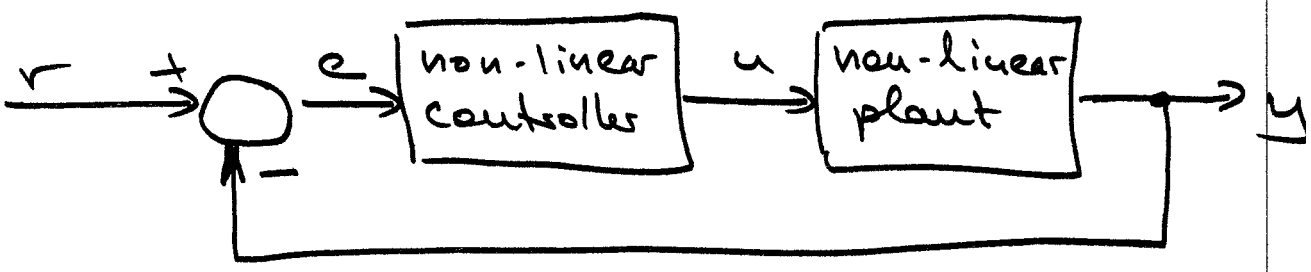
Fuzzy Control (Non-linear Control):

Until now, we only looked at the control of linear plants. How can we proceed if the plant model contains arbitrary non-linearities?

Let us look at the simple SISO system below:



For any given error signal, e, we need to find the controller output (plant input), u, that drives the plant output, y, to where we want it to be; in general, make e as small as possible.

This is an optimization problem. Since, for arbitrary non-linear systems, there is no closed-form synthesis rule available, we have to tune the controller by optimizing a performance index.

There are a number of options as to how this can be accomplished.

(I) Neural Control (NC):

The problem with the above problem is that we don't know how to optimize an unknown relationship. What we do know is how to tune a set of parameters. Yet, we cannot use this approach, unless we

first parametrize the problem. This means, we need to make some assumption about the structure of the controller, keep a number of unknown parameters in that structure, and then tune these parameters to optimize the performance of the controlled system.

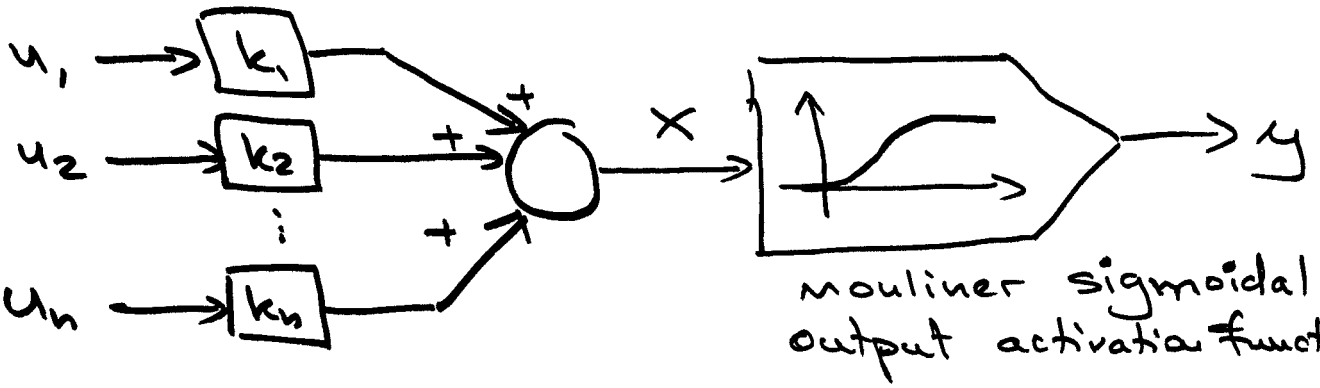
How do we choose the best possible structure, without already compromising the optimality of the underlying parameter optimization problem?

NC offers a solution to this problem: Assume a structure that is as generic as possible with many parameters, and

solve the overparametrized problem by finding a set (among many) of parameters that optimizes the system performance.

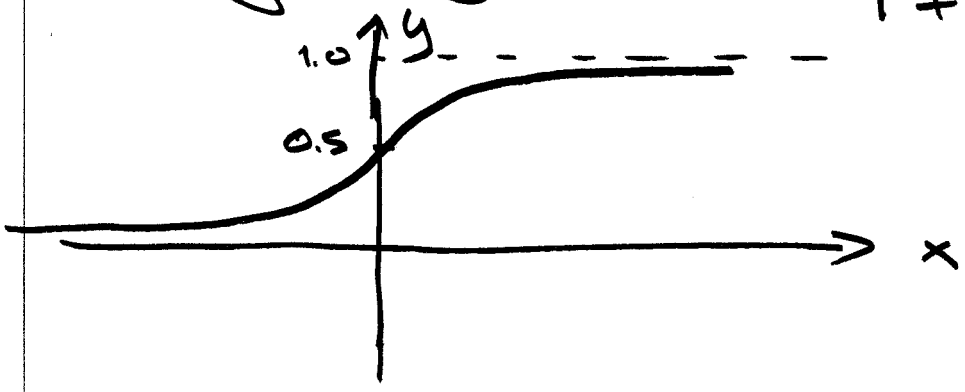
The fact that NC borrows its generic structure from an idealized version of the human brain is irrelevant! What counts is that NC offers a structure that is sufficiently generic to fit any curve!

