KON 426E
INTELLIGENT CONTROL SYSTEMS

LECTURE 7

04/04/2022

Neural Networks in Control

Inverse Learning (Generalized Learning)

Consider the nonlinear plant: $\vec{x}(k+1) = \vec{f}(\vec{x}(k), \vec{u}(k))$

(Assuming the order of the plant is known and all state variables are measurable.)

$$\vec{x}(k+2) = \vec{f}(\vec{x}(k+1), \vec{u}(k+1)) = \vec{f}(\vec{f}(\vec{x}(k), \vec{u}(k)), \vec{u}(k+1))$$

.....

$$\vec{x}(k+n) = \vec{F}(\vec{x}(k), \vec{U})$$

where $\vec{U} = [\vec{u}(k)\, \vec{u}(k+1), \ldots \ldots \vec{u}(k+n)]$

Assume the inverse dynamics of the plant exists, then:

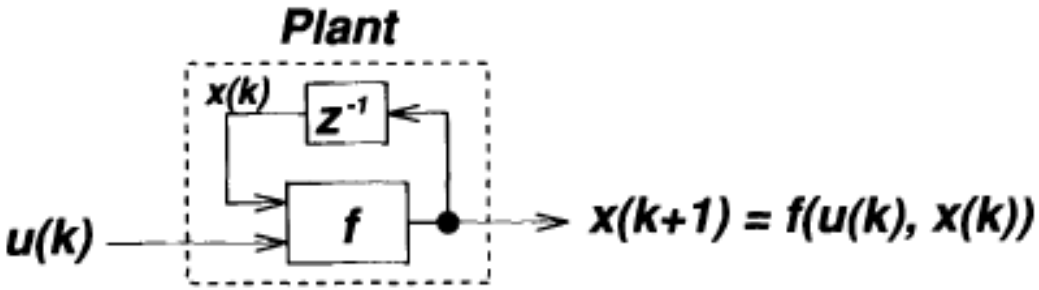
$$\vec{U} = \vec{G}(\vec{x}(k), \vec{x}(k+n))$$

This means that there is a unique input sequence $\vec{U}$ specified by mapping $\vec{G}$ that can drive the plant from state $\vec{x}(k)$ to $\vec{x}(k+n)$ in *n* time steps. How do we find this $\vec{G}$?

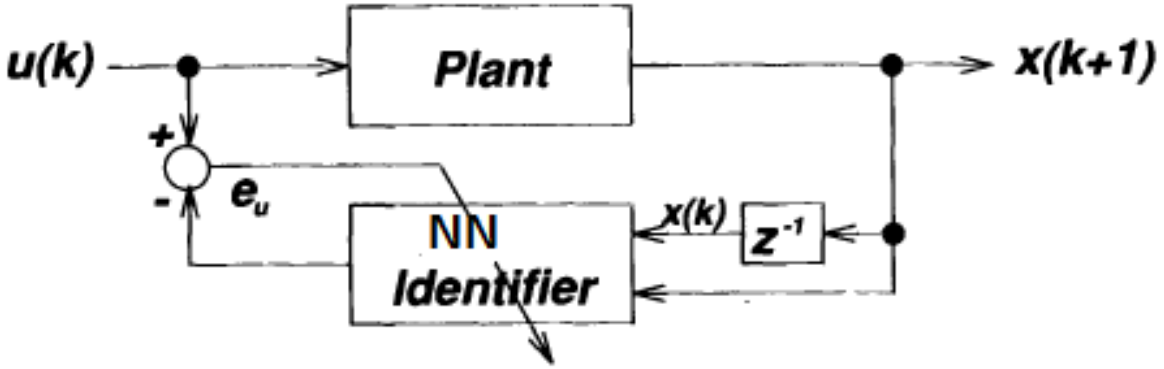
$\vec{G}$ exists by assumption but it does not always have an analytic solution.

