Advanced Computational Intelligence for Identification, Control and Optimization of Nonlinear Systems

Ganesh Kumar Venayagamoorthy, PhD

Associate Professor of Electrical and Computer Engineering
& Director of Real-Time Power and Intelligent Systems Laboratory
University of Missouri-Rolla, USA

http://www.umr.edu/~ganeshv
www.ece.umr.edu/RTPIS
gkumar@ieee.org

UMR

Acknowledgment

Support from the National Science Foundation, USA, CAREER Grant ECS # 0348221, University of Missouri Research Board and University of Missouri-Rolla, USA is gratefully acknowledged.

UMR

Intelligence

- Ability to comprehend, to understand and profit from experience, to interpret intelligence, having the capacity for thought and reason (especially, to a higher degree).
- Creativity, skill, consciousness, emotion and intuition.

UMR

Computational Intelligence (1)

- Computational Intelligence (CI) is the study of adaptive mechanisms to enable or facilitate intelligent behavior (intelligence) in complex and changing environments.
- These mechanisms include paradigms (AI) that exhibit an ability to learn or adapt to new situations, to generalize, abstract, discover and associate.
- Turing Test - 1950

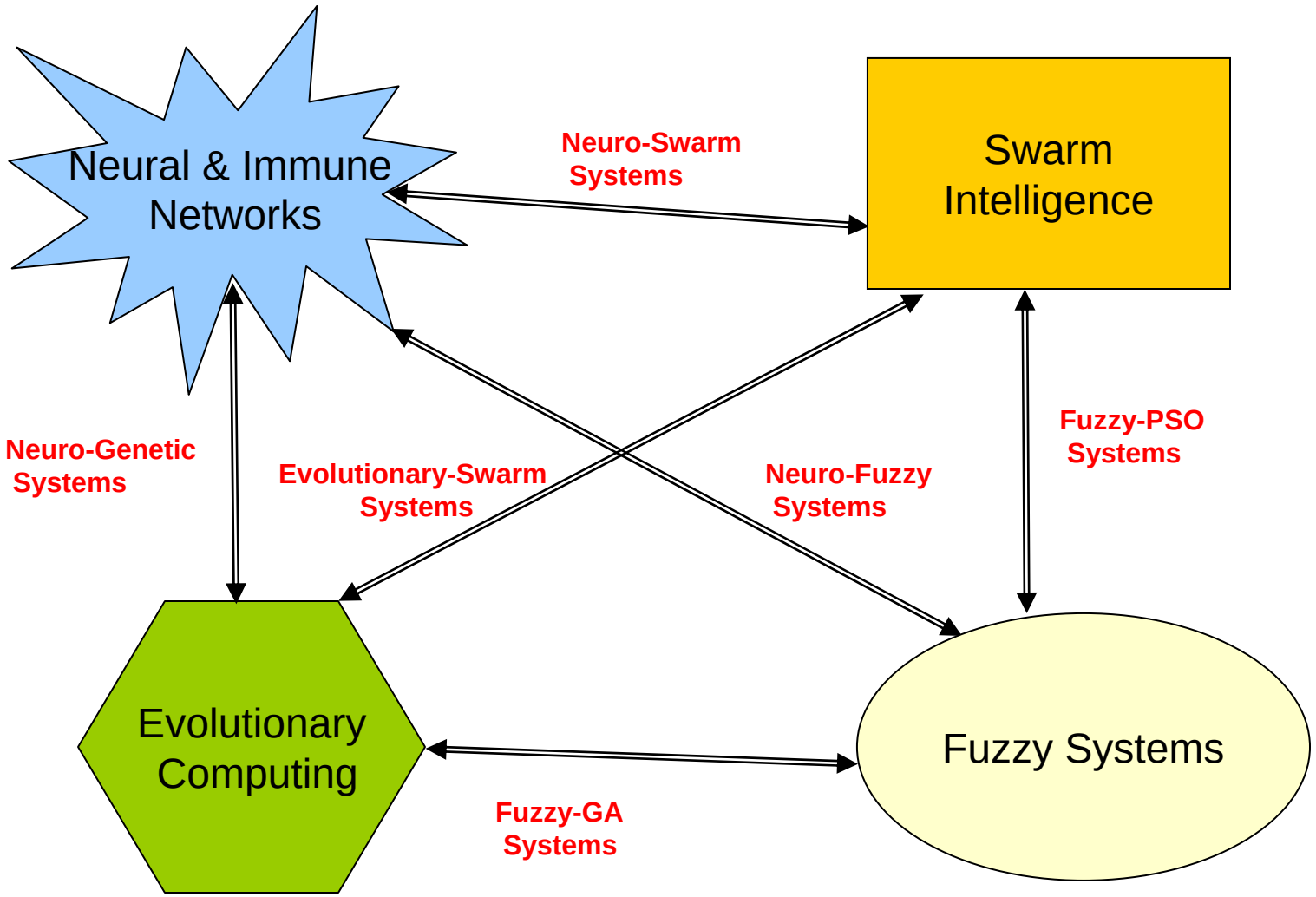
UMR

Computational Intelligence (2)

- Computational Intelligence (CI) can be defined as *the computational models and tools of intelligence capable of inputting raw numerical sensory data directly, processing them by exploiting the representational parallelism and pipelining the problem, generating reliable & timely responses and withstanding high fault tolerance.*

UMR

Computational Intelligence Paradigms



UMR

Neural Networks

- A neural network can be defined as a massively parallel distributed processor made up of simple processing units, which has the natural propensity for strong experiential knowledge and making it available for use.
- The neural network resembles the brain in two aspects –
  - *Knowledge is acquired by the network from its environment through a learning process.*
  - *Interneuron connection strengths, known as synaptic weights, are used to store acquired knowledge.*

UMR

Artificial Immune Systems

- Artificial Immune Systems (AIS) are biologically inspired models for immunization of engineering systems.
- The pioneering task of AIS is to detect and eliminate non-self materials, called antigens such as virus or cancer cells.
- The AIS also plays a great role to maintain its own system against dynamically changing environment.
- The immune systems thus aim at providing a new methodology suitable for dynamic problems dealing with unknown/hostile environments.

UMR

Evolutionary Computing

- Evolutionary Computing has as its objective the model of natural evolution, where the main concept is survival of the fittest: the *weak must die*, *the elites move to the next level*.
- In natural evolution, survival is achieved through reproduction. Offspring, reproduced from two parents, contain genetic material of both parents – hopefully the best characteristics of each parent.
- Those individuals that inherit the bad characteristics are weak and lose the battle to survive.
- In some bird species, one hatchling manages to get more food, gets stronger, and at the end kicks out all its siblings from the nest to die.
- GAs, GP, EP, ES

UMR

Swarm Intelligence

- Swarm intelligence originated from the study of colonies (ants, bees, termites) or swarms of social organisms – flock of birds, school of fish.
- Studies of the social behavior of organisms (individuals) in swarms prompted the design of very efficient optimization and clustering algorithms.
- SI is an innovative distributed intelligent paradigm for solving optimization problems.

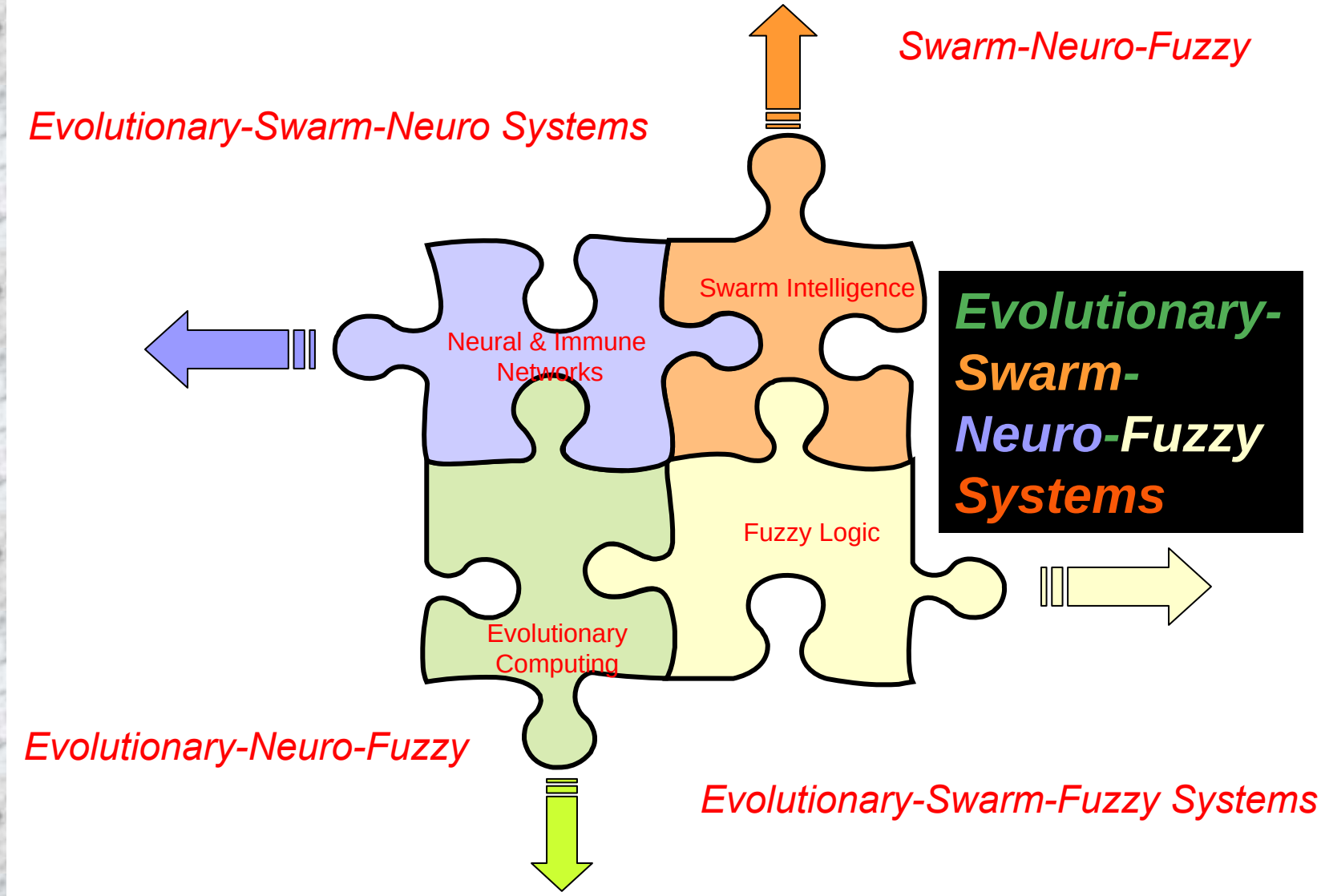
UMR

Fuzzy Systems

- Traditional set theory requires elements to be either part of a set or not. Similarly, binary-valued logic requires the values of parameters to be either 0 or 1, with similar constraints on the outcome of an inferencing process.
- Fuzzy sets and fuzzy logic allow what is referred to as approximate reasoning.
- With fuzzy sets, an element belongs to a set to a certain degree of certainty.
- Fuzzy logic allows reasoning with these uncertain facts to infer new facts, with a degree of certainty associated with each fact.
- In a sense, fuzzy sets and logic allow the modeling of common sense.

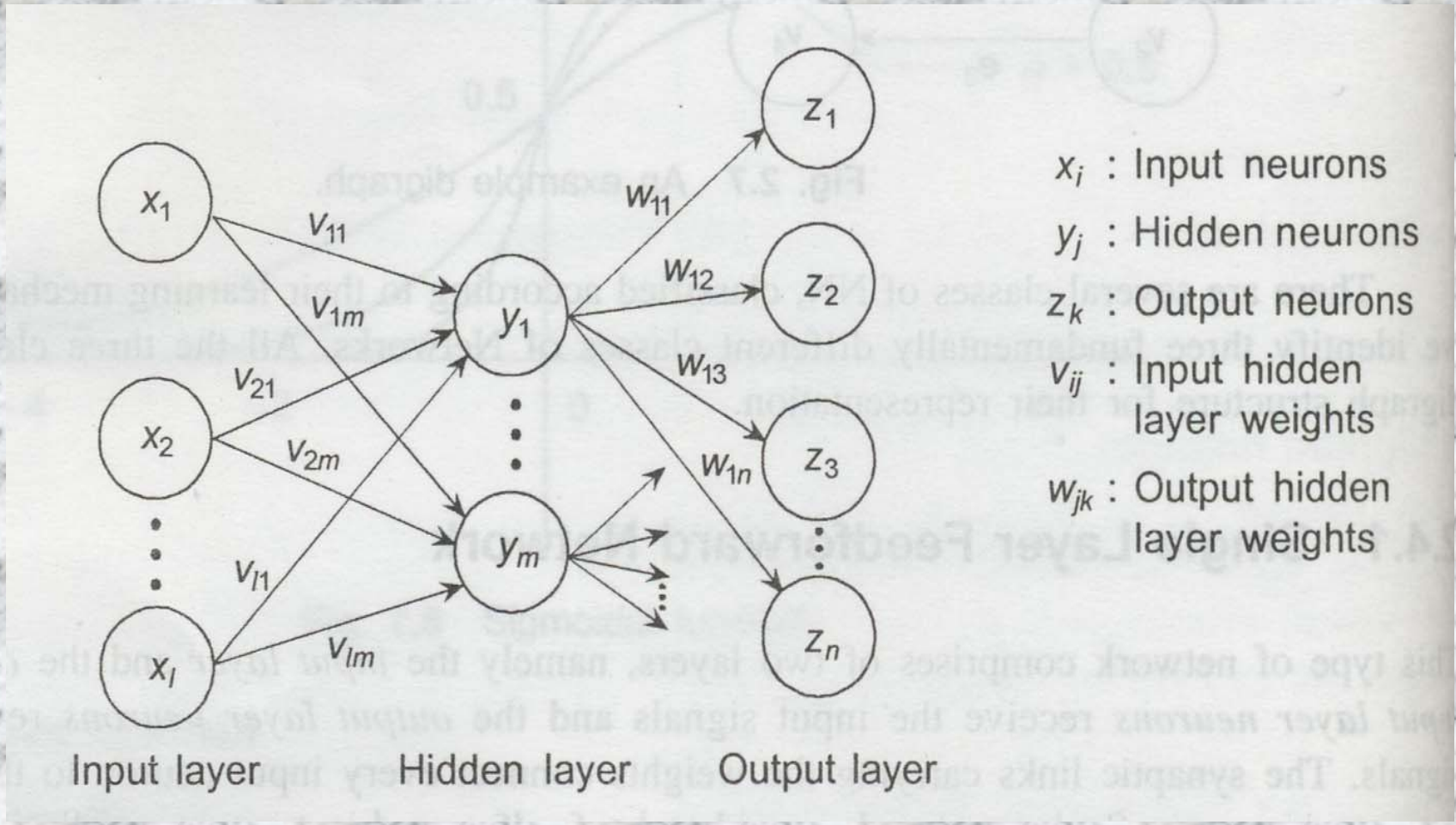
UMR

Computational Intelligence



UMR

Multi-Layer Feedforward Networks



x_i : Input neurons

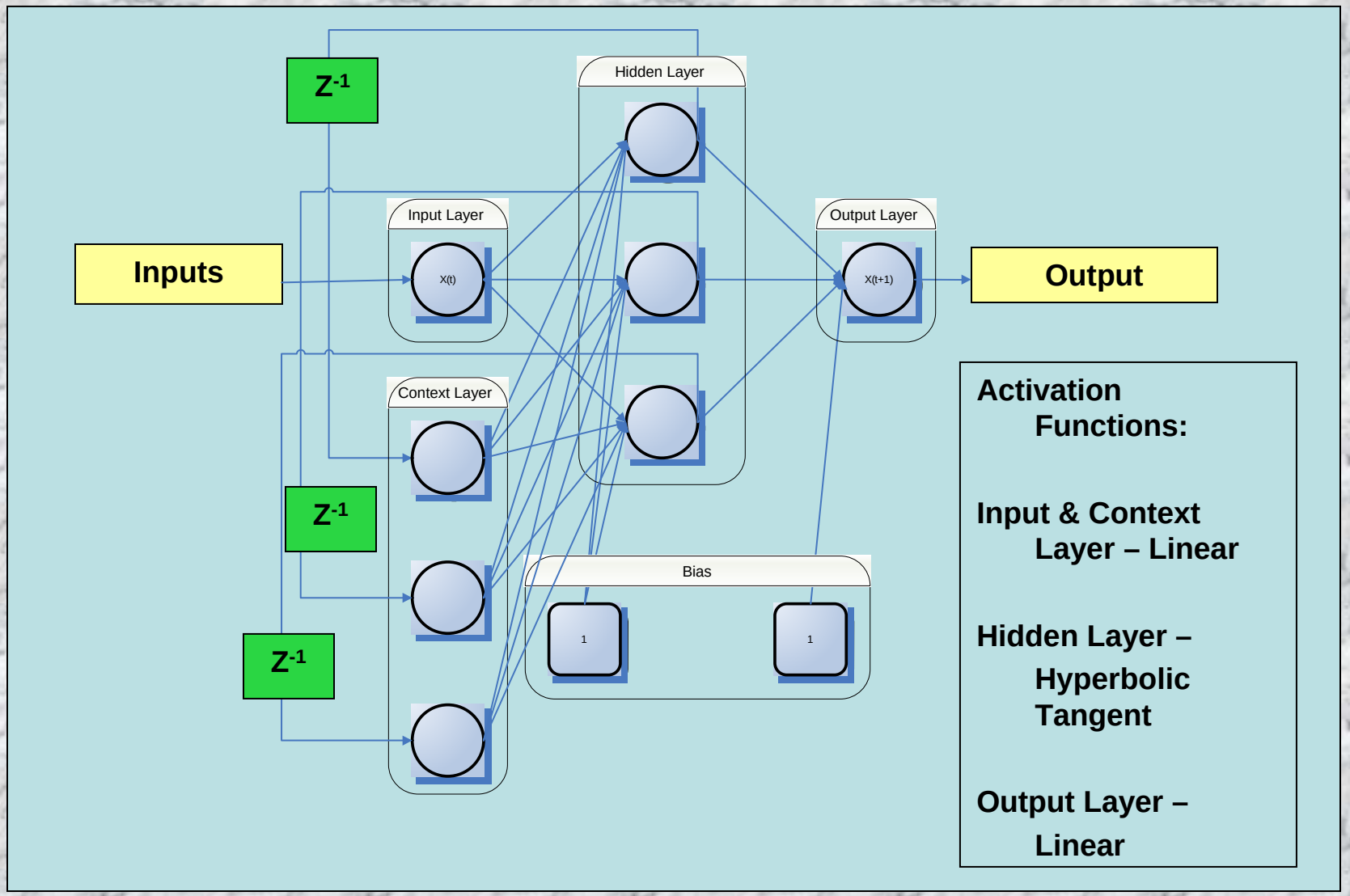
y_j : Hidden neurons

z_k : Output neurons

v_{ij} : Input hidden layer weights

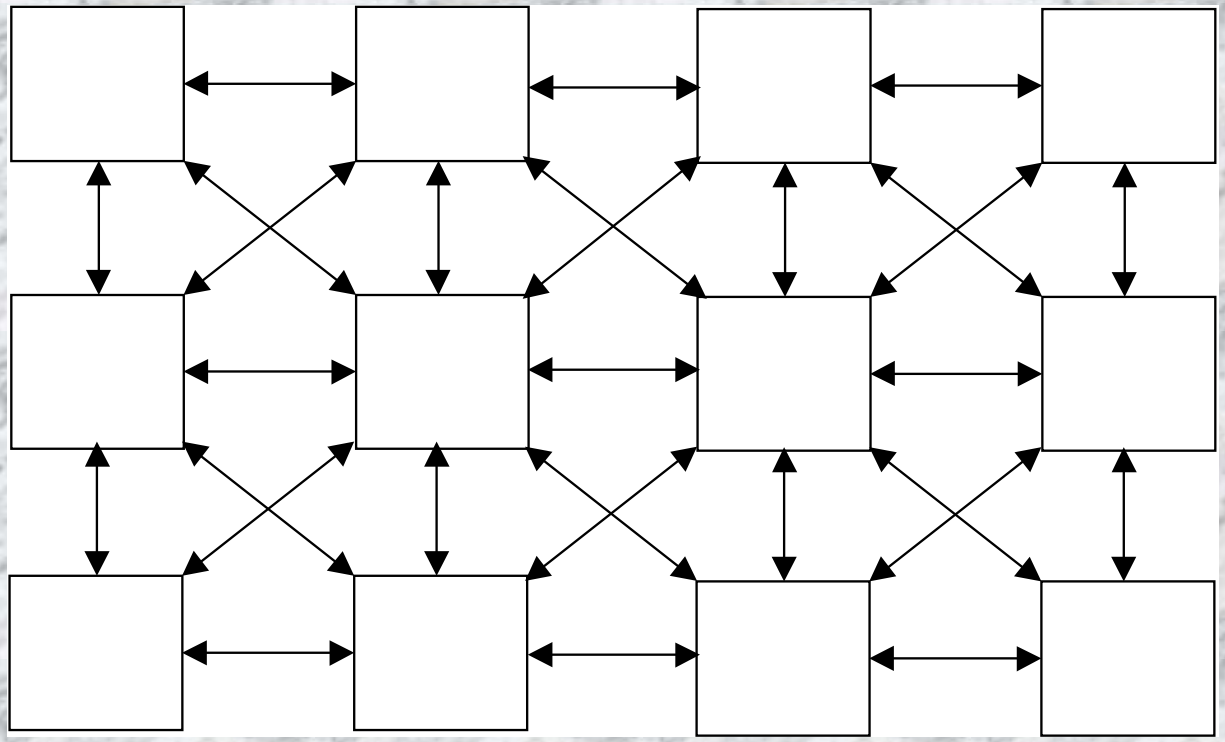
w_{jk} : Output hidden layer weights

Feedback (Recurrent) Networks



UMR

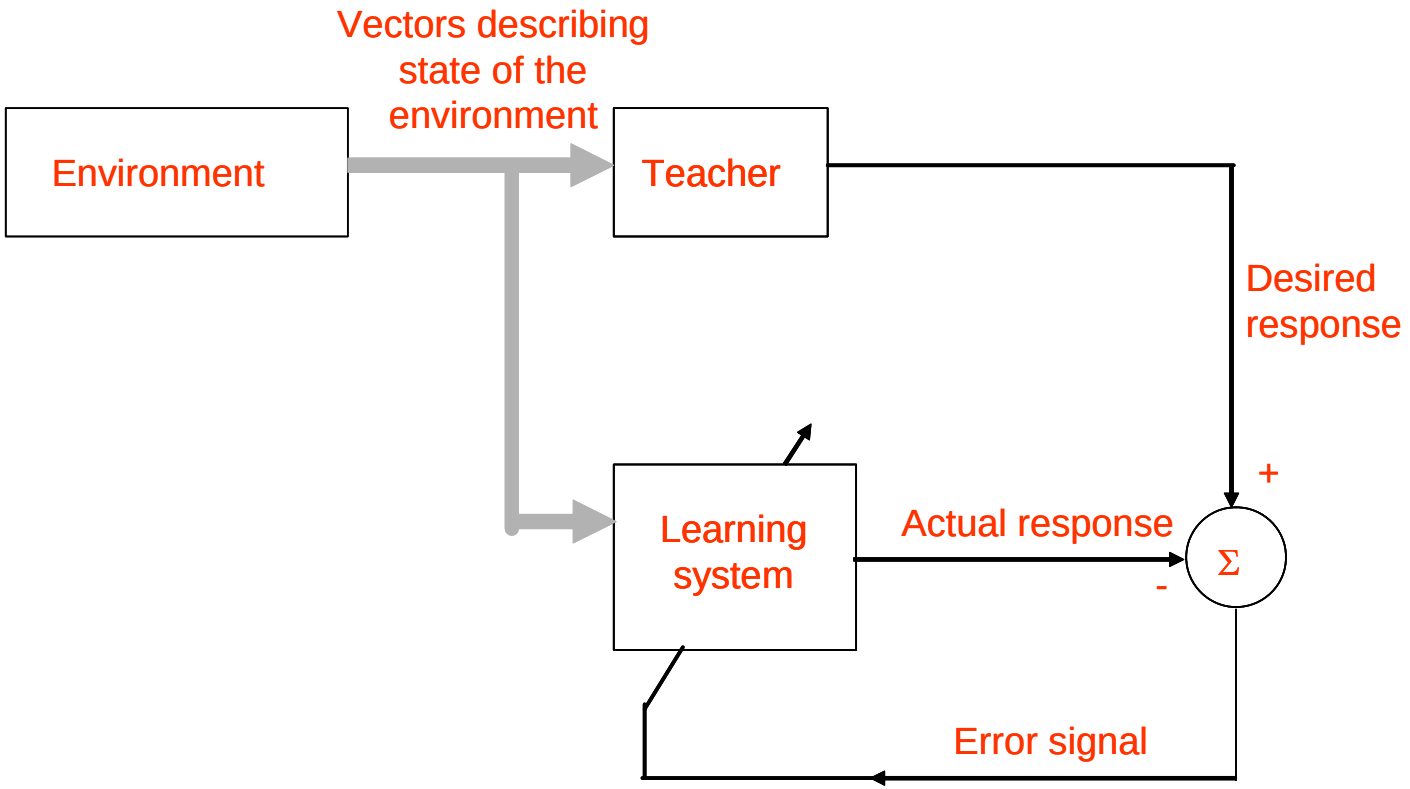
Cellular Architectures



Learning Methods

Learning Methods can be broadly classified into three basic types: supervised, unsupervised, and reinforcement.

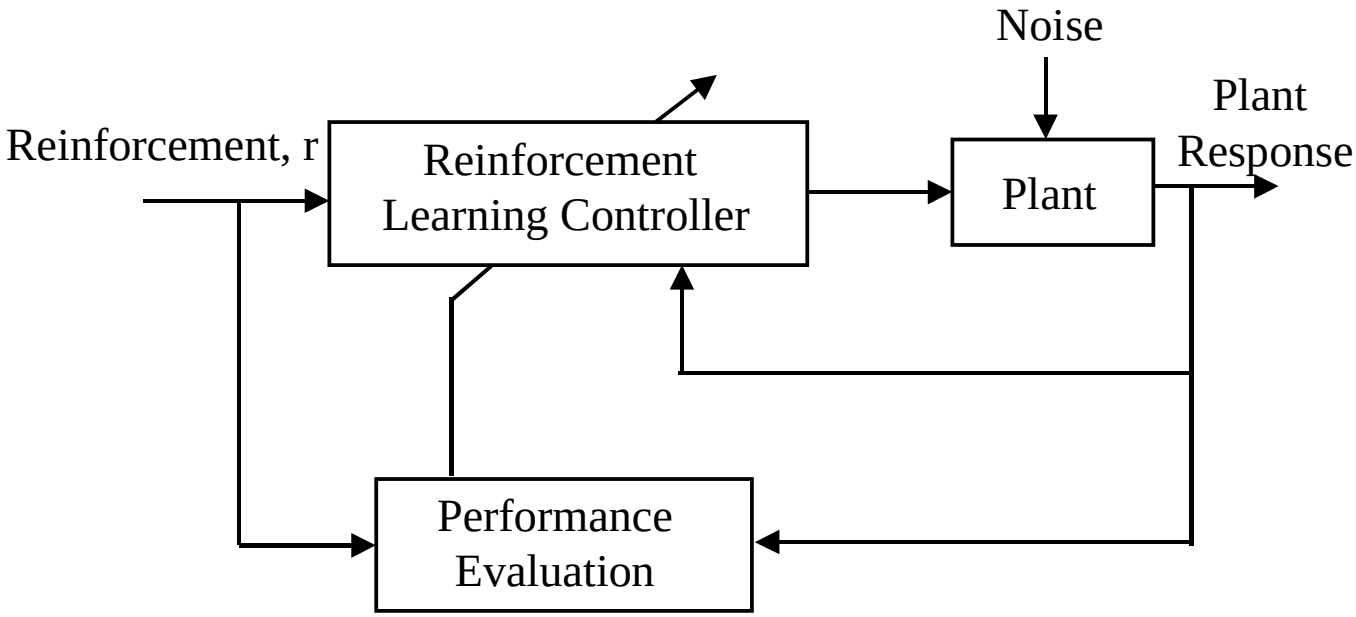
Supervised Learning:-



Learning Methods

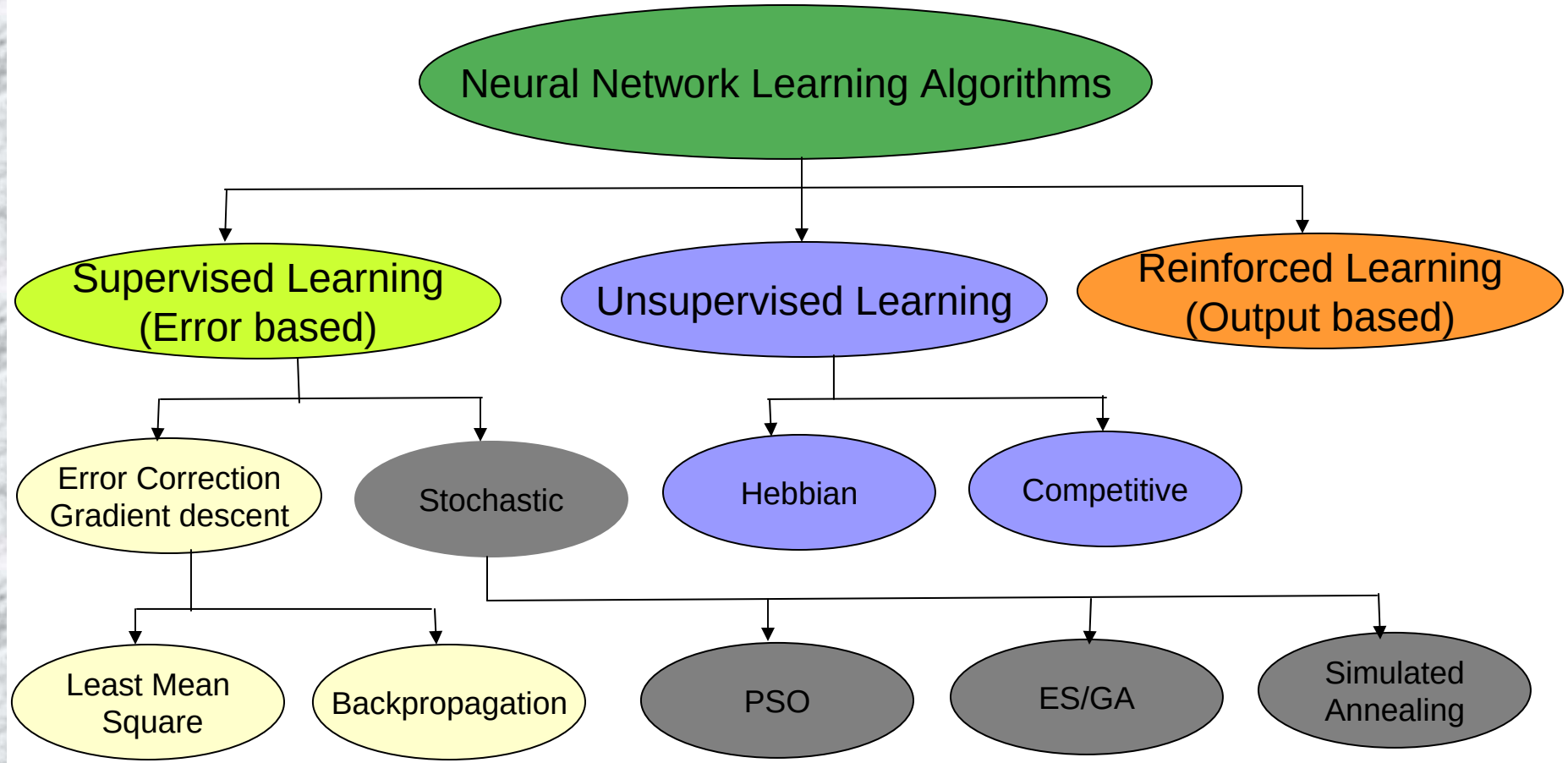
Unsupervised Learning:- In contrast to supervised learning, the objective of unsupervised learning is to discover patterns or features in the input data with no help from a teacher, basically performing a clustering of input space.

