Intelligent Control Systems
(0640734)

Lecture (5)

Neural Networks in Control Systems

Prof. Kasim M. Al-Aubidy
Philadelphia University-Jordan

Introduction:

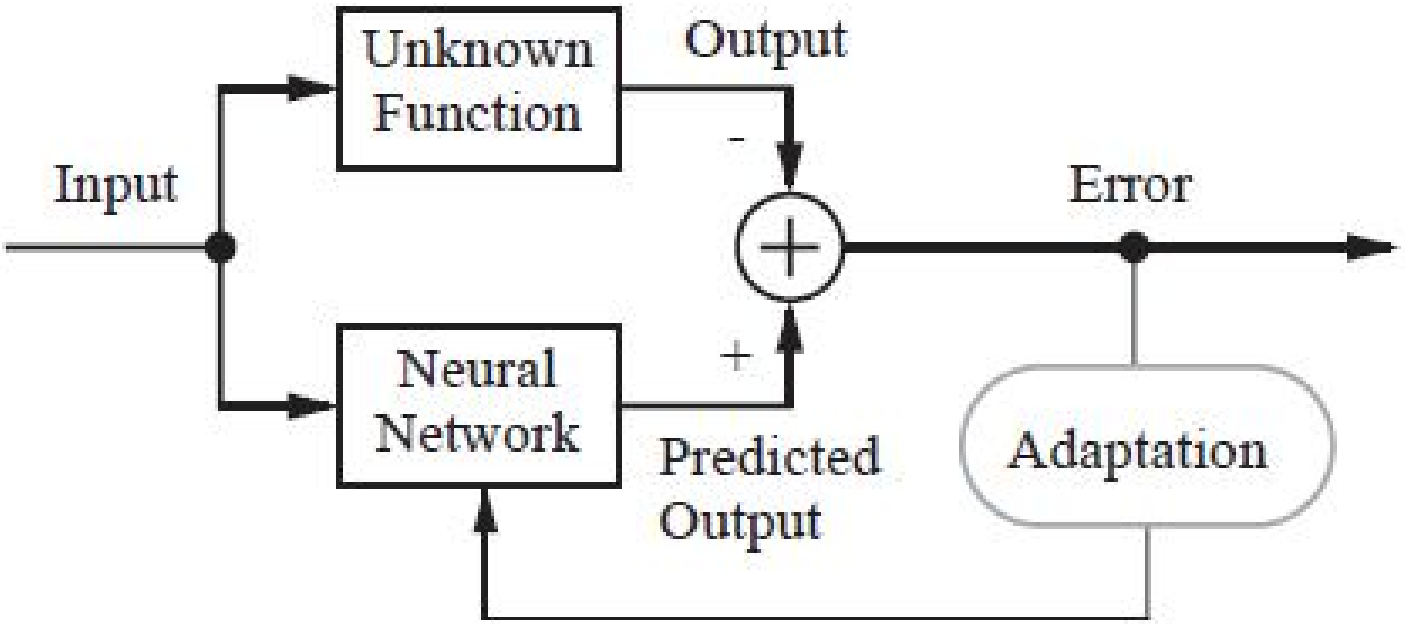
- The use of NN in control systems was first proposed by Werbos [1989] and Narendra [1990].
- NN control has had two major objectives:
  - Approximate Dynamic Programming, which uses NN to approximately solve the optimal control problem, and
  - NN in closed-loop feedback control.
- The challenge in using NN for feedback control purposes is to select a suitable control system structure, and then to demonstrate using mathematically acceptable techniques how the NN weights can be tuned so that closed-loop stability and performance are guaranteed.

Why Neural Control?

- When mathematical models of the plant dynamics are not available, NNs can provide a useful method for designing controllers, provided we have numerical information about the system behavior in the form of input-output data.
- NN can be used as a "black box" model for a plant.
- Controllers based on NNs will benefit from NNs. learning capability that is suitable for adaptive control where controllers need to adapt to changing environment.
- NN controllers have proved to be most useful for time-invariant systems.
- To build a NN based controller that can force a plant to behave in some desirable way, we need to adjust its parameters from the observed errors.
- Adjustment of the controller's parameters will be done by propagating back these errors across the NN structure. This is possible if the mathematical model of the plant is known.
- NN identified models are used in indirect neural control designs.
- NNs are massive parallel computation structures. This allows calculations to be performed at a high speed, making real-time implementations feasible.
- NNs can also provide significant fault tolerance, since damage to a few weights need not significantly impair the overall performance.

Function Approximator:

- Neural Network is used as a Function Approximator.
- We have some unknown function that we wish to approximate.
- We want to adjust the parameters of the network so that it will produce the same response as the unknown function, if the same input is applied to both systems.



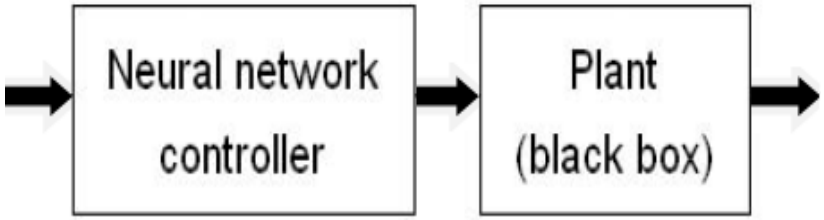
Neural Network Control Topologies:

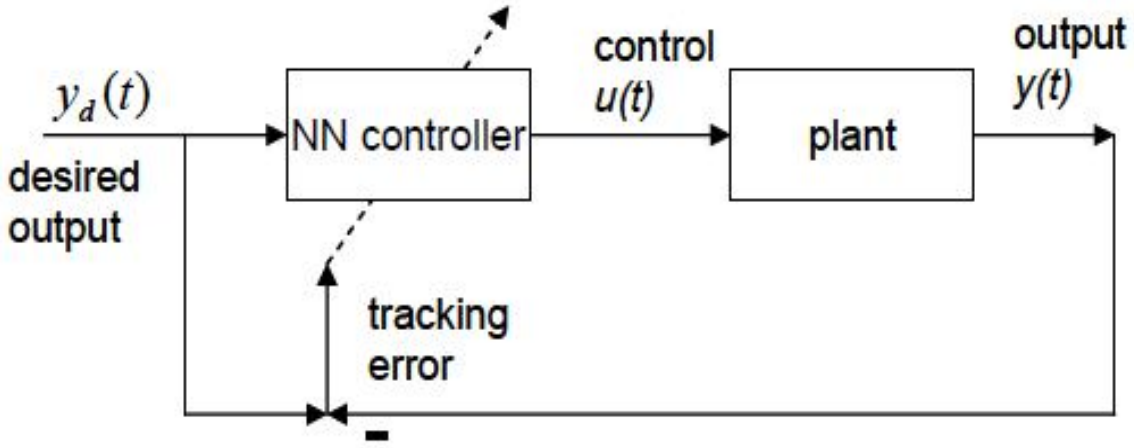
There are two design methods for neural control, these are;

- **Direct Method:** in which the controller itself is a neural network.
- **Indirect Method:** in which controllers are designed based on a neural network model of the plant.

1. Direct Method:

Direct design means that a neural network directly implements the controller (Neural network).
The network must be trained according to some criteria, using either numerical I/O data or a mathematical model of the system.



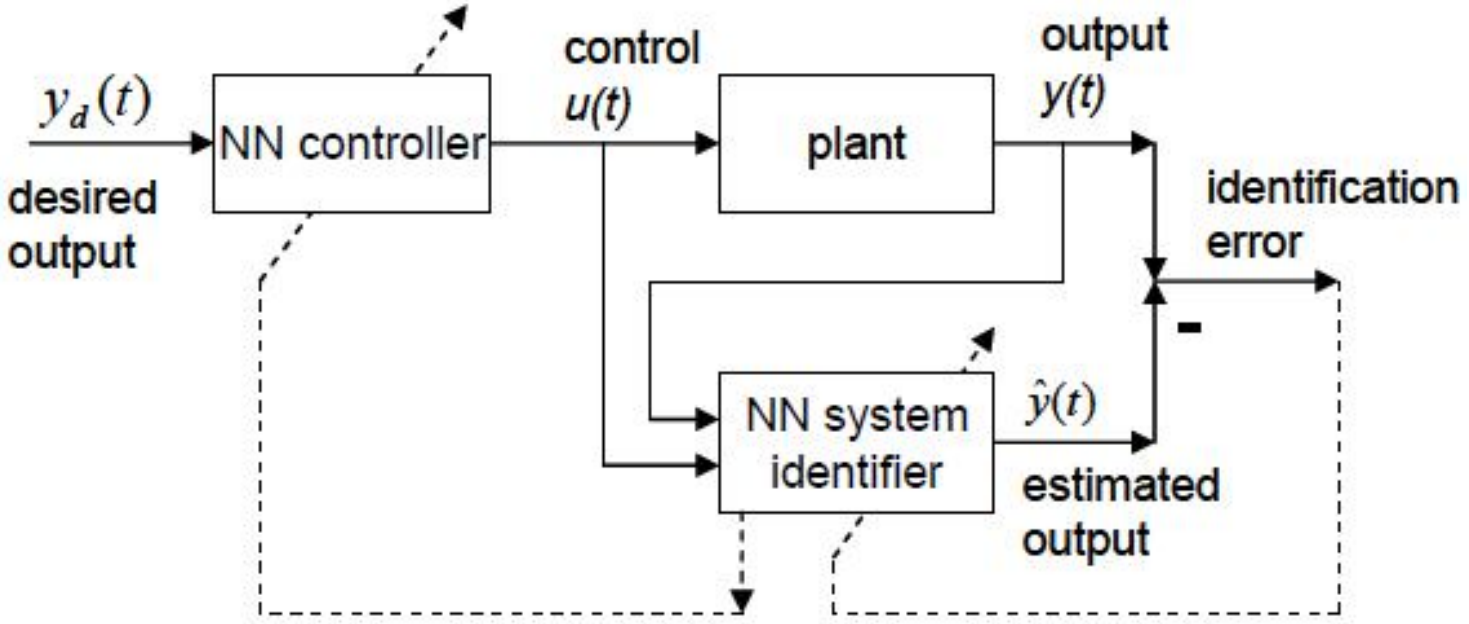


Indirect NN Control:

Indirect neural control design is based on a neural network model of the system to be controlled. In this case, the controller itself may not be a neural network, but it is derived from a plant that is modeled by a neural network.

There are two phases;

- NN System Identifier Unit: the NN is tuned to learn the dynamics of the unknown plant, and
- NN Controller Unit: the NN uses this information to control the plant.



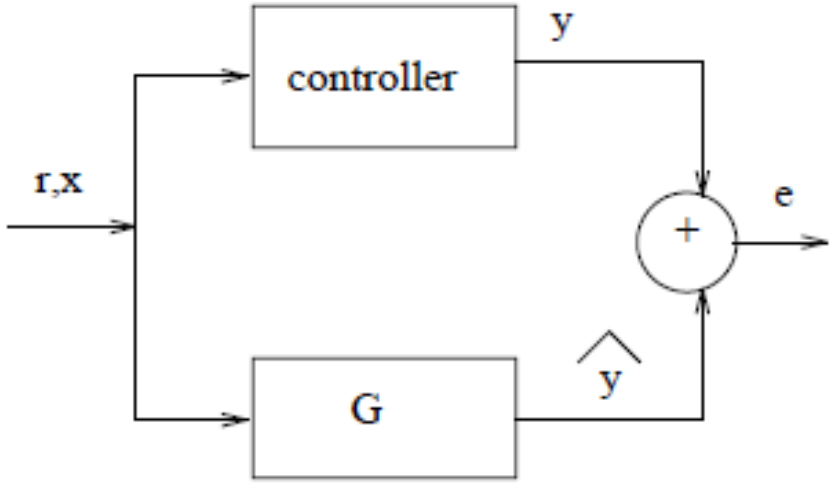
Neural Control:

Control methods utilize NNs in conjunction with classical controllers;

- Control by emulation.
- Inverse and open-loop control.
- Feedback control.

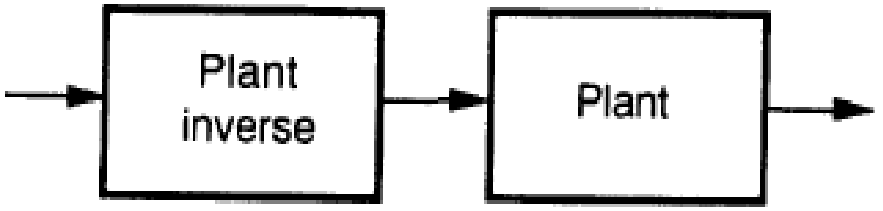
1. Expert Emulator:

If a controller exists, we can emulate it.
There is no feedback, and we can train NN
With standard PB.
This is useful if the controller is a human and
We want to automate the process.