We will estimate $\hat{U}$ by using a NN: $\hat{U} = \hat{G}(\vec{x}(k), \vec{x}_d(k+n))$

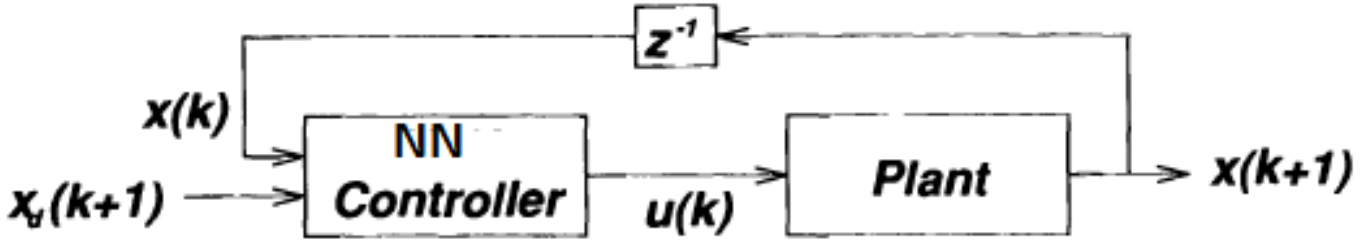
Assume n=1



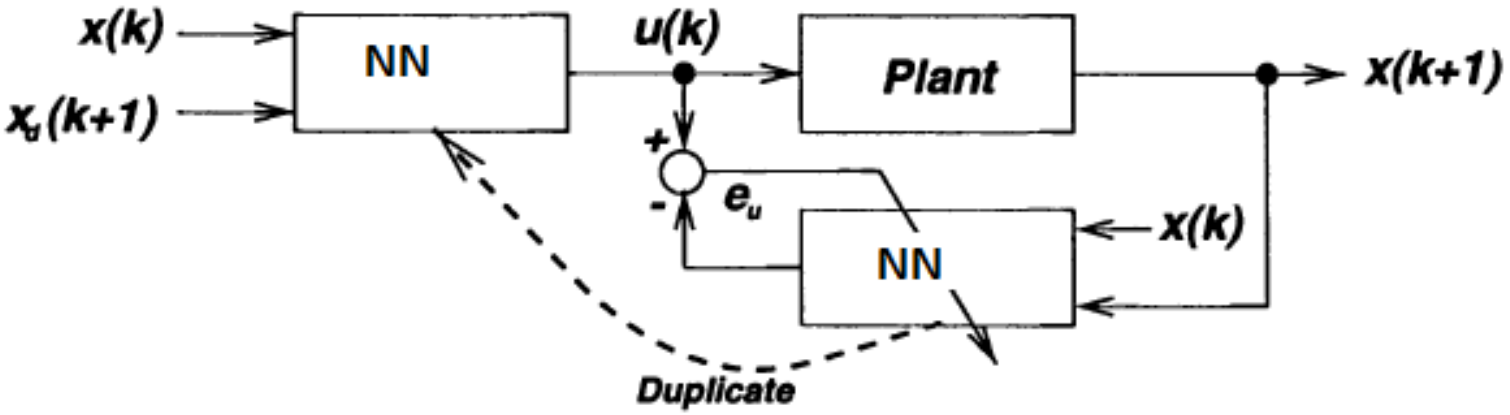
Training Phase



Application Phase



Combination of training and application phases for on-line inverse learning

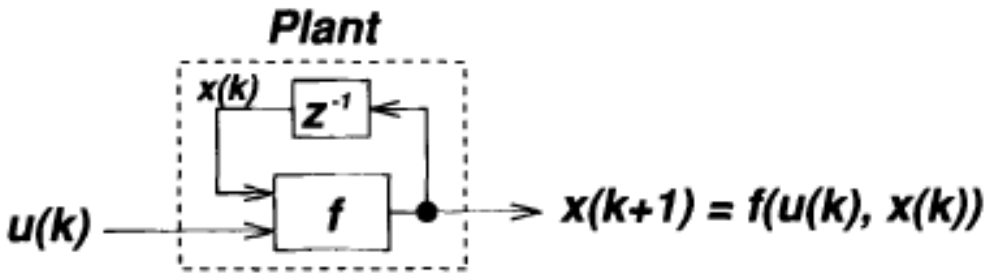


Problems Associated with Inverse Learning

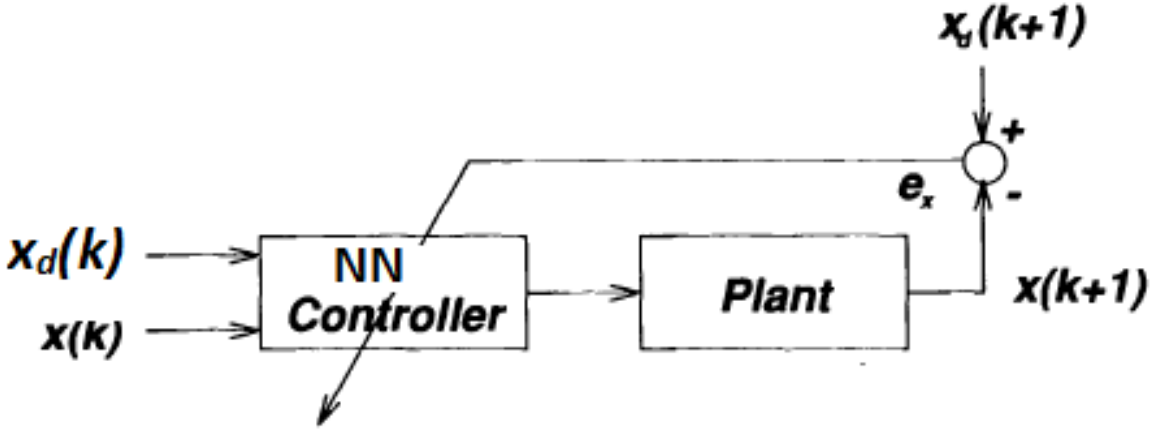
- An inverse model does not always exist
- Inverse learning is an indirect approach.
- It tries to minimize **NN output error** instead of the **overall system error**.
- NN output error: $\|U - \hat{U}\|^2$
- System error: $\|\vec{x}_d(k) - \vec{x}(k)\|^2$ (difference between desired and actual trajectories)
- Minimization of the NN error $\|U - \hat{U}\|^2$ does not guarantee the minimization of overall system error $\|\vec{x}_d(k) - \vec{x}(k)\|^2$