Reinforcement Learning:- In this method, a teacher though available, does not present the expected answer but only indicates if the computed output is correct or incorrect. The information provided helps the network in its learning process. A reward is given for a correct answer computed and a penalty for a wrong answer.



UMR

Classification of Learning Algorithms

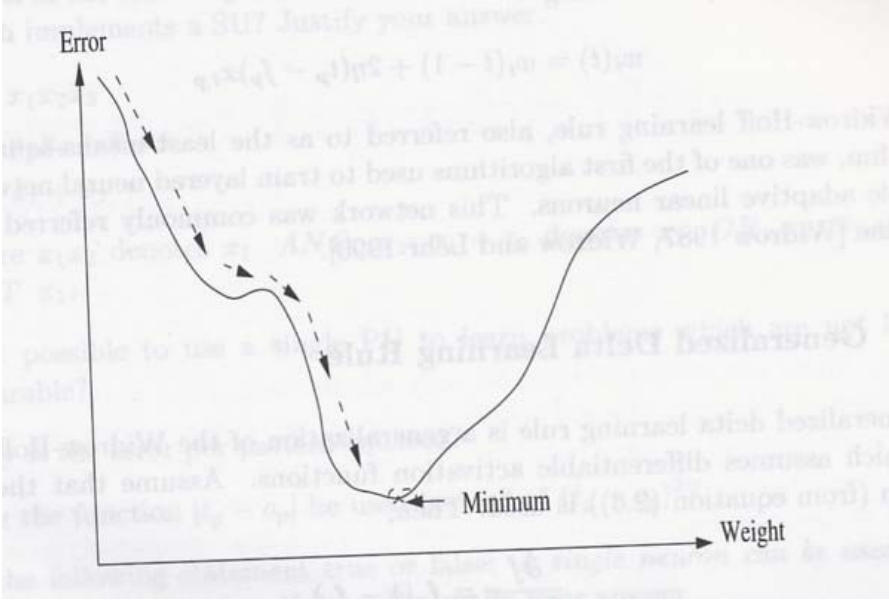


UMR

Gradient Descent Learning Rule – Neuron

$$f_j = \psi\left(\sum_{i=1}^{n+1} w_{ji} x_i\right) \qquad Error = \sum_{p=1}^{P} (t_p - f_p)^2$$

where t_p and f_p are respectively the target and actual output for patterns p, and P is the total number of input-target vector pairs (patterns) in the training set.



Gradient Descent Learning Rule – Neuron

The weights are updated using:

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$with \quad \Delta w_i(t) = \eta(-\frac{\partial E}{\partial w_i})$$

$$Error = \sum_{p=1}^{P} (t_p - f_p)^2$$

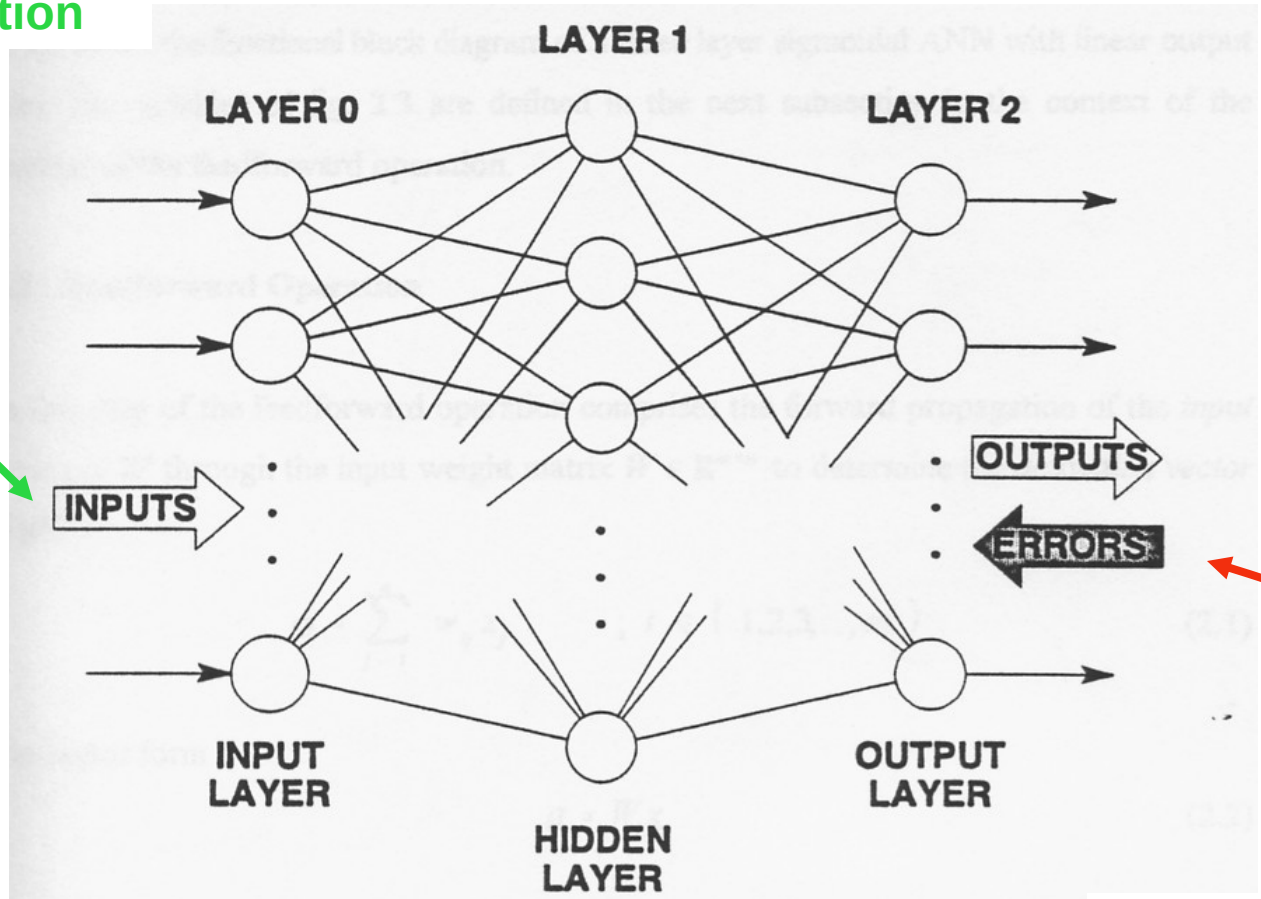
$$where \quad \frac{\partial E}{\partial w_i} = -2(t_p - f_p)\frac{\partial f}{\partial u_p} x_{i,p}$$

where η is the learning rate and $w_i(t+1)$ is the new weights.

Feedforward Neural Networks

UMR

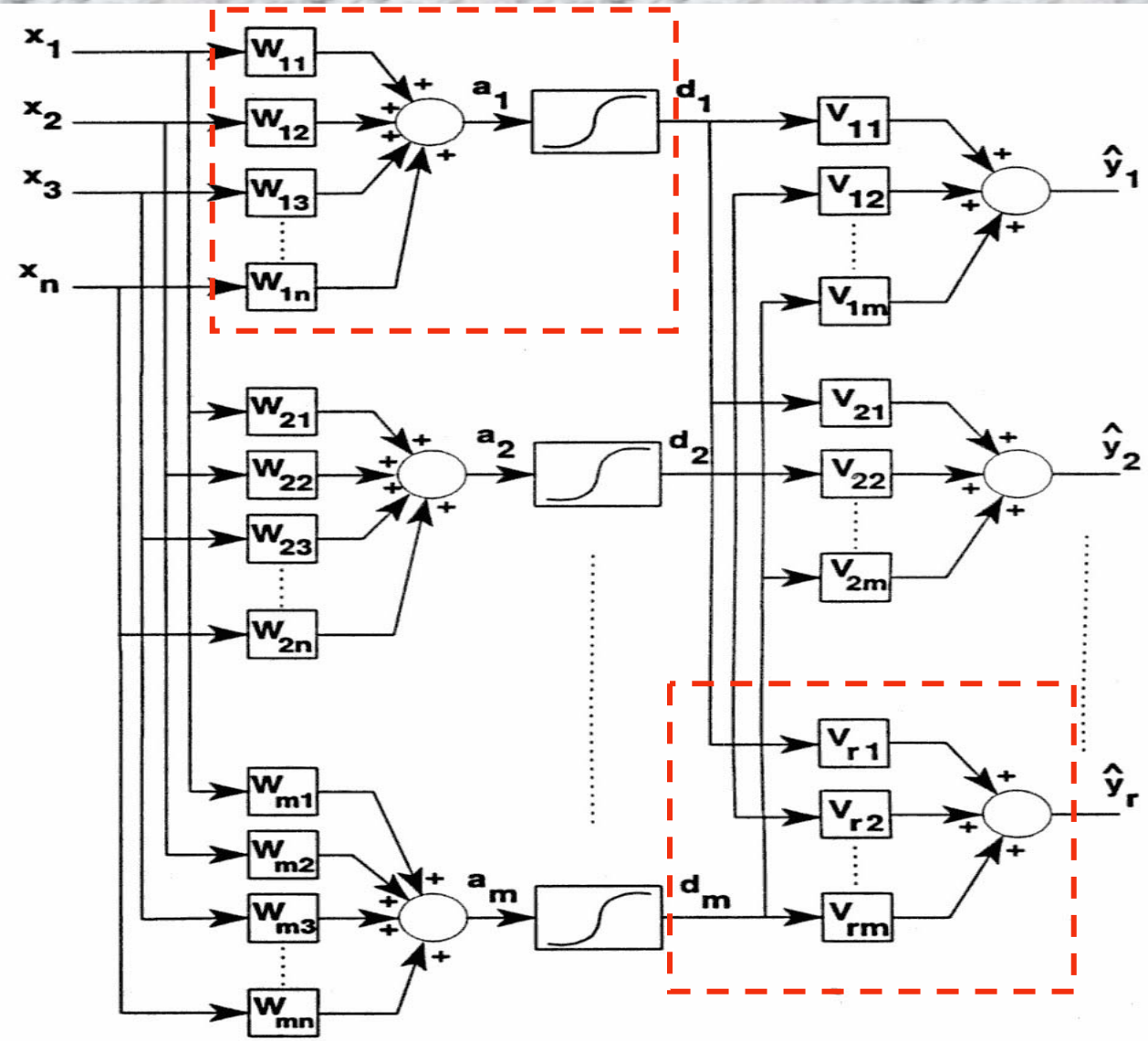
Feedforward Operation



Backpropagation Operation

UMR

FFNN/MLP Functional Block Diagram



FFNN Feedforward Operation

Input vector x_j where $j = 1$ to n *(number of inputs).*
Input weight matrix W_{ij} where $i = 1$ to m *(hidden neurons).*

Step 1: Activation vector a_i is given by

$$a_i = \sum_{j=1}^{n} w_{ij} x_j$$

$$\underline{a} = W \underline{x}$$

Decision vector d_i is given by

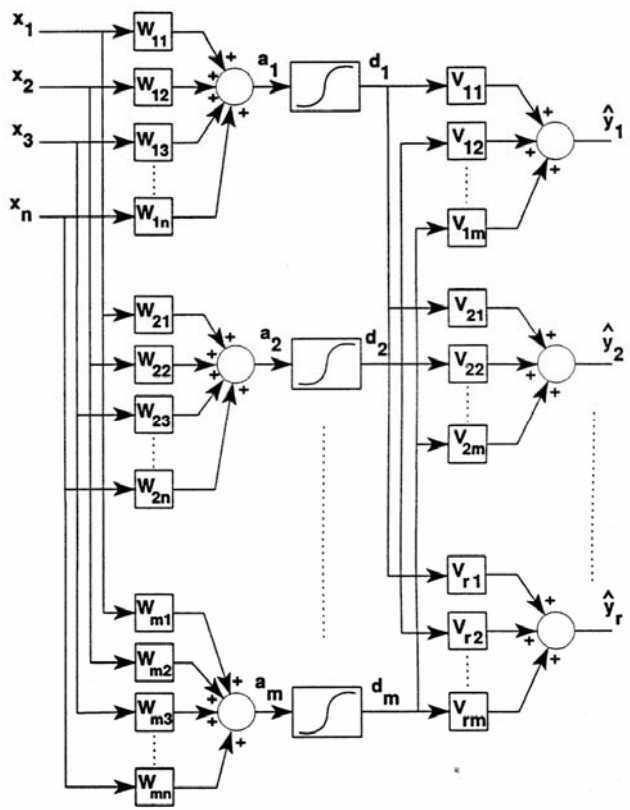
$$d_i = sig(a_i)$$

$$where \quad sig(.) = \frac{1}{1+e^{-(.)}}$$

Step 2: Output vector $\hat{y}_i$ is given by (r is no. of outputs)

$$\hat{y}_h = \sum_{i=1}^{m} v_{hi} d_i \quad ; \quad h \in \{1,2,3,...r\}$$

$$\underline{\hat{y}} = V \underline{d}$$



FFNN Backpropagation Operation

UMR

The error function used to derive the backpropagation training algorithm is based on the principle of gradient descent and is given as half the square of the Euclidean norm of the ANN output error vector.

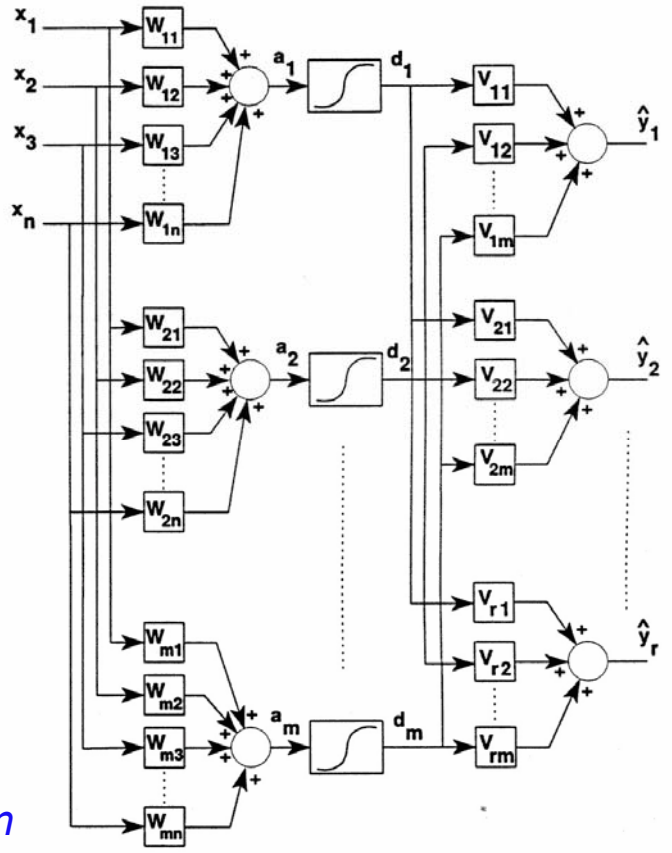
$$E(.) \equiv E(\underline{e}_y) \equiv E(x, W, V, y) \equiv \frac{1}{2}\left|\underline{e}_y\right|^2$$

This is called the objective function for ANN learning to be optimized by the optimization method.

The square law results in more sensitivity to larger errors than smaller errors. Thus,

-Large parameter adjustments for fast reduction of large errors, and

-Fine parameter adjustments for settling into Desired error function minima.



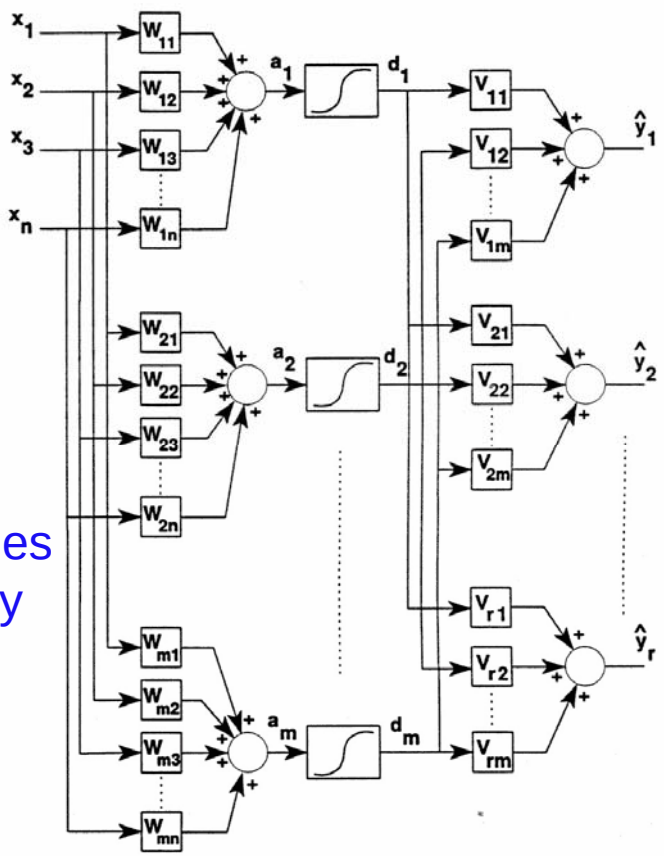
UMR

Weight Adjustments/Updates

Two types of supervised learning algorithms exist, based on when/how weights are updated:

- Stochastic/Delta/(online) learning, where the weights are adjusted after each pattern presentation. In this case the next input pattern is selected randomly from the training set, to prevent any bias that may occur due to the sequences in which patterns occur in the training set.

- Batch/(offline) learning, where the weight changes are accumulated and used to adjust weights only after all training patterns have been presented.



FFNN Backpropagation Operation

UMR

Step 1: The output error vector is given by

$$\underline{e}_y = \underline{y} - \underline{\hat{y}}$$

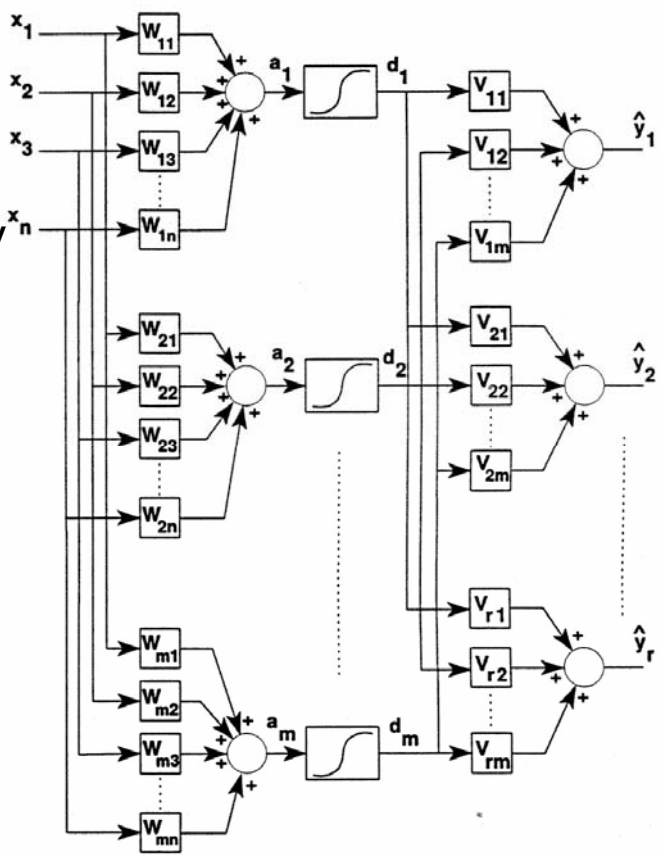
Step 2: The decision error vector is given by

$$\underline{e}_d = V^T \underline{e}_y$$

The activation error vector is given by

$$\underline{e}_{ai} = \frac{d}{da_i} d_i \underline{e}_{di} \qquad \frac{d}{da_i} d_i = d_i(1-d_i)$$

$$\underline{e}_{ai} = d_i(1-d_i)\underline{e}_{di}$$



Step 3: The weights changes are given by

$$\Delta V(k) = \gamma_g \underline{e}_y(k)\underline{d}^T(k) + \gamma_m \Delta V(k-1)$$

$$\Delta W(k) = \gamma_g \underline{e}_a(k)\underline{x}^T(k) + \gamma_m \Delta w(k-1)$$

where γ_g *and* γ_m *are learning and momentum gains respectively.*

UMR

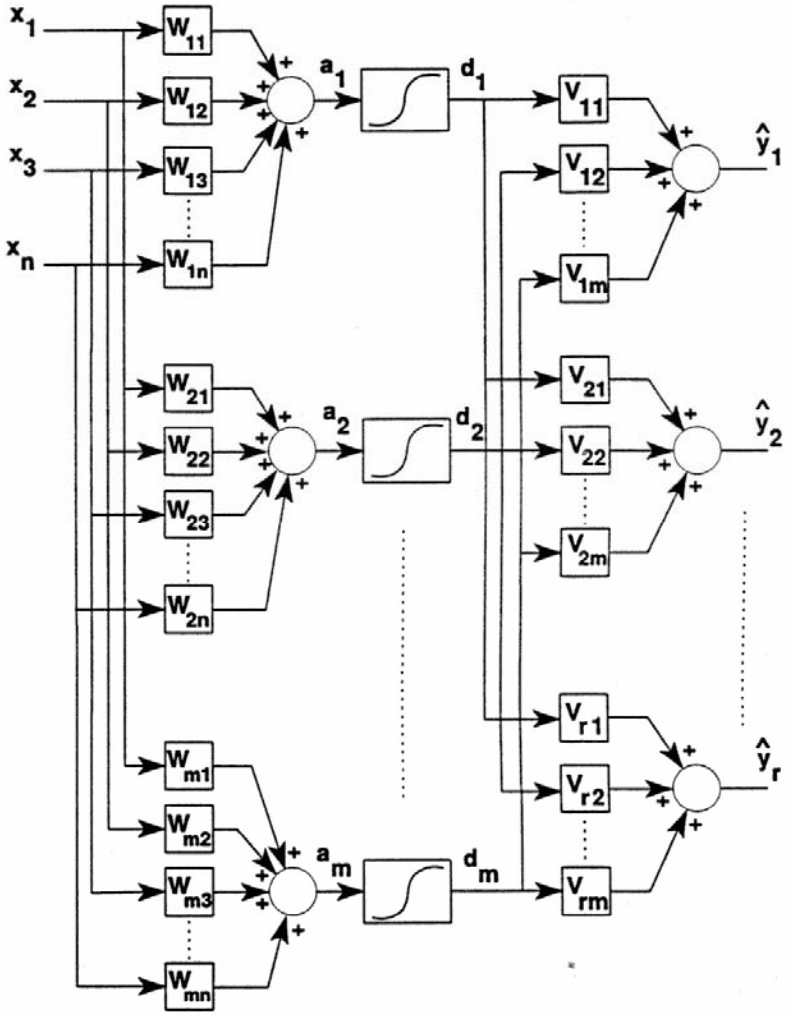
FFNN Weight Updates

Step 3: The weights updates are given by

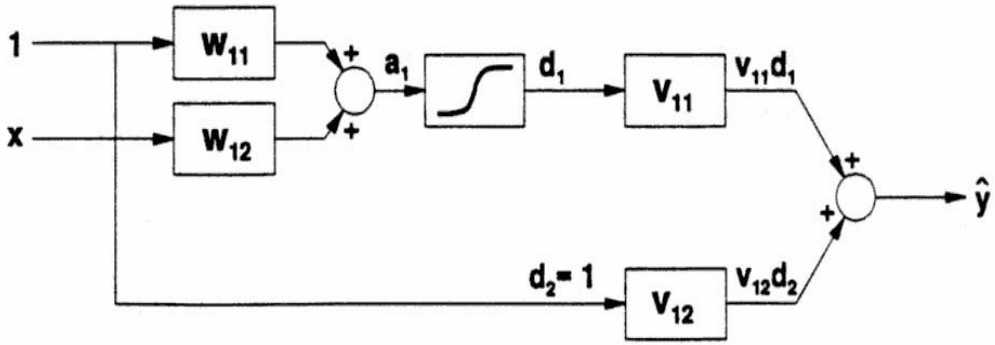
$$V(k+1) = V(k) + \Delta V(k)$$

$$W(k+1) = W(k) + \Delta W(k)$$

One set of weight modifications is called an *epoch*, and many of these may be required before the desired accuracy of approximation is reached.