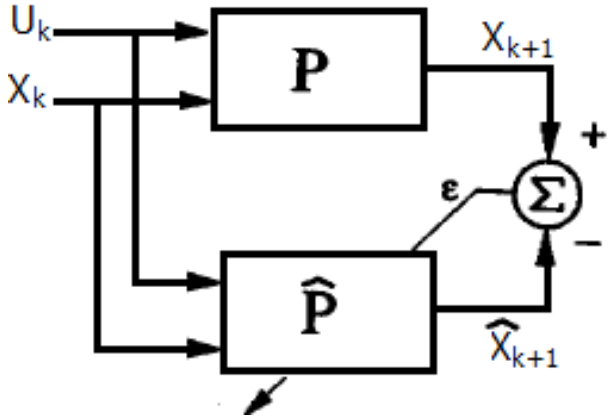
2. Open Loop/ Inverse Control:

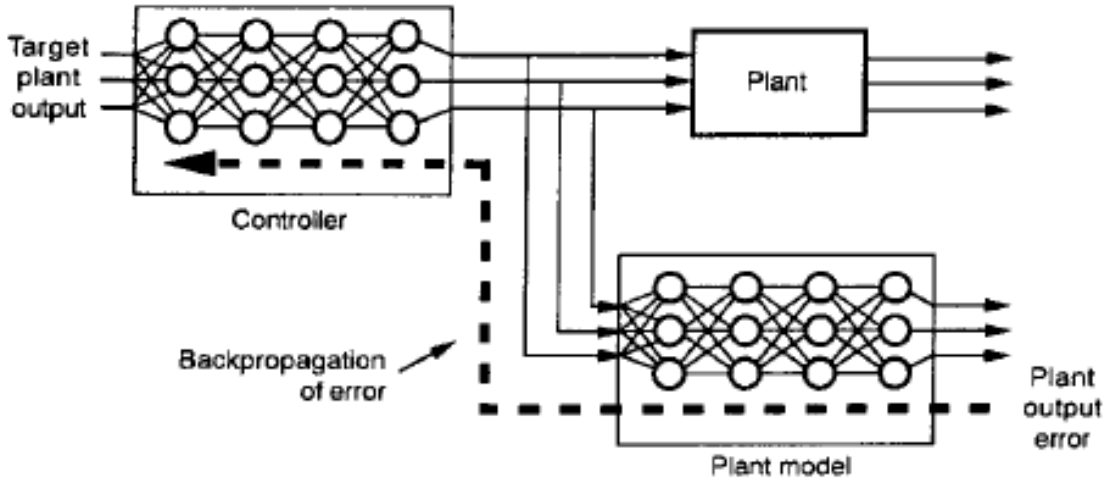
Early systems were designed open loop making the neural controller approximately the inverse of the plant to be controlled.



Note:

It is useful to build a plant emulator, then train the Neural Controller parameters as shown bellow.

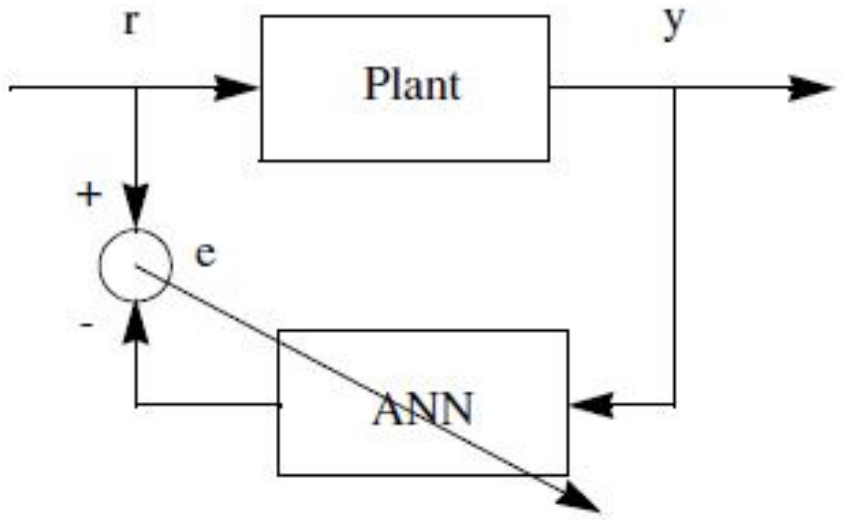




NN Training:

There are several possibilities for training the NN, depending on the architecture used for learning;

1. **Generalized learning architecture:** The error (e) is used to adjust the neural network parameters. NN training involves supplying different inputs to the plant and teaching the network to map the corresponding outputs back to the plant inputs.

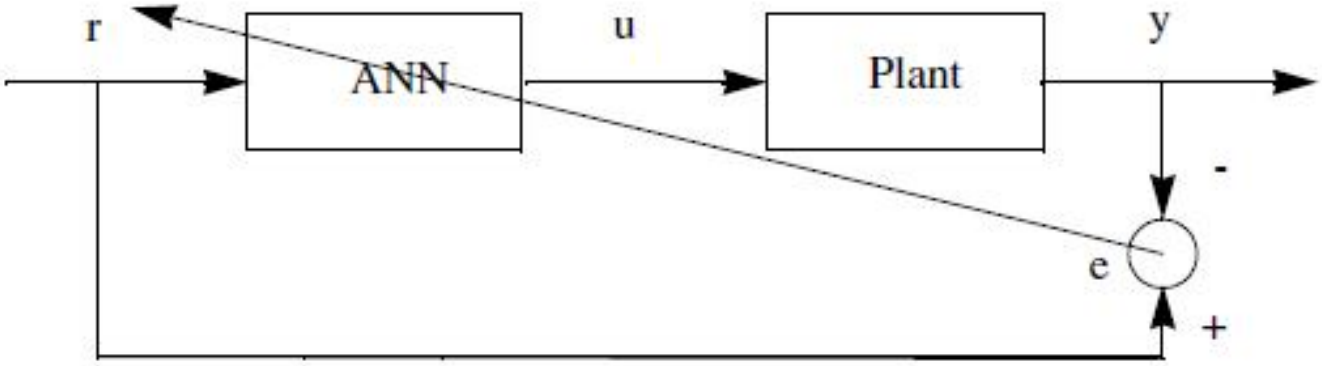


Drawbacks:

- It works well but has the drawback of not training the network over the range where it will operate, after having been trained.
- The NN may have to be trained over an extended operational range.
- It is impossible to train the network on-line.

2. Specialized learning architecture:

It is used to overcome the disadvantages of the generalized learning approach. The NN learns how to find the inputs (u) that drive the system outputs (y) towards the reference signal (r). The NN is trained in the region where it will operate, and can also be trained on-line.



Drawbacks:

The problem with this architecture lies in the fact that although the error (e= r-y) is available, there is no explicit target for the control input (u) to be applied in the training of the NN.

To overcome these difficulties, research paper [] proposed either replacing the Jacobian elements by their sign, or performing a linear identification of the plant by an adaptive least-squares algorithm.

Inverse dynamics:

An ideal control law describes the inverse dynamics of a plant.

Consider, the time-invariant linear system described in discrete time by the equations

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + B\mathbf{u}(k),\ k = 0, 1, 2, ...$$

where A is an $n \times n$ matrix and B is $n \times 1$. Then we have

$$\begin{aligned}
\mathbf{x}(k+2) &= A\mathbf{x}(k+1) + B\mathbf{u}(k+1) \\
&= A(A\mathbf{x}(k) + B\mathbf{u}(k)) + B\mathbf{u}(k+1) \\
&= A^2\mathbf{x}(k) + AB\mathbf{u}(k) + B\mathbf{u}(k+1) \\
\mathbf{x}(k+3) &= A\mathbf{x}(k+2) + B\mathbf{u}(k+2) \\
&= A\left(A^2\mathbf{x}(k) + AB\mathbf{u}(k) + B\mathbf{u}(k+1)\right) + B\mathbf{u}(k+2) \\
&= A^3\mathbf{x}(k) + A^2B\mathbf{u}(k) + AB\mathbf{u}(k+1) + B\mathbf{u}(k+2)
\end{aligned}$$

$$
\begin{aligned}
\mathbf{x}(k+4) &= A\mathbf{x}(k+3) + B\mathbf{u}(k+3) \\
&= A\left(A^3\mathbf{x}(k) + A^2B\mathbf{u}(k) + AB\mathbf{u}(k+1) + B\mathbf{u}(k+2)\right) + B\mathbf{u}(k+3) \\
&= A^4\mathbf{x}(k) + A^3B\mathbf{u}(k) + A^2B\mathbf{u}(k+1) + AB\mathbf{u}(k+2) + B\mathbf{u}(k+3)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{x}(k+n) &= A^n\mathbf{x}(k) + A^{n-1}Bu(k) + \cdots + ABu(k+n-2) + Bu(k+n-1) \\
&= A^n\mathbf{x}(k) + WU
\end{aligned}
$$

where $W = \begin{bmatrix} A^{n-1}B & A^{n-2}B & \cdots & A^2B & AB & B \end{bmatrix}$

$U = \begin{bmatrix} u(k) & u(k+1) & \cdots & u(k+n-2) & u(k+n-1) \end{bmatrix}^T$

If the matrix W is nonsingular, we can compute U directly from W and $\mathbf{x}$:

$$
\begin{aligned}
U &= W^{-1}\left(\mathbf{x}(k+n) - A^n\mathbf{x}(k)\right) \\
&= \varphi\left(\mathbf{x}(k), \mathbf{x}(k+n)\right)
\end{aligned}
$$

This is a control law for the system that is derived directly from the plant dynamics by computing the inverse dynamics.

Example *Take* $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -5 & -2 & -3 \end{bmatrix}$ *and* $B = \begin{bmatrix} 0 \\ 0 \\ 9 \end{bmatrix}$. *Then*

$$W = \begin{bmatrix} A^2B & AB & B \end{bmatrix} = \begin{bmatrix} 9 & 0 & 0 \\ -27 & 9 & 0 \\ 63 & -27 & 9 \end{bmatrix}$$

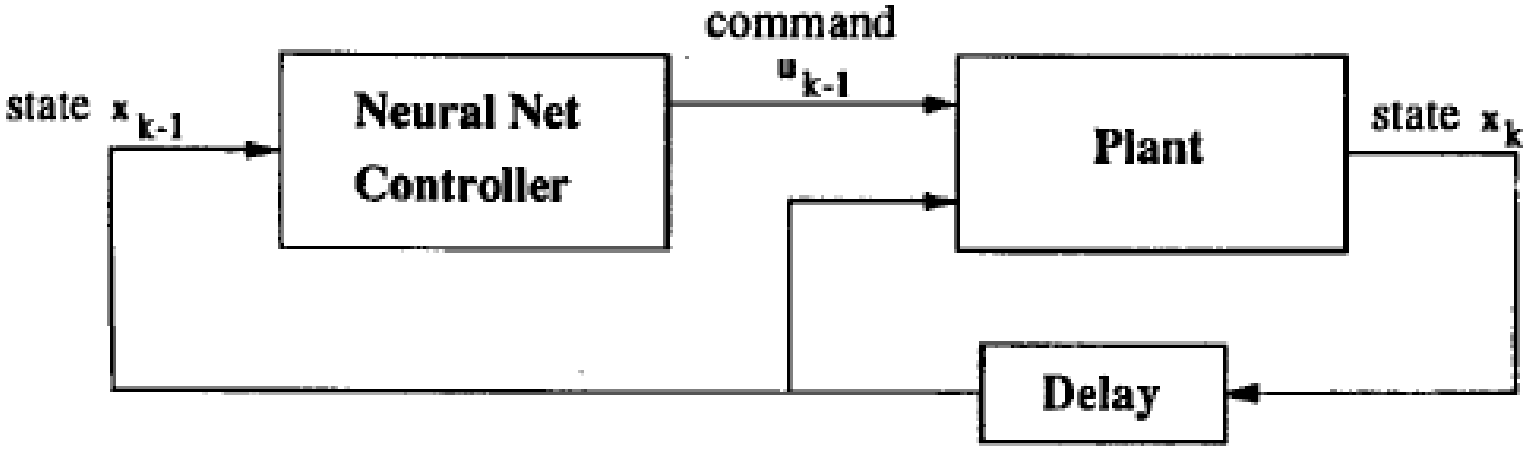
and

$$\begin{aligned} U &= \begin{bmatrix} \frac{1}{9} & 0 & 0 \\ \frac{1}{3} & \frac{1}{9} & 0 \\ \frac{2}{9} & \frac{1}{3} & \frac{1}{9} \end{bmatrix} \left(\mathbf{x}(k+3) - \begin{bmatrix} -5 & -2 & -3 \\ 15 & 1 & 7 \\ -35 & 1 & -20 \end{bmatrix} \mathbf{x}(k) \right) \\ &= \begin{bmatrix} \frac{1}{9} & 0 & 0 \\ \frac{1}{3} & \frac{1}{9} & 0 \\ \frac{2}{9} & \frac{1}{3} & \frac{1}{9} \end{bmatrix} \mathbf{x}(k+3) - \begin{bmatrix} -\frac{5}{9} & -\frac{2}{9} & -\frac{1}{3} \\ 0 & -\frac{5}{9} & -\frac{2}{9} \\ 0 & 0 & -\frac{5}{9} \end{bmatrix} \mathbf{x}(k) \\ &= \begin{bmatrix} \frac{1}{9}x_1(k+3) + \frac{5}{9}x_1(k) + \frac{2}{9}x_2(k) + \frac{1}{3}x_3(k) \\ \frac{1}{3}x_1(k+3) + \frac{1}{9}x_2(k+3) + \frac{5}{9}x_2(k) + \frac{2}{9}x_3(k) \\ \frac{2}{9}x_1(k+3) + \frac{1}{3}x_2(k+3n) + \frac{1}{9}x_3(k+3) + \frac{5}{9}x_3(k) \end{bmatrix} \\ &= \varphi(\mathbf{x}(k), \mathbf{x}(k+n)) \end{aligned}$$

When the inverse dynamics of a plant exist, you can control the plant by modeling its inverse dynamics.