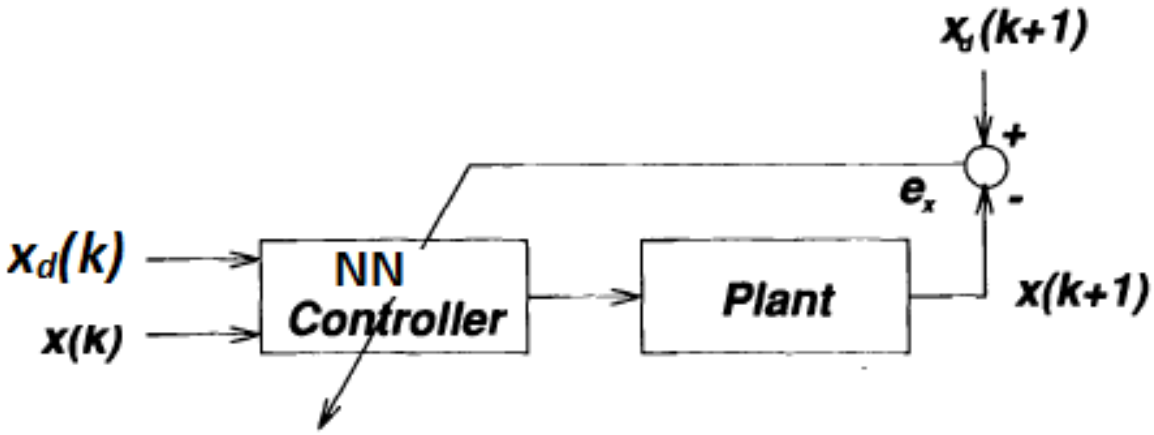
Specialized Learning

The plant



Specialized Learning





Plant $$\vec{x}(k+1) = \vec{f}\left(\vec{x}(k), \vec{u}(k)\right)$$

NN output $$\hat{u}(k) = \vec{F}(\vec{x}_d(k), \vec{x}(k), \vec{w})$$

Set $$\vec{u}(k) = \hat{u}(k)$$

$$\vec{x}(k+1) = \vec{f}\left(\vec{x}(k), \vec{F}(\vec{x}_d(k), \vec{x}(k), \vec{w})\right)$$

Minimize $$E(\vec{w}) = \sum_k \left\| \vec{x}_d(k+1) - \vec{f}\left(\vec{x}(k), \vec{F}(\vec{x}_d(k), \vec{x}(k), \vec{w})\right) \right\|$$

Problems Associated with Specialized Learning

In order to update the weights, you have to compute:

$$\frac{\partial E}{\partial \vec{w}} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial \vec{x}}\frac{\partial \vec{x}}{\partial u}\frac{\partial u}{\partial \vec{w}}$$

You can easily compute $\frac{\partial E}{\partial e}$, $\frac{\partial e}{\partial \vec{x}}$ and $\frac{\partial u}{\partial \vec{w}}$

However, computation of $\frac{\partial \vec{x}}{\partial u}$ (the **Jacobian**) is not easy.

Jacobian defines the dynamics of the plant, and most of the time you need to use a second NN (or another intelligent structure) to calculate it.