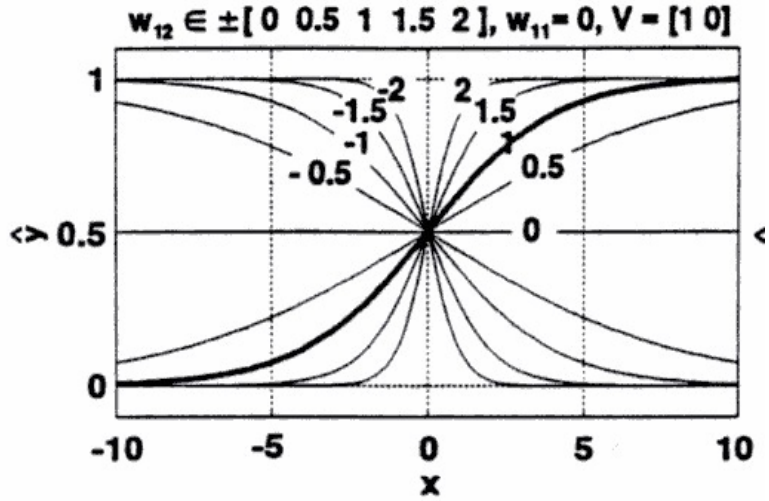


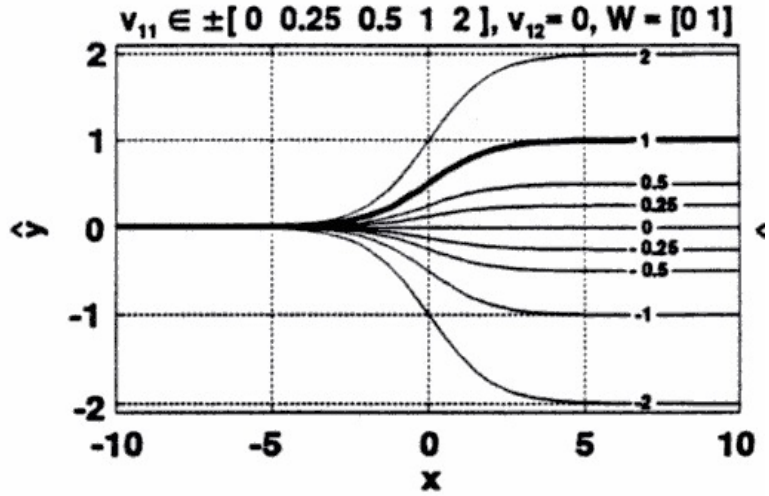
Feedforward Neural Networks



$$\hat{y} = v_{11}\,sig(w_{12}x + w_{11}) + v_{12}$$

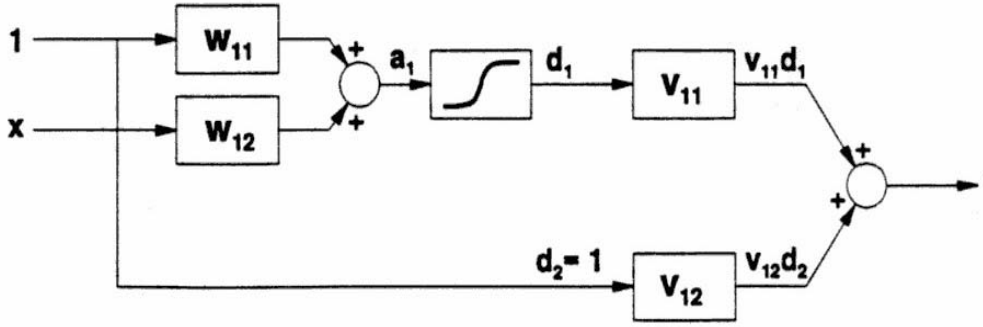
$$sig(.) = \frac{1}{1 + e^{(.)}}$$





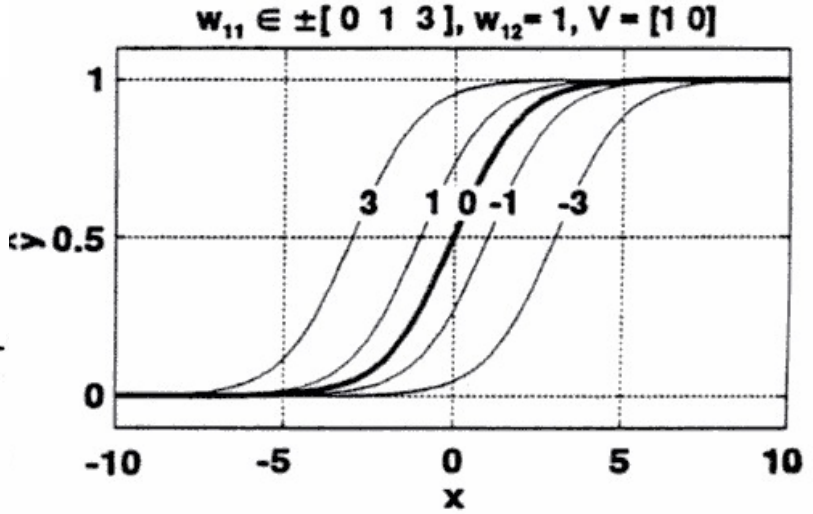
UMR

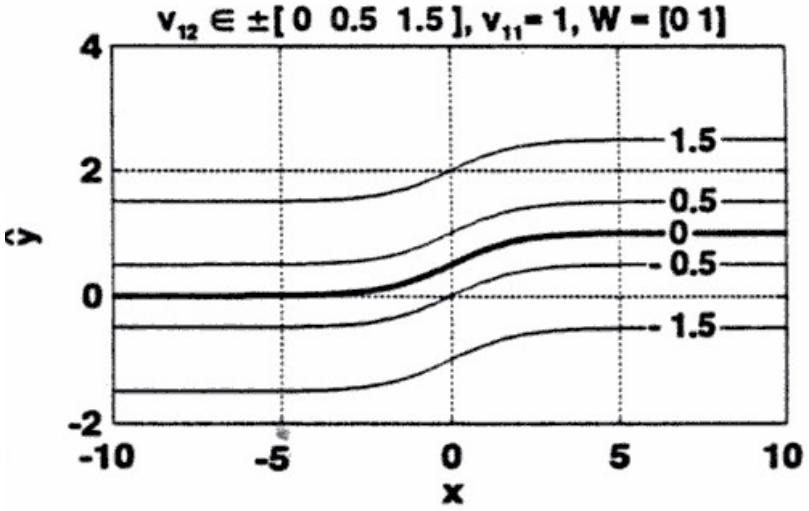
Feedforward Neural Networks



$$\hat{y} = v_{11}\,\mathrm{sig}(w_{12}x + w_{11}) + v_{12}$$

$$\mathrm{sig}(.) = \frac{1}{1 + e^{(.)}}$$





Evolutionary Computing

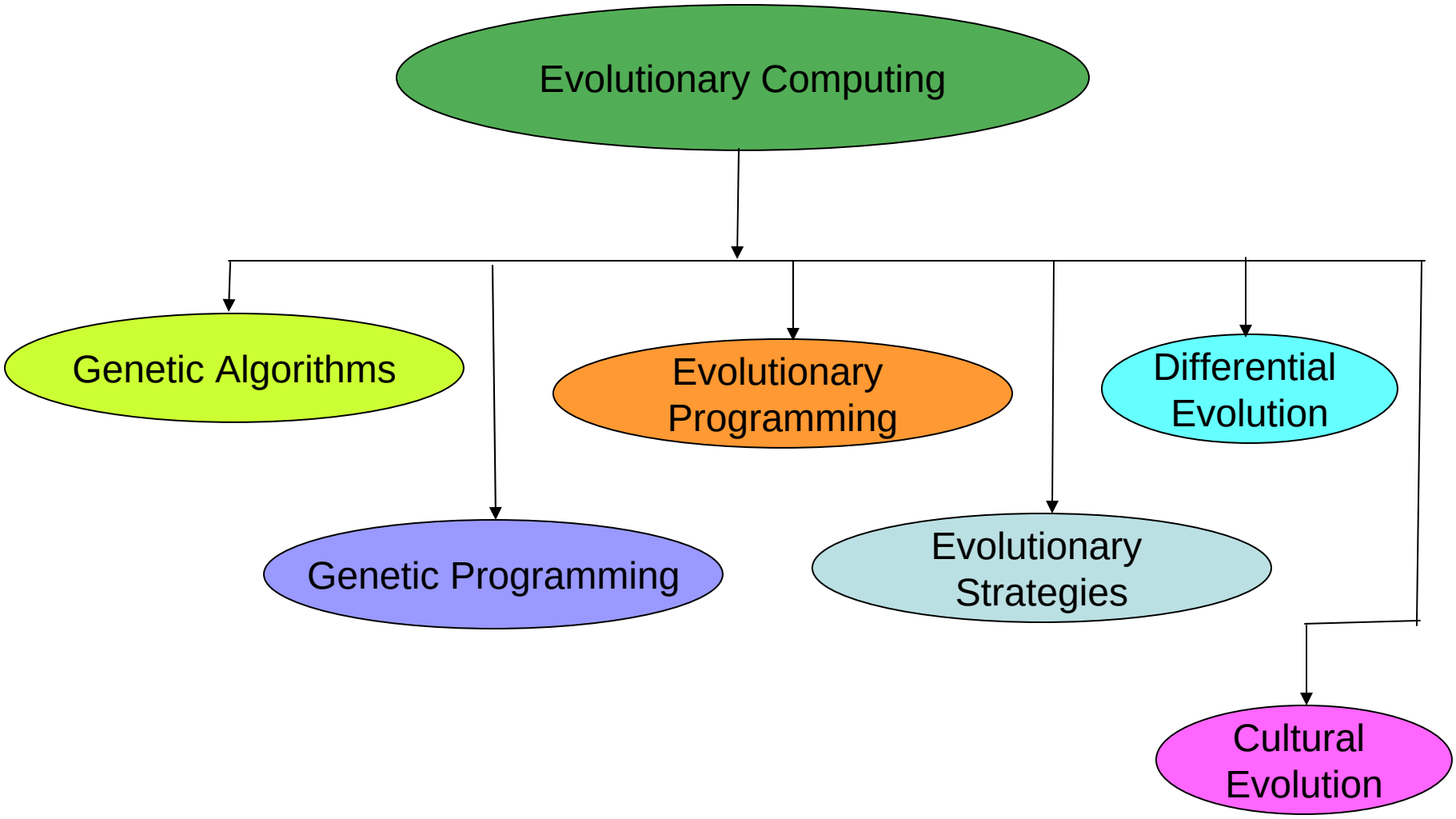
1. Evolution is the process of adaptation with the aim of improving the survival capabilities through processes such as natural selection, survival of the fittest, reproduction, mutation, competition and symbiosis.

2. Evolution is an optimization process.

3. EC is a field of CI which models the processes of natural evolution.

UMR

Evolutionary Computing



Main Steps in Evolutionary Computation

1. Encoding of solutions to the problem as a chromosome.
2. Fitness function – evaluation of individual strength.
3. Initialization of the initial populations.
4. Selection operators.
5. Reproduction operators.

UMR

Encoding Process

1. Population of individuals – each individual is a candidate solution.

2. Characteristics of an individual are represented by a *chromosome*, or *genome.*

3. Characteristics of a chromosome represented in two classes – *genotypes* and *phenotypes*.

4. A genotype describes the genetic make up of an individual as inherited from its parents. Experience of parents stored in genotypes.

5. A phenotype is the expressed behavioral traits/physical characteristics of an individual in a specific environment.

Fitness Function

1. Important part for EA to be successful.
2. Fitness is a scalar quantity.
3. Fitness function quantifies the quality of a potential solution, i.e how close is the solution to the optimal solution.
4. Selection, cross-over, mutation and elitism operators are based on fitness function values.
5. The fitness function should include all criteria to be optimized.
6. Penalty can be imposed on those individuals that violate constraints within the fitness function, in the initialization, reproduction and mutation operators.

UMR

Selection Operators

1. Random selection – good or bad individuals have equal chance.

2. Proportional Selection – chance of individuals selected is proportional to their fitness. Roulette wheel selection is used.

   (Probability of i^{th} individual = fitness of ith individual /sum of all individual fitness)

3. Tournament Selection – a group of *k* individuals are randomly selected to take part in a tournament. The winner is selected.

4. Rank-Based Selection – Ranking is given either in decreasing or increasing order based on the fitness values.

5. Elitism – Best individuals are copied into the next generation.

General Evolutionary Algorithm

1. Initialize a population of N individuals.

2. While no convergence:
   - Evaluate the fitness of each individual in the population
   - Perform cross-over
     - select 2 individuals
     - produce offspring
   - Perform mutation
     - select one individual
     - mutate
   - Select the new generation
   - Evolve the next generation.

- Convergence is reached when: max. generations is exceeded, acceptable fitness is evolved, fitness does not change significantly over past *x* generations.

UMR

Evolutionary Computing versus Classical Optimization

1. *No-Free-Lunch theorem [Wolpert and Macready 1996] states that cannot exist any algorithm for solving all problems that is on average superior to any other algorithm.*

2. Thus, the motivation for research into new optimization especially EC.

3. Classical optimization algorithms are very successful for linear, quadratic, strongly convex, unimodal problems.

4. EAs are more efficient for discontinuous, nondifferentiable, mutlimodal and noisy problems.

5. COs use deterministic rules to move from one point to other in the search space while ECs use probabilistic transition rules.

Evolutionary Computing versus Classical Optimization

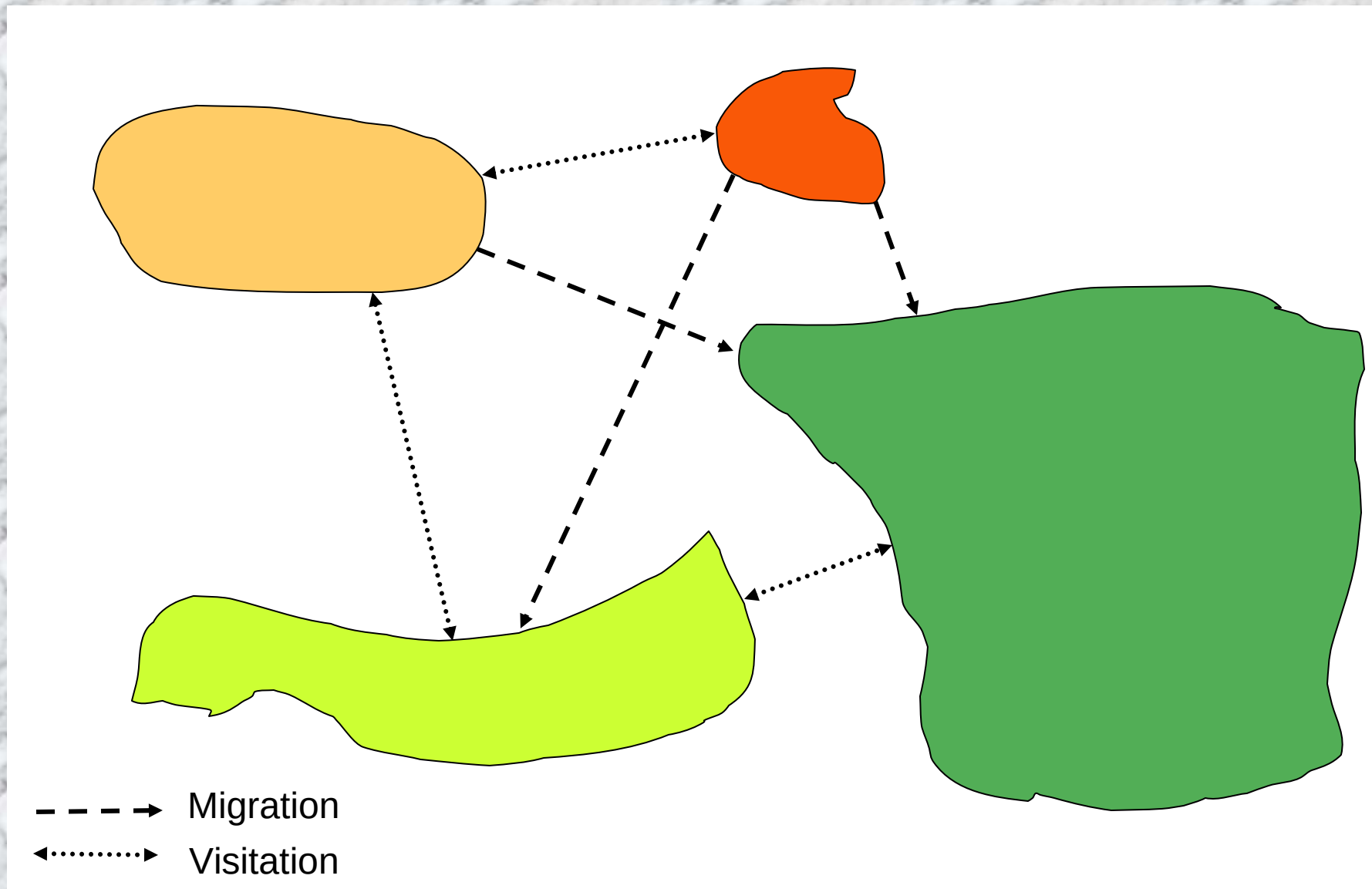
1. COs use sequential search while EAs use parallel search.

2. COs use derivative information, using first order or second order, of the search space to guide the path to the optimum.

3. ECs use no derivative information. Only fitness values of individuals are used to guide the search.

UMR

Islands of Population Based Algorithms



UMR

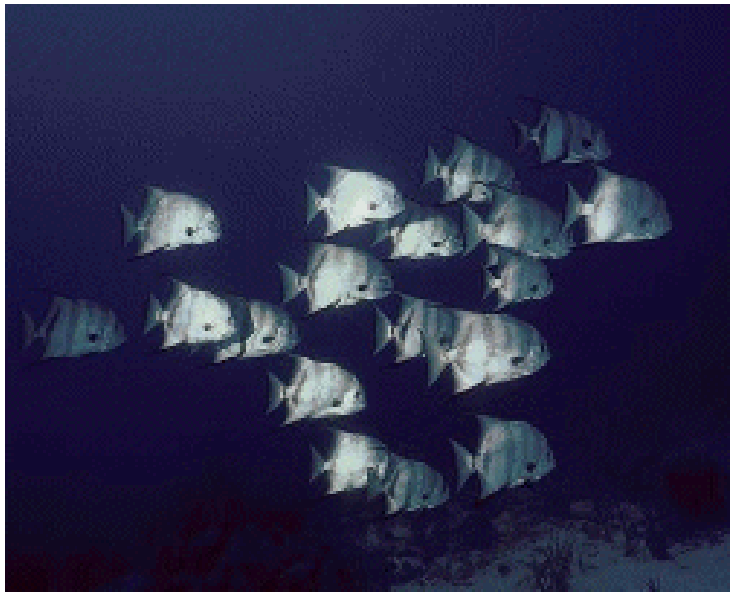
- Swarm Intelligence (SI) is the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge.

UMR

Basic Principles of Swarm Intelligence (Mark Millonas, Santa Fe Institute)

- Proximity principle: the population should be able to carry out simple space and time computations.
- Quality principle: the population should be able to respond to quality factors in the environment.
- Diversity principle: the population should not commit its activities along excessively narrow channels.
- Stability principle: the population should not change its mode of behavior every time the environment changes.
- Adaptability principle: the population must be able to change behavior mode when it's worth the computational price.

School of Fish/ Flock of Birds



"… and the thousands off fishes moved as a huge beast, piercing the water. They appeared united, inexorably bound to a common fate. How comes this unity?" – Anonymous, 17th century

The motion of a flock of birds is one of nature's delights.

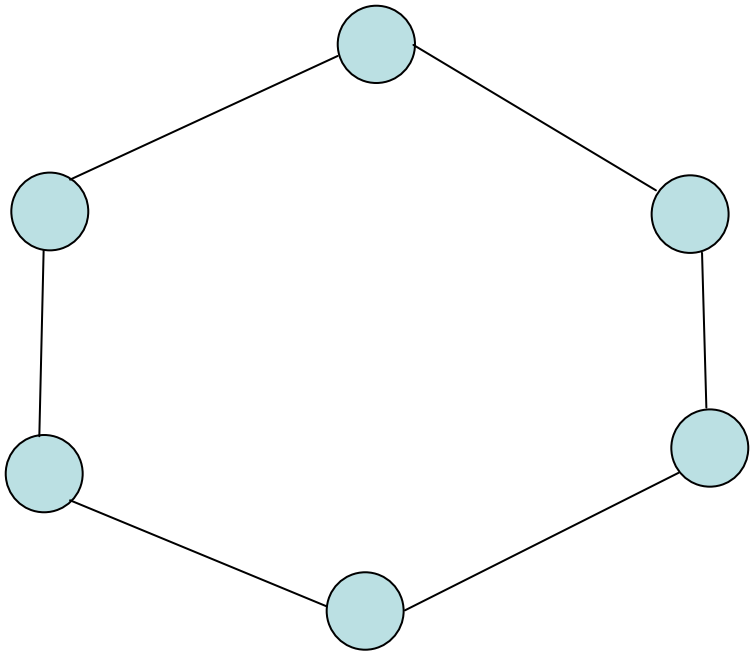
UMR

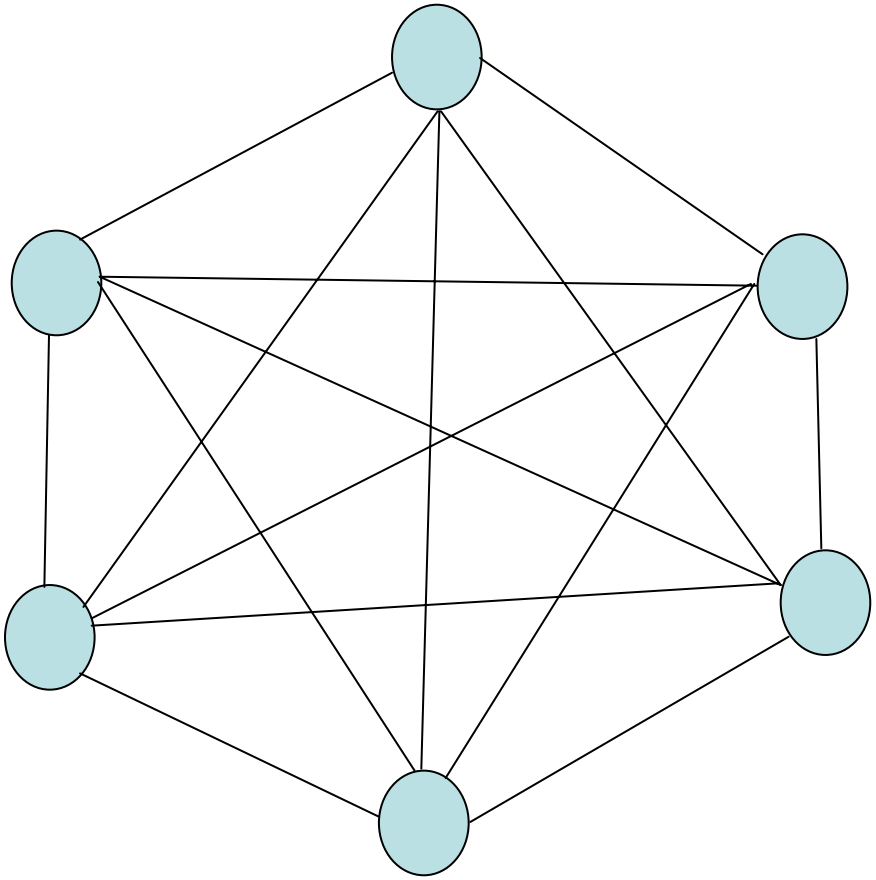
Particle Swarm Optimization (PSO)

- A concept applicable to optimizing nonlinear functions
- Has roots in artificial life and evolutionary computation
- Developed by Kennedy and Eberhart (1995)
- Key points -
  - Simple in concept
  - Easy to implement
  - Computationally efficient
  - Effective on a variety of problems

UMR

Swarm Topologies

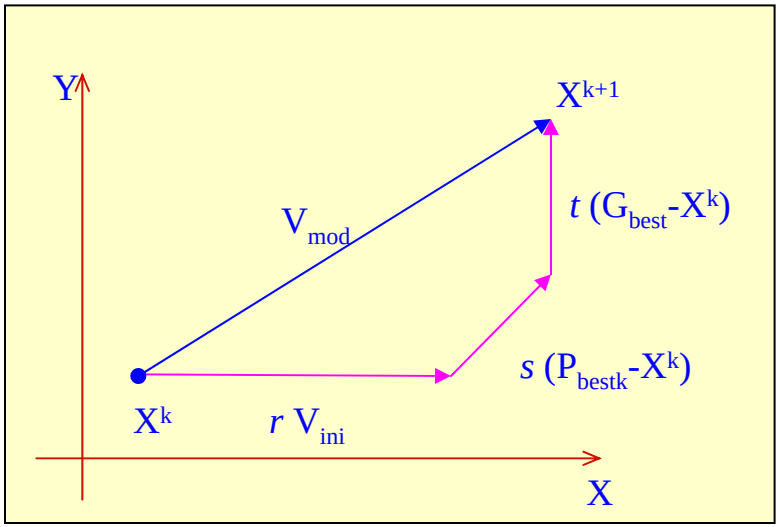




Particle Swarm Optimization

- The system initially has a population of random solutions called particles
- Each particle has random velocity and memory that keeps track of previous best position and corresponding fitness
- The previous best value of the particle position is called the 'p_{best}'
- It has another value called 'g_{best}', which is the best value of all the '*pbest*' positions in the swarm
- Basic concept of PSO lies in accelerating each particle towards its p_{best} and the g_{best} locations at each time step
- In local PSO, the 'g_{best}' is changed to 'l_{best}' where 'l_{best}' is the best value of all the particles in local neighborhood.

PSO Equations



The velocity of the particles is given as follows

$$V_{id} = w \times V_{id} + c_1 \times rand_1 \times (P_{bestid} - X_{id}) + c_2 \times rand_2 \times (G_{bestid} - X_{id})$$

The position vector of the particles is changed as follows

$$X_{id} = X_{id} + V_{id}$$

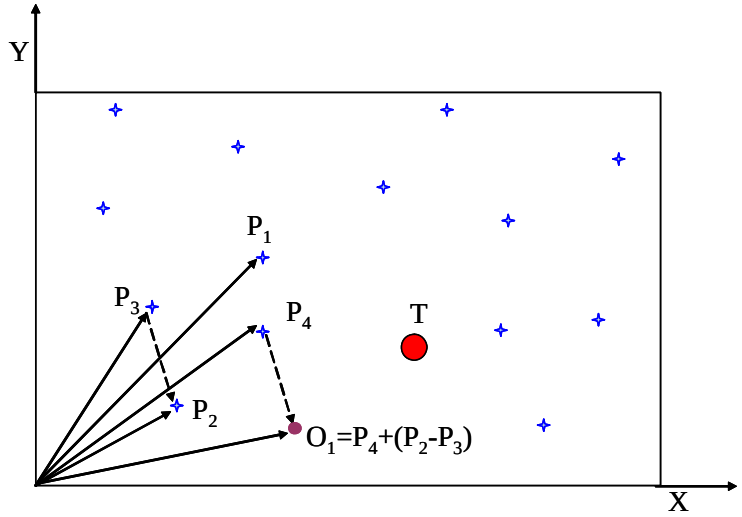
UMR

Differential Evolution

1. Initialize a population of N individuals.

2. In every generation, for each individual select 3 distinct individuals randomly from the remaining population.

$$O_{1,\,j} = P_{4,\,j} + \gamma \times \left(P_{2,\,j} - P_{3,\,j} \right)$$



3. Compute a random number to determine whether to mutate or not.

$$O_{1,\,j} = P_{1,\,j}$$

4. For each parent and its offspring, the individual with the greater fitness is passed on to the next generation.

5. Test for convergence. If all the individuals have not converged the procedure is repeated from (2).

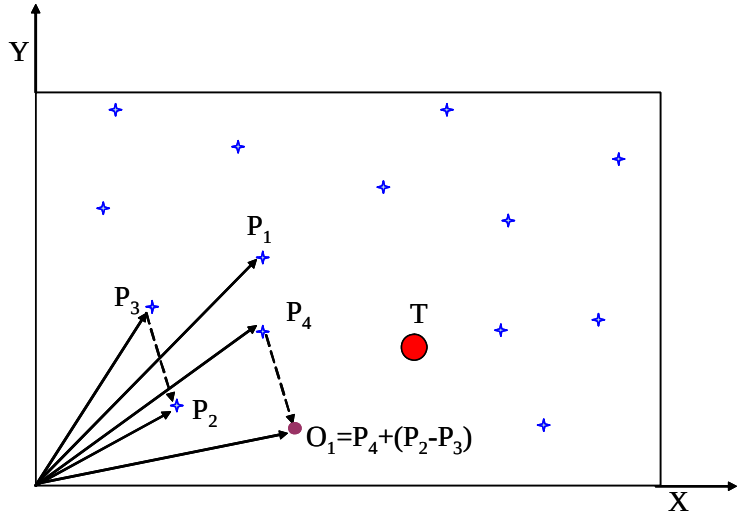
UMR

Differential Evolution

1. Initialize a population of N individuals.

2. In every generation, for each individual select 3 distinct individuals randomly from the remaining population.

$$O_{1,\,j} = P_{4,\,j} + \gamma \times \left(P_{2,\,j} - P_{3,\,j} \right)$$



3. Compute a random number to determine whether to mutate or not.

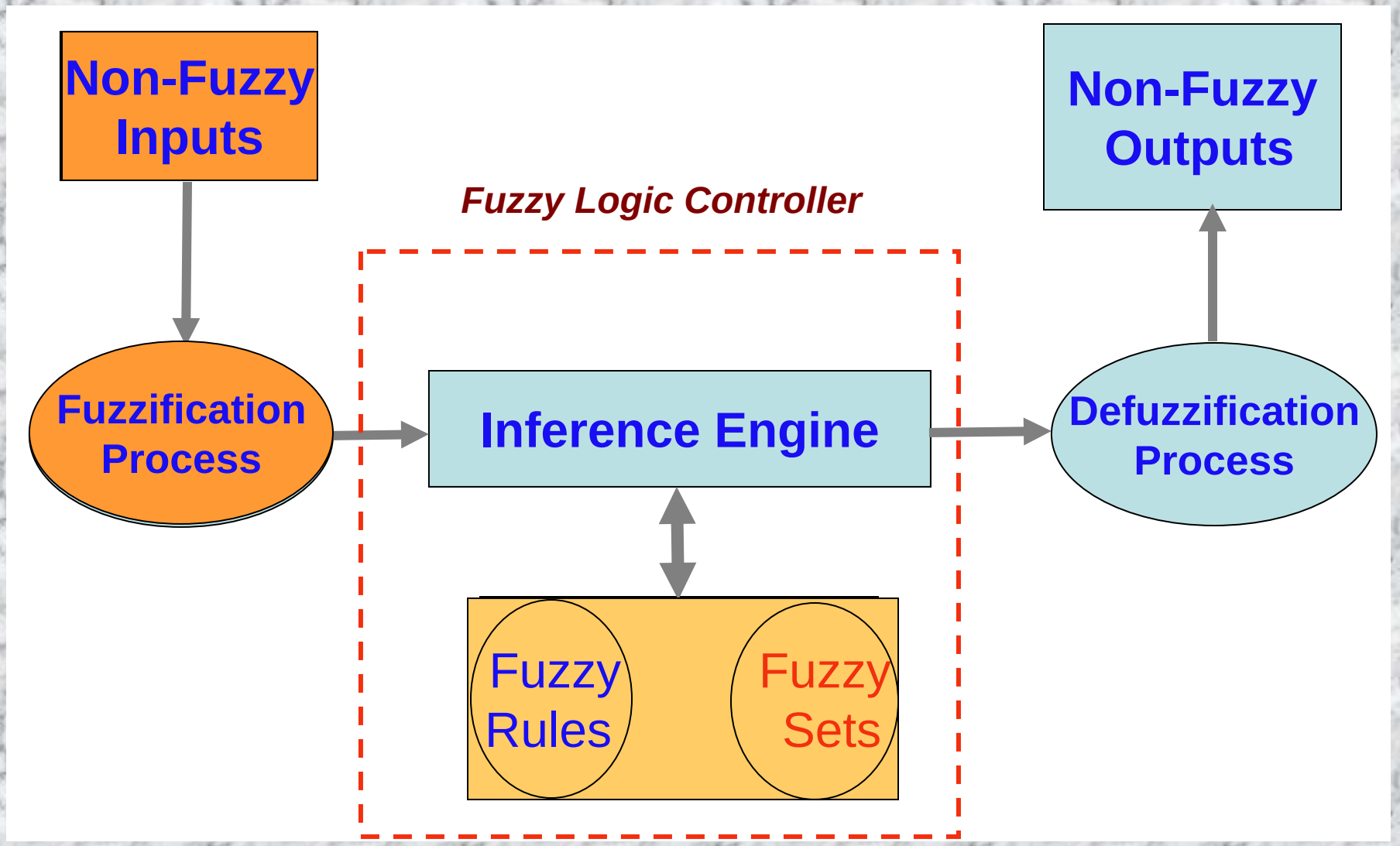
$$O_{1,\,j} = P_{1,\,j}$$

4. For each parent and its offspring, the individual with the greater fitness is passed on to the next generation.

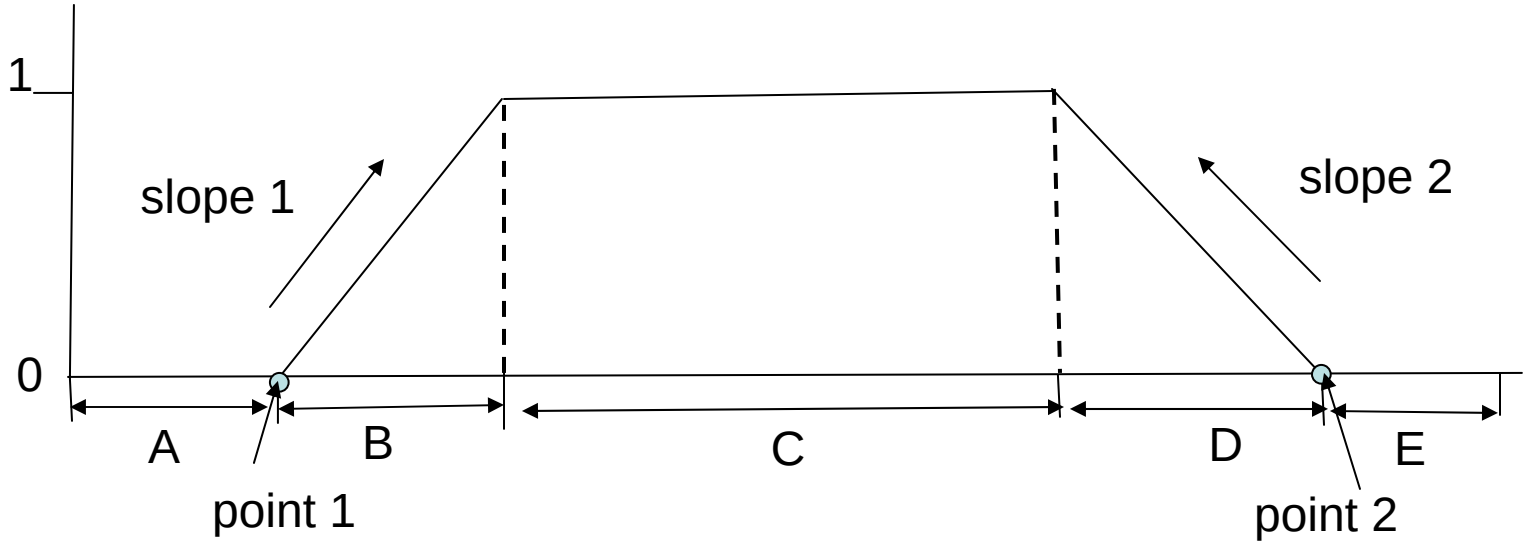
5. Test for convergence. If all the individuals have not converged the procedure is repeated from (2).