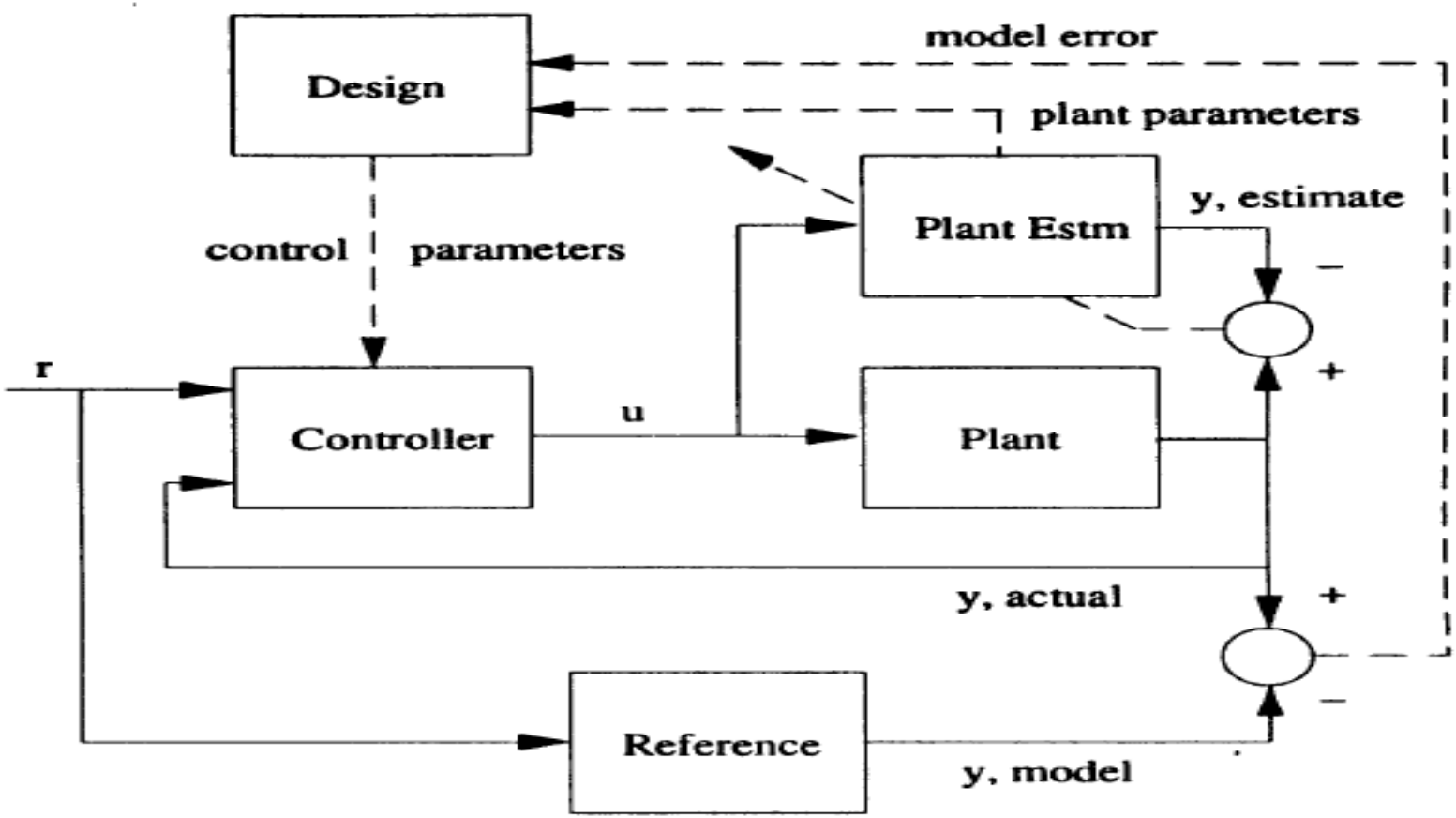
3. Classical Feedback Control:

These methods use neural nets to improve dynamics in conjunction with classical methods. The feedback is ignored during designing the controller.



4. Neural Feedback Control:

Incorporating feedback, neural system can be trained with similar objectives and cost functions, as classical control but solution are iterative and approximate. Optimal control techniques can be applied which are constrained to be approximate by nature of training and NN architectural constraints.



CONTROL SYSTEM APPLICATIONS:

Ref: M.T. HAGAN, H.B. DEMUTH & O. DE JESÚS, "AN INTRODUCTION TO THE USE OF NEURAL NETWORKS IN CONTROL SYSTEMS", Available online on net.

This paper outlines a brief introduction to the use of neural networks in control systems.

MLP NNs have been applied successfully in the identification and control of dynamic systems.

There are three well-known typical neural network controllers:

- Model Predictive Control,
- Feedback Linearization Control, and
- Model Reference Control.

There are two steps involved when using NNs for control:

System Identification: develop a neural network model of the plant that we want to control.

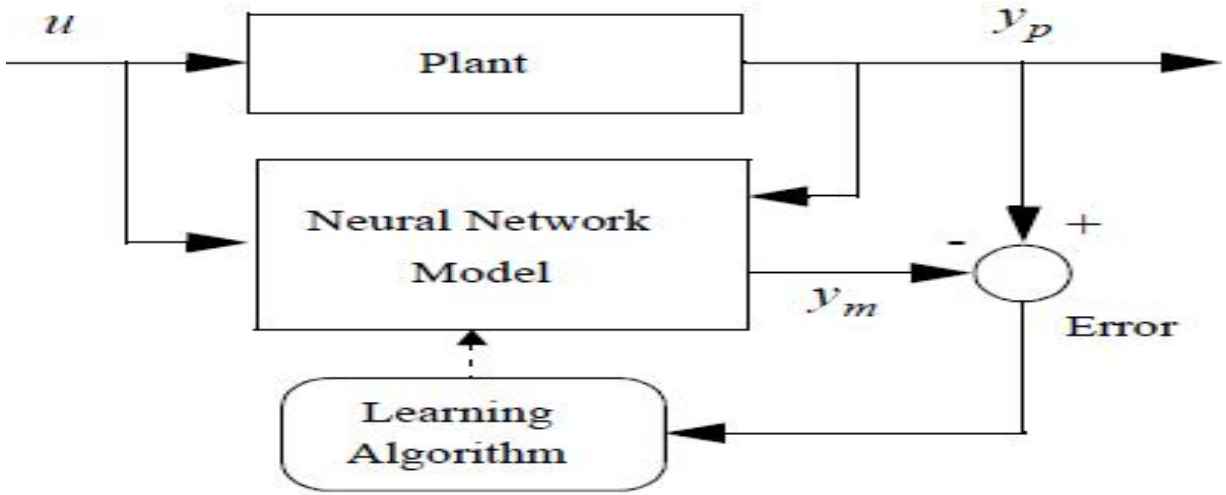
Control Design: use the neural network plant model to design (or train) the controller.

1. Model Predictive Control:

- The first step in model predictive control is to determine the NN plant model (system identification). Next, the plant model is used by the controller to predict future performance.

System Identification:

- The first stage of model predictive control is to train a NN to represent the forward dynamics of the plant.
- The prediction error between the plant output and the NN output is used as the NN training signal.
- The neural network plant model uses previous inputs and previous plant outputs to predict future values of the plant output.



Predictive Control:

- The NN model predicts the plant response over a specified time horizon.
- The predictions are used by a numerical optimization program to determine the control signal that minimizes the following performance criterion;

$$J = \sum_{j=N_1}^{N_2} (y_r(k+j) - y_m(k+j))^2 + \rho \sum_{j=1}^{N_u} (u'(k+j-1) - u'(k+j-2))^2$$

where;

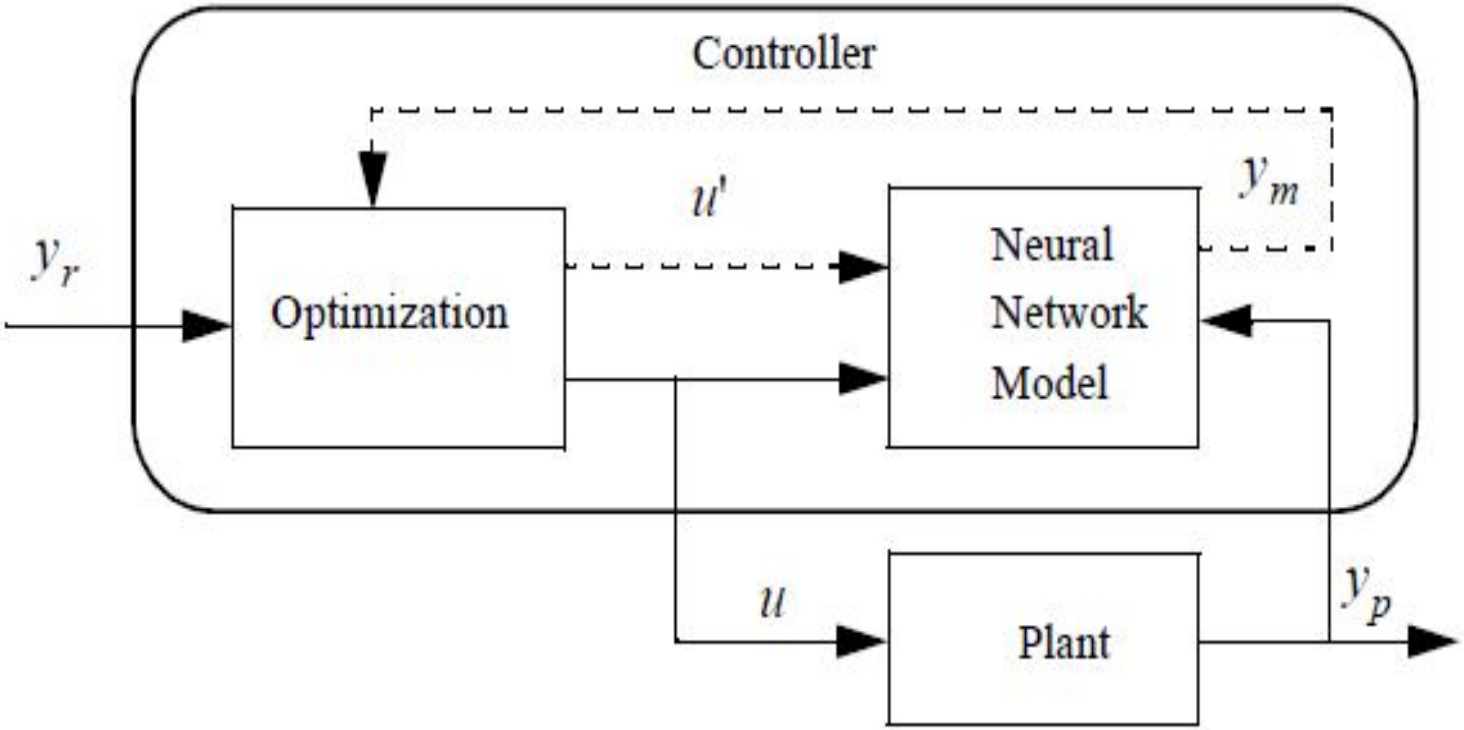
N1, N2, and Nu: define the horizons over which the tracking error and the control increments are evaluated.

ŭ: is the control signal,

Yr: is the desired response and ym is the network model response.

ρ: determines the contribution that the sum of the squares of the control increments has on the performance index.

Neural Network Predictive Control:



The controller consists of the NN plant model and the optimization block. The optimization block determines the values of ŭ that minimize J , and then the optimal u is input to the plant.

2. Feedback Linearization Control:

- The idea of this controller is to transform nonlinear system dynamics into linear dynamics by canceling the nonlinearities.
- The next control input is computed to force the plant output to follow a reference signal.
- The NN plant model is trained with static backpropagation. The controller is a rearrangement of the plant model, and requires minimal online computation.