Specialized Learning with Model Referencing

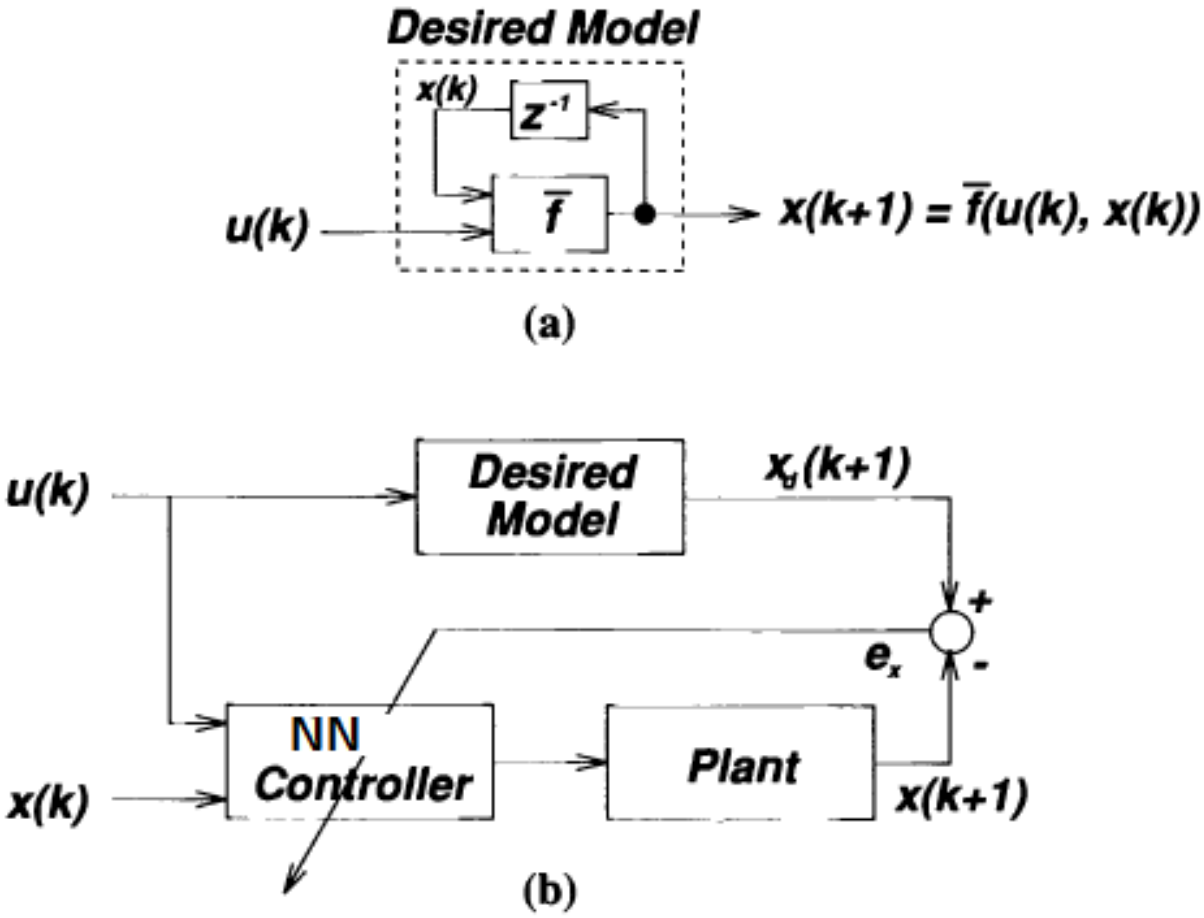


Figure 17.10. *(a) Desired model block; (b) specialized learning with model referencing.*

Different types of modelling for nonlinear dynamical systems