UMR

Fuzzy System



Computing Degree of Membership



System Input Range	Degree of Membership
A	0
B	(X – point 1) × slope 1
C	max
D	(point 2 - X) × slope 2
E	0

Fuzzy Rule Based System

- Example
  - *If* there is *heavy rain* and *strong winds then* there must be *severe flood* warning.
  - Fuzzy sets = {heavy, strong, severe}
  - Fuzzy variables = {rain, winds, flood}
- If the conclusion C to be drawn from a rule base R is the conjunction of all the individual consequents C_i of each rule, then

$$C = C_1 \cap C_2 \cap C_3 \cap \ldots \cap C_n$$

where

$$\mu_C(y) = \min(\mu_{C1}(y), \mu_{C2}(y), \mu_{C3}(y), \ldots, \mu_{Cn}(y))$$

Inference Engine (Firing)

- The task of the inference engine is to carry out the inferencing process which is to map the fuzzified inputs to the rule base, and to produce a fuzzified output for each rule.
- Fuzzy inference is referred to as *approximate reasoning* (evaluating linguistic descriptions).
- Rules that are not activated have a zero firing strength.

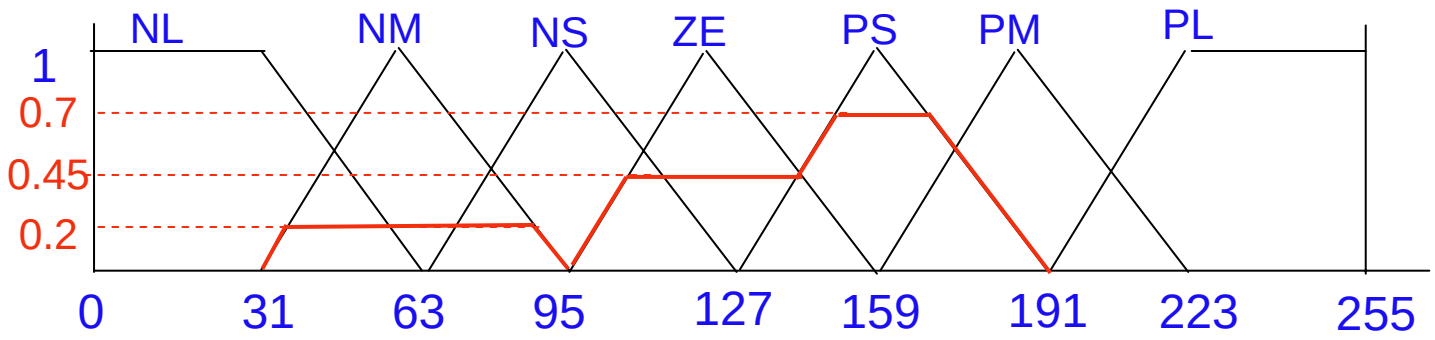
UMR

Defuzzification

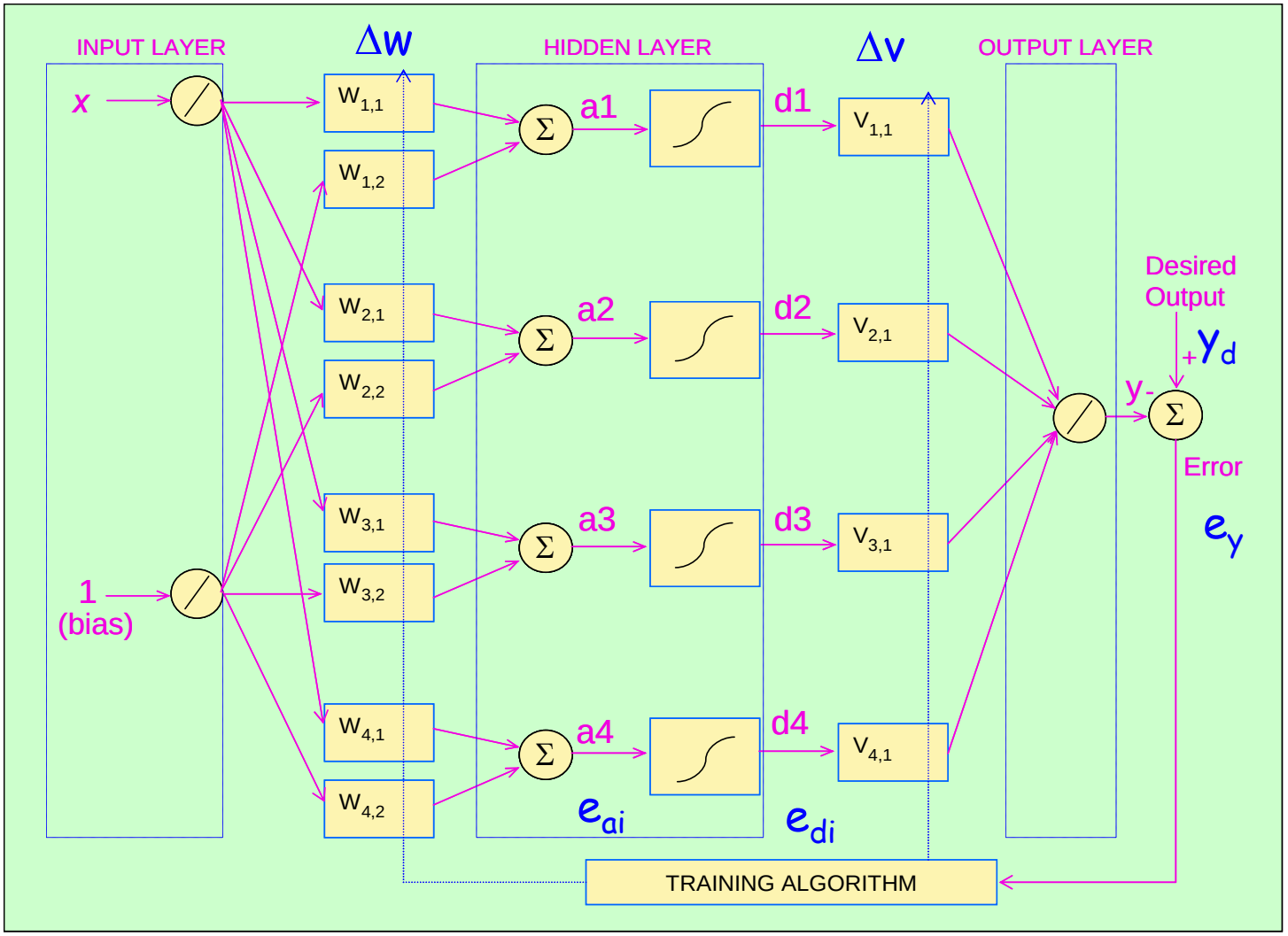
- This is the reverse process to fuzzification.
- The fuzzy output of the inference engine (fuzzy rules) is converted into scalar, or non-fuzzy value.
- Defuzzification resolves conflicts between competing actions such as '**output to be set to positive medium**' and '**output to be set to positive large**' (*example illustrates this*). In this case, defuzzification employs compromising techniques to resolve both the vagueness and conflict issues.
- Several methods exist for finding an approximate scalar value, namely:
  - Max-min method
  - Averaging method
  - Root-sum-square method
  - Clipped center of gravity method

Defuzzification

- Clipped center of gravity method
  - Each membership is clipped at the corresponding rule firing strengths. The centroid of the composite area is calculated and its x coordinate is the output of the controller.



Training of a Feedforward Neural Network



UMR

PSO for Neural Network Training

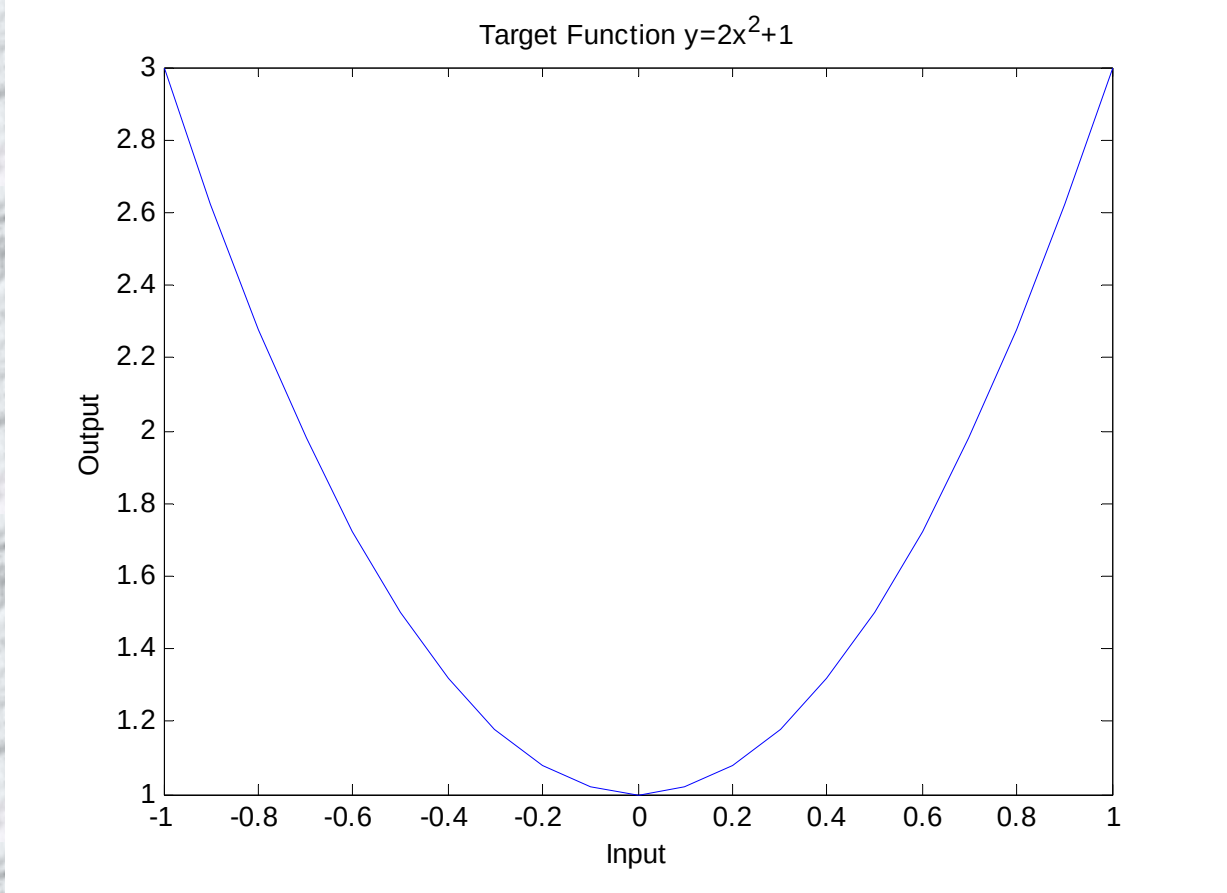
For training a neural network using the PSO:

- The fitness value of each particle of the swarm is the value of the error evaluated at the current position of the particle

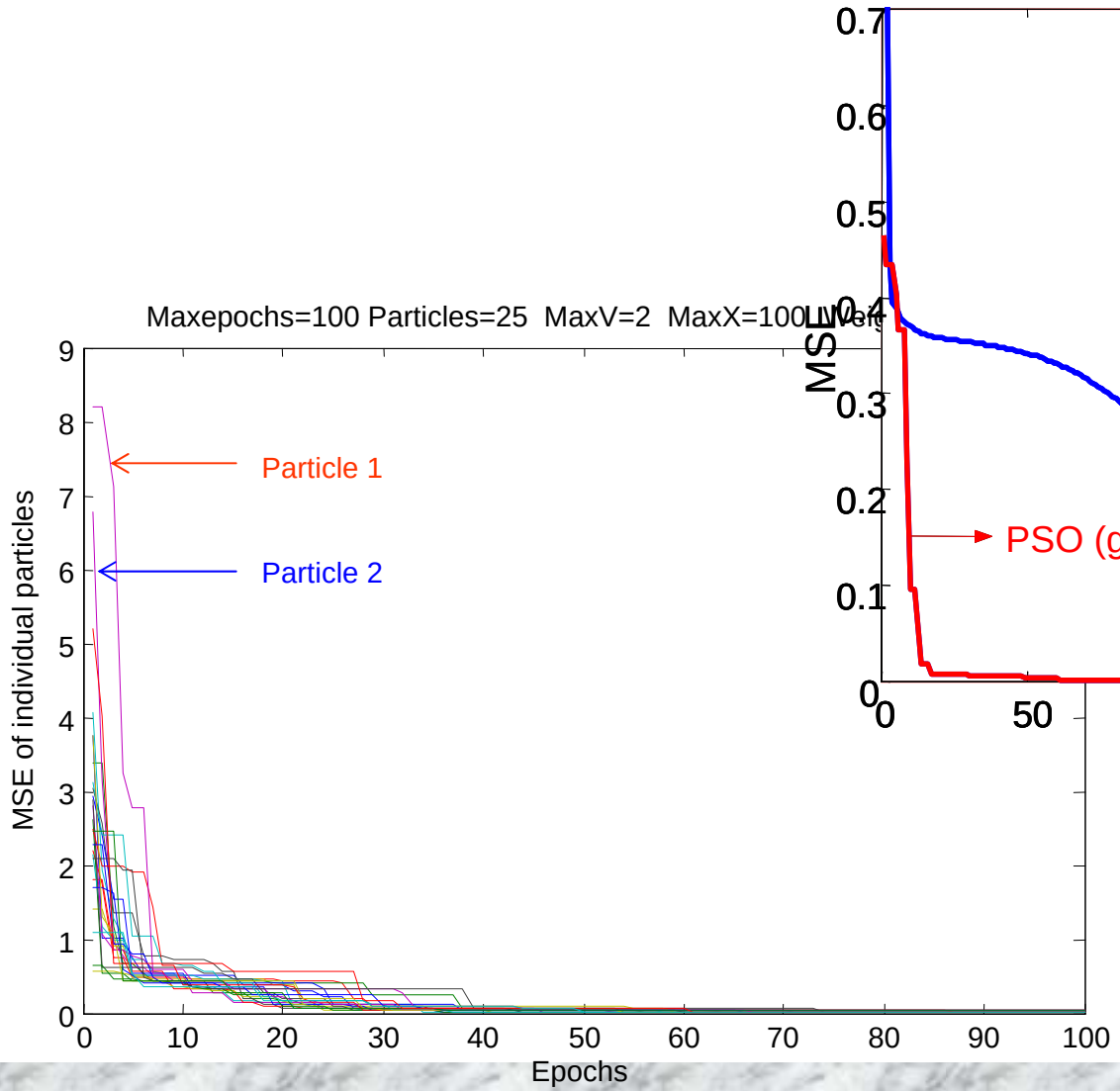
- The position vector of the particle corresponds to the weight matrix of the neural network.

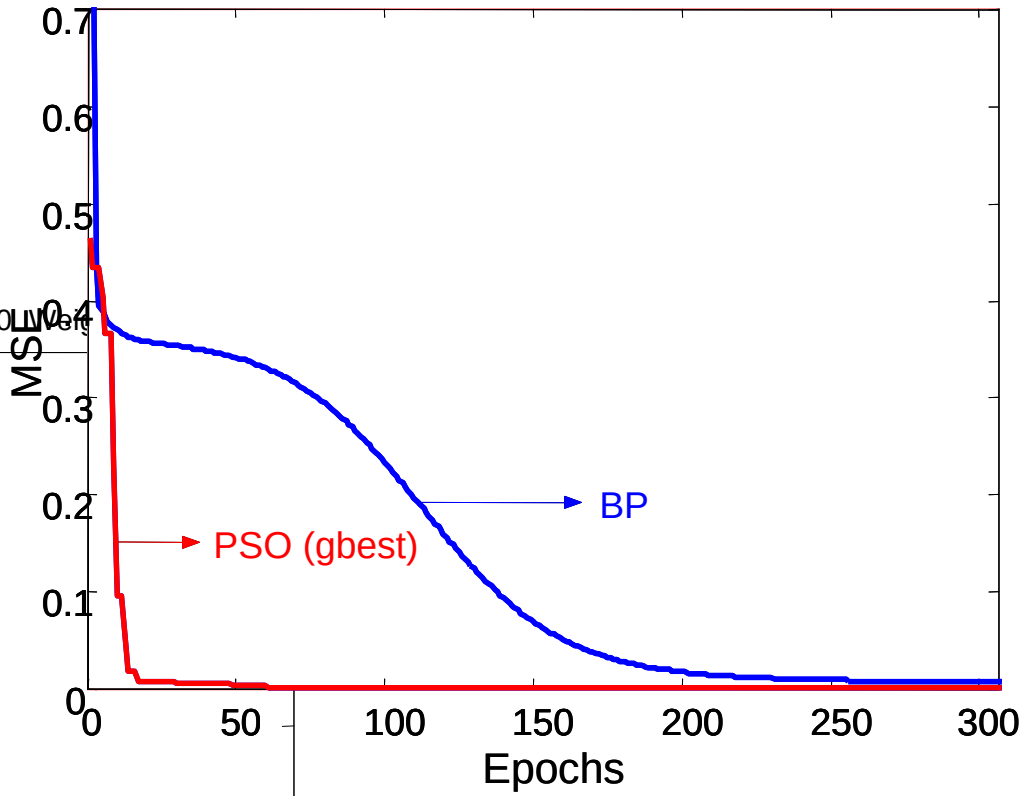
UMR

Problem 1:
Target Function for Neural Network Training
$Y = 2x^2 + 1$

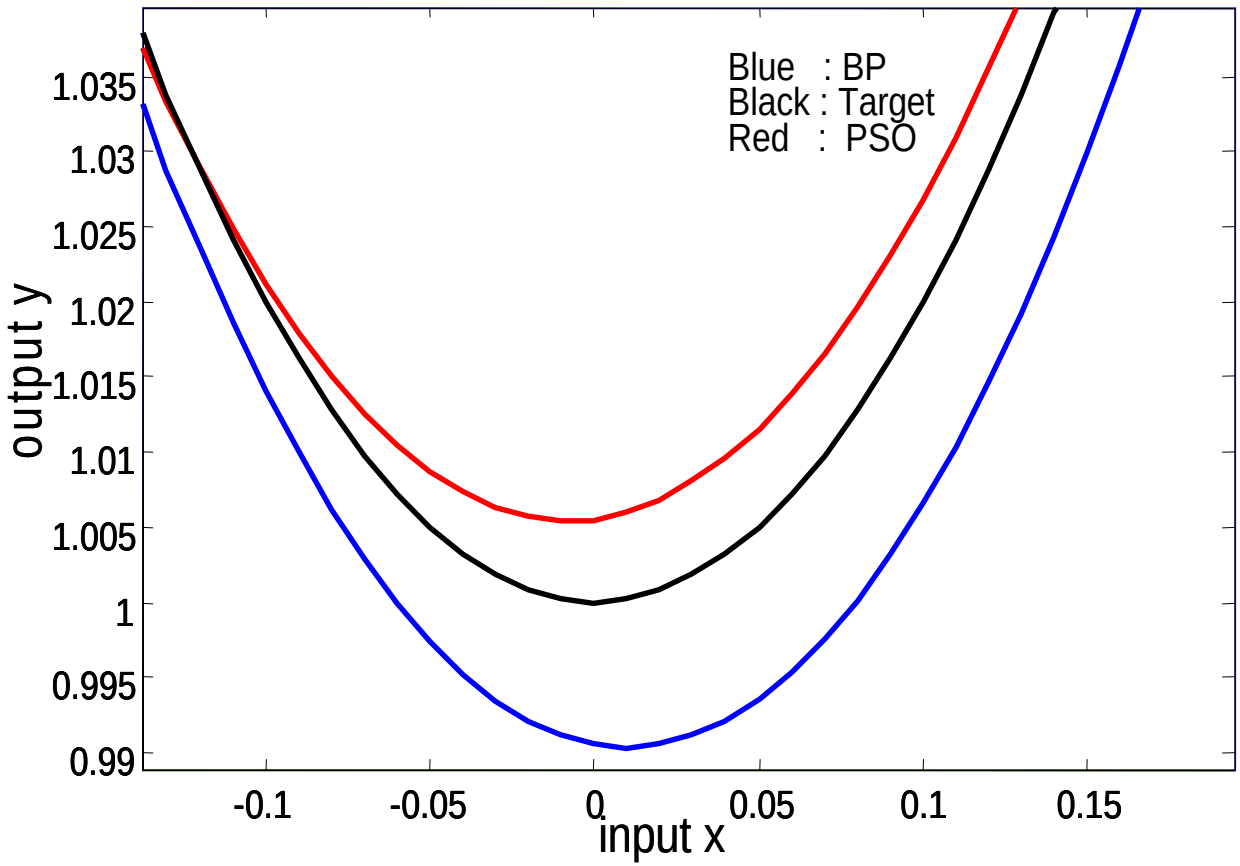


Training MSE of the PSO Particles – $Y = 2x^2+1$





Magnified Test Results for Neural Networks with Fixed Weights Trained with BP and PSO - Bias 1



UMR

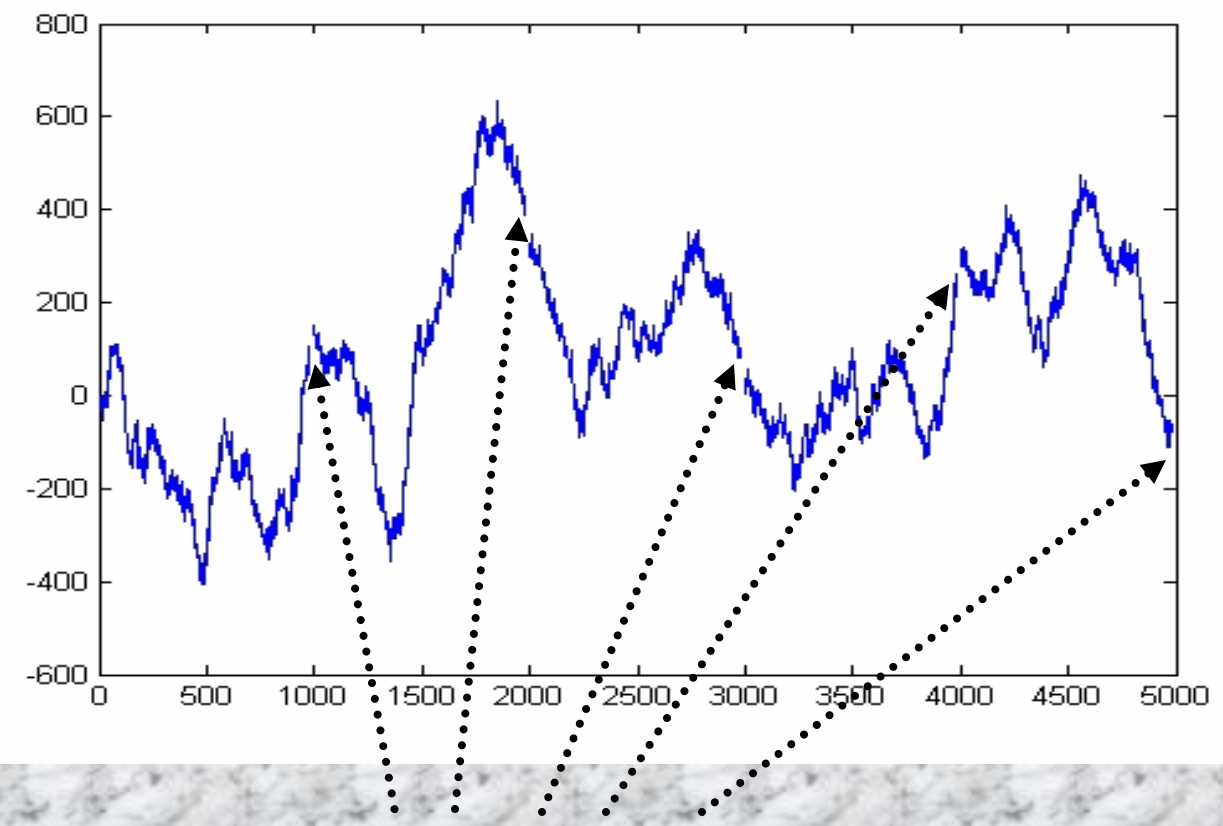
Comparison of Number of Computations in Training a Neural Network $2 \times 4 \times 1$ (Bias 1)

Error=0.001	PSO	BP	PSO	BP
Patterns (input *x*)	-1: 0.1:1		-1: 0.01:1	
Iterations	194	9836	116	962
Forward path (additions+ multiplications)	2546250	4750788	14572500	4447326
Backward path (additions+ multiplications)	523800	14045808	835200	13148616
Total (Forward + Backward)	3070050	18796596	15407700	17595942
Ratio of computations	6.1226		1.1420	

With bias 2

Error=0.001	PSO	BP
Patterns (input *x*)	-1: 0.1: 1	
Iterations	194(83	9836(307
Forward path (additions+ multiplications)	348600	148281
Backward path (additions+ multiplications)	71712	438396
Total (Forward + Backward)	420312	586677
Ratio of computations	1.3958	