Identification Stage:

As with model predictive control, the first step is to identify the system to be controlled. The approximate model is given by:

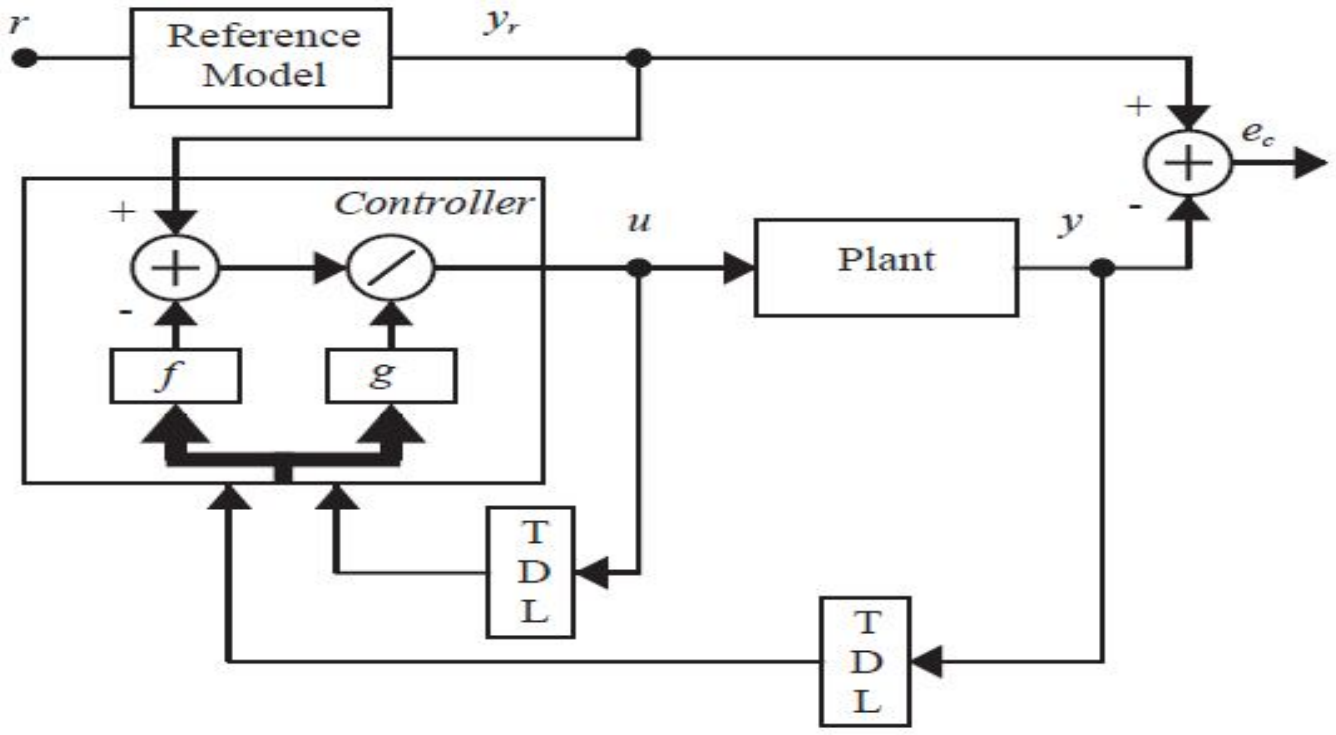
$$\hat{y}(k+d) = f[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-m+1)] + g[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-m+1)] \cdot u(k)$$

Control Stage:

The advantage of the NN model of the plant is that you can solve for the control input that causes the system output to follow a reference signal: $y(k+d) = y_r(k+d)$

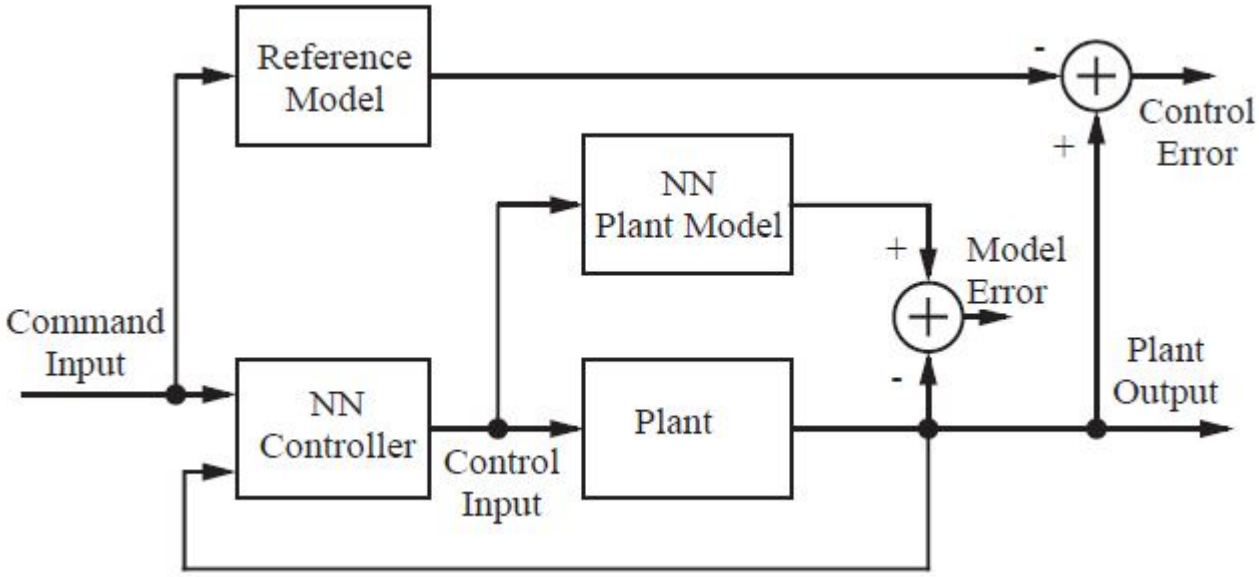
The resulting controller would have the form:

$$u(k) = \frac{y_r(k+d) - f[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-m+1)]}{g[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-m+1)]}$$



3. Model Reference Control:

- This architecture uses two NNs: a controller network and a plant model network.
- A neural network plant model is first developed. The plant model is then used to train a neural network controller to force the plant output to follow the output of a reference model.
- This control architecture requires the use of dynamic backpropagation for training the controller. This generally takes more time than training static networks with the standard backpropagation algorithm.

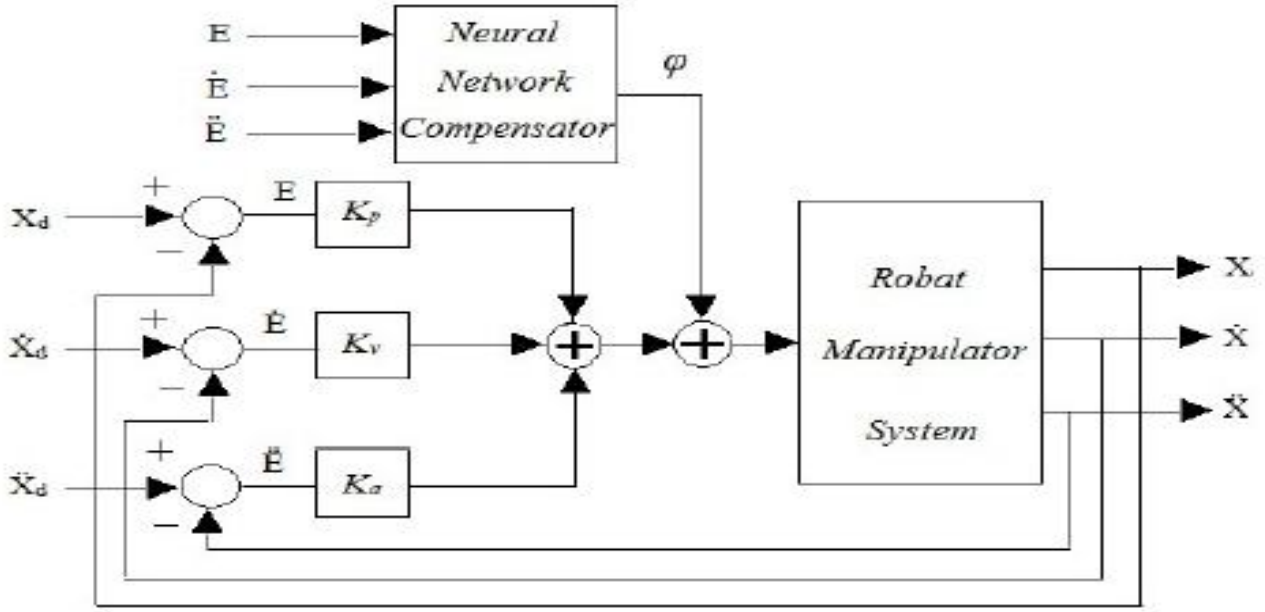


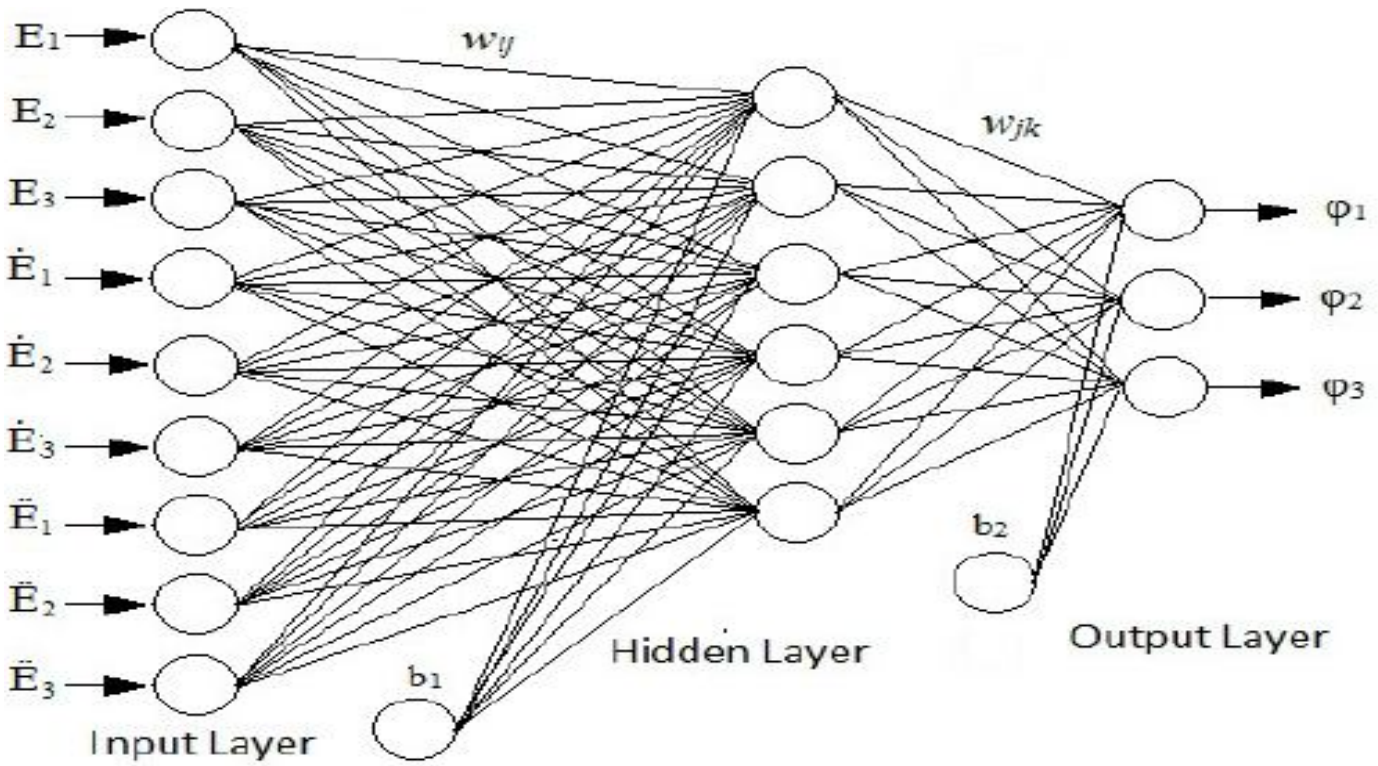
Applications:

NN PID Controller of a Robot Manipulator:

Ref: S. Pezeshki1, S. Badalkhani and A. Javadi, "Performance Analysis of a Neuro-PID Controller Applied to a Robot Manipulator", Int J Adv Robotic Sy, 2012, Vol. 9.

- A NN PID controller is used to improve the performance of a robot manipulator.
- The NN is applied to compensate the effect of the uncertainties of the robot model.
- The NN output φ cancels out the uncertainties caused by an accurate robot model. The output signal φ of the NN is added to the control input U whose dimension equals the number of degrees of freedom of the robot manipulator. Therefore, the vector φ is three dimensional.





- The three layer feedforward neural network structure is used as the compensator.
- It is composed of linear input layer, output layer and a nonlinear intermediate hidden layer which uses a sigmoid function.

Applications: NN-Based PID Auto-tuning:

The majority of the controllers used in industry are of the PID type. These controllers are tuned using simple empirical rules such as the Ziegler-Nichols tuning rule. The controllers are often poorly tuned due to neglect or lack of time.

The continuous PID controller transfer function is;

$$U(s) = k_c\left(1 + T_i s + \frac{T_d s}{1 + T_f s}\right)E(s)$$

The digital PID controller can be written as:

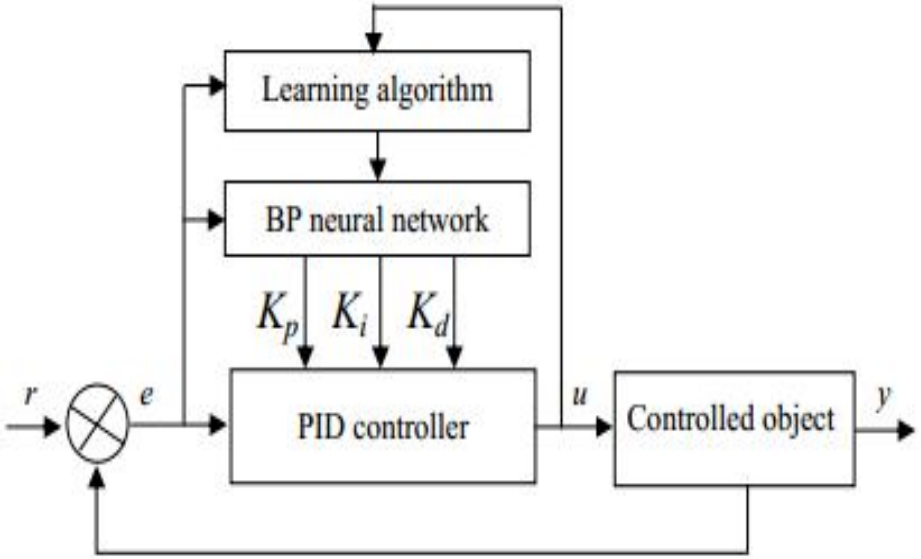
$$u[k] = k_D\left(1 + \frac{T_s}{T_{DI}}\frac{1}{q-1} + \frac{T_{DD}}{T_s}\frac{q-1}{q+\gamma}\right)e[k]$$