Neuron Model:

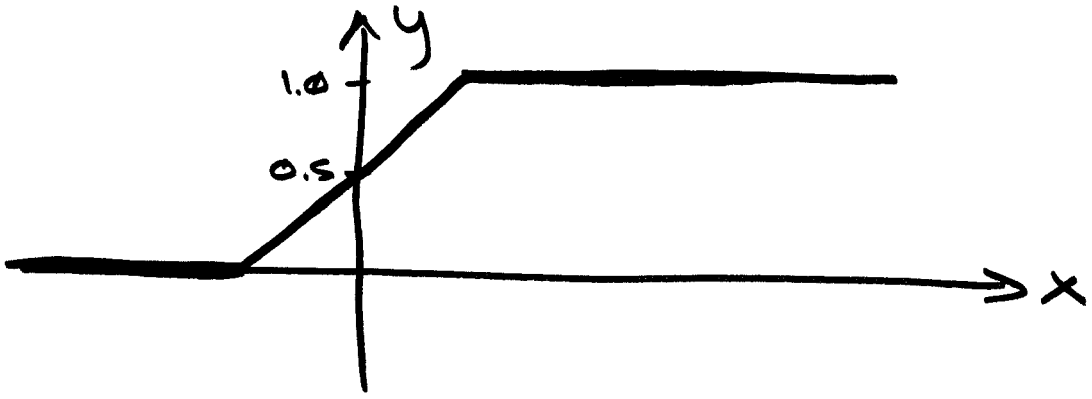


Although it is possible to parametize the output activation function, this is rarely done. Usually, the activation function is a <u>fixed</u> function, e.g.

$$y = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



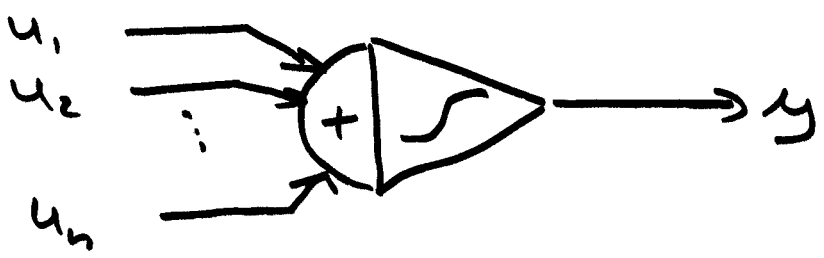
or: $y = \text{limit}(x + 0.5,\ 0.0,\ 1.0)$



and only the input gains (the "weights") are used as parameters:

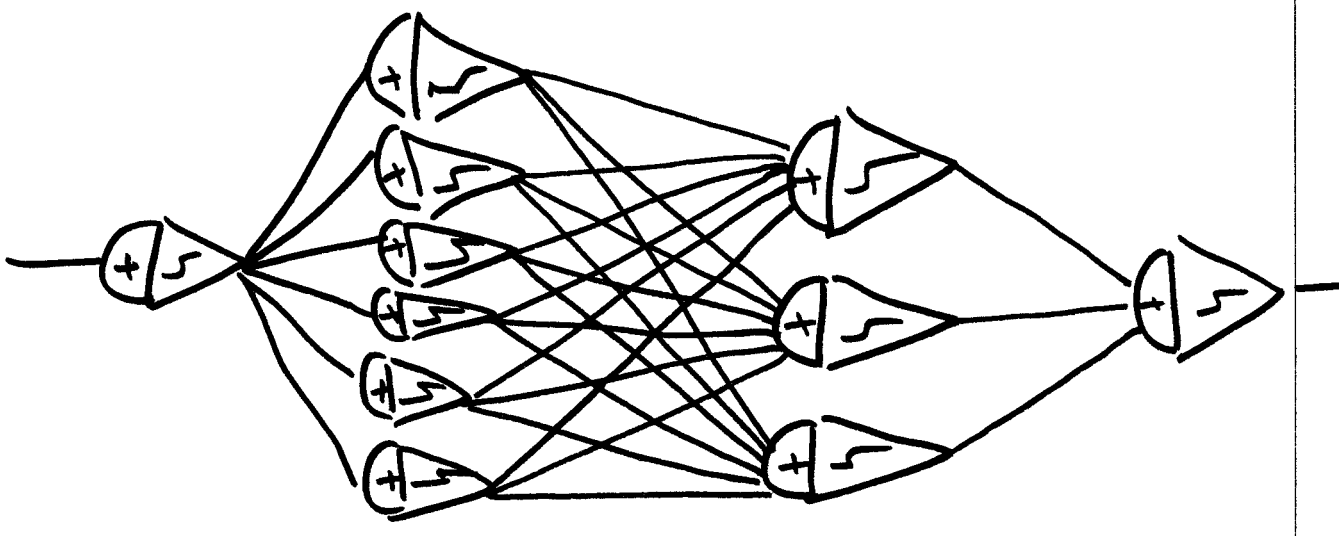
$$k_i \in [k_{i\,min}, k_{i\,max}] \quad .$$