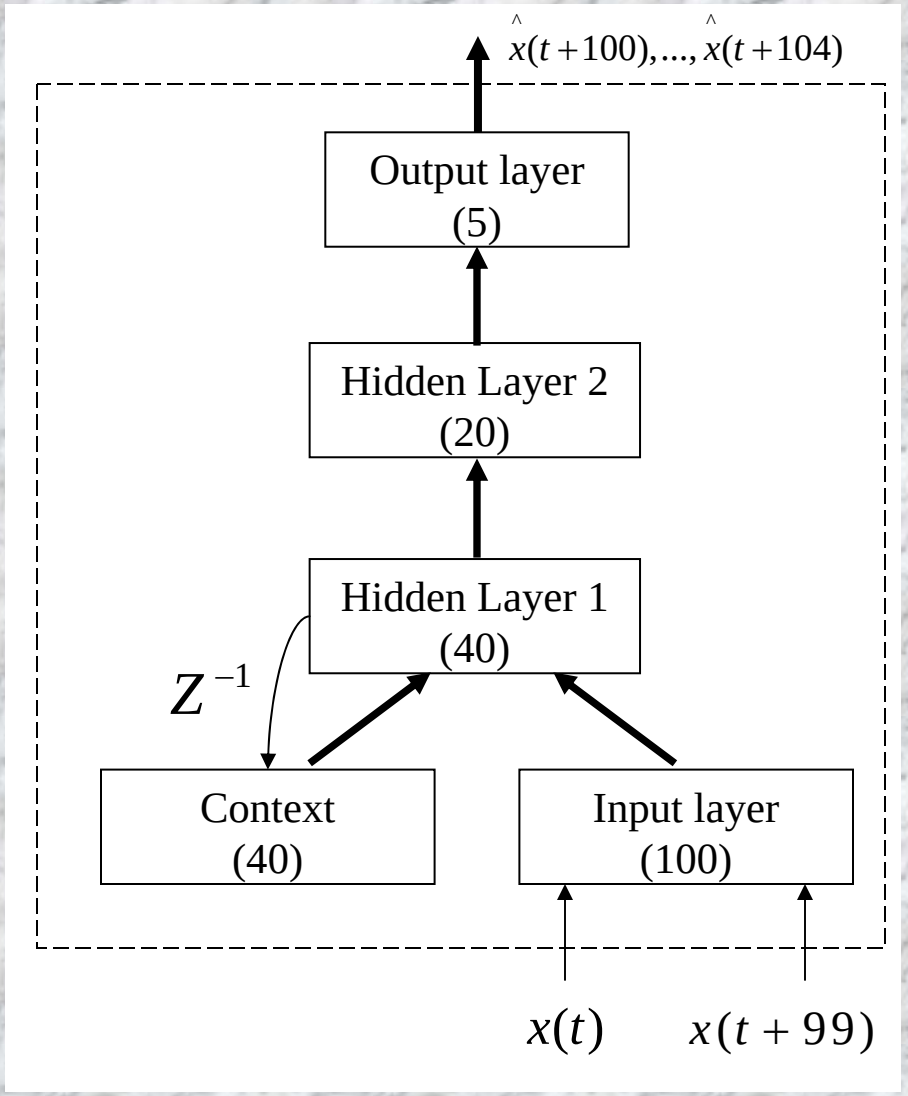
IJCNN04 CATS Problem



Missing data to be predicted

Recurrent Neural Network

- An Elman RNN is chosen as the predictor
- Novel training algorithm

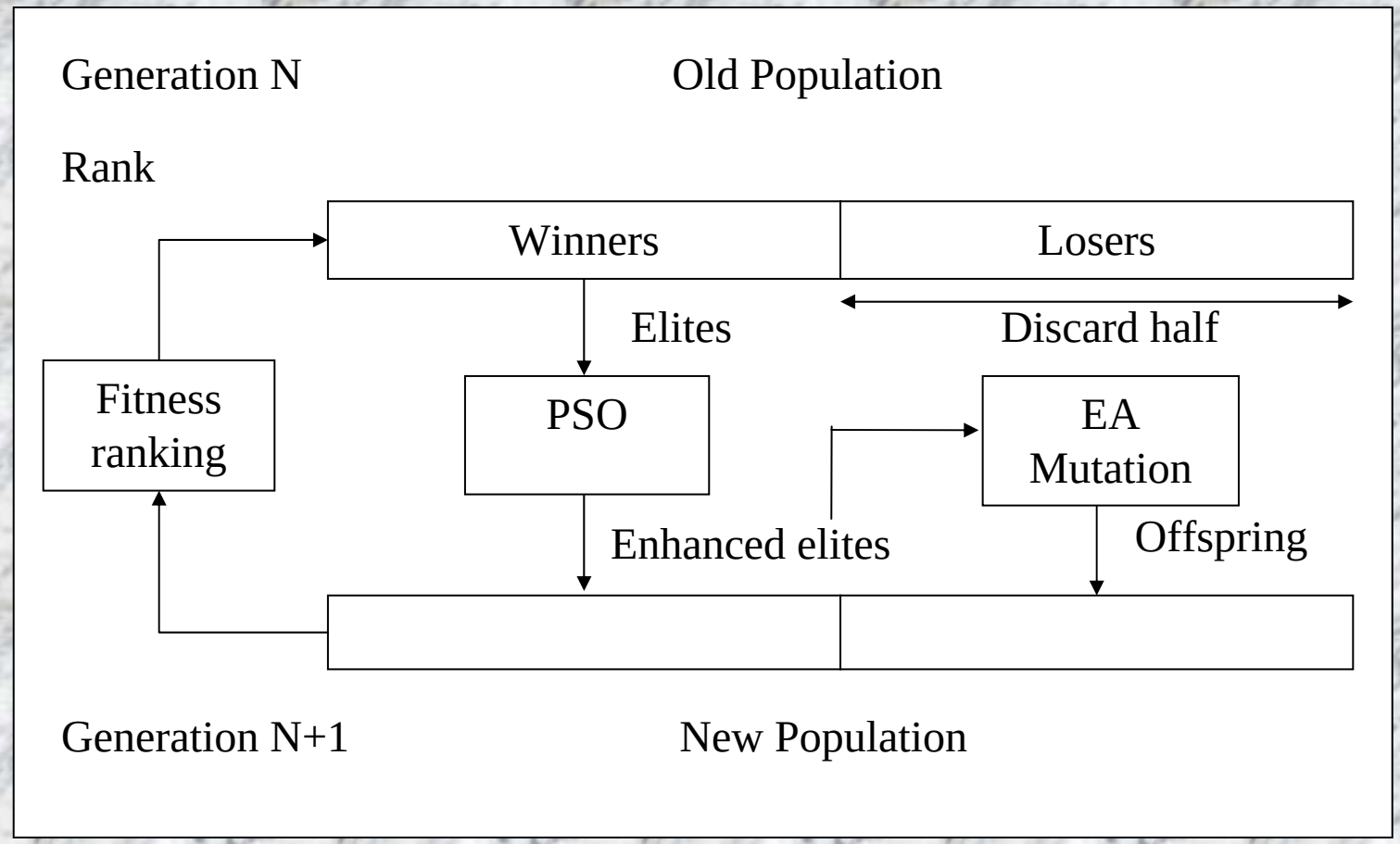


UMR

Hybrid PSO+EA

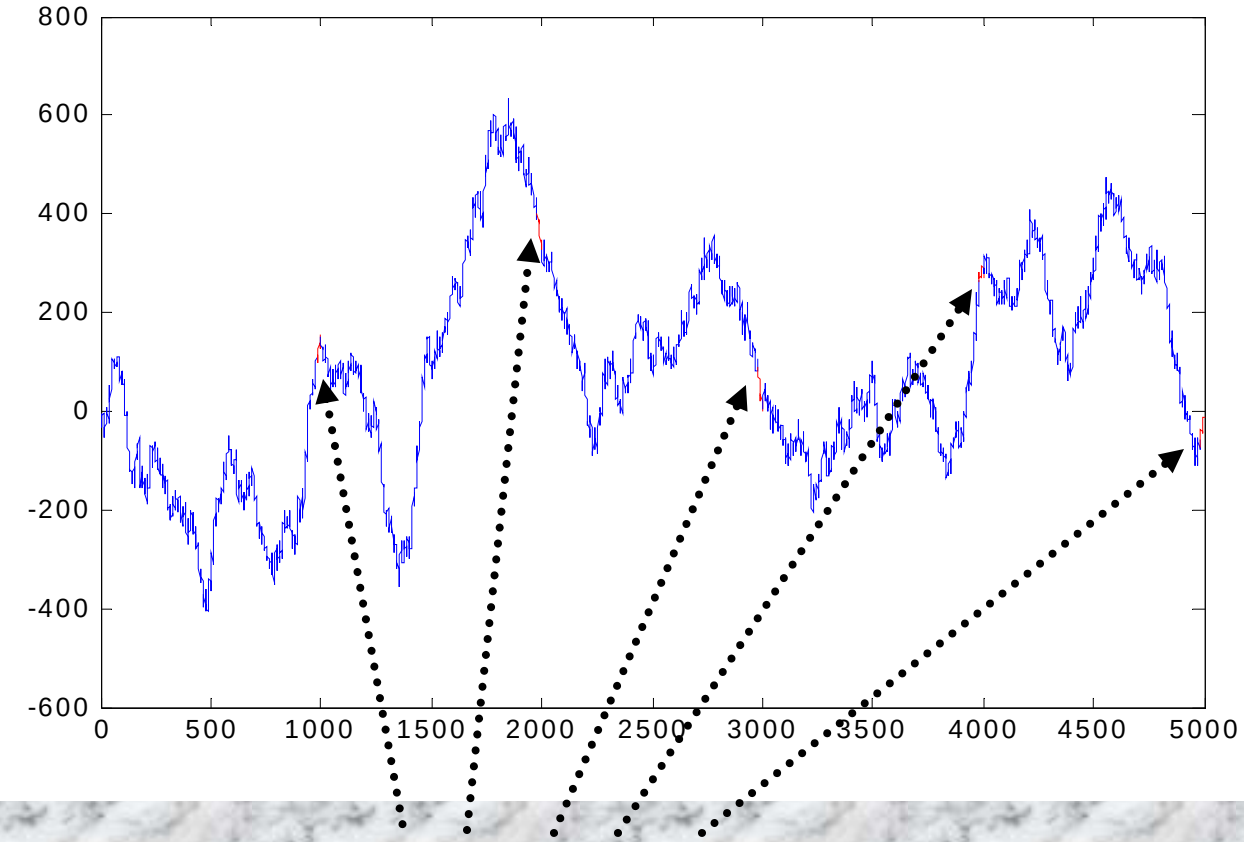
- Hybrid = Co-operative + Competitive
  - (PSO + EA)
- Apply evolutionary operators to PSO
  - Selection, crossover and mutation
- Benefits:
  - Focus on good region
  - Increase the diversity of the population
  - Save computation

Hybrid PSO+EA



UMR

Prediction



Predicted missing data

UMR

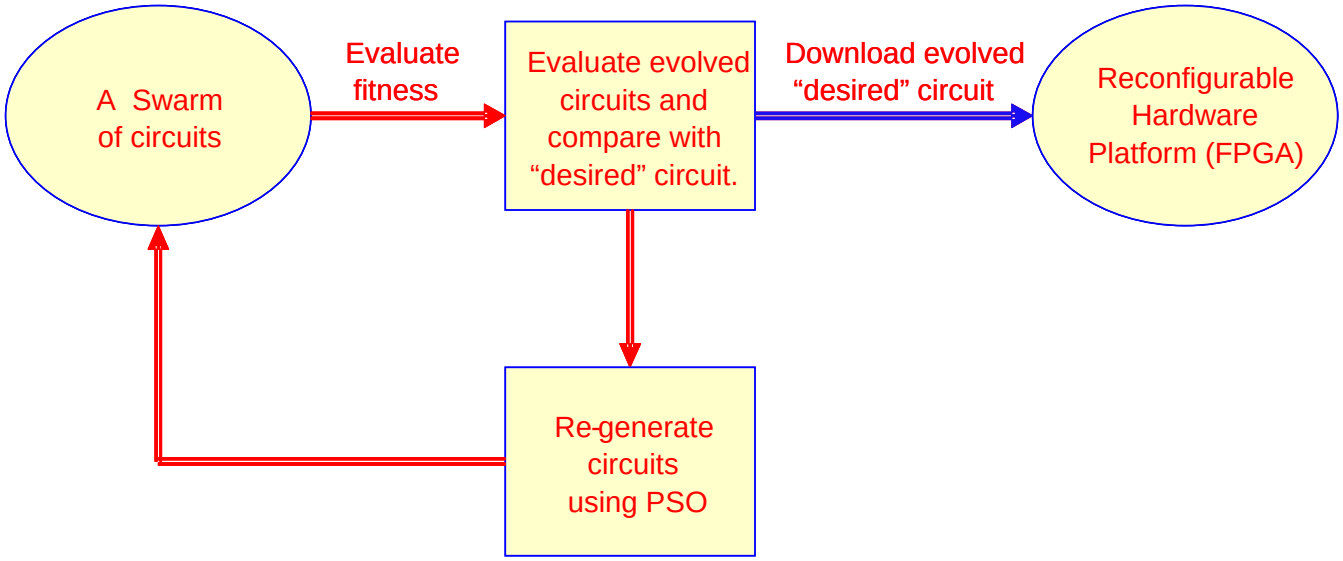
Author	MSE (100) - E_1	MSE (80) E_2	MSE (Last 20)
[10]	408	346	656
[11]	441	402	597
[12]	502	418	838
[13]	530	370	1170
[14]	577	395	1305
[15]	644	542	1052
[16]	653	351	1861
[17]	660	442	1532
[18]	676	677	672
[19]	725	222	2737
[20]	928	762	1592
[21]	954	994	794
[22]	1037	402	3577
[23]	1050	278	4138
[24]	1156	995	1800
[25]	1247	1229	1319
[26]	1425	894	3549

UMR

Evolvable Hardware

- There are many methods to design combinational circuits, generally K-maps, Quine-McCluskey methods are used
- The problem with the human designs is that they become cumbersome and problematic with the complexity of the function
- Design of circuits based on the principles of Darwinian evolution is known as *Evolvable Hardware* (EHW)
- To design conventional hardware, it is necessary to know all the specifications of the hardware functions in advance. In contrast to this, EHW can configure itself without such specifications known in advance
- One of the goals of EHW is to evolve complex designs, not achievable with the traditional design methods

“Desired” Circuit Hardware Evolution

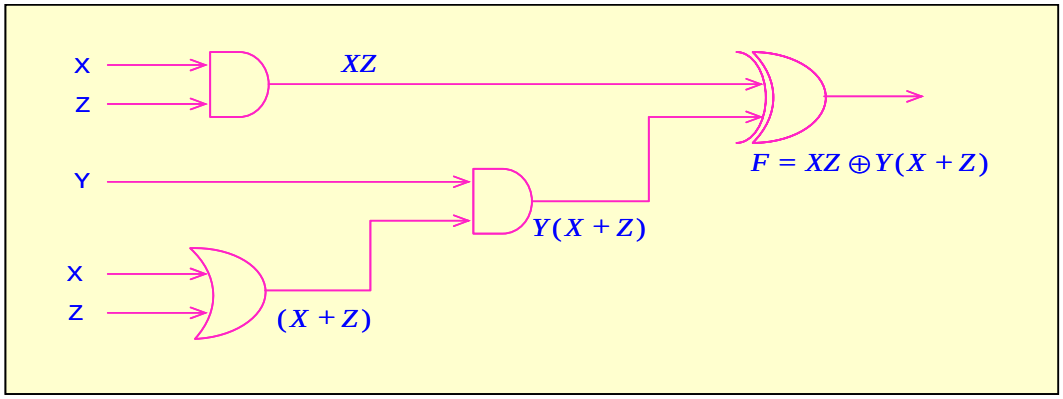


The “desired” circuit refers to the circuit required to map 100 % exactly the outputs for corresponding inputs, typically given by a truth table for a digital function, with the least number of gates.

3 input
Truth Tables for the examples taken

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

HD1	NGA	MGA	PSO
$F = Z(X \oplus Y) + Y(X \oplus Z)$	$F = Z(X + Y) \oplus (XY)$	$F = (X + Y)Z \oplus XY$	$F = XZ \oplus Y(X + Z)$
5 gates	4 gates	4 gates	4 gates
2 AND,1 OR, 2 XOR	2 AND,1 OR, 1 XOR	2 AND,1 OR, 1 XOR,	2 AND,1 OR, 1 XOR,



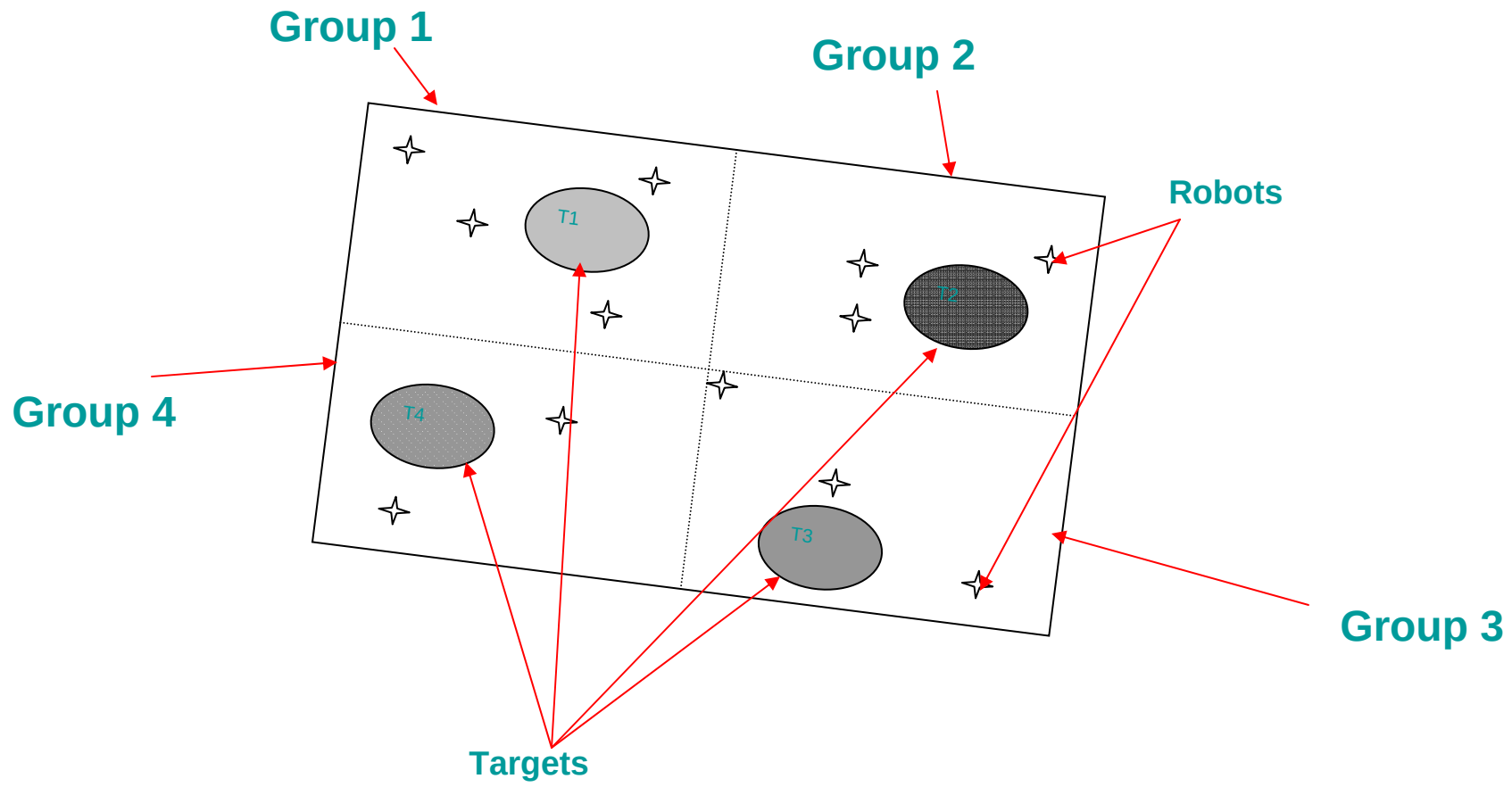
UMR

Collective Robotic Search

- Unmanned vehicles/mobile robots are used to explore environments inhospitable to humans such as remote areas, military surveillance applications, seismic activity detection, planetary exploration, toxic area exploration, etc.
- A large number of mobile robots are used for these applications.

Multiple Targets

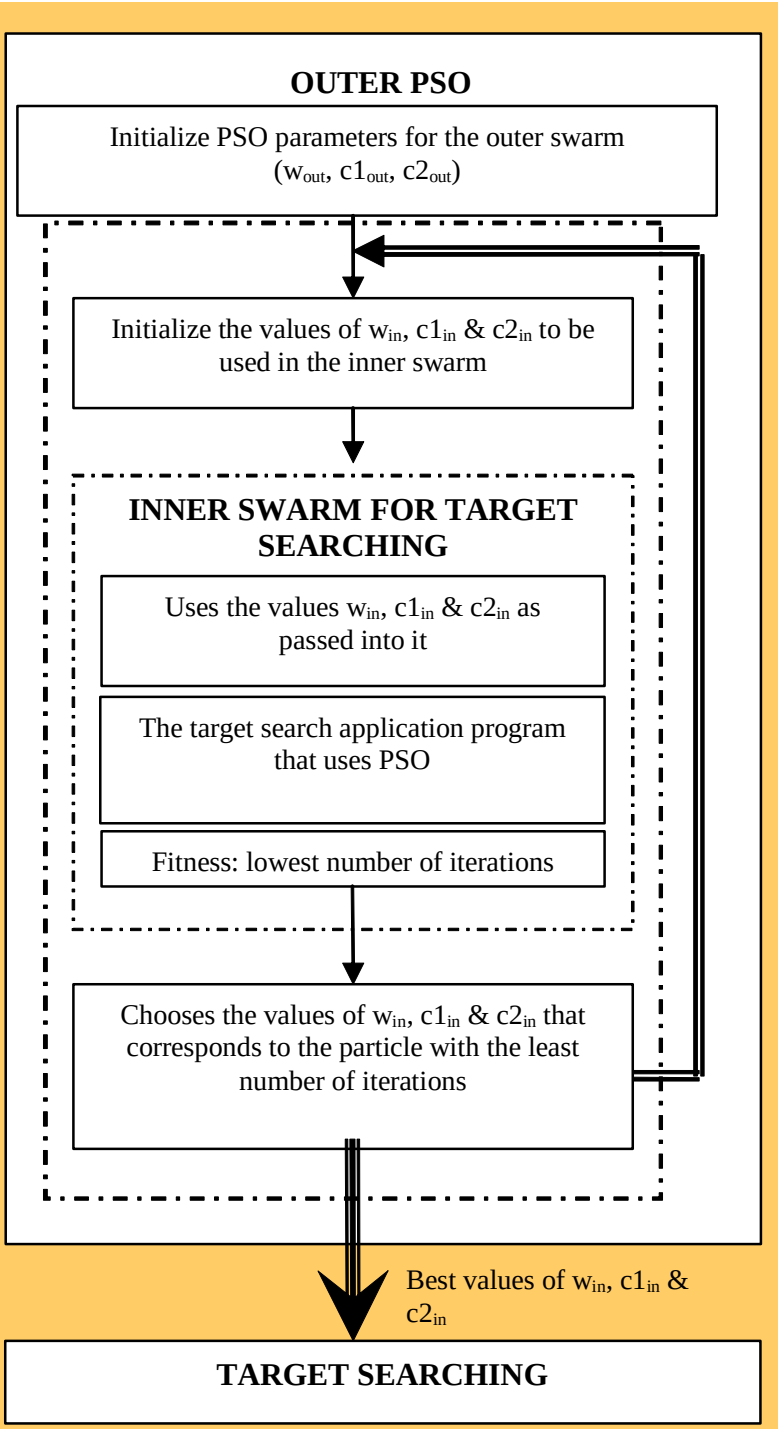
Graphical representation of a multiple target case – each robot is equipped with sensors to measure desired intensities.



UMR

Flow chart for Optimal PSO

- Optimization is done in an offline environment
- 2-level hierarchy
- Inner Swarm:
  - Specific application
  - Fitness: intensity, Euclidean distance, etc
- Outer Swarm:
  - Optimizes parameters of the application
  - Fitness: number of iterations of inner swarm



UMR

Comparison of PSO & Optimal PSO

Case	Parameter Values	# of Iterations			
Single Target (Unoptimized)	w=0.6 c_1=0.5 c_2=2	70.7			
Single Target (Optimized)	w =0.45 c_1 =0.60 c_2 =1.50	39.5			
Multiple Targets (Unoptimized)	w=0.6 c_1=0.5 c_2=2	137	144	138	139
Multiple Targets (Optimized)	w =0.55 c_1 =0.55 c_2=2.23	123	128	112	124

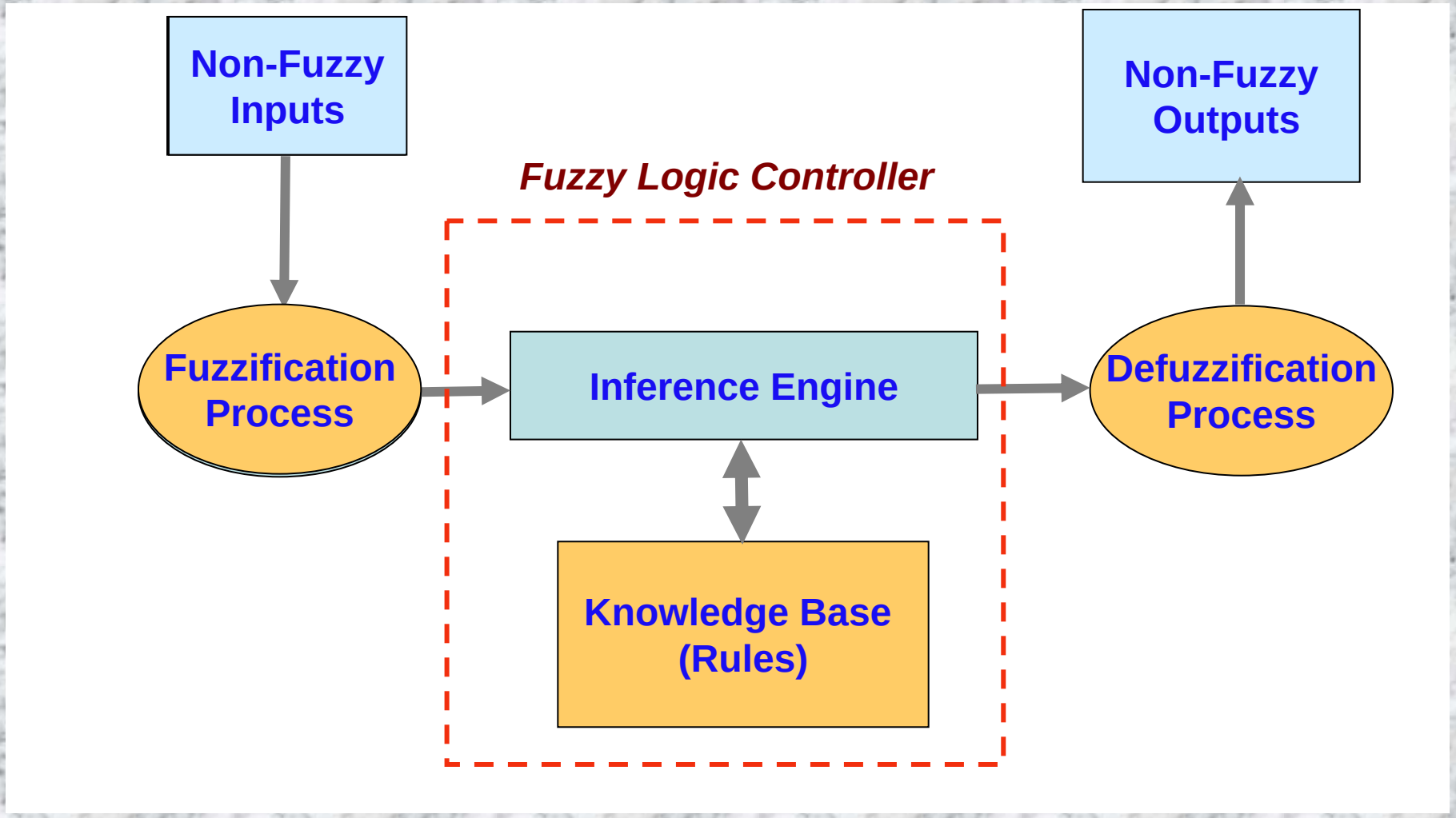
UMR

Comparison PSO, DE and DEPSO

Case	Evaluations	Convergence
PSO	11796	100% (84%)
DE	3624	100% (94%)
DEPSO	5723	100% (100%)

UMR

Fuzzy Logic Control



UMR

The Fuzzy System

- Equations:
  - $P_i = (X_i + \Delta X_i , Y_i + \Delta Y_i)$
    - $\Delta X_i = f(I_i, Gdx)$
    - $\Delta Y_i = f(I_i, Gdy)$

Where,

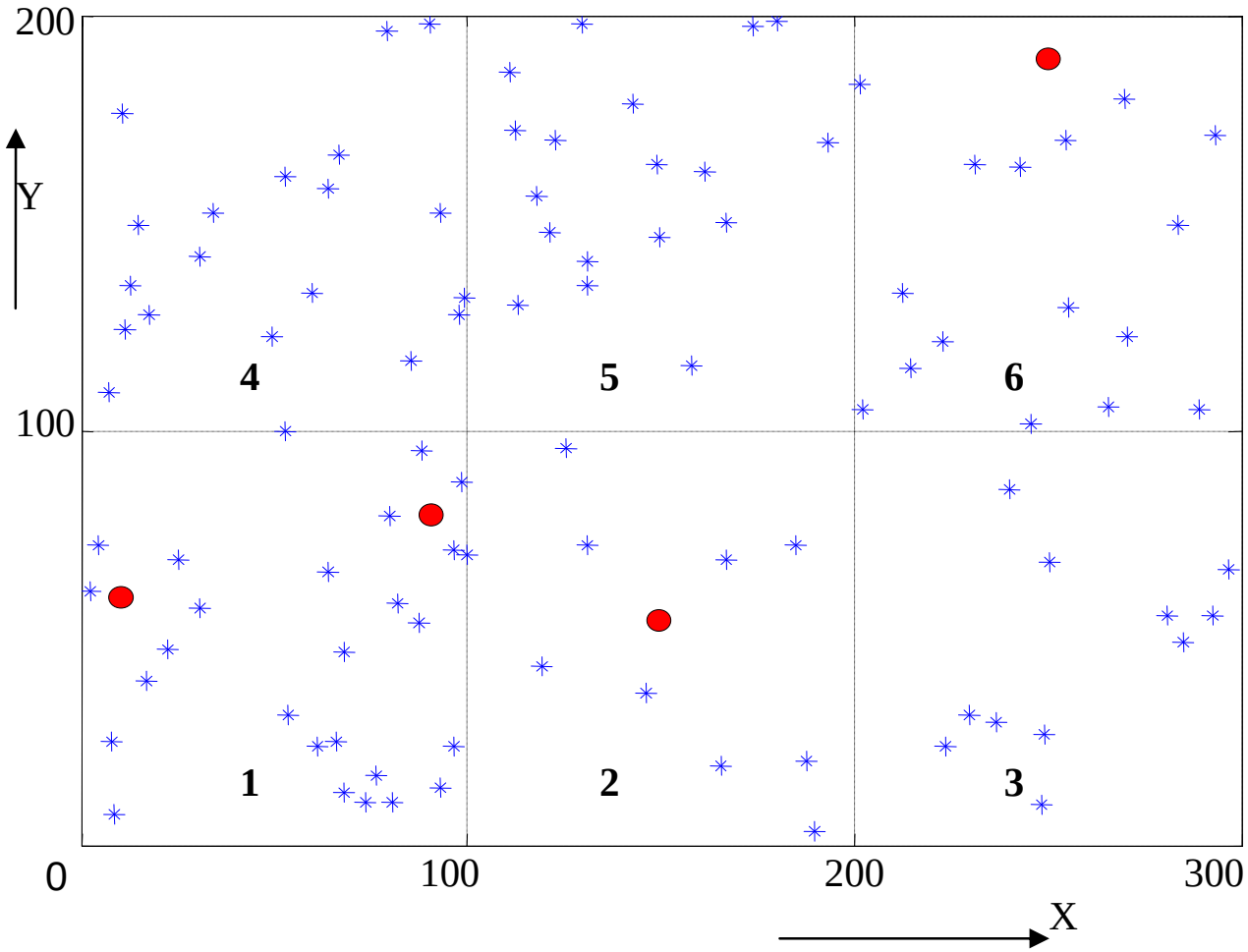
- $Gdx = L_{best}x - Px$ and $Gdy = L_{best}y - Py$
- Px and Py are the X-Y coordinates of the sensors current position
- $L_{best}x$ and $L_{best}y$ are the X-Y coordinates of the L_{best} of the swarm.