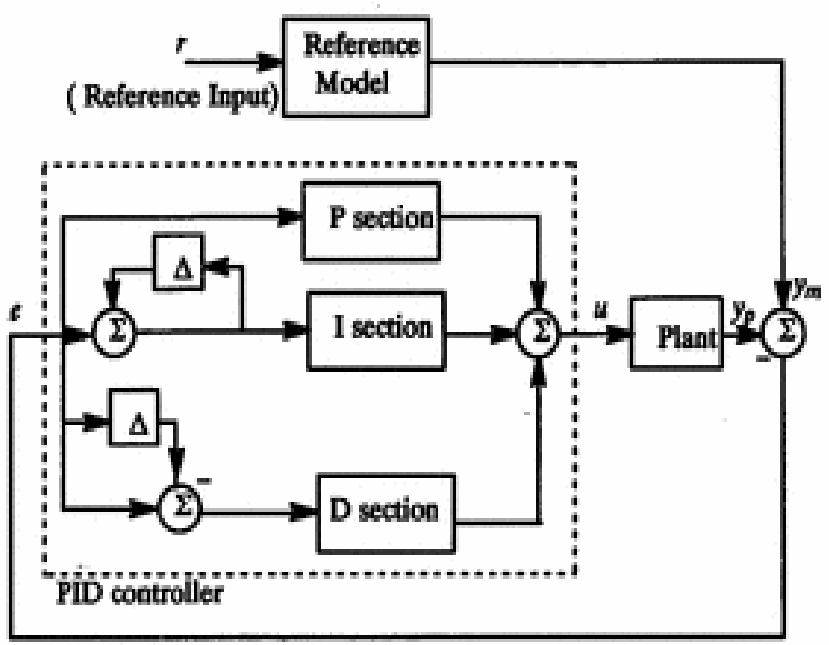
and γ is given by: $\gamma = -e^{-\frac{T_s}{T_f}}$

$$\Delta u[k] = u[k] - u[k-1] = k_D\left(1 - q^{-1} + \frac{q^{-1}T_s}{T_{DI}} + \frac{T_{DD}(1 - 2q^{-1} + q^{-2})}{T_s(1 + \gamma q^{-1})}\right)e[k]$$

Methods of PID auto-tuning:

- Ziegler and Nichols methods.
- On-line parameter estimation.
- Exert and fuzzy logic tuning methods.
- NN-based PID auto-tuner.



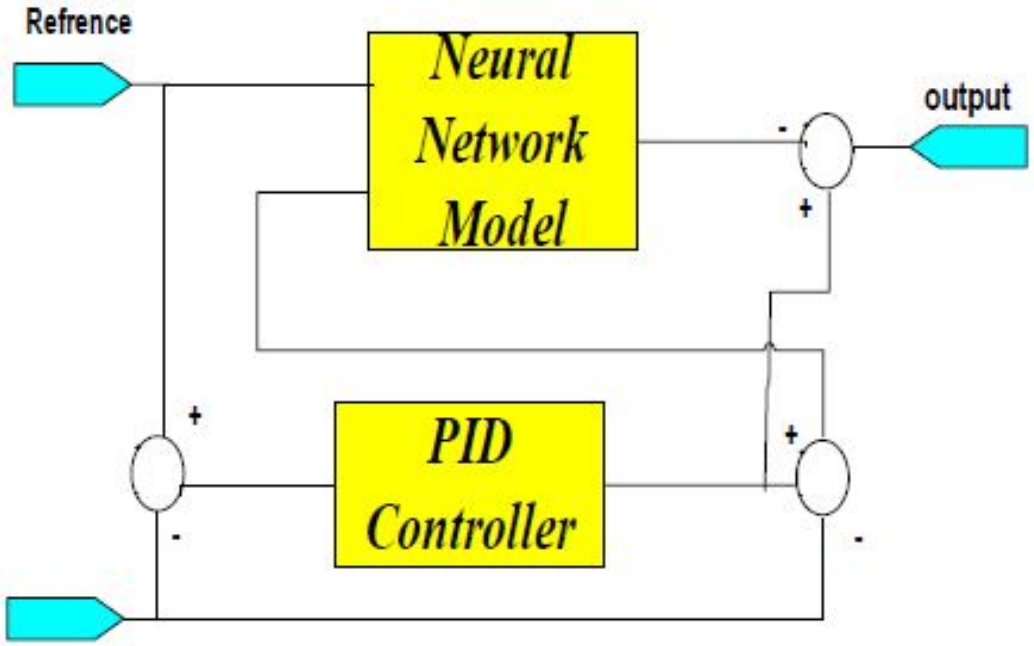


Ref:

1. *L. Luoren, L. Jinling, "Research of PID Control Algorithm Based on Neural Network", Energy Procedia 13 (2011) 6988 – 6993*
2. *M. Bahrami & K.E. Tait, "A neural network-based proportional integral derivative controller", Neural Computing & Applications, 1994, Vol.2, No.3, pp 134-141*

Neural Network controllers based on PID controllers:

This paper presents the control of two link robotic manipulator systems using Neural Network. It covers PD controller, PID Controller, NN Controller based on PD Controller and NN Controller based on PID Controller.



Ref: Leila Fallah Araghi, M. Habibnejad Korayem, Amin Nikoobin, Farbod Setoudeh, "Neural Network Controller Based on PID Controller for Two links- Robotic Manipulator Control", Proceedings of the World Congress on Engineering and Computer Science 2008 WCECS 2008, October 22 - 24, 2008, USA.

Neural PID Auto-tuner:

In this approach the open loop step response of a plant is discretized, its samples being used as inputs to a MLP. The role of the NN is, based on these inputs, to determine the corresponding PID parameters.

The MLP has therefore three outputs, corresponding to the three PID parameters.

Some observations can be made concerning this method of PID auto-tuning:

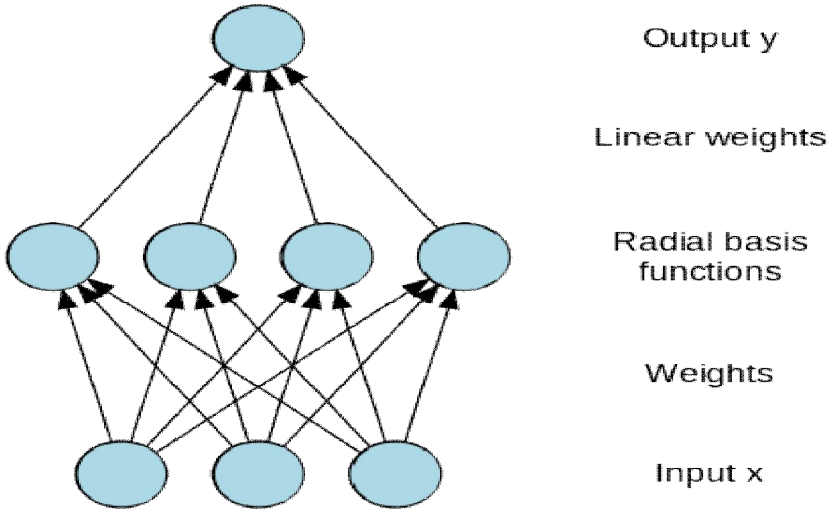
- It is an open loop technique, it can not be applied in closed loop.
- Because samples of the open loop step response are used as inputs to the MLPs,
- obtaining good results might require a high number of samples. This leads to a large number of parameters in the MLP, resulting in large training times.
- This technique is dependent on the sampling time chosen.
- The responses obtained with Ziegler-Nichols tuning rule are not well damped. It seems sensible to obtain the target PID values using a better criterion.
- If the Ziegler-Nichols technique is used to derive the target PID values then there is no need to consider three outputs for the MLP, since the integral and derivative time constants are linearly interdependent.

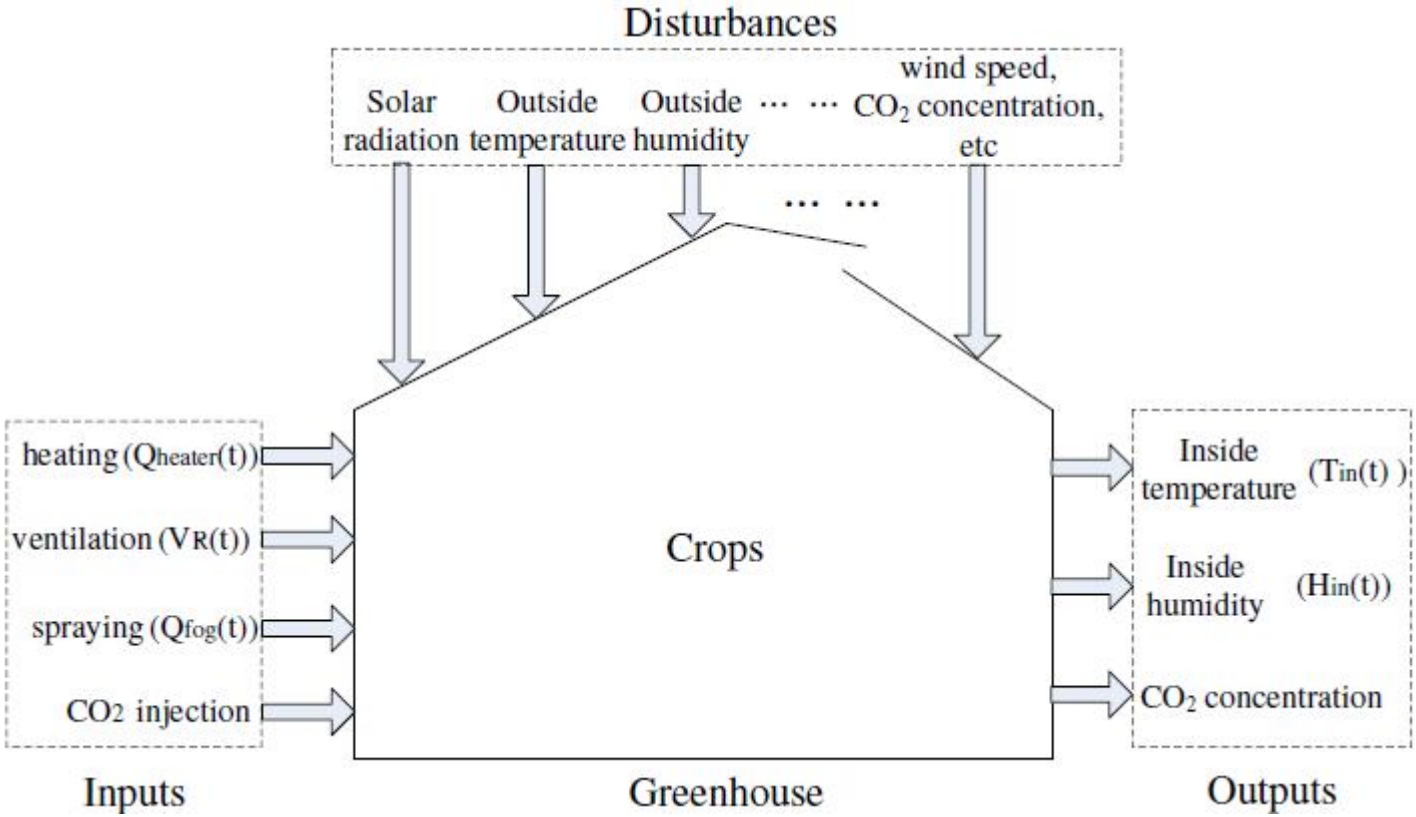
Applications: Nonlinear Adaptive PID Control

S. Zeng, H. Hu, L, Xu & G. Li , " Nonlinear Adaptive PID Control for Greenhouse Environment Based on RBF Network", Sensors 2012, 12, pp.5328-5348.

- A hybrid controller combining Radial Basis Function (RBF) neural network with PID controllers for the greenhouse climate control.
- A model of nonlinear plant (Greenhouse climate) is formulated.
- RBF NN is used to tune and identify all PID gain parameters online and adaptively.
- a RBF NN is an ANN that uses **radial basis function** as activation functions.
- A **radial basis function** is a real-valued function whose value depends only on the distance from the origin, so that ; or alternatively on the distance from some other point *c*, called a *center*, so that .

- RBF NN Architecture: An input vector (X) is used as input to all radial basis functions, each with different parameters. The output of the network is a linear combination of the outputs from radial basis functions.





