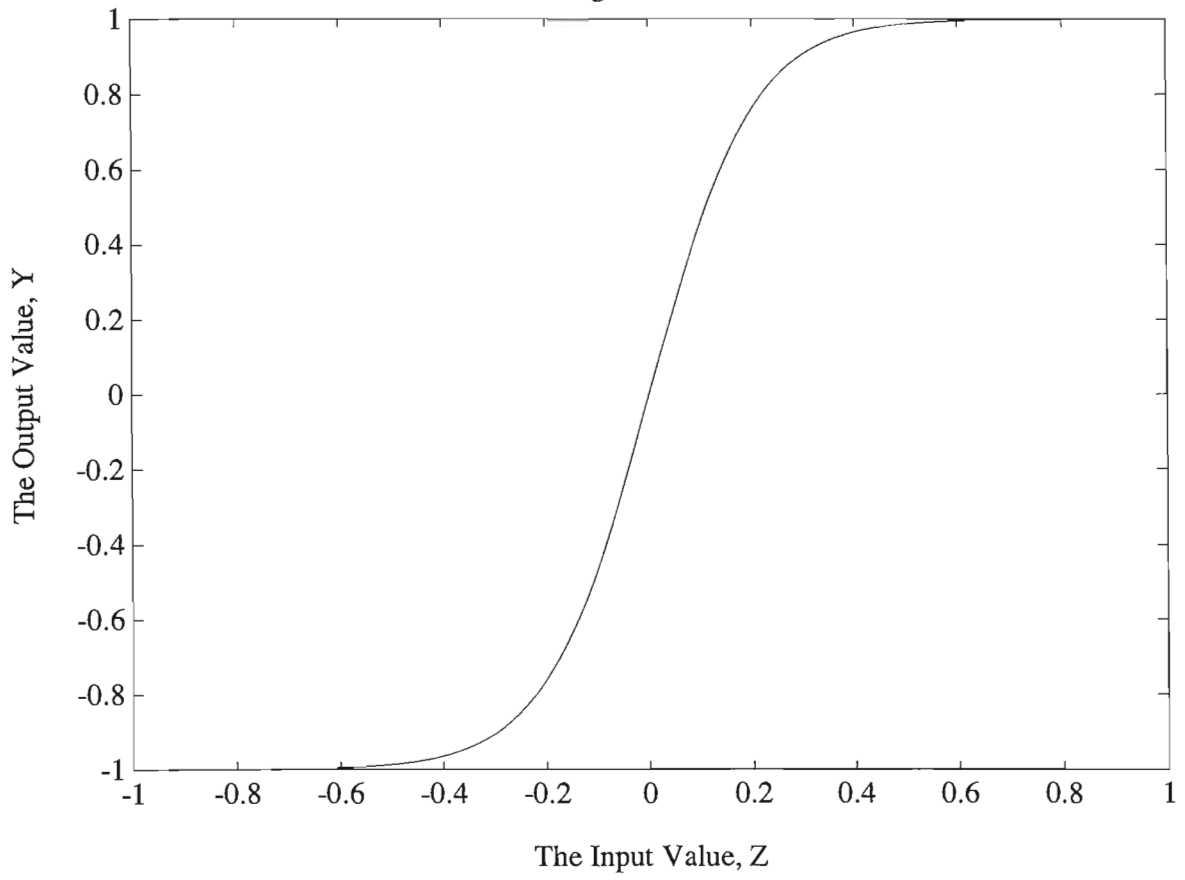
The function  $f(\cdot)$  is a soft limiter. The sigmoid function is frequently used:

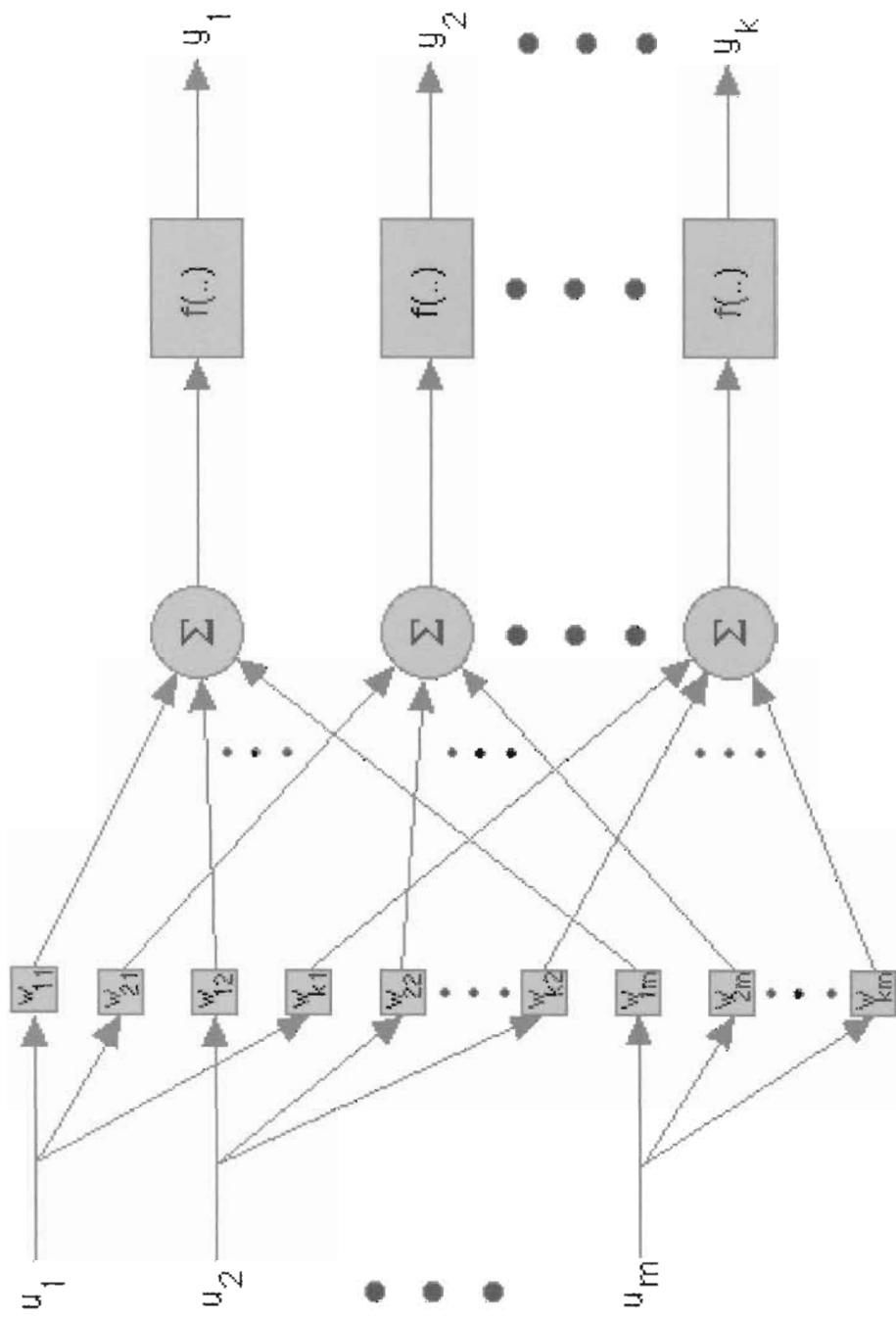
$$y = \frac{1 - e^{-\alpha z}}{1 + e^{-\alpha z}} \quad ; \quad \alpha > 0$$

A typical sigmoid is shown on the next page.

Many of these neurons are typically configured in parallel, with every input providing a signal to every neuron through a weight. An example parallel NN with  $m$  inputs and  $k$  outputs is shown, having  $mn$  parameters (weights).