UMR

Optimized Fuzzy System

- PSO was used to optimize the Fuzzy Parameters
  - Membership Functions
  - Coarse and Fine Rule Set
    - IF (*a set of conditions is satisfied*), then (*a set of consequences can be inferred*).

Problem Search Space



Results for PSO Based Navigation

w	c_1	c_2	# of Iterations	Time (sec)	Convergence
0.8	2	2	1181.58	6.35	0%
	0.5	2	1106.09	5.88	0%
	2	0.5	985.39	4.98	26%
	0.5	0.5	924.10	3.99	93%
0.6	2	2	910.60	4.90	94%
	0.5	2	859.40	4.79	99%
	2	0.5	889.36	5.07	32%
	0.5	0.5	841.85	5.24	8%

UMR

Results for the Fuzzy Based Navigation (100% Convergence)

Simulation case study	Membership Function	Coarse Rule Set	Fine Rule set	Iterations	Time (sec)
1	Original (heuristics)	Original (heuristics)	Original (heuristics)	**355.80**	**349.97**
2	PSO	Original (heuristics)	Original (heuristics)	**308.41**	**328.24**
3	Original (heuristics)	PSO	Original (heuristics)	**356.96**	**356.18**
4	Original (heuristics)	Original (heuristics)	PSO	**307.44**	**325.65**
5	PSO	PSO	PSO	**306.15**	**326.78**

UMR

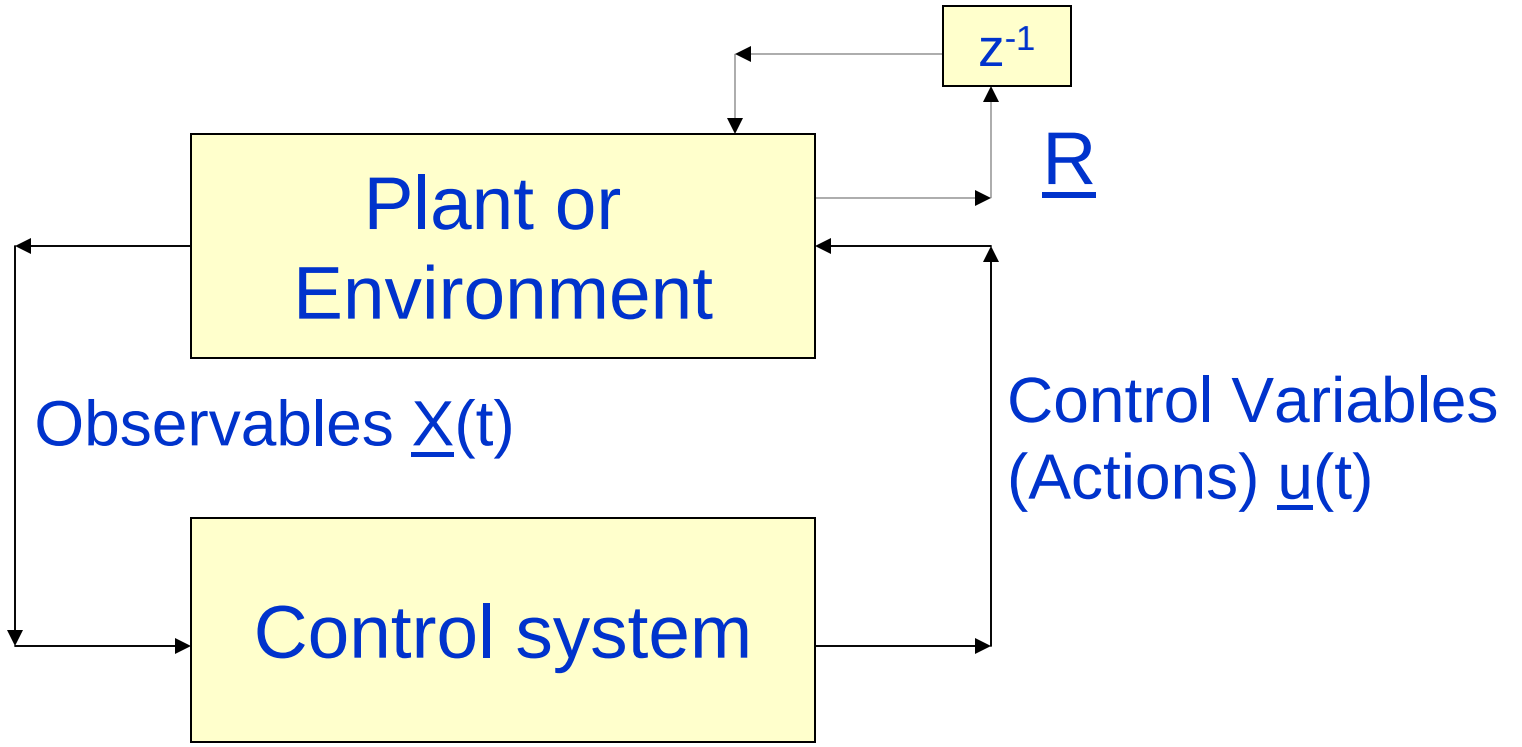
Nonlinear System Identification and Control & Applications

Ganesh Kumar Venayagamoorthy, PhD

Associate Professor of Electrical and Computer Engineering
& Director of Real-Time Power and Intelligent Systems Laboratory
University of Missouri-Rolla, USA

http://www.umr.edu/~ganeshv
www.ece.umr.edu/RTPIS
gkumar@ieee.org

What is Control?



- t may be discrete (0, 1, 2, ...) or continuous
- "Decisions" may involve multiple time scales

What is Intelligent Control?

Intelligent Control is a form of control is defined as the ability of a system to **comprehend**, **reason**, and **learn** about

- processes
- disturbances and
- operating conditions.

UMR

What are the Goals of Intelligent Control?

The fundamental goals of intelligent control may be described as follows:

- Full utilization of knowledge of a system and/or feedback from a system to provide reliable control in accordance with some preassigned performance criterion
- Use of the knowledge to control the system in an intelligent manner, as a human expert may function in light of the same knowledge
- Improved ability to control the system over time through the accumulation of experiential knowledge (i.e., learning from experience).

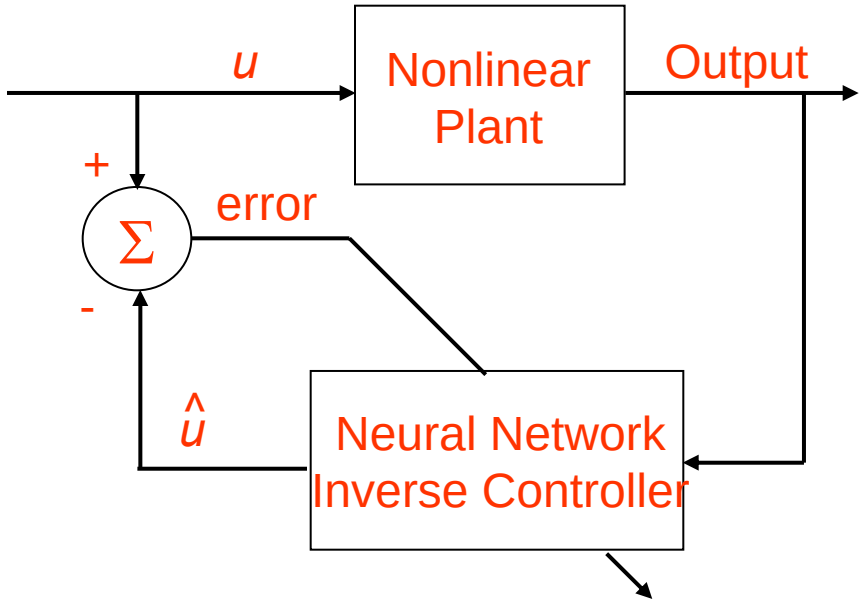
UMR

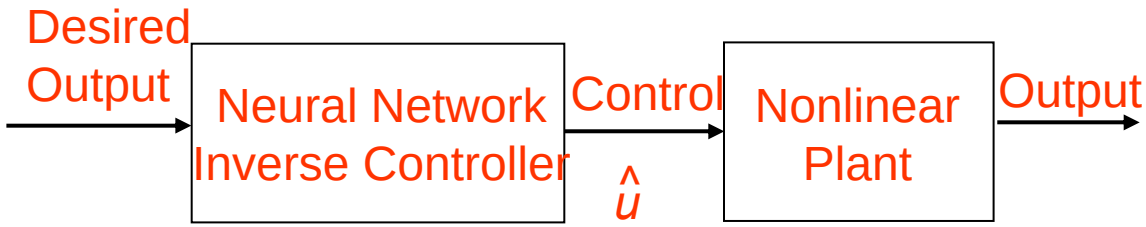
Neural Network Controller Design Approaches

Neural Network Controller designs fall mostly into the following five categories:

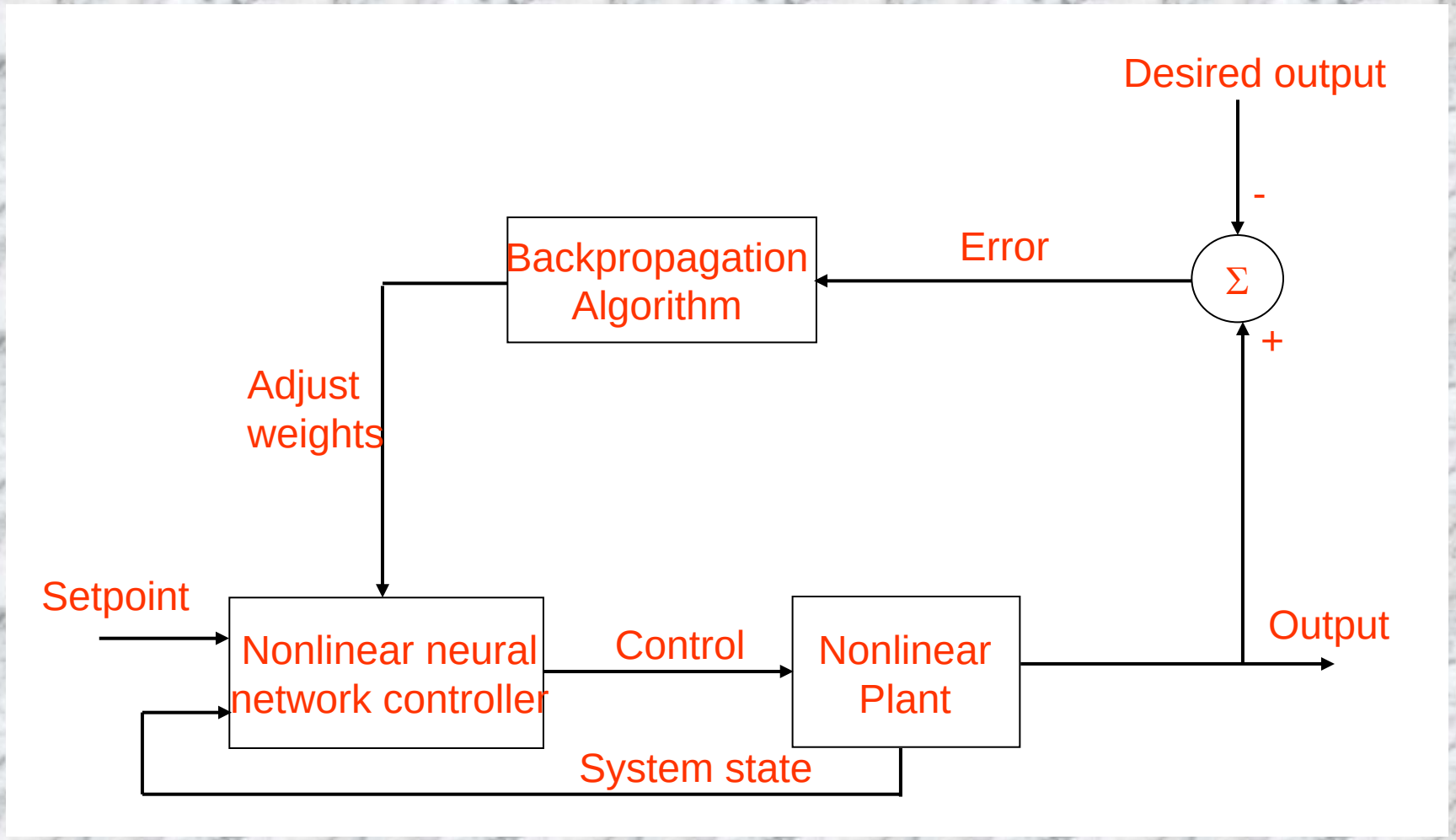
- Supervised Control
- Direct Inverse Control
- Neural Adaptive Control
- Backpropagation Through Time (BPTT)
- Adaptive Critic Designs (ACDs)

Direct Inverse Control

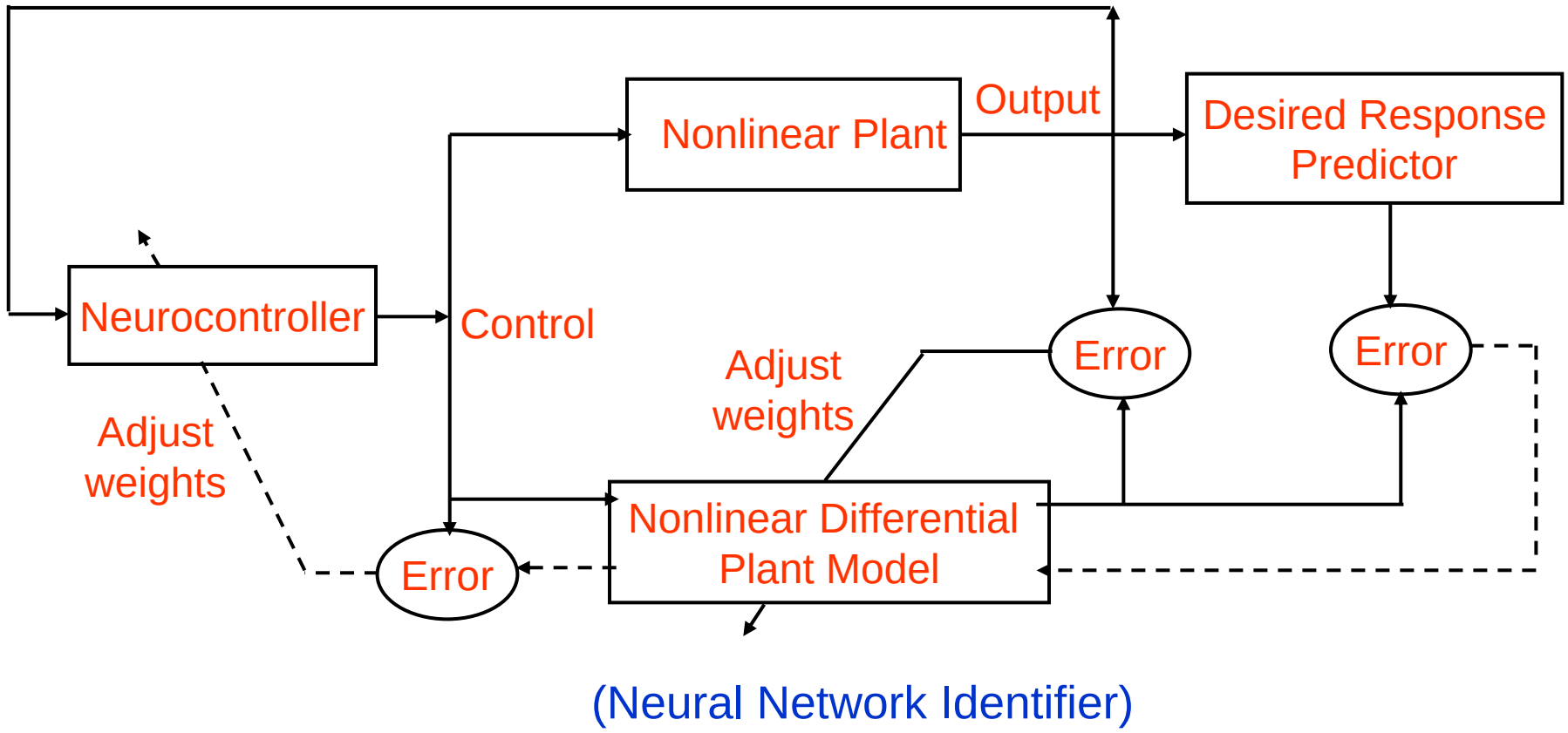




Neural Direct Adaptive Control

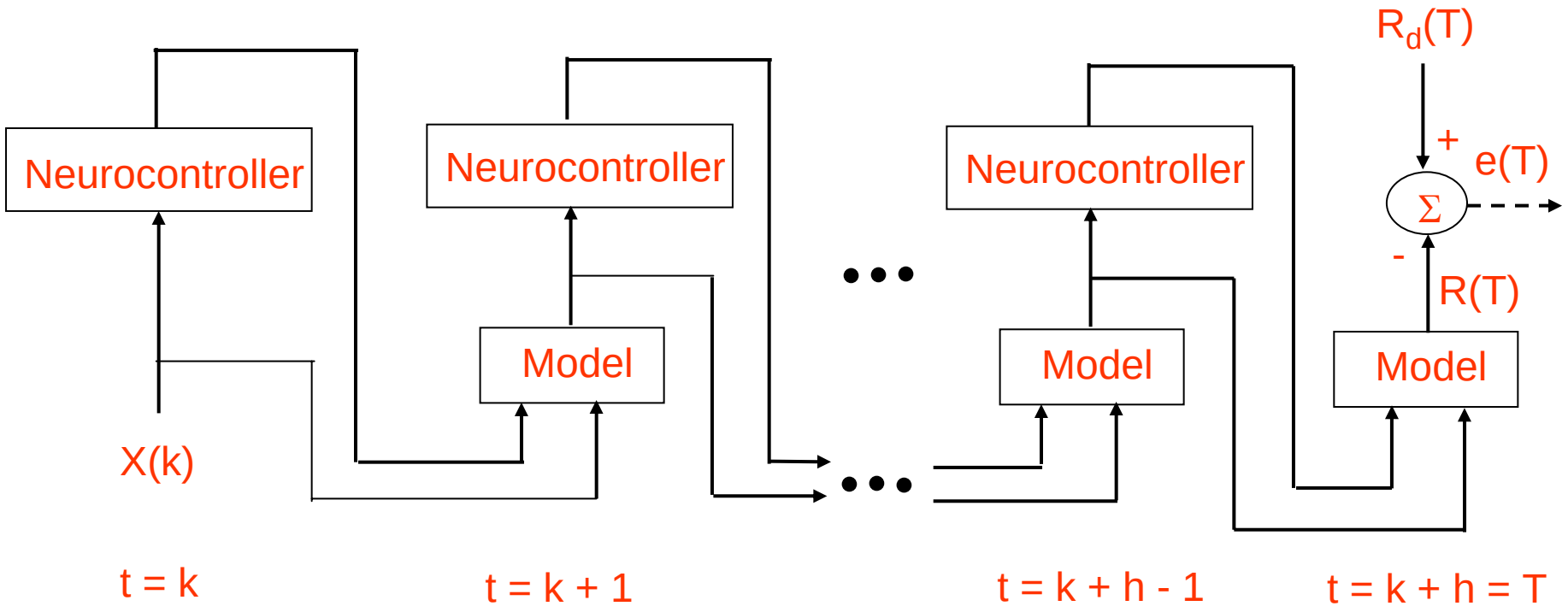


Neural Indirect Adaptive Control

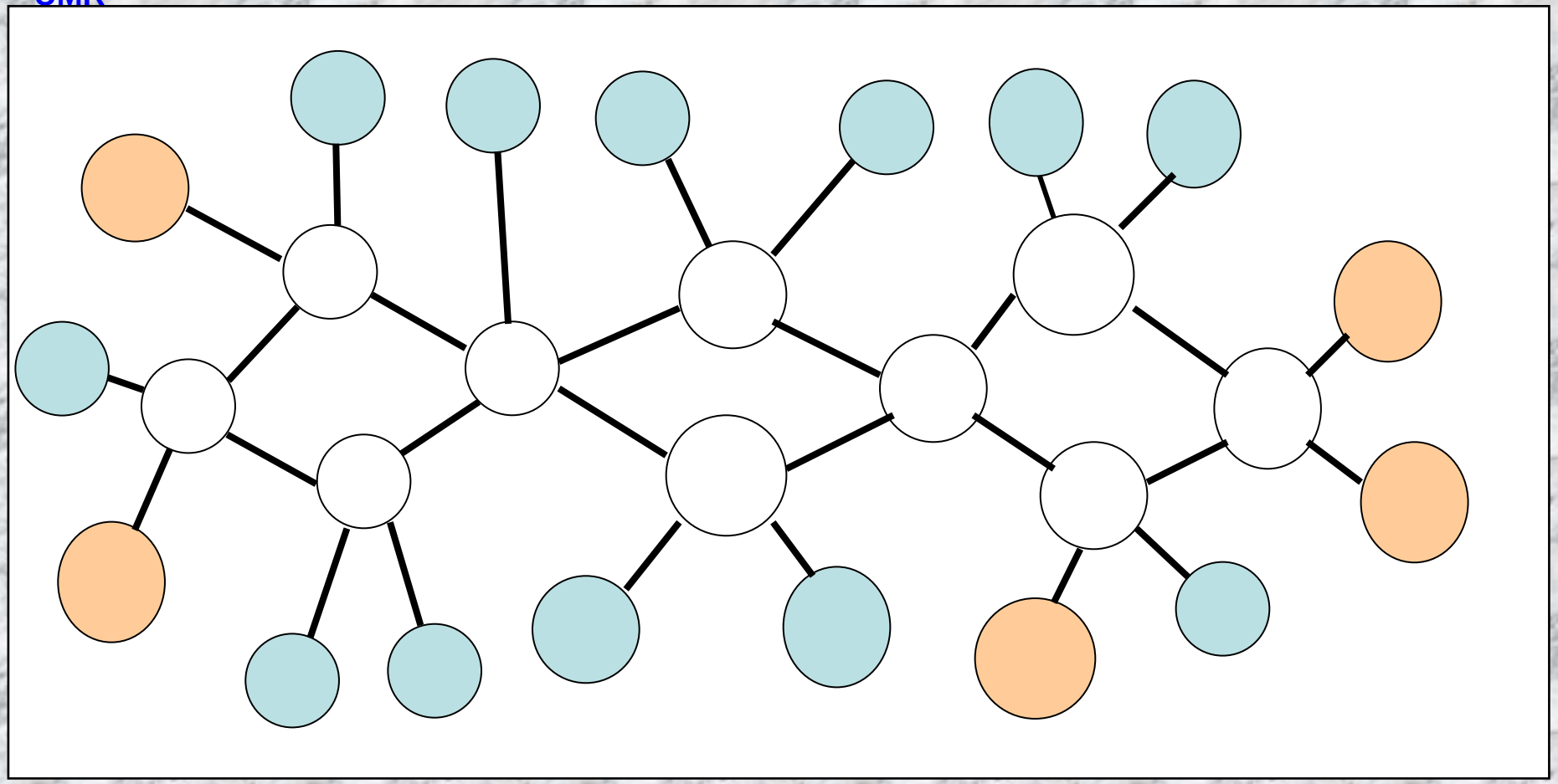


(Neural Network Identifier)

Backpropagated Through Time (BPTT) Based Neurocontrol



Power Grid



Generators

Loads

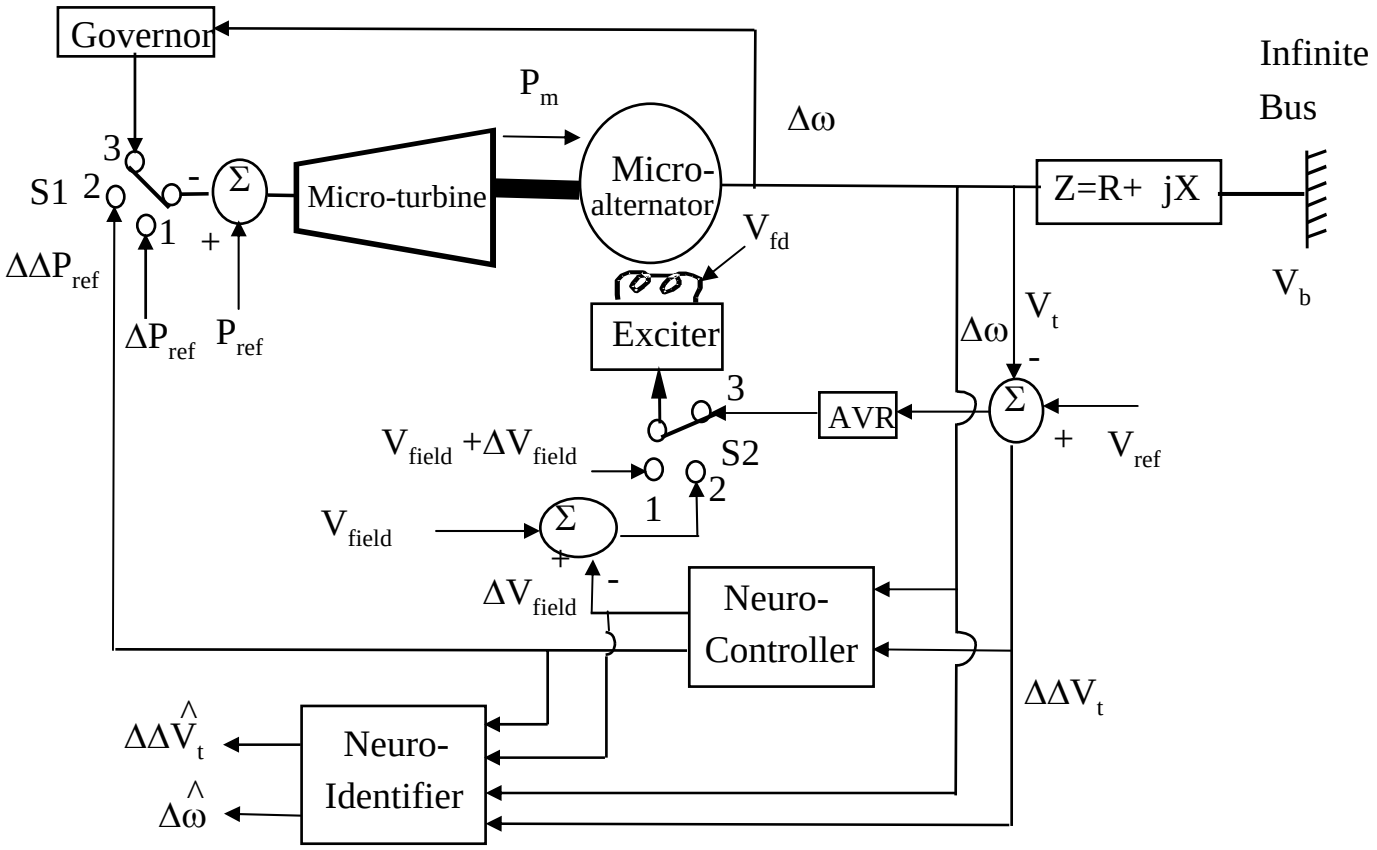
Buses

Transmission Lines

A CONTINUALLY ONLINE TRAINED NEUROCONTROLLER FOR EXCITATION AND TURBINE CONTROL OF A TURBOGENERATOR

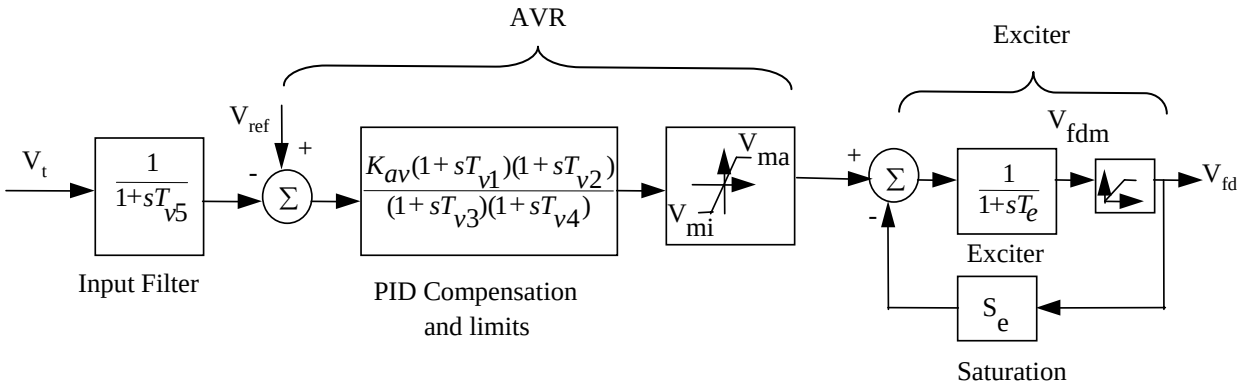
***IEEE Transactions on Energy Conversion*,**
vol. 16, no.3, September 2001, pp. 261-269.