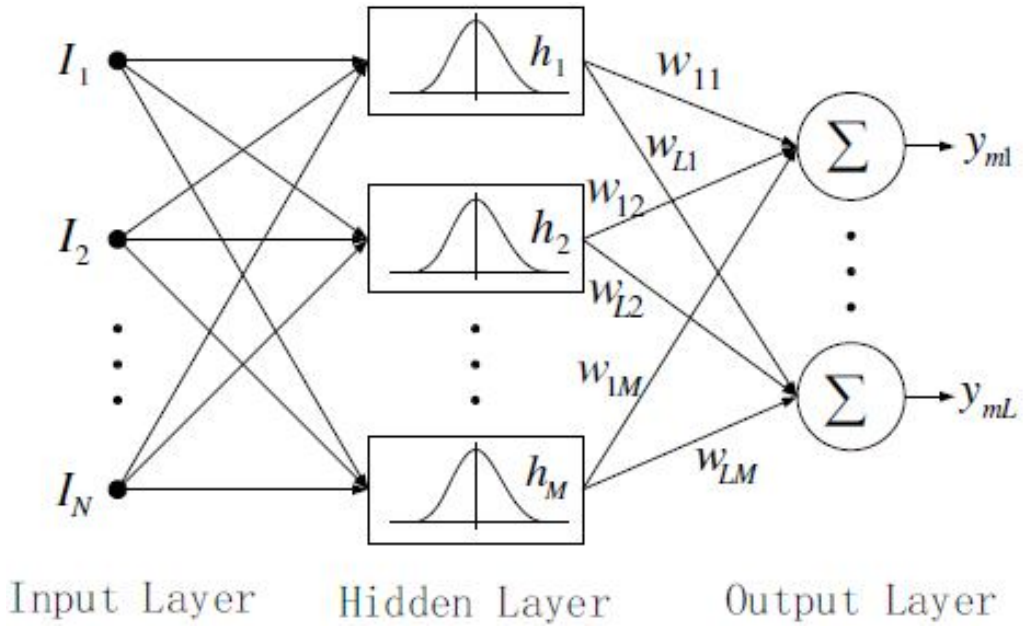
$$\dot{x}_1(t) = \frac{UA}{\rho C_p V_T} x_1(t) - \frac{1}{V_T} x_1(t) u_1(t) - \frac{\lambda}{\rho C_p V_T} u_2(t) + \frac{1}{\rho C_p V_T} v_1(t) + \frac{UA}{\rho C_p V_T} v_2(t) + \frac{1}{V_T} u_1(t) v_2(t)$$

$$\dot{x}_2(t) = \frac{\beta_T}{\rho V_H} x_2(t) + \frac{1}{\rho V_H} u_2(t) + \frac{\alpha}{\lambda \rho V_H} v_1(t) - \frac{1}{\rho V_H} x_2(t) u_1(t) + \frac{1}{\rho V_H} u_1(t) v_3(t)$$

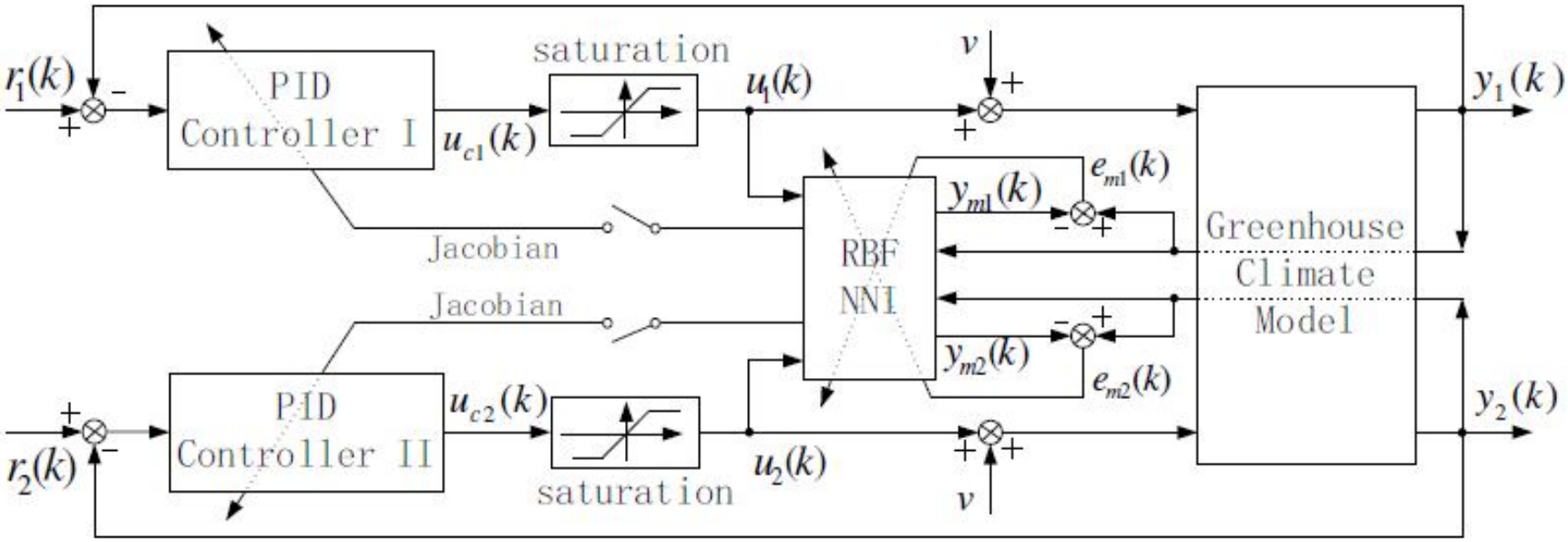
Design of Adaptive Neuro-PID Controller for Greenhouse Climate:

RBF Network Structure:

- RBF neural networks have an input layer, a hidden layer and an output layer.
- The neurons in the hidden layer contain Gaussian transfer functions whose outputs are inversely proportional to the distance from the center of the neuron.



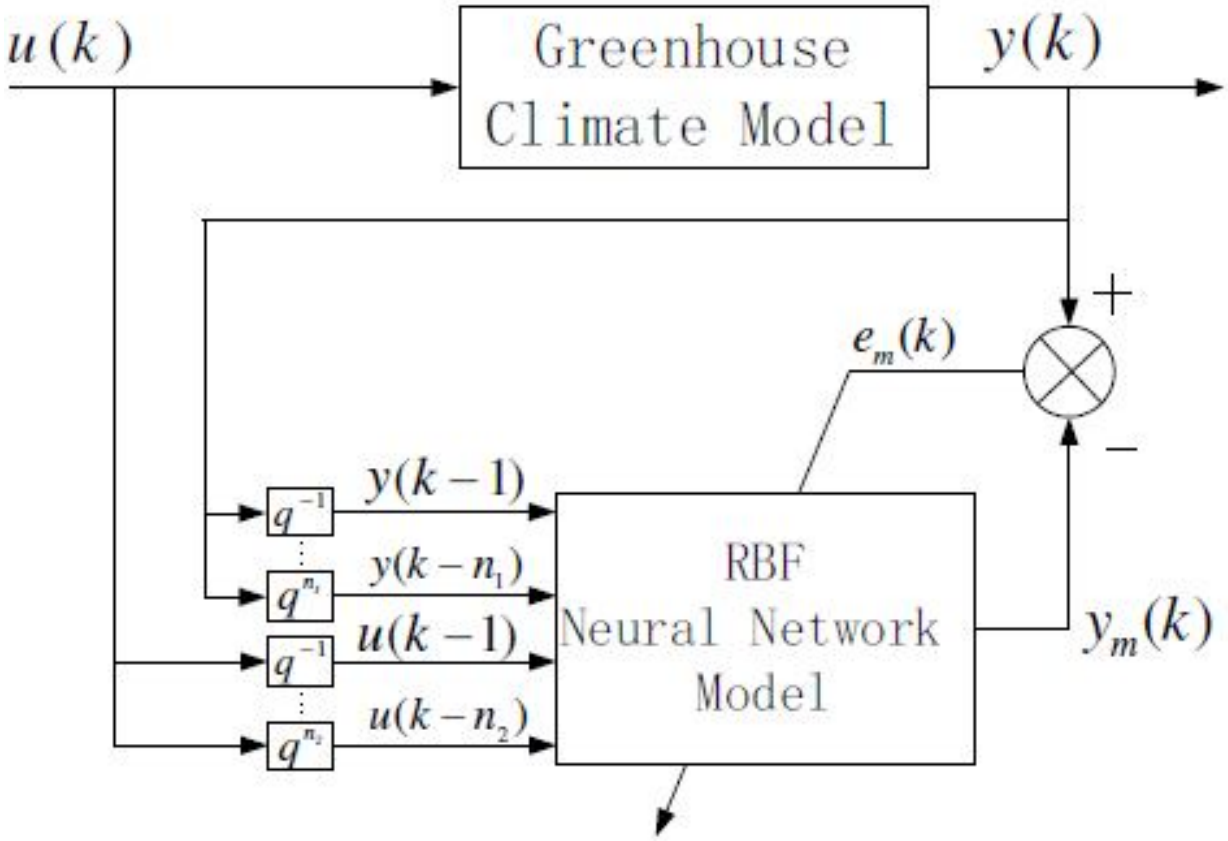
Hybrid Control Strategy:



- The greenhouse dynamic system is a two-input and two-output continuous time nonlinear system. We adopt a hybrid control strategy by combining RBF network with the conventional PID controller, in which there is one neural network model with two outputs (namely, inside temperature and humidity).
- RBF network is used to tune the parameters of the conventional PID controller through Jacobian information.

Greenhouse Climate Model:

The RBF neural network has been used to obtain a model for the plant because it has excellent adaptability, learning ability and powerful ability to approximate nonlinear functions.



Remarks:

The results given in the paper outline that the proposed adaptive hybrid control scheme by combining RBF network with a conventional PID controller is a promising method with the following features;

- It has a satisfactory control performance characterized by adaptability, robustness and good set-point tracking.
- It can achieve the real-time online control of various MIMO systems.
- It can be applied in nonlinear dynamic control systems such as greenhouse climate system.
- The approach is not limited to greenhouse applications and could easily be extended to other applications.

References:

1. Eric A. Wan, "Control Systems: Classical, Neural and Fuzzy", Oregon Graduate Institute, Lecture Notes, 1998.
2. Saerens, M., Soquet, A., 'Neural-controller based on back-propagation algorithm', IEE Proceedings, Part F, Vol. 138, No 1, 1991, pp. 55-62.
3. Moonyong Lee and Sunwon Park, "Process Control Using a Neural Network Combined with the Conventional PID Controllers", ICASE: The Institute of Control, Automation and Systems Engineers, KOREA Vol. 2, No. 3, September, 2000.
4. *S. Pezeshki1, S. Badalkhani and A. Javadi, "Performance Analysis of a Neuro-PID Controller Applied to a Robot Manipulator", Int J Adv Robotic Sy, 2012, Vol. 9.*
5. FRANCKLIN RIVAS-ECHEVERRÍA, ADDISON RÍOS-BOLÍVAR, JEANETTE CASALES-ECHEVERRÍA, "Neural Network-based Auto-Tuning for PID Controllers", http://www.wseas.us/e-library/conferences/crete2001/papers/658.pdf
6. Jun Kanga,b, Wenjun Menga, , Ajith Abrahamc,d,e, Hongbo Liuc,e,, "An Adaptive PID Neural Network for Complex Nonlinear System Control", http://www.softcomputing.net/neucom13_2.pdf
7. *S. Zeng, H. Hu, L, Xu & G. Li , " Nonlinear Adaptive PID Control for Greenhouse Environment Based on RBF Network", Sensors 2012, 12, pp.5328-5348.*

UNIT– II

Artificial Neural Networks and Applications

- Different artificial neural network models,
- learning in artificial neural networks,
- neural network applications in control systems.

Different artificial neural network models

There are various Artificial Neural Network Model. Main ones are

- **Multilayer Perceptron –** It is a feedforward artificial neural network model. It maps sets of input data onto a set of appropriate outputs.
- **Radial Basis Function Network –** A radial basis function network is an artificial neural network. It uses radial basis functions as ***activation functions.***

Both of the above are being **supervised learning** networks used with 1 or more dependent variables at the output.

- **The Kohonen Network –** It is an **unsupervised learning** network used for *clustering.*

Multilayer Perceptron

As we saw above, A multilayer perceptron is a feedforward artificial neural network model. It maps sets of input data onto a set of appropriate outputs. In feed-forward neural networks, the movement is only possible in the forward direction.
An **MLP** consists of many layers of nodes in a directed graph, with each layer connected to the next one. Each neuron is a linear equation like linear regression as shown in the following equation

$$y_i = w_0 + w_1 * x_1 + w_2 * x_2 + \ldots + w_n *$$

The equation is the transfer function in a neural network. This linear weight sum would be a threshold at some value so that output of neuron would be either 1 or 0.
The multilayer perceptron networks are suitable for the discovery of complex nonlinear models. On the possibility of approximating any regular function with a sum of sigmoid its power based.
MLP utilizes a supervised learning technique called **backpropagation** for training the network. This requires a known, desired output for each input value to calculate the loss function gradient.
MLP is a modification of the standard linear perceptron and can distinguish data that are not linearly separable.

Do you know about ANN Applications

2.2. Radial Basis Function Network

A **Radial Basis Function (RBF) network** is a supervised learning network like MLP which it resembles in some ways. But, RBF network works with only one hidden layer. It accomplishes this by calculating the value of each unit in the hidden layer for an observation. It uses the distance in space between this observation and the center of the unit. Instead of the sum of the weighted values of the units of the preceding level.
Unlike the weights of a multilayer perceptron. The centers of the hidden layer of an RBF network are not adjusted at each iteration during learning.
In RBF network, hidden neurons share the space and are virtually independent of each other. This makes for faster convergence of RBF networks in the learning phase, which is one of their strong points.

Let's revise learning rules in Neural Network

Response surface of a unit of the hidden layer of an RBF network is a hypersphere. The response of the unit to an individual (xi) is a decreasing function G of the distance between the individual and its hypersphere.
As this function Γ generally a Gaussian function. The response surface of the unit, after the application of the transfer function, is a Gaussian surface. In other words, it is a 'bell-shaped' surface.
Learning of RBF involves determining the number of units in the hidden layer. Like a number of radial functions, their centers, radii, and coefficients.

2.3. The Kohonen Network

A **self-organizing map (SOM)** is a type of ANN that trained using unsupervised learning. It is also called as self-organizing feature map (SOFM). It produces a low-dimensional discretized representation of the input space of the training samples called a map.

Have a look at SVM in ML
The Finnish professor Teuvo Kohonen describes the model first as an ANN and it is sometimes called a Kohonen map or network.
The Kohonen network is the most common unsupervised learning network. It is also called self-adaptive or self-organizing network because of it 'self-organizes' input data.
Like any neural network, it is being made up of layers of units and connections between these units. The major difference from the rest of neural networks is that there is no variable that can predict.
The purpose of the network is to 'learn' the structure of the data so that it can distinguish clusters in them. By the following two levels the Kohonen network composed.