Symbol:



shall symbolize one neuron.

A neural network is a connection of neurons, e.g.

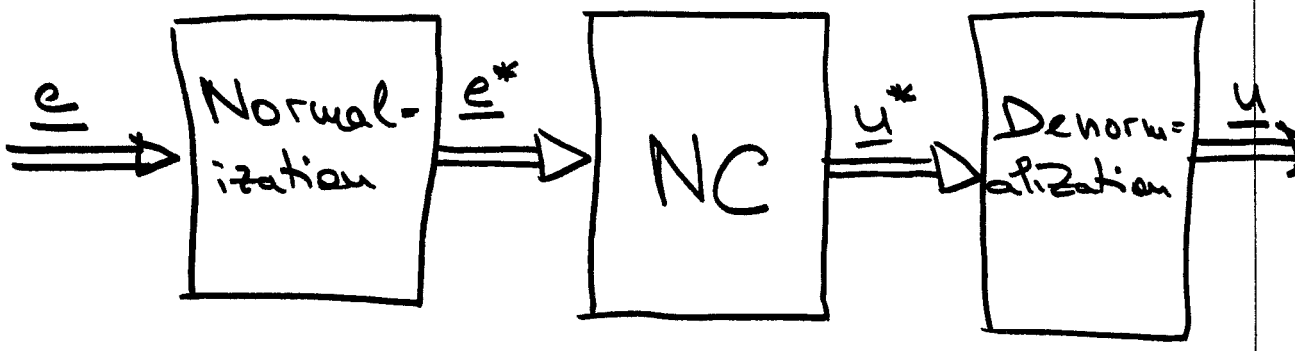


could be a <u>feedforward neural network</u> for a SISO system.

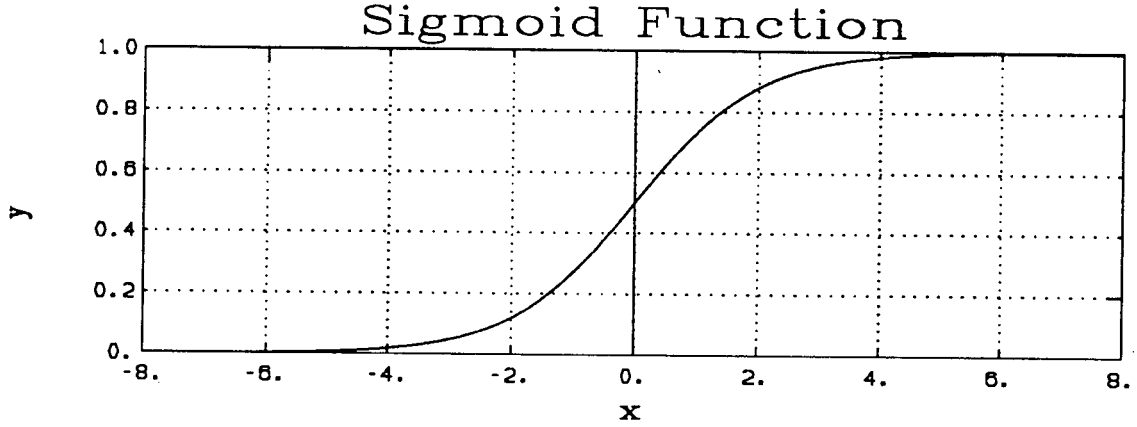
The above example has 4 layers: an input layer, two intermediate layers, and an output layer. The input layer has as many neurons as the system has inputs, the output layer has as many neurons as the system has outputs, and the intermediate layers have as many neurons as needed to provide the desired generality.

The neural network literature sometimes doesn't count the input layer, and then talks about a 3-layer network.