Power System Model

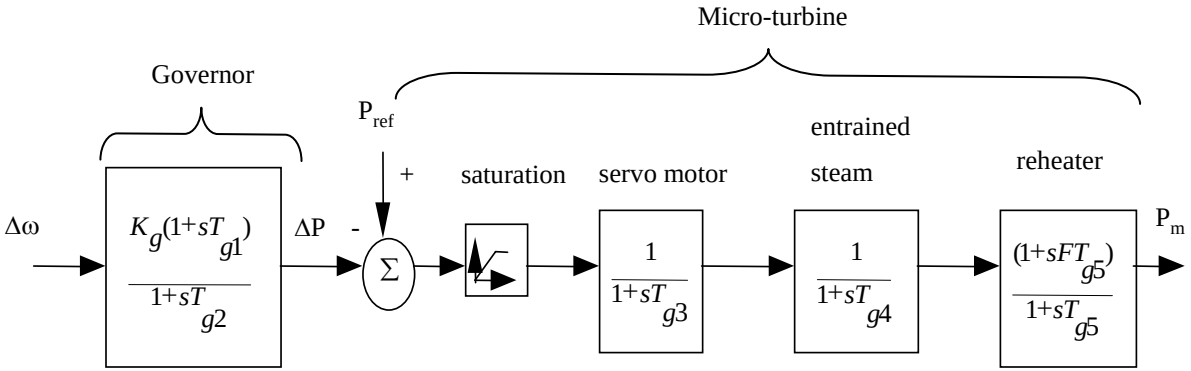


Conventional Controllers

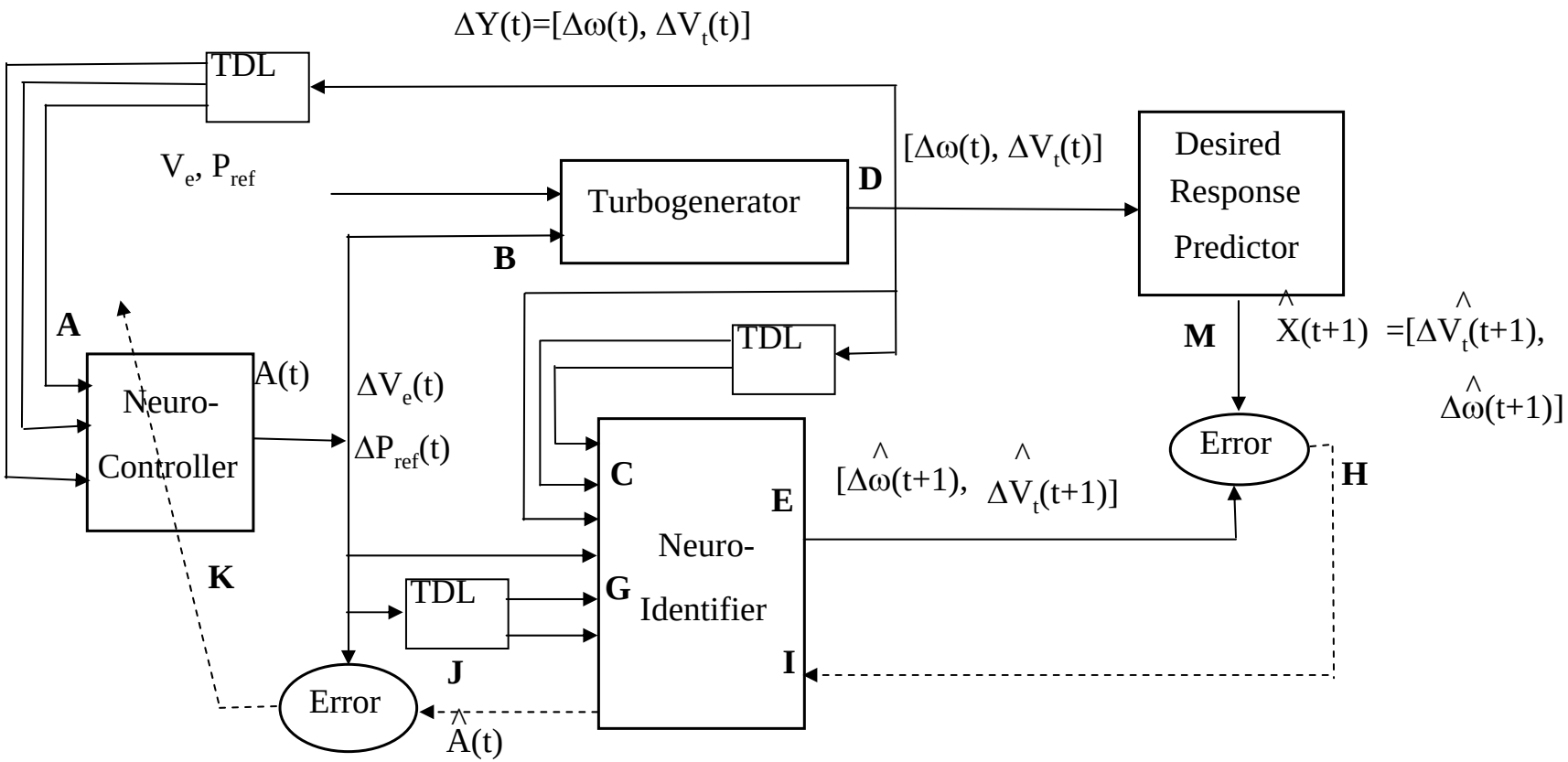
Automatic voltage regulator and exciter combination



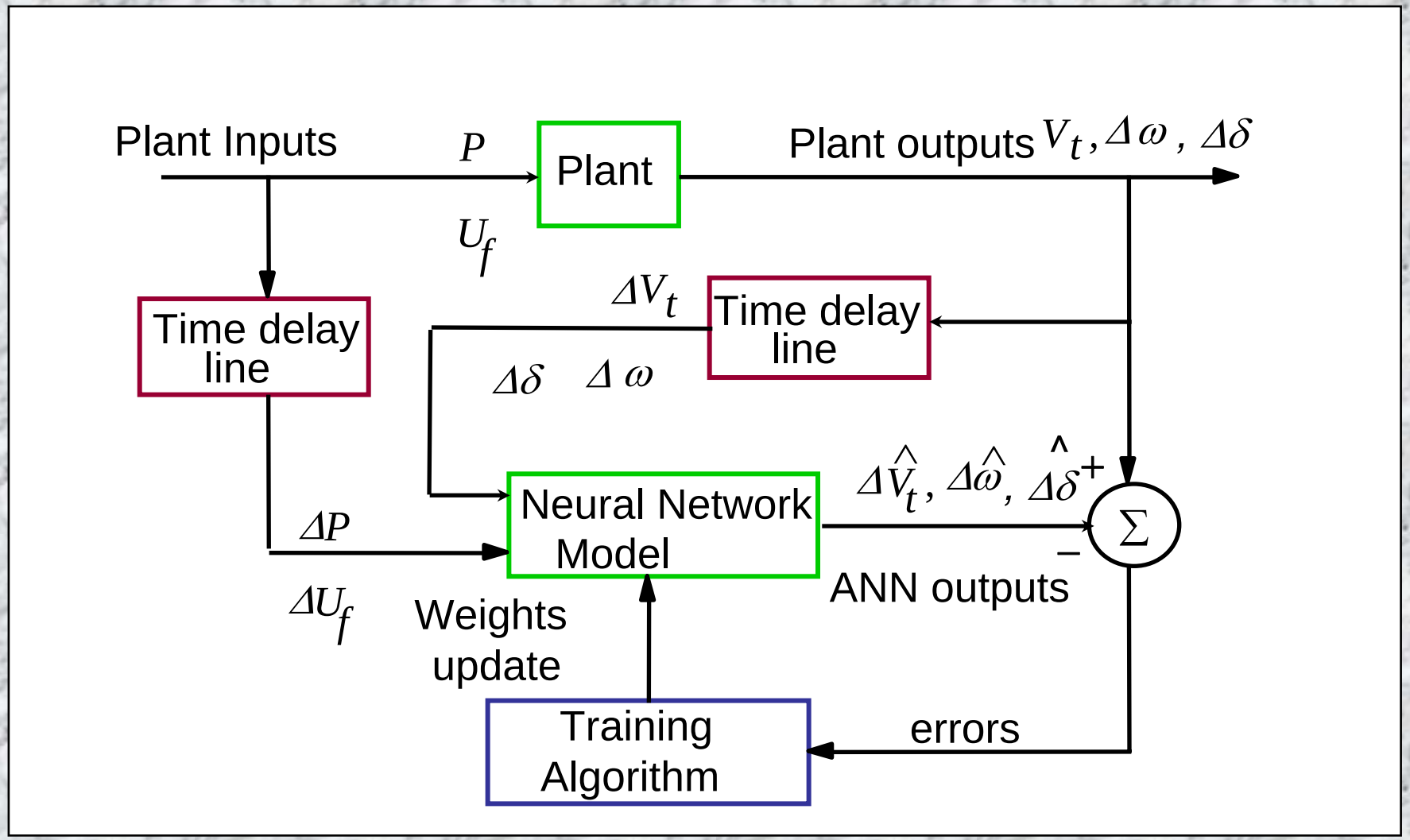
Micro-turbine and governor combination



Adaptive Neurocontroller



ANN Identifier/Model



ANN Identifier/Model

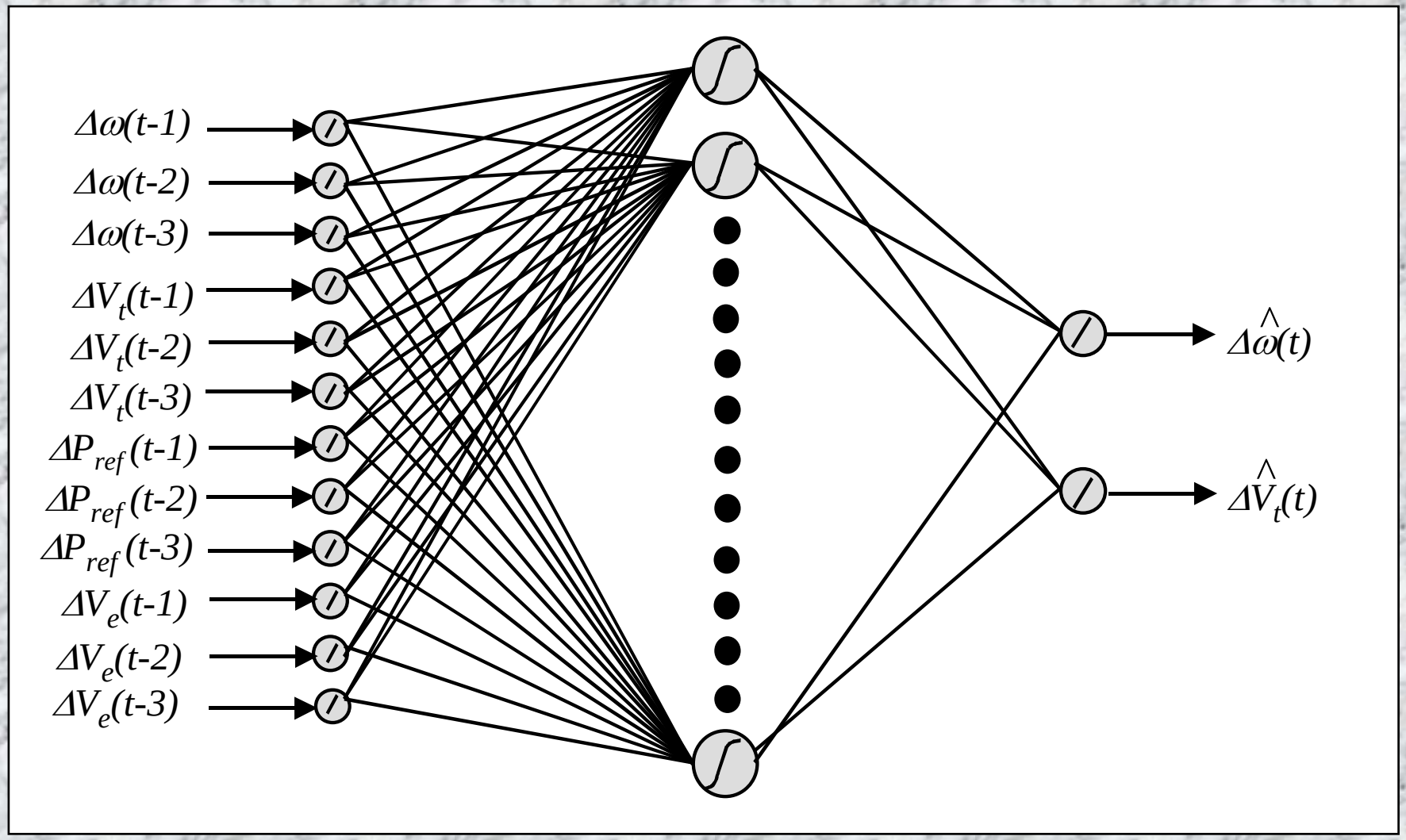
Series-parallel Nonlinear Auto Regressive Moving Average (NARMA) model

$$\hat{y}(t+1) = f\begin{bmatrix} y(t),\ y(t-1),\ \ldots,\ y(t-n+1), \\ u(t), u(t-1), \ldots, u(t-m+1) \end{bmatrix}$$

The NARMA model has been chosen in preference to other system identification models because online learning is desired to identify the dynamics of the power systems (as shown in the following slides).

UMR

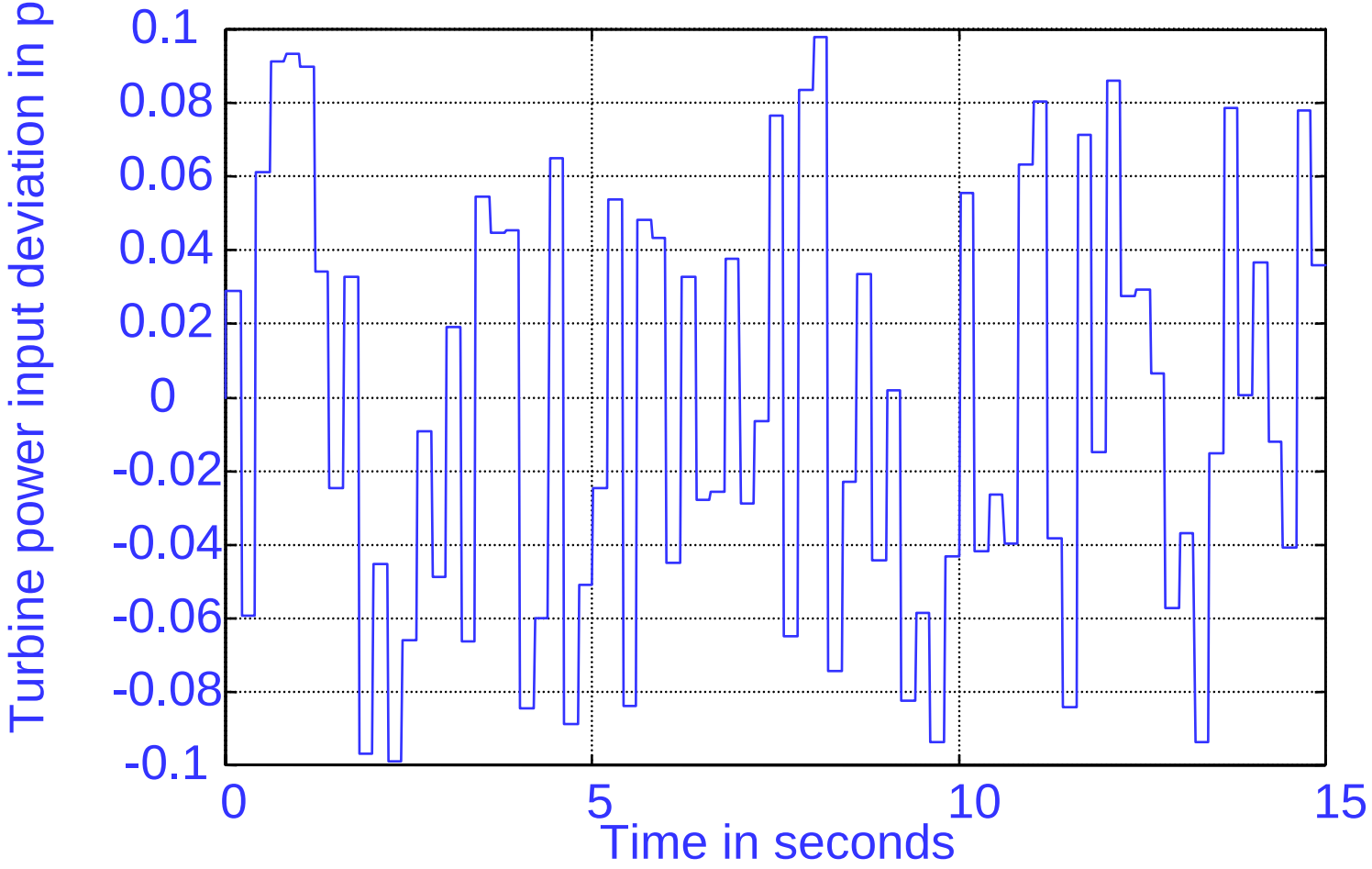
ANN Identifier/Model



UMR

Training Signals

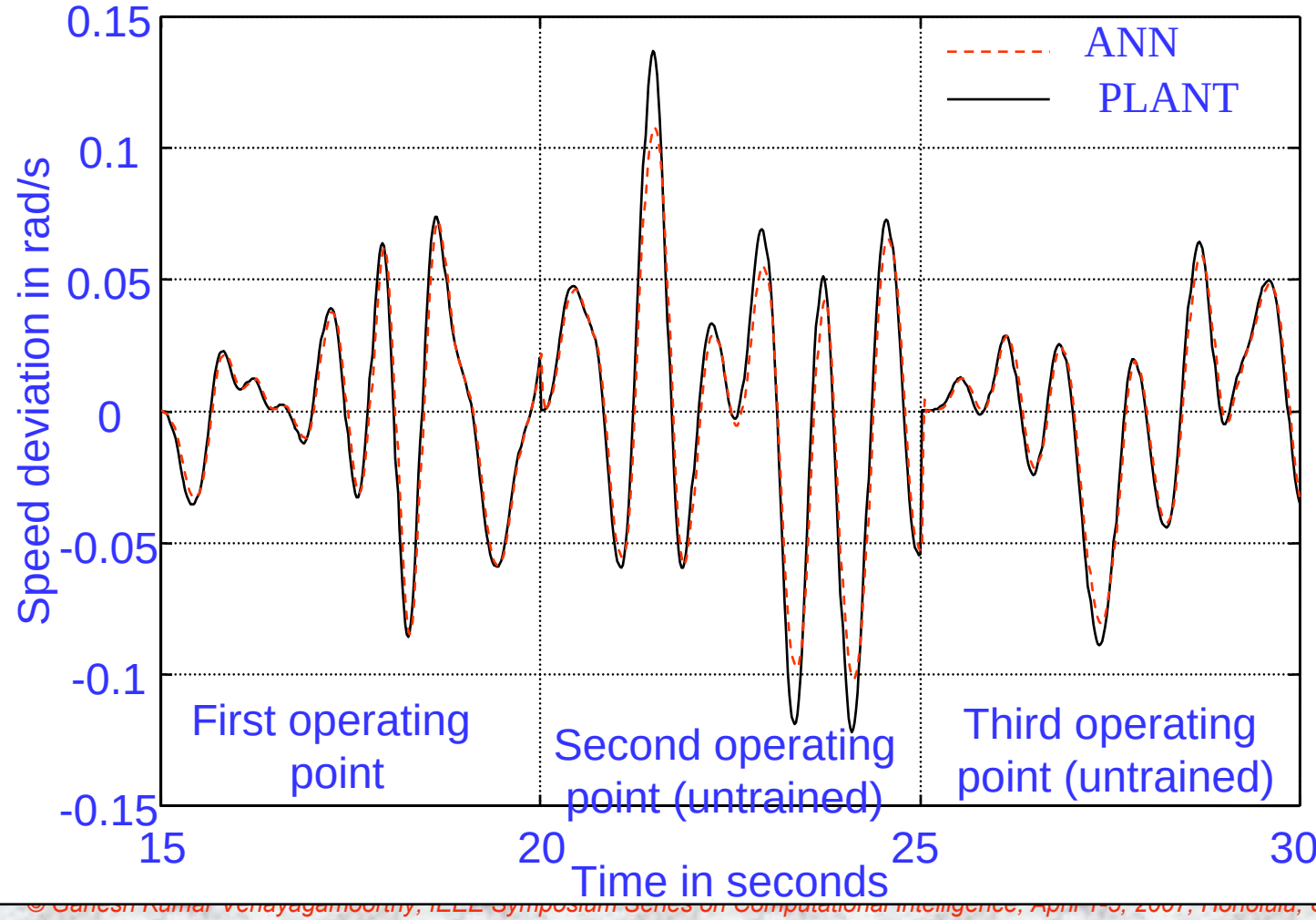
Power deviations at inputs to the turbine and the ANN



UMR

Plant and ANN Outputs

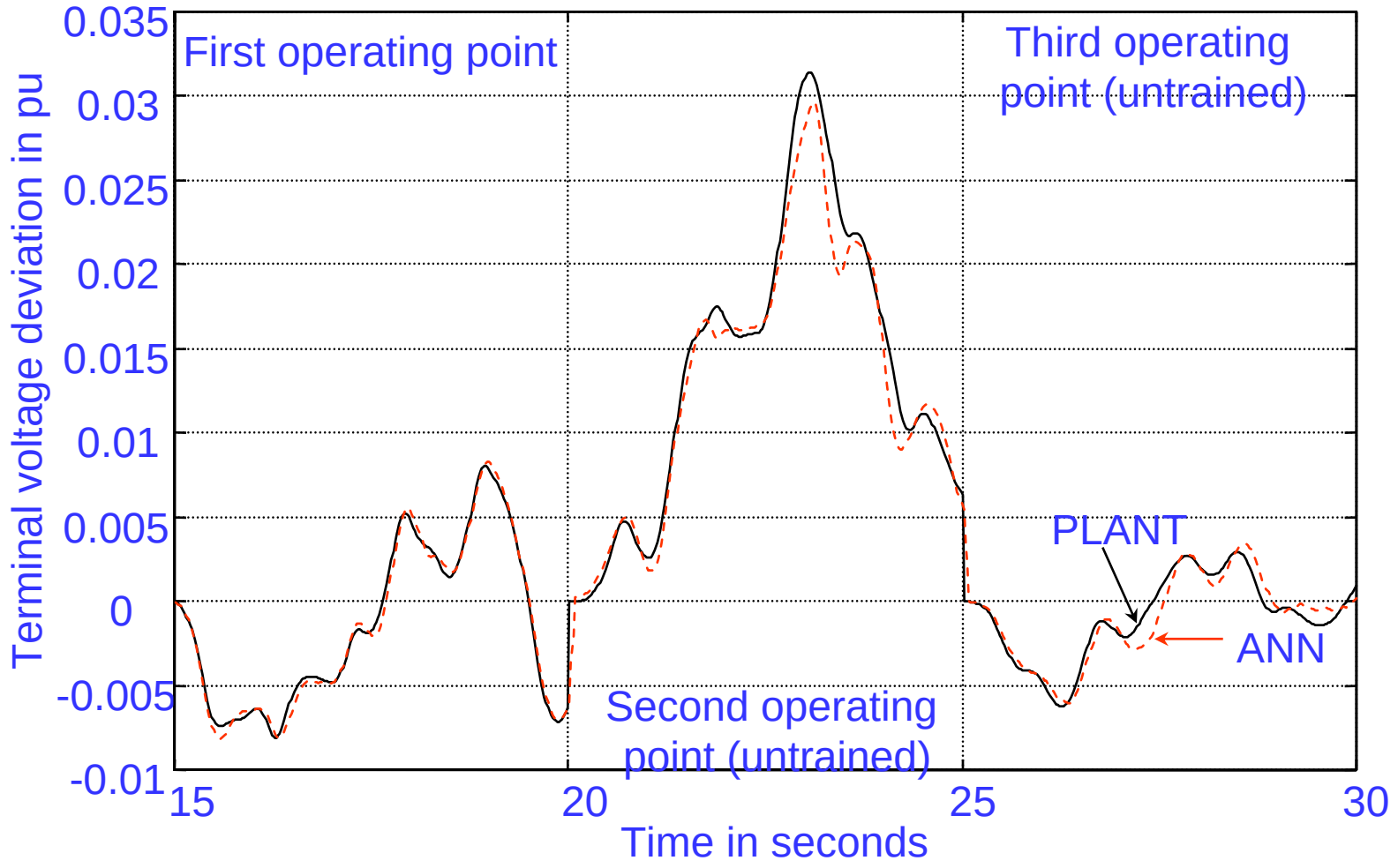
Speed deviations at outputs of the PLANT and the ANN