- **Input layer** with a unit for each of the n variables used in clustering
- **Output layer –** Its units are generally in a square or rectangular grid of l*m units. Each of these l*m units is connected to each of n units of the input layer.

The key related to Kohonen networks:

- For each individual, only one output unit (the 'winner') is activated – The Kohonen Net has a competitive layer of neurons. There is also an input layer. The input layer is fully connected to the competitive layer. The units in the competitive layer sum their weighted inputs to find a single winner.
- It adjusts the weight of the winner and its neighbors.
- The adjustment is such that two close placed output units correspond to two close placed individuals.
- At the output Groups (clusters) of units forms.

Do you know about Kernel Functions

In the application phase, Kohonen network operates by representing each input individual by the unit of the network. The network which is closest to it in terms of a distance defined above. This unit will be the cluster of the individual.

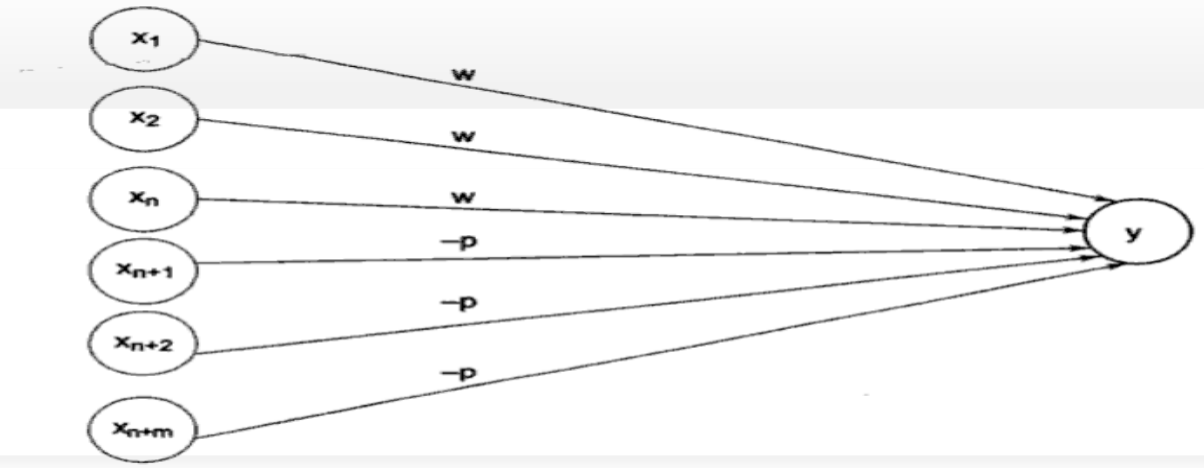
McCulloch Pitts Neuron Model

Warren McCulloch and Walter Pits presented the first mathematical model of a single idealized biological neuron in 1943. This model is quite simple because it does not require learning or adoption. The neurons are binary activated. If the neuron fires, it has an activation of 1, if the neuron does not fire, it has an activation of 0. The neurons are connected by directed weighted paths. The connection paths maybe excitatory or inhibitory. Excitatory connection has positive weights, and inhibitory connection has negative weights. All the excitatory connections in a particular neuron have the same weight. Each neuron has a fixed threshold such that if the network input to the neuron is greater than the threshold the neuron should fire. The threshold is set such that the inhibition is absolute. This means any non-zero inhibitory input will prevent the neuron from firing. it takes one-time step for a signal to pass over one connection link.

The McCulloch Pitts neuron that receives 'n' signals through excitatory connection and 'm' signals through inhibitory connection. The excitatory path has weight w>0 and the inhibitory connection path has weight '-p'. The activation function

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{-in} \geq \theta \\ 0 & \text{if } y_{-in} < \theta \end{cases}$$

Architecture of McCulloch Pitts Model

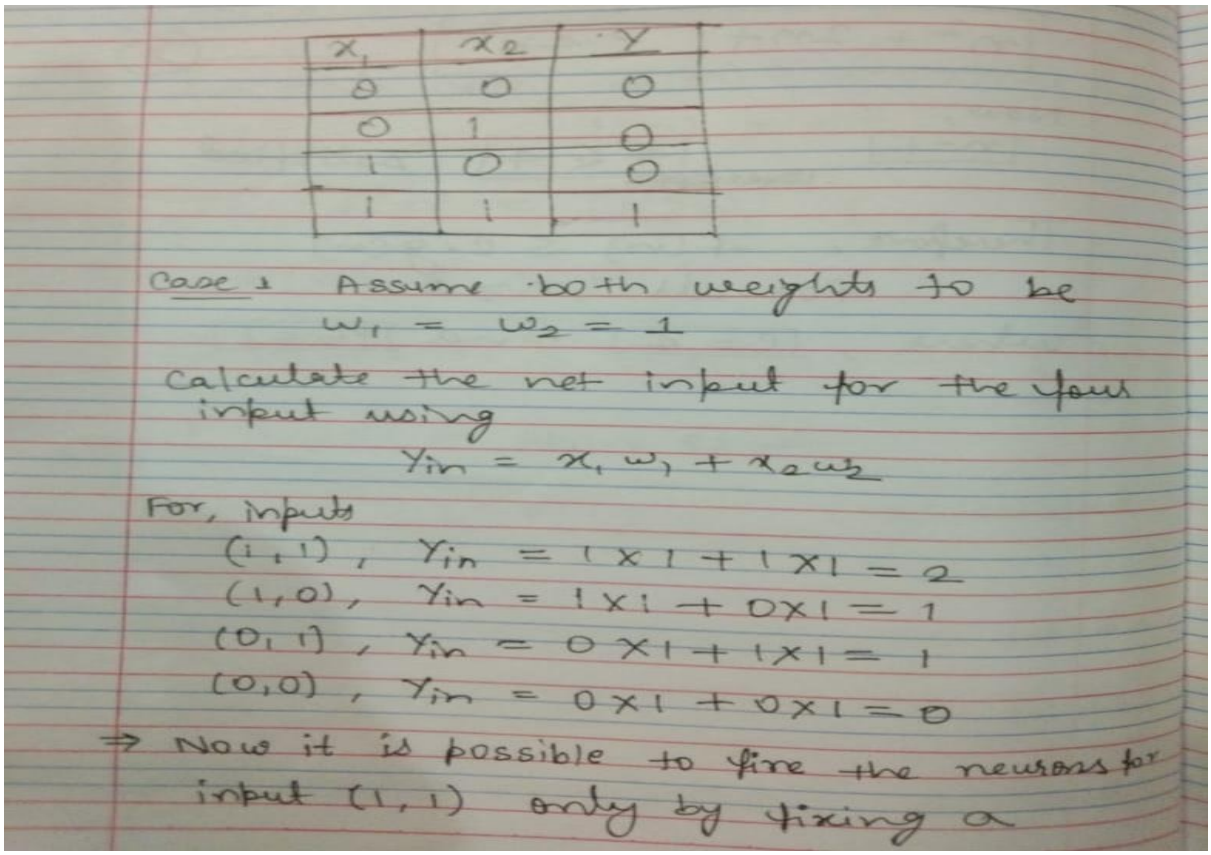


- 'Y' is the McCulloch Pitts neuron, it can receive signal from any

number of other neurons.

- The connection weight from x_1, x_2, …x_n are excitatory.
- The connection weight from x_{n+1}…x_{n+m} are inhibitory

Realize the AND function using McCulloch Pitts neuron



x_1	x_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Case 1: Assume both weights to be $w_1 = w_2 = 1$

Calculate the net input for the four input using

$$Y_{in} = x_1 w_1 + x_2 w_2$$

For, inputs

(1,1), $Y_{in} = 1 \times 1 + 1 \times 1 = 2$

(1,0), $Y_{in} = 1 \times 1 + 0 \times 1 = 1$

(0,1), $Y_{in} = 0 \times 1 + 1 \times 1 = 1$

(0,0), $Y_{in} = 0 \times 1 + 0 \times 1 = 0$

⇒ Now it is possible to fire the neurons for input (1,1) only by fixing a

x_1	x_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Case 1 : Assume both weights to be
$w_1 = w_2 = 1$

Calculate the net input for the four input using

$$Y_{in} = x_1 w_1 + x_2 w_2$$

For, inputs

(1,1), $Y_{in} = 1 \times 1 + 1 \times 1 = 2$
(1,0), $Y_{in} = 1 \times 1 + 0 \times 1 = 1$
(0,1), $Y_{in} = 0 \times 1 + 1 \times 1 = 1$
(0,0), $Y_{in} = 0 \times 1 + 0 \times 1 = 0$

⇒ Now it is possible to fire the neurons for input (1,1) only by fixing a

Realize the ANDNOT function using McCulloch Pitts neuron.

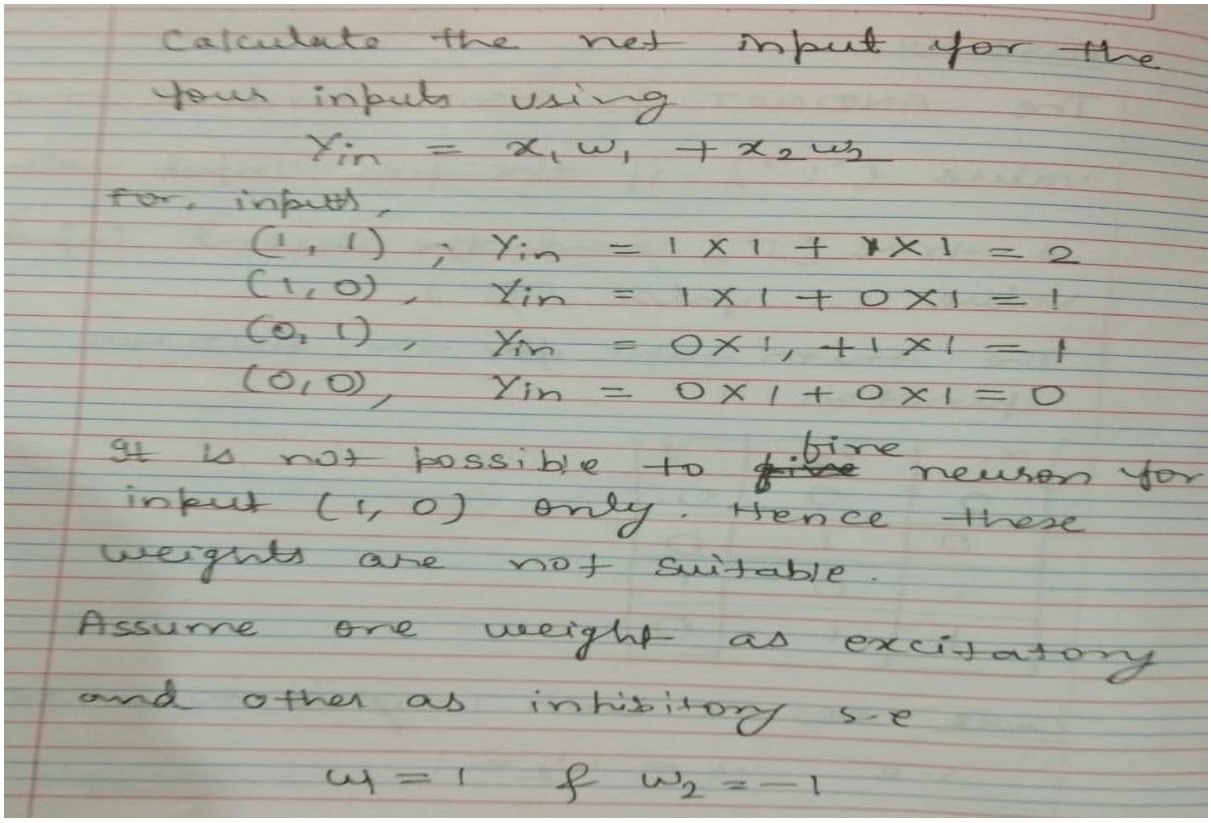
The ANDNOT function returns a true value ('1') if the first input value is true ('1') and the second input value is false ('0').

The truth table for the AND NOT function is,

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

Case 1 : Assume both weights to be excitatory

i.e $w_1 = w_2 = 1$



Learning

i. Learning is relatively permanent change in behavior brought out by experience.

ii. The occurrence of learning can be observed only by observing the performance.

iii. Given a set of input output pairs, a neural network is made to learn by learning the relationship of transforming inputs into outputs.

iv. A neuron is an adaptive element, as the weight of a neuron adapts based on the input signals it receives, the output signals it produces and the teacher's response signals.

v. If a neuron adapts its weight based on input and output signals, then it will be an unsupervised learning as there is no teacher's signal

Categories of learning

1. Supervised, reinforcement and unsupervised
2. Off-line and on-line
3. Deterministic, stochastic and **fuzzy**
4. Discrete and continuous

Supervised Vs. Unsupervised learning

In supervised learning the weight adjustment is determined based on the deviation of the desired output from the actual output. Supervised learning may be used for ***structural learning or for temporal learning.*** Structural learning is concerned with capturing in the weights the relationship between the given input-output pattern pairs. Temporal learning is concerned with

capturing in the weights the relationship between neighboring patterns in a sequence of patterns.