Nonlinear autoregressive exogenous model (NARX)

$$x(k+1) = f(x(k), x(k-1), \ldots\ldots, x(k-n), u(k), u(k-1), \ldots\ldots\ldots u(k-m)\ )$$

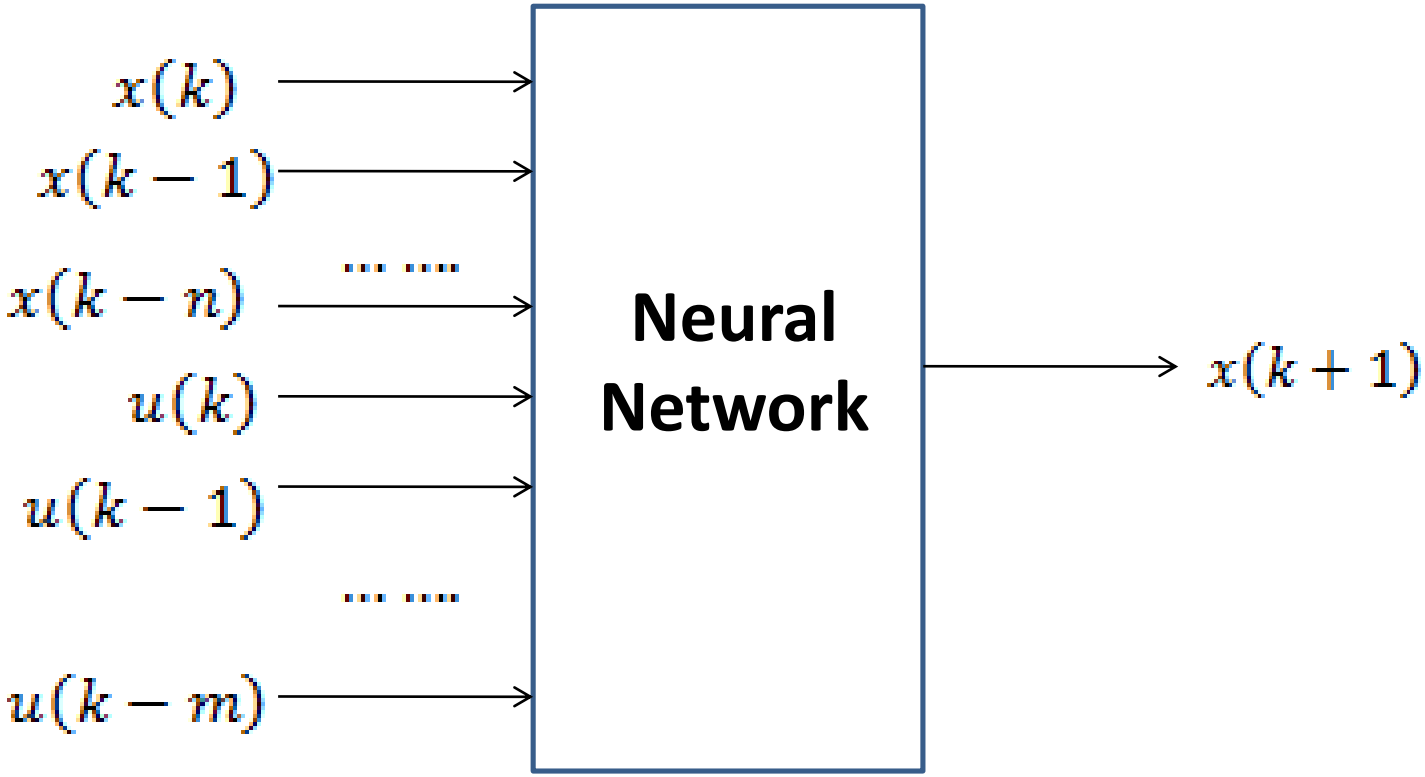
Nonlinear autoregressive moving average (NARMA)