The Sigmoid Function





## Vector Representation of Signals

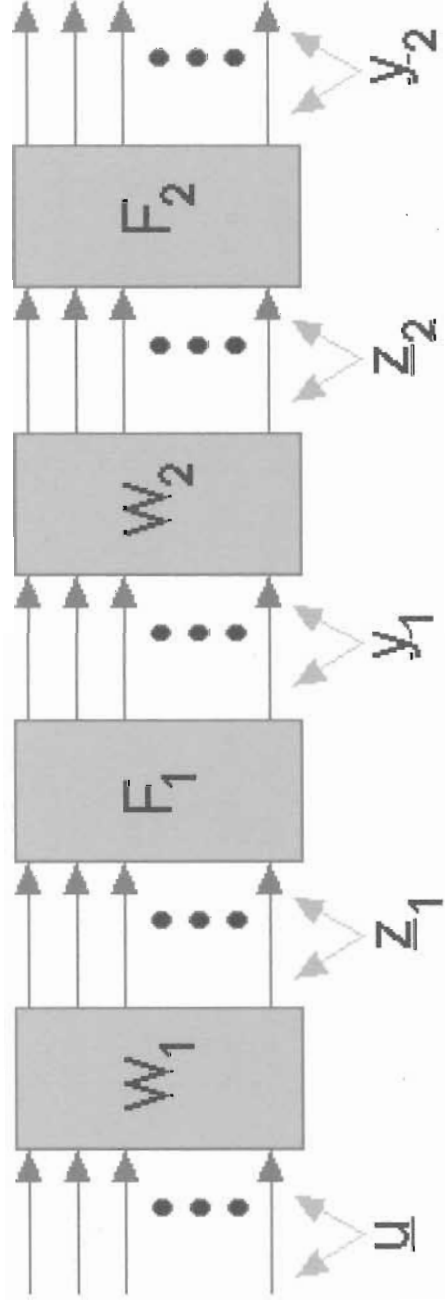
To compress the notation, signals are grouped into vectors:

$$\underline{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix} = \underline{W} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}$$

and

$$\underline{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \underline{F}(\underline{z}) \quad \text{where} \quad \underline{F}(\underline{z}) = \begin{bmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_k) \end{bmatrix}$$

Each of these parallel collections of neurons is called a layer, and layers can be connected in series, with the outputs of one layer providing the inputs to the next.



Multiple Layer Neural Network

## Input and Output Signal Coding

Input and output signal values are usually represented by a binary code. The coding scheme chosen has an impact on performance and training, but this is not well-understood.

The implementation is constructed by replacing  $f(\cdot)$  by  $h(f(\cdot))$  where

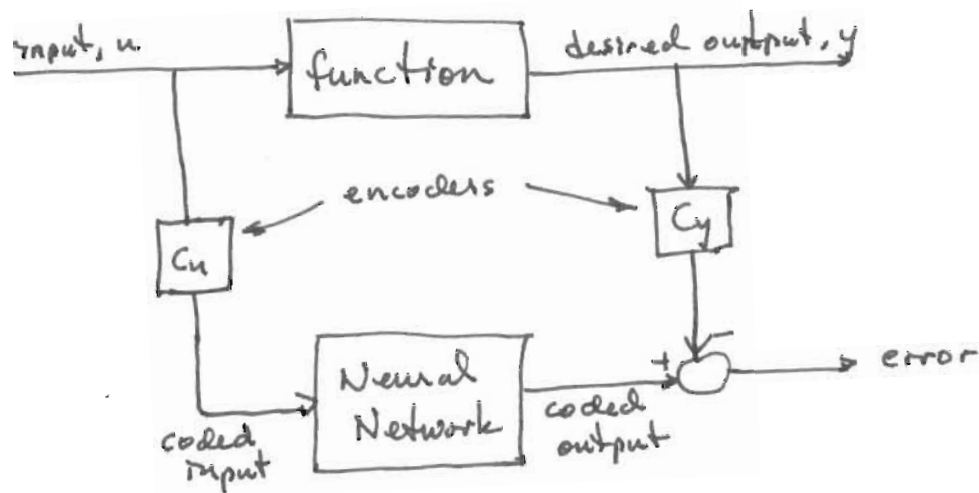
$$h(w) = \begin{cases} +1 & \text{if } w \geq 0.5 \\ 0 & \text{if } w < 0.5 \end{cases}$$

$y = +1$  is binary 1, and  
 $y = -1$  is binary zero.

Other coding schemes can be adopted, but they need to be consistent with the choice of limiter function.

## Error Model

Remember that a NN is just an approximation of a function.



Given an error model, as a function of the training sets and weights, the training problem is to select all weights so the error is small.

If the error model's value changes differentially with the weights, its value and gradient w.r.t. the weighting vector  $\underline{W}$  can be used to adjust (train) the weights.

This algorithm is called back propagation; it is essentially gradient search.

Normally, error is measured as a difference between the computed and actual function values.

The coded values are used instead.

Definition of Total Error Relative to a Training Set

Removing the discretization function,  $h$ , from the threshold functions,  $F$ , the neural network's outputs will take values in the unit interval.