Unsupervised learning discovers features in a given set of patterns, and organizes the patterns accordingly. There is no externally specified desired output in this case. Unsupervised learning uses mostly local information to update the weights. The local information consists of signal or activation values of the units at either end of the connection for which the weight update is being made.

off-line or on-line

In an off-line learning all the given patterns are used together to determine the weights. On the other hand, in an **on-line learning** the information in each new pattern is incorporated into the network by incrementally adjusting the weights. Thus an on-line learning allows the neural network to update the information continuously. However, an off-line learning provides solutions better than an on-line learning since the information is extracted using all the training samples in the case of off-line learning.

Deterministic, stochastic and fuzzy

In practice, the training patterns can be considered as samples of random processes. Accordingly, the activation and output states could also be considered as samples of random processes. Randomness in the output state could also result if the output function is implemented in a probabilistic manner rather than in a deterministic manner. These input, activation and output variables may also be viewed as fuzzy quantities instead of crisp quantities. Thus we can view the learning process as **deterministic or stochastic or fuzzy or a combination of these characteristics.**

Discrete and continuous

Finally, in the implementation of the learning methods the variables may be **discrete or continuous. Likewise, the update of** weight values may be in discrete steps or in continuous time. All these factors influence not only the convergence of weights, but also the ability of the network to learn from the training samples.

Learning Rules

1. Hebbian Learning Rule
2. Perceptron Learning Rule
3. Competitive Learning Rule
4. Memory based Learning
5. Delta Learning Rule (Widrow-Hoff Rule or Least Mean Square (LMS)Rule)

Hebbian Learning Rule

Rule:

This rule, one of the oldest and simplest, was introduced by Donald Hebb in his book *The Organization of Behavior* in 1949. It is a kind of feed-forward, unsupervised learning.

Basic Concept:

This rule is based on a proposal given by Hebb, who wrote −

"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."

From the above postulate, we can conclude that the connections between two neurons might be strengthened if the neurons fire at the same time and might weaken if they fire at different times.

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.

2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

Such a synapse is called a Hebbian synapse. We define a Hebbian synapse as a synapse that uses a time dependent, highly local, and strongly interactive mechanism to increase synaptic efficiency as a function of the correlation between the presynaptic and postsynaptic activities.

The simplest form of Hebbian learning is described by,

$\Delta w = x_i y$

This at Hebbian learning rule represents a purely feed forward, unsupervised learning. It states that if the cross product of output and input is positive, this results in increase of weight, otherwise the weight decreases.

From this definition we may deduce the following four key mechanisms (properties) that characterize a Hebbian synapse

1. **Time-dependent mechanism**: This mechanism refers to the fact that the modifications in a Hebbian synapse depend on the exact time of occurrence of the presynaptic and postsynaptic signals.

2. **Local mechanism:** By its very nature, a synapse is the transmission site where information-bearing signals (representing ongoing activity in the presynaptic and postsynaptic units) are in spatial temporal contiguity. This locally available information is used by a Hebbian synapse to produce a local synaptic modification that is input specific.

3. **Interactive mechanism**: The occurrence of a change in a Hebbian synapse depends on signals on both sides of the synapse. That is, a Hebbian form of learning depends on a "true interaction" between presynaptic and postsynaptic signals in the sense that we cannot make a prediction from either one of these two activities by itself.

4. **Conjunctional or correlational mechanism:** One interpretation of Hebb's postulate of learning is that the condition for a change in synaptic efficiency is the conjunction of presynaptic and postsynaptic signals. Thus, according to this interpretation, the co-occurrence

of presynaptic and postsynaptic signals (within a short interval of time) is sufficient to produce the synaptic modification. It is for this reason that a Hebbian synapse is sometimes referred to as a conjunctional synapse.

Perceptron Learning Rule

For the perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response. This type of learning is supervised.

The perceptron learning rule states that for a finite '*n*' number of input training vectors,

$x(n)$ where $n = 1$ to N

each with an associated target value,

$t(n)$ where $n = 1$ to N

which is +1 or – 1, and an activation function $y - f(y_{-in})$, where,

$$y = \begin{cases} 1 & \text{if } y > \vartheta \\ 0 & \text{if } -\vartheta \le y_{in} \le \vartheta \\ -1 & \text{if } y_{in} < -\vartheta \end{cases}$$

the weight updation is given by

if $y \neq t$, then

$$w_{new} = w_{old} + tx$$

if $y = t$, then there is no change in weights.

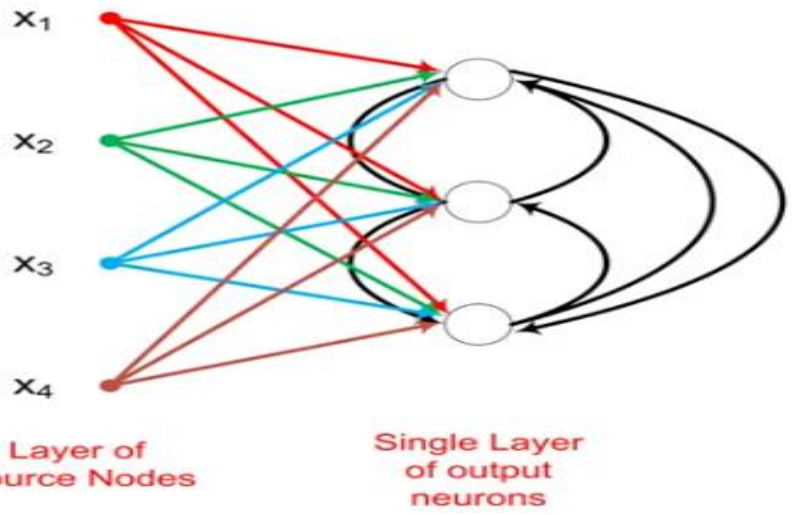
The perceptron learning rule is of central importance for supervised learning of neural networks. The weights can be initialized at any values in this method.

There is a perceptron learning rule convergence theorem which states, "If there is a weight vector w* such that f(x(p) w*) = t(p) for all p, then for any starting vector w_1 the perceptron learning rule will converge to a weight vector that gives the correct response for all training patterns, and this will be done in a finite number of steps".

Competitive Learning

Definition

- The output neurons of a neural network compete among themselves to become active(fired).
- Only one neuron is active at any time which won the competition.
- It is used to classify a set of input patterns.



Basic Elements

- A set of neurons that are all the same except for some randomly distributed synaptic weights and which therefore respond differently to a given set of input patterns.
- A limit imposed on the strength of each neuron.
- A mechanism that permits neurons to compete such that only one neuron is active at a time and it is called winner takes all neurons.

Winning Criterion

- For a neuron k to be the winning neuron, its induced local field for a specified input pattern must be the largest among all the neurons in network

$$y_k = \begin{cases} 1 & v_k > v_j \; for all j; j \neq k \\ 0 & otherwise \end{cases}$$

- Here represents the combined action of all the excitatory (feed forward) and inhibitory (feed back) inputs to k.

This rule is suited for unsupervised network training. The winner-takes-all or the competitive learning is used for learning statistical properties of inputs. This uses the standard Kohonen learning rule.

Let w_{ij} denote the weight of input node j to neuron i. Suppose the neuron has a fixed weight, which are distributed among its input nodes;

$$\sum_j w_{ij} = 1 \quad \text{for all } i.$$

Weight Updation

A neuron then learns by shifting weights from its inactive to active input modes. If a neuron does not respond to a particular input pattern, no learning takes place in that neuron. If a particular neuron wins the competition, its corresponding weights are adjusted.

Weight Updation

- According to standard competitive learning rule, the change in weight is defined as

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) if\ "k"\ wins \\ 0 if\ "k"\ looses \end{cases}$$

where

η =learning rate parameter

x_j =jth node input

w_{kj} =synaptic weight from jth node to kth node

Memory Based Learning

- In this learning, we mimic the way, the brain recollects the information.

Mathematical Expressions

In this Rule, all of the past experiences are explicitly stored in a large memory of correctly classified input-output examples.

$$\{(x_i, d_i)\}_{i=1}^{N}$$

Where

x_i is the input vector and

d_i is corresponding desired response

Example

Binary Pattern Classification

- Two classes e_1 and e_2.
- $d_i = 0$ for class e_1
- $d_i = 1$ for class e_2
- When the classification of a test vector x_{test} is required, the algorithm responds by retrieving and analyzing the training data in a "local neighbourhood" of x_{test}.

Solution

- **All memory based learning algorithms involve two essential ingredient.**
  - Criterion used for defining the local neighbourhood of x_{test}.
  - Leaning rule applied to training examples in local neighbourhood of x_{test}

Nearest Neighbour Rule

- The local neighbourhood is defined as the training example that lies in the immediate neighbourhood of the test vector x_{test}
- The vector $x_N' \in \{x_1, x_2, \ldots x_N\}$ is to be the nearest neighbour of x_{test} if

$$\min_i d(x_i, x_{test}) = d(x_N', x_{test})$$

where $d(x_i, x_{test})$ is the Euclidean distance of x_i and x_{test}

Problems

- Cover and Hart (1967) have studied formally the nearest neighbour rule and based on two assumptions.
  - The classified examples are independently and identically distributed.
  - The sample size N is infinitely large.
- Under these assumptions, The minimum probability of error over all decision rules.

K-Nearest Neighbour Classifier

- Identify the k classified patterns that lie nearest to the test vector x_{test} for some integer k.
- Assign x_{test} to the class hat is the most frequently represented in the k nearest neighbours to x_{test} (use a majority vote for classification)

Delta Rule or Widrow Hoff Rule or LMS or Error Correction Rule

Delta learning rule:

The delta rule in an artificial neural network is a specific kind of backpropagation that assists in refining the machine learning/artificial intelligence network, making associations among input and outputs with different layers of artificial neurons. The Delta rule is also called the Delta learning rule.

Generally, backpropagation has to do with recalculating input weights for artificial neurons utilizing a gradient technique. Delta learning does this by using the difference between a target activation and an obtained activation. By using a linear activation function, network connections are balanced. Another approach to explain the Delta rule is that it uses an error function to perform gradient descent learning.

Delta rule refers to the comparison of actual output with a target output, the technology tries to discover the match, and the program makes changes. The actual execution of the Delta rule will fluctuate as per the network and its composition. Still, by applying a linear activation function, the delta rule can be useful in refining a few sorts of neural networks with specific kinds of backpropagation.

Delta rule is introduced by **Widrow and Hoff**, which is the most significant learning rule that depends on supervised learning.

This rule states that the change in the weight of a node is equivalent to the product of error and the input.

Mathematical equation:

The given equation gives the mathematical equation for delta learning rule:

$\Delta w = \mu.x.z$

$\Delta w = \mu(t-y)x$

Here,

$\Delta \mathbf{w}$ = weight change.

$\boldsymbol{\mu}$ = the constant and positive learning rate.

X = the input value from pre-synaptic neuron.

z= (t-y) is the difference between the desired input t and the actual output y. The above mentioned mathematical rule cab be used only for a single output unit.

The different weights can be determined with respect to these two cases.

Case 1 - When $t \neq k$, then

w(new) = w(old) + Δw

Case 2 - When t = k, then

No change in weight

For a given input vector, we need to compare the output vector, and the final output vector would be the correct answer. If the difference is zero, then no learning takes place, so we need

to adjust the weight to reduce the difference. If the set of input patterns is taken from an independent set, then it uses learn arbitrary connections using the delta learning rule. It has examined for networks with linear activation function with no hidden units. The error squared versus the weight graph is a paraboloid shape in n-space. The proportionality constant is negative, so the graph of such a function is concave upward with the least value. The vertex of the paraboloid represents the point where it decreases the error. The weight vector is comparing this point with the ideal weight vector. We can utilize the delta learning rule with both single output units and numerous output units. When we are applying the delta learning rule is to diminish the difference between the actual and probable output, we find an error.

Hebbian learning rule:

The Hebbian rule was the primary learning rule. In 1949, **Donald Hebb** created this learning algorithm of the unsupervised neural network. We can use this rule to recognize how to improve the weights of nodes of a network.

The Hebb learning rule accepts that if the neighboring neurons are activated and deactivated simultaneously, then the weight associated with these neurons should increase. For neurons working on the contrary stage, the weight between them should diminish. If there is no input signal relationship, the weight should not change.

If inputs of both the nodes are either positive or negative, then a positive weight exists between the nodes. If the input of a node is either positive or negative for others, a solid negative weight exists between the nodes.

In the beginning, the values of all weights are set to zero. This learning rule can be utilized for both easy and hard activation functions. Since desired reactions of neurons are not utilized in the learning process, this is the unsupervised learning rule. The absolute values of the weights are directly proportional to the learning time, which is undesired.

According to the Hebbian learning rule, the formula to increase the weight of connection at each time frame is given below.

$$\Delta\omega ij(t) = \alpha pi(t)*qj(t)$$

Here,

$\Delta\omega ij(t)$ **=** increment by which the connection of the weight increases at the time function t.

α = constant and positive learning rate.

pi(t) = input value from pre-synaptic neuron at function of time t.

qj(t) = output of pre-synaptic neurons at the same function of time t.

We know that each association in a neural network has an associated weight, which changes throughout the learning. It is an example of supervised learning, and the network begins its learning by assigning a random variable to each weight. Here we can evaluate the output value on the basis of a set of records for which we can know the predicted output value. **Rosenblatt**

introduces this rule. This learning rule is an example of supervised training in which the learning rule is given with the set of examples of proper network behavior:

{X1,t1} , {x2,t2},…,{xq,tq}

Where,

Xq = input to the network.

tq = target output.

As each input is given to the network, the network output is compared with the objective of the network. Afterward, the learning rule changes the weights and biases the network in order to move the network output closer to the objective.