$$x(k+1) = f\big(x(k)\big) + g\big(x(k)\big)u(k)$$

Finding the NARX model of a a system using neural network

$$x(k+1) = f(x(k), x(k-1), \ldots\ldots, x(k-n), u(k), u(k-1), \ldots\ldots\ldots u(k-m))$$

The required NN architecture to identify $x(k+1)$



Self-Tuning Adaptive Control (NARMA-L2)

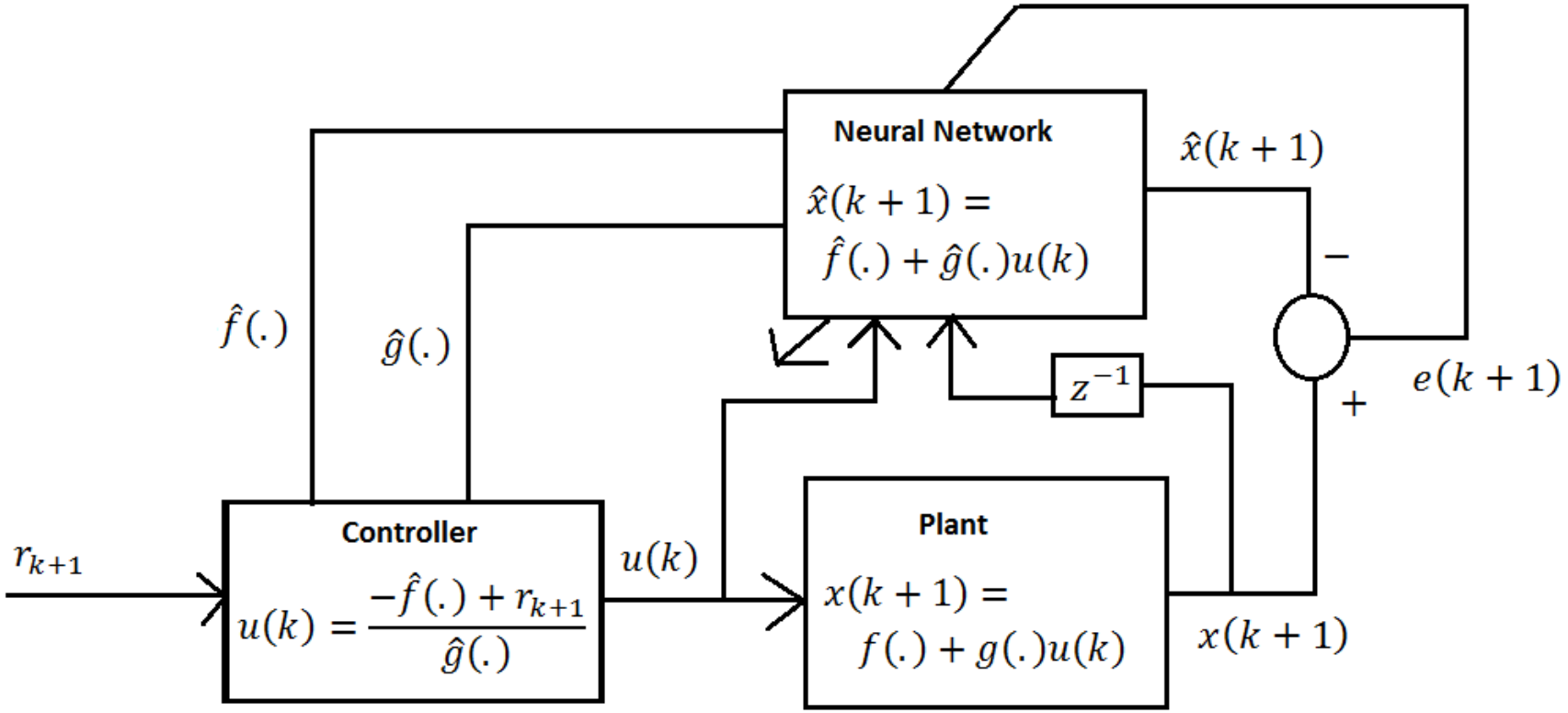
$$x(k+1) = f\big(x(k)\big) + g\big(x(k)\big)u(k)$$