Define error as the sum of the squared errors over all NN output signals and over all elements of the training set:

$$\begin{aligned} \text{total error} = J &= \sum_{(u_i, y_i) \in T} \|\hat{y}_i - \bar{y}_i\|^2 \\ &= \sum_{(u_i, y_i) \in T} \|\hat{y}_i - C_y A u_i\|^2 \end{aligned}$$

where the "hat" refers to the NN's output, the "bar" refers to the coded signal, and

$$T = \left\{ (u_i, y_i) \right\}_{i=1}^M$$

is the training set.

Training set: the error function will be used to train the NN to closely match the input/output map of the function on a discrete set of values,  $T$ .

The Training Problem can be precisely stated as:  
"Find weights  $W$  which do a good job of minimizing the error function  $J$ ."

The Training set must be representative of the function's behavior in the sense that if the network performs well on the training set, we can expect similar performance on any other set of input/output pairs of like size and characteristics.

Why use the term "good job" rather than "minimize"  $J$ ?

This is a hard optimization problem:

1. It is not convex, except for simple cases.  
This implies a local minimum is not necessarily globally minimal, and minima are not unique.
2. It is big! There are lots of weights, and comp. of the error function and its gradient is expensive.

## Training Algorithm

The most common training algorithm is back propagation, which is an implementation of gradient search, or steepest descent.

### Back propagation

- (a) finds a local minimum, which is often good enough, and
- (b) is very slow.

## Steepest Descent Algorithm

To minimize  $J(\underline{x})$ , starting at  $\underline{x}_0$ :

(1) Let  $i = 0$  and  $\underline{x}_i = \underline{x}_0$ .

(2) Compute the gradient of  $J$  at  $\underline{x}_i$ ,

$$[\nabla_{\underline{x}} J](\underline{x}_i)$$

(3) Update the estimate of the minimizing  $\underline{x}$ , to  $\underline{x}_{i+1}$ , by moving in the direction opposite to  $[\nabla_{\underline{x}} J](\underline{x}_i)$  a small distance:

$$\underline{x}_{i+1} = \underline{x}_i - \eta [\nabla_{\underline{x}} J](\underline{x}_i)$$

(4) ~~Repeat~~ Increment  $i$  by 1.

(5) Repeat (2) - (4) until either  $\|\underline{x}_i - \underline{x}_{i+1}\|$  or  $\|J(\underline{x}_i) - J(\underline{x}_{i+1})\|$  is small.

# Back Propagation

Back propagation is an algorithm for computing  $[\nabla_{\mathbf{x}} J](\mathbf{x}_i)$  and applying the update.

Model a single layer NN as

$$y = F[\underline{W}\underline{u}]$$

Then for a single member  $(\underline{u}_k, y_k)$  of the training set  $T$ ,

$$J = \langle \hat{y}_k - y_k, \hat{y}_k - y_k \rangle = \langle F[\underline{W}\underline{u}_k] - y_k, F[\underline{W}\underline{u}_k] - y_k \rangle.$$

The components of  $\underline{W}^v$ ,  $w_{ij}$ , are the parameters of the error function  $J$  we wish to select in order to minimize  $J$ .

Therefore, we need  $\frac{\partial J}{\partial w_{ij}}$ , which is

$$\frac{\partial J}{\partial w_{ij}} = z (\hat{y}_k - y_k)^T \frac{\partial y}{\partial w_{ij}}$$

and

$$\frac{\partial y}{\partial w_{ij}} = \frac{\partial F}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}} = \frac{\partial F}{\partial z_i} u_j$$

Defining

$$\underline{e}_k = \hat{y}_k - y_k, \text{ then}$$

$$\frac{\partial J}{\partial w_{ij}} = z \underline{e}_k^T \frac{\partial F}{\partial z_i} u_j \quad \text{where} \quad \frac{\partial F}{\partial z_i} = \begin{bmatrix} \frac{\partial f_1}{\partial z_1} \\ \vdots \\ \frac{\partial f_m}{\partial z_i} \end{bmatrix}$$

Writing out this inner product, the contribution of the error due to a single member of the training set toward the gradient of the error is

$$\frac{\partial J}{\partial w_{ij}} = 2 \sum_{l=1}^m e_{k,l} \frac{\partial f_l}{\partial z_i} u_{kj}$$