In a control application, the inputs to the controller can be of different magnitude, and the outputs (i.e., plant inputs) can also be of arbitrary magnitude. Thus, a NC usually has a <u>normalization stage</u> and a <u>denormalization stage</u>:



$$y = \text{sigmoid}(x) = \frac{1.0}{1.0 + \exp(-x)} \qquad (14.16)$$



$$\underline{e}^* = D_e \cdot \underline{e} + \underline{b}_e$$

Diagonal matrix — bias vector

$D_e, \underline{b}_e$ chosen such that

$$\underline{e}^* \in [-5, 5]$$

if the sigmoid function is used.

$$\underline{u}^* \in [0, 1]$$

$$\Rightarrow \underline{u} = D_u \cdot \underline{u}^* + \underline{b}_u$$

$D_u, \underline{b}_u$ chosen such that the plant inputs have the desired magnitudes.

$D_e, D_u, \underline{b}_e, \underline{b}_u$ are usually preselected, and <u>not</u> parameters to be optimized.

It can be shown that a single intermediate (so-called "hidden") layer suffices to approximate any convex non-linear function, and that 2 hidden layers suffice to approximate any arbitrarily shaped m-dimensional function:

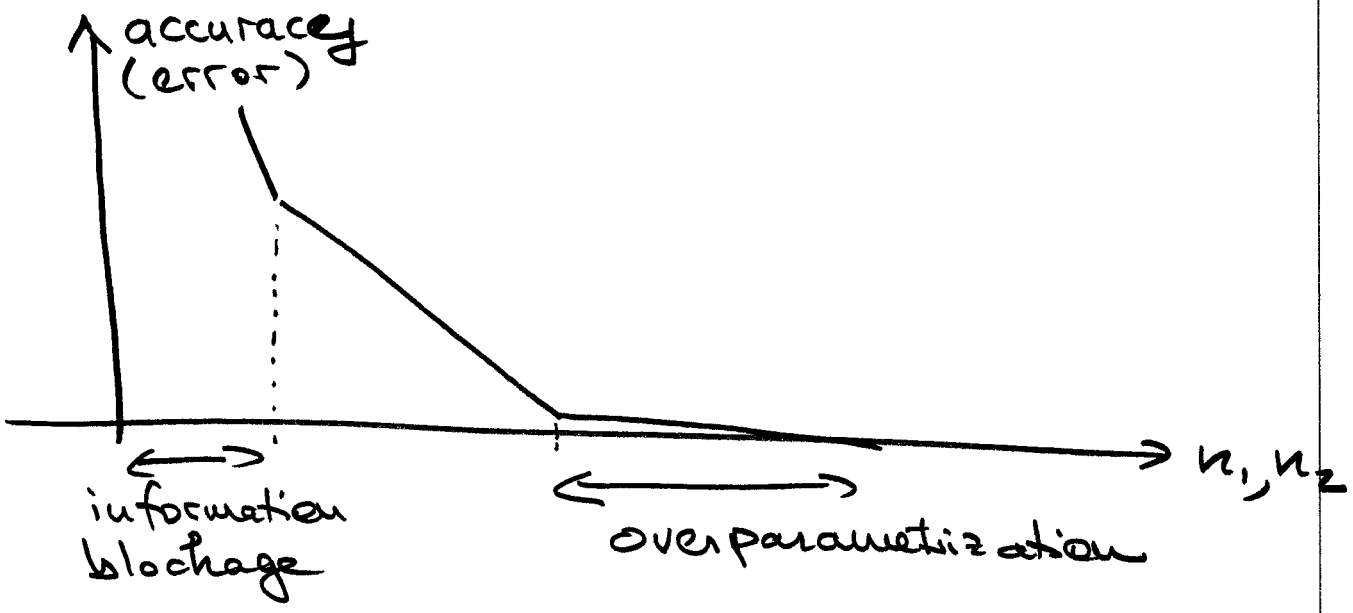
$$[y_1, y_2, \ldots, y_p] = \overline{\mp}(u_1, u_2, \ldots, u_m)$$

↑ arbitrary function (single-valued)

can be approximated by

layer	# of neurons
input	m
hidden 1	n_1
hidden 2	n_2
output	p

n_1 and n_2 control the <u>accuracy</u> of the approximation.



There are two "knees" in the curve. If n_1, n_2 are too small, information is blocked, and the approximation doesn't work. If n_1, n_2 are too large, the optimization problem is over-parametrized, and adding more neurons doesn't help.

Some papers try to come up with formulae to estimate

where the two lines are, i.e., how many neurons should be used.

A feedforward neural network is a generic static function generator. Once the structure has been chosen (i.e., the number of hidden layers and the number of neurons per layer have been set), we are left with a parameter optimization problem to determine the weight matrices at the input of each of the layers.