$$u(k) = \frac{-\hat{f}(.) + r_{k+1}}{\hat{g}(.)}$$

$$x(k+1) = f\big(x(k)\big) + g\big(x(k)\big)\frac{-\hat{f}(.) + r_{k+1}}{\hat{g}(.)}$$

$$x(k+1) = r_{k+1}$$

NARMA-L2 Controller



Certainty Equivalence Principle-

Using the estimated variable instead of the real one

Universal Approximation Theorem
(Haykin, 1999, pg. 230-231)

Let $\varphi(.)$ be a non-constant, bounded and monotone- increasing continuous function. Let I_{m_0} denote the m_0- dimensional unit-hypercube $[0,1]^{m_0}$. The space of continuous functions on I_{m_0} is denoted by $C(I_{m_0})$.

Then, given any function $f \ni C(I_{m_0})$ and $\varepsilon > 0$, there exist an ineteger m_1 and sets of real constants α_i , b_i and w_{ij} where $i = 1,2,\ldots..m_1$ and $j = 1,2,\ldots..m_0$ such that we may define:

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi\left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i\right)$$

as an approximate realization of the function $f(.)$ that is:

$$|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \epsilon$$

for all $x_1, x_2, \ldots, x_{m_0}$ that lie in the input space.

The universal approximation theorem is directly applicable to multilayer perceptrons (MLPs).

- The activation function $\varphi(.)$ e.g. $\frac{1}{1+e^{-v}}$ is a non-constant, bounded, monotone-increasing function.
- The network has m_0 input nodes and a single hidden layer consisting of m_1 neurons. The inputs are: $x_1, \ldots\ldots x_{m_0}$
- Hidden neuron i has synaptic weights $w_{i_1}, \ldots\ldots w_{m_0}$ and bias b_i.
- The network output is a linear combination of the outputs of the hidden neurons with $\alpha_1, \alpha_2, \ldots\ldots\ldots \alpha_{m_1}$ defining the synaptic weights of the output layer.

- This is an existence theorem.
- It states that a single hidden layer is sufficient for a multilayer perceptron to compute a uniform approximation to a given training set represented by the set of inputs $x_1, \ldots\ldots x_{m_0}$ and a desired output $f(x_1, \ldots\ldots x_{m_0})$
- **BUT** the theorem **<u>DOES NOT</u>** say that a single hidden layer is optimum in the sense of learning time, ease of implementation or generalization.