If  $\frac{\partial J}{\partial \underline{w}}$  is written as a matrix with  $(ij)$  component  $\frac{\partial J}{\partial w_{ij}}$ , then

$$\frac{\partial J}{\partial \underline{w}} = 2 \sum_{l=1}^m e_{k,l} \frac{\partial f_l}{\partial \underline{z}} \underline{u}_k^T = 2 \sum_{l=1}^m \delta_{k,l} \underline{u}_k^T$$

where

$$\delta_{k,l} = e_{k,l} \frac{\partial f_l}{\partial \underline{z}}$$

Summing over all members of the training set,

$$\frac{\partial J}{\partial \underline{w}} = 2 \sum_{k=1}^M \sum_{l=1}^m \delta_{k,l}^k \underline{u}_k^T$$

with

$$\delta_{k,l}^k = e_{k,l}^k \frac{\partial f_l}{\partial \underline{z}}$$

and

$$e^k = y - y_k$$

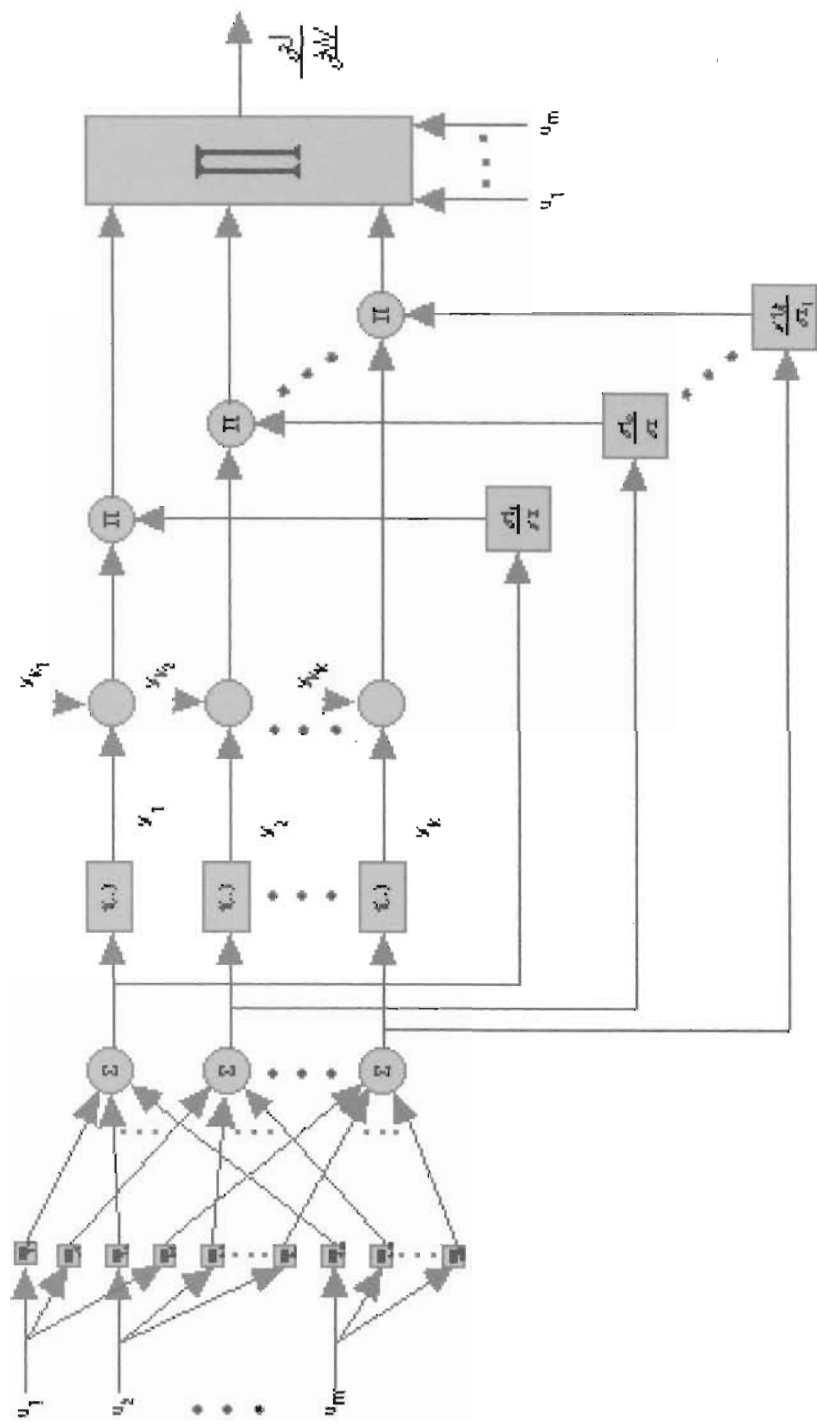
Therefore, the back propagation algorithm:

- (1) Applies, one at a time, inputs  $\underline{u}_k$  from the training set to the NN.
- (2) Subtracts the corresponding output (from the training set),  $y_{i(k)}$ , from the NN's output to obtain  $\underline{e}_k$ .
- (3) Multiplies each component of the error vector,  $e_{k,i}$ , by the corresponding component of  $\frac{\partial F}{\partial z_i}$  to obtain the vector  $\underline{\delta}^k$ .
- (4) Right multiply (outer product)  $\underline{\delta}^k$  by  $\underline{u}_k^T$ .
- (5) And sum over all elements of the training set to obtain  $\frac{\partial J}{\partial \underline{W}}$ .

$\frac{\partial J}{\partial \underline{W}}$  is then used to update  $\underline{W}$ , the matrix of coefficients, by

$$\underline{W}_{i+1} = \underline{W}_i - \eta [\nabla_{\underline{W}} J](\underline{W}_i)$$

A network diagram is shown on the next page to illustrate the process, followed by Matlab code.



Oct 19, 04 12:31

testnn.m

Page 1/1

```

% This file provides a simple test of back propagation for
% training a neural network.  The NN is trained to emulate
% the characteristics of a 2-input / 2-output (diagonal)
% function.  The number of examples in the training set
% is set by changing the number of columns of u (and at other
% places (poor design)).  For one column, training is very
% rapid, but for more elements of the training set, training
% fails (as with 5 elements).  The back propagation algorithm,
% however, is converging; this is an excellent example of a
% problem where the NN structure is not sufficiently rich to
% adequately describe the problem.
%
errnorm=zeros(100,1);
% define eta for convergence rate of conj. grad. algorithm
eta=0.2;
% define the number of elements in the training set.
kelem = 2;
% define input vector (2x2 problem)
m=2;
u=rand(m,kelem);
% true value of the output (training vector)
ytrue=f(u);
% initially random weights
w=rand(m,m);
% run 100 times and record the error
j=0;
for i=1:1024,
    j=(j+1);
    if (j>kelem), j=1;end
    z=w*u(:,j);
    y=sigmoid(z);
    e=y-ytrue(:,j);
    errnorm(i)=norm(e);
    djdw=delj(e,z,u(:,j));
    w=w-eta*djdw;
end;
% compute average error over each pass through the training set
navg=floor(max(size(errnorm))/kelem);
avgerr=zeros(navg,1);
for i=1:navg,
    avgerr(i)=mean(errnorm((kelem*(i-1)+1):(kelem*i)));
end;

```

Oct 19, 04 12:31

delj.m

Page 1/1

```
function djdw=delj(err,z,u)
% computes the gradient of the error function at W, given
% z and u
djdw = (err.*dsigmoid(z))*u';
```

Oct 19, 04 12:31

dsigmoid.m

Page 1/1

```
function y=dsigmoid(u)
a=3;
m=max(size(u));
y=zeros(u);
for i=1:m,
    x=exp(-a*u(i));
    y(i)=2*a*x/(1+x)^2;
end
```

```
function y=f(u)
% function which is used to train the NN
y=sin(5*u);
```

Oct 19, 04 12:31

sigmoid.m

Page 1/1

```
function y=sigmoid(u)
% evaluate sigmoid function at a real input value
a=3;
m=max(size(u));
y=zeros(u);
for i=1:m,
    x=exp(-a*u(i));
    y(i)=(1-x)/(1+x);
end
```

## When to update the Weights?

There's a choice:

- (1) Use each element of  $T$  to generate a gradient and update  $\underline{W}$ , or
- (2) Complete one pass of  $T$ , accumulating the gradient by summing all the gradients produced by each input/output pair, and then use the accumulated gradient to update  $\underline{W}$ .

Either way, multiple passes through the training set are required. In practice, (1) is probably used more often.