UMR

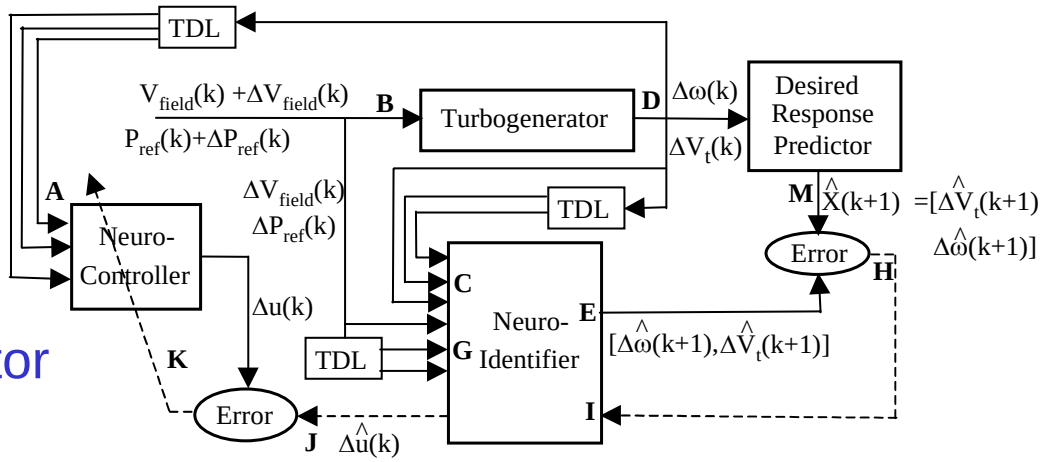
Plant and ANN Outputs

Voltage deviations at outputs of the PLANT and the ANN



Pre-training of the Neurocontroller

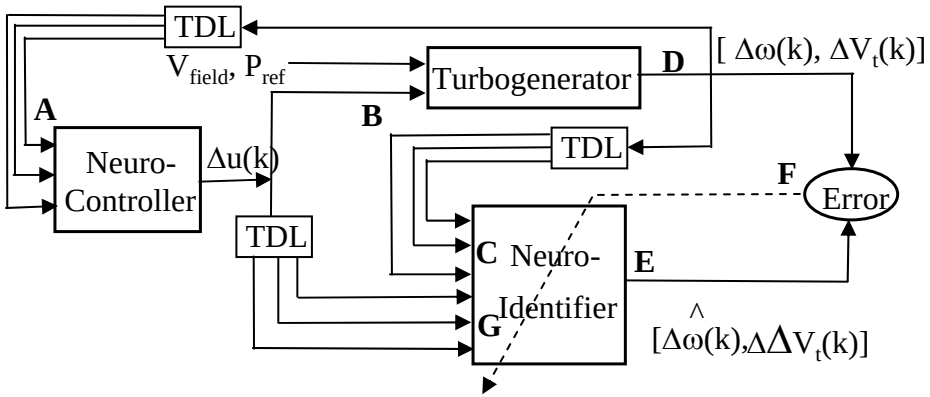
- Inputs to the Neurocontroller
  - Delayed values of Terminal voltage deviation ΔV_t
  - Delayed values of Speed deviation $\Delta\omega$
  - Outputs of the Neurocontroller
  - Deviation in Field voltage ΔV_{field}
  - Deviation in Turbine Power ΔP_{ref}
- Neuroidentifier weights
  - Fixed
  - Backpropagation of errors at H
- Desired Response Predictor



Post-control Training

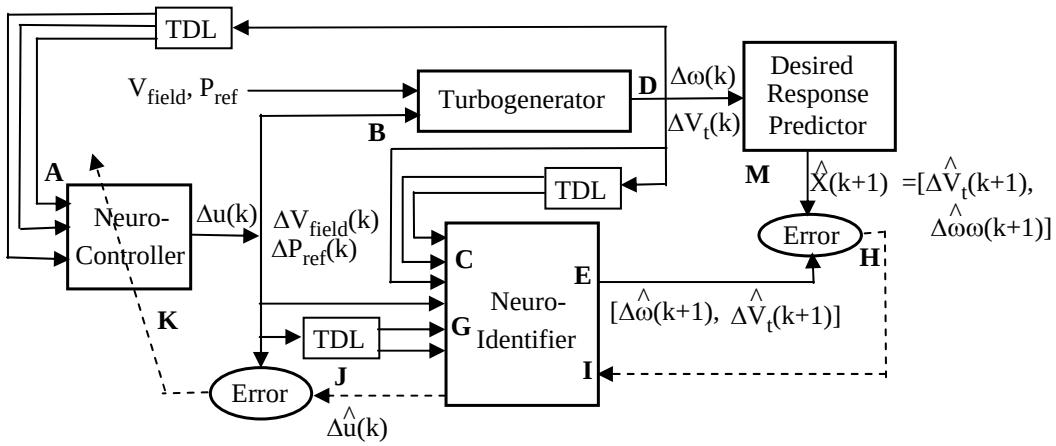
- Online training continues
- Three procedures are carried out every sampling period:
  - Training the neuroidentifier
  - Training the neurocontroller
  - Controlling the turbogenerator

- First Procedure: Training the Neuroidentifier



Post-control Training

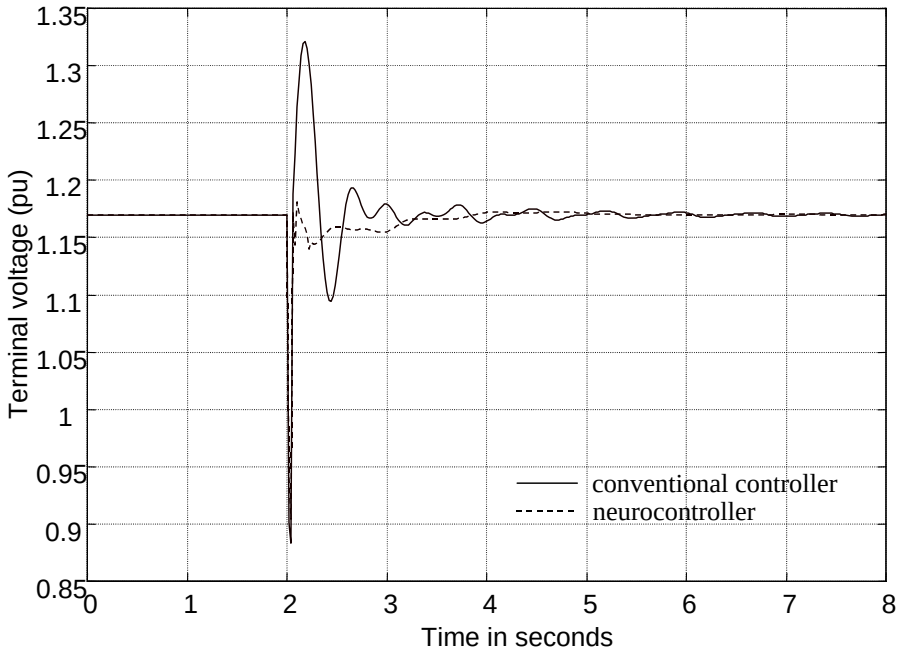
- Second Procedure: Training the Neurocontroller

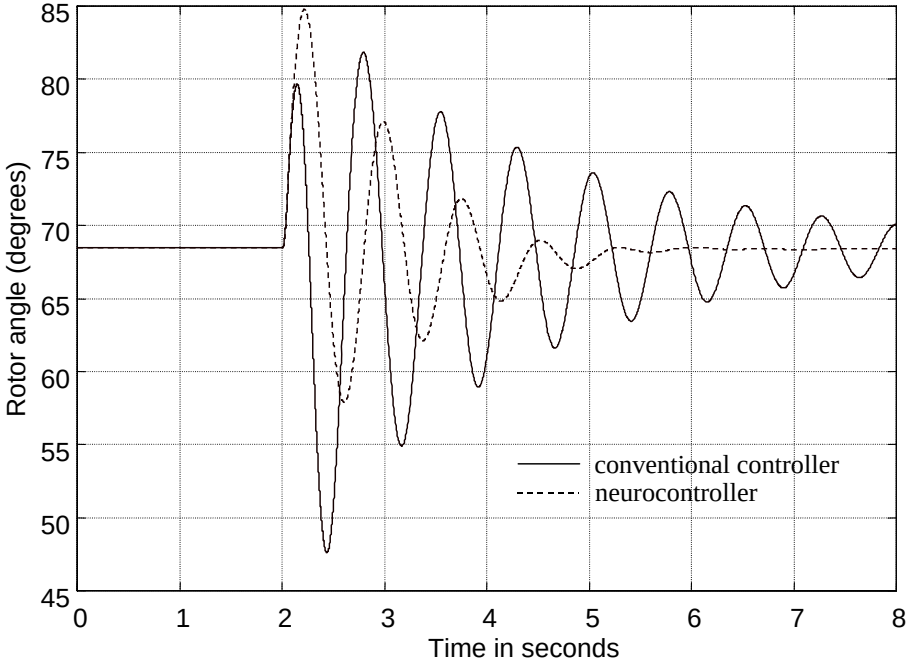


- Third Procedure: Controlling the turbogenerator
  - New control signals Δu are calculated using the updated weights from the second procedure and are applied at time (k+1) to the turbogenerator at B.

Simulation Results: Three Phase Short Circuit at the Infinite Bus

- The short circuit test is carried out at: Z = 0.025 + j 0.6 at P = 1 pu & Q = 0.62 pu





Adaptive Critic Designs and Applications in Power Systems

Ganesh Kumar Venayagamoorthy, PhD

Associate Professor of Electrical and Computer Engineering
& Director of Real-Time Power and Intelligent Systems Laboratory
University of Missouri-Rolla, USA

http://www.umr.edu/~ganeshv
www.ece.umr.edu/RTPIS
gkumar@ieee.org

UMR

Adaptive Critic's Based Neurocontrol

- The Adaptive critic designs have the potential of replicating critical aspects of human intelligence:
  - ability to cope with a large number of variables in parallel, in real time, in a noisy nonlinear non-stationary environment.

- The ACDs show a family of promising methods to solve optimal control problems.

- The origins of ACDs are ideas synthesized from dynamic programming, reinforcement learning and backpropagation.

What is Reinforcement Learning?

- Learning from interaction – theories of learning and intelligence.
- Learning is an *active process*.
- Goal-oriented learning than other approaches of ML.
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal

Reinforcement Learning

- Reinforcement learning is defined by Barto as follows:

  If an action taken by a learning system is followed by a satisfactory state of affairs, then the tendency of the system to produce that particular action is strengthen or reinforced. Otherwise, the tendency of the system to produce that action is weaken.

  Reinforcement Learning is a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives when interacting with a complex, uncertain environment.

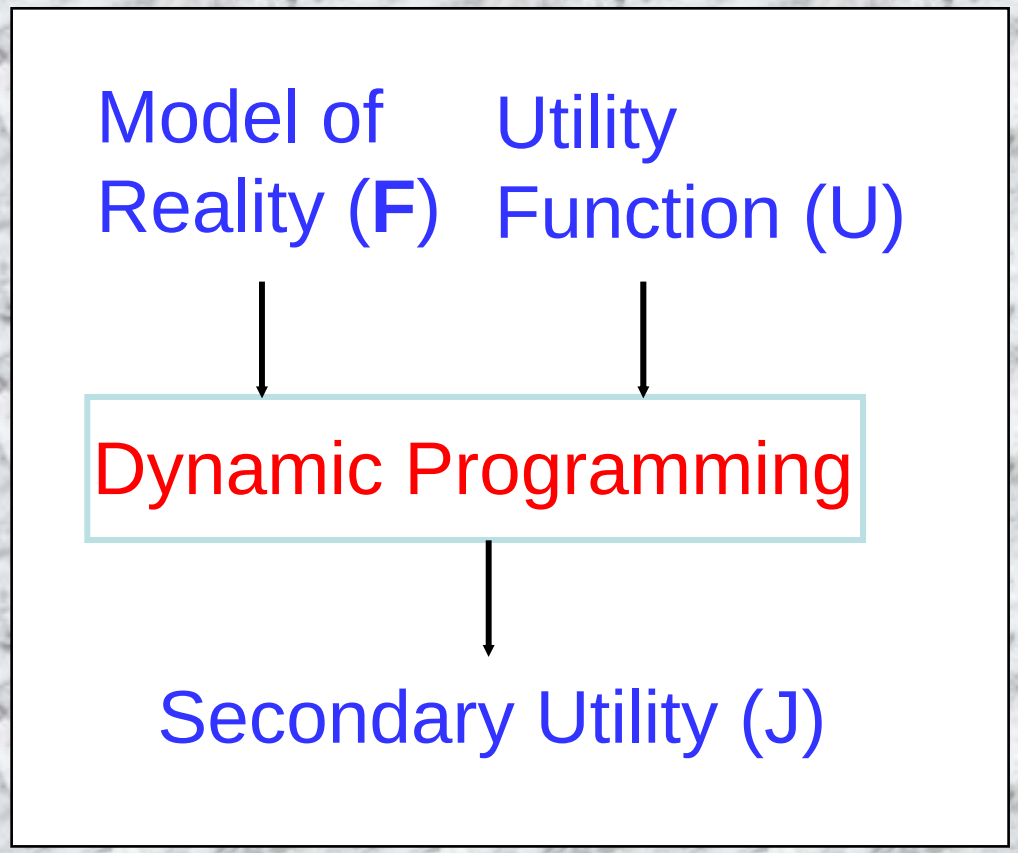
- There are two types of reinforcement learning: non-associative and associative.

- The simplest and most frequently used reinforcement learning methods is the *Q – learning* (Watkins).

- Reinforcement learning control system may be used where the correct actions are not known.

Dynamic Programming

- The basic concept of all forms of dynamic programming can be summarized as follows:



Dynamic Programming

- Bellman's equation of dynamic programming

$$J(t) = \sum_{k=0}^{\infty} \gamma^{k} U(t+k)$$

- Approximate dynamic programming is obtained using a neural network called 'the Critic network' to approximate the J - function.

ACDs as Supervised and Reinforcement Learning

Types of primary reinforcement:

- 1) Explicit targets for system outputs are provided at every step.
- 2) Explicit differentiable cost as a function of system variables is provided at every step.
- 3) A graded cost is provided at each step but explicit relationship with system states is not given.
- 4) Ungraded reinforcement is provided when appropriate, e.g. binary outcome at the end of a game.

ACDs are supervised learning systems in the cases of (1) & (2).
They are reinforcement learning systems in the cases of (3) & (4).
Critic may be thought of as a transformer of cases (3) & (4) to case (2).

Adaptive Critic Designs

- A family of adaptive critic designs was proposed by Werbos in 1977 as a new optimization technique combining concepts of reinforcement learning and approximate dynamic programming.
- The adaptive critic method determines optimal control laws for a system by successively adapting two neural networks, namely an *Action neural network* (which dispenses control signals) and a *Critic neural network* (which learns the desired performance index for some function associated with the performance index).
- These two neural networks approximate the Hamilton-Jacobi-Bellman equation associated with optimal control theory.

Summary: The goals of Intelligent Control with ACDs

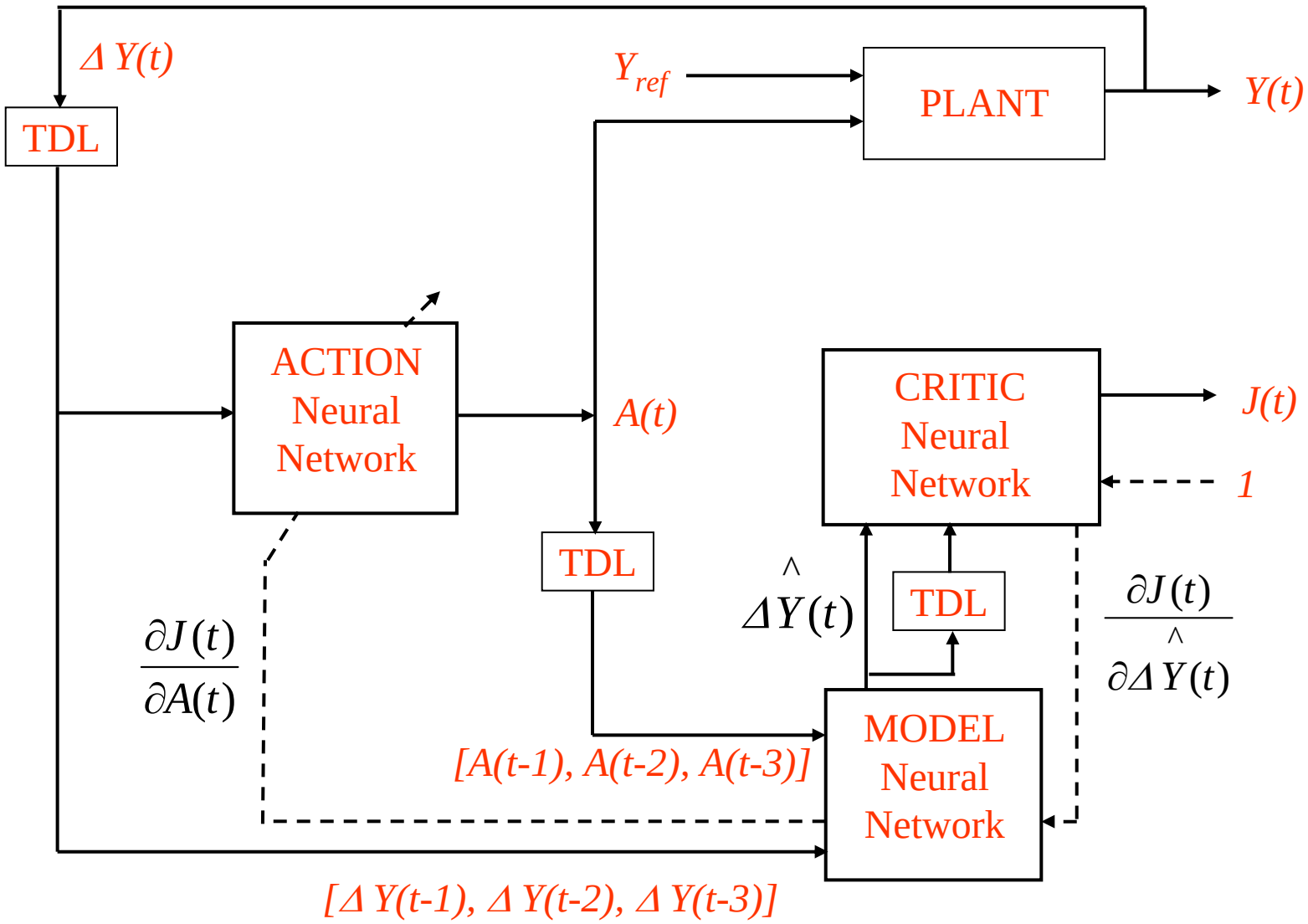
- Adaptive Critic designs, which are rarely studied but very powerful design techniques that give brain-like intelligence to controllers
  - MIMO system
  - Nonlinear system
  - Model uncertainties
  - Random disturbance
  - Learns over time
  - Adaptive
  - Robustness (Hamilton-Jacobi-Bellman equation, basic equation of stochastic optimal control)

UMR

Adaptive Critic Designs

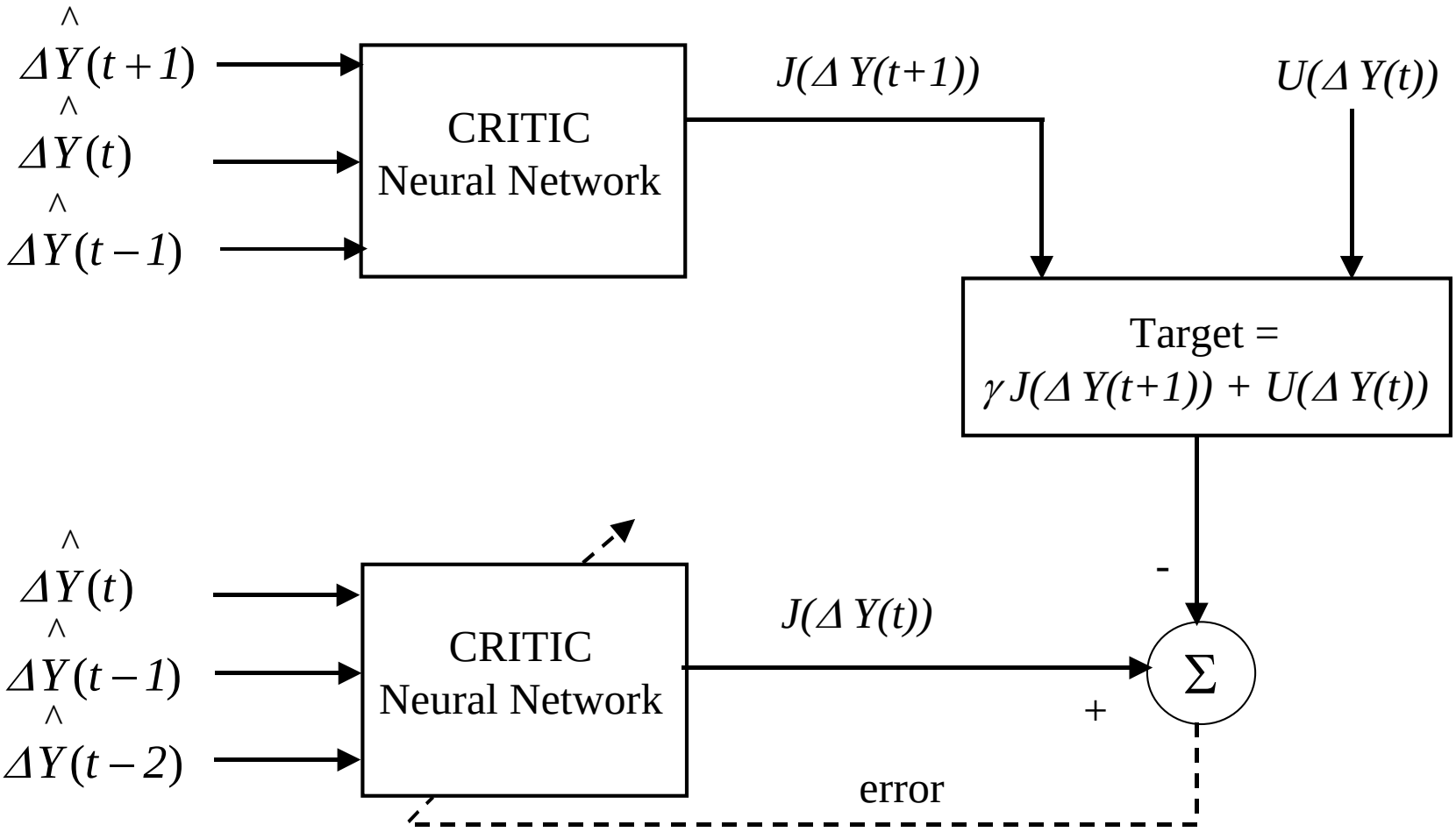
- To actually build an adaptive critic control system, the following will have to be decided:
  - Exactly what the Critic network is supposed to approximate, and how it will adapted;
  - How the Action network will be adapted in response to the information coming out of the Critic network.

ANN Controller based on Adaptive Critic Designs - HDP

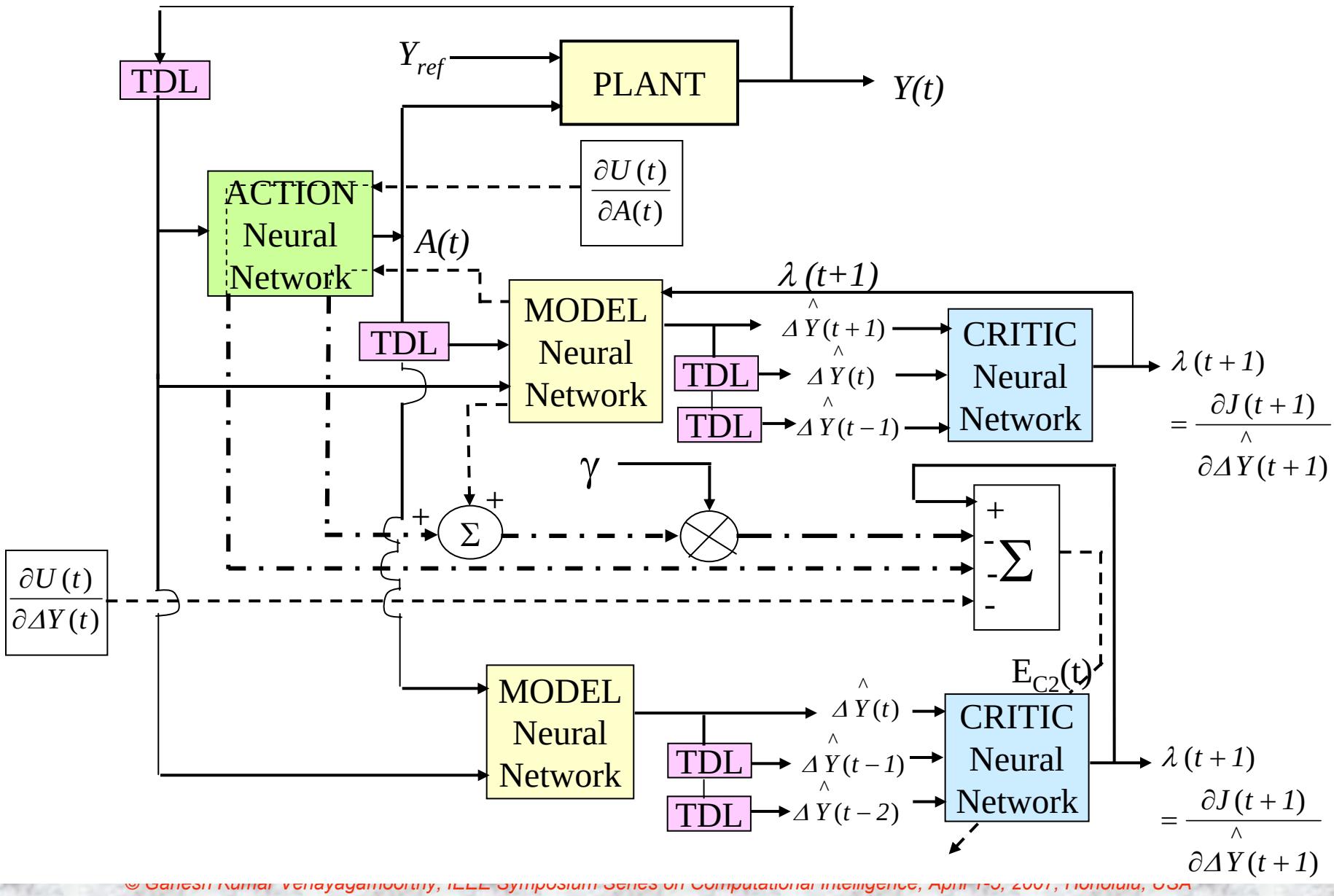


Critic Neural Network

$$J(Y(t)) = \sum_{k=0}^{\infty} \gamma^k U(Y(t+k))$$

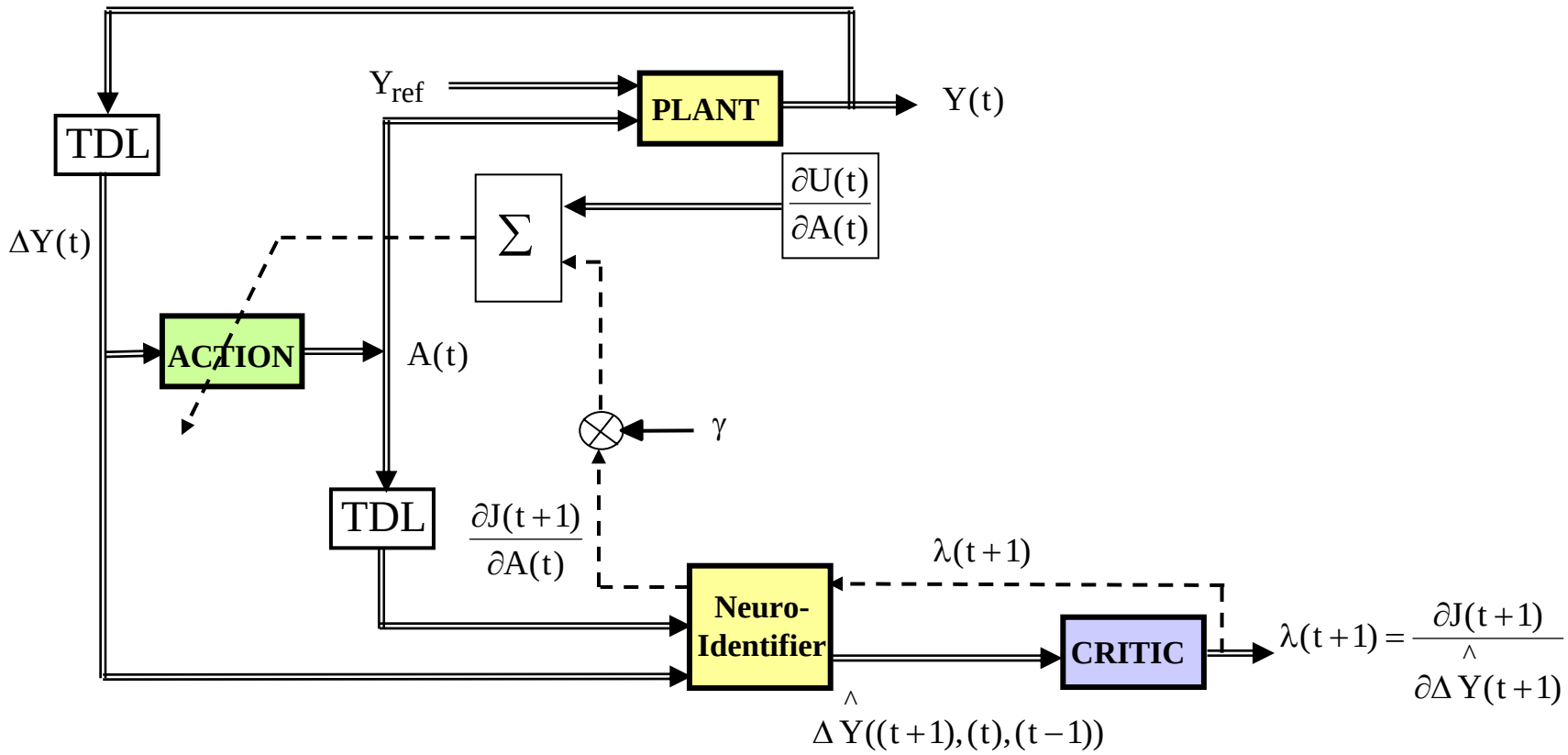


DHP Critic Network Adaptation



Action Network Adaptation

$$\frac{\partial^+ J(t)}{\partial A(t)} = \frac{\partial U(t)}{\partial A(t)} + \gamma \frac{\partial^+ J(t+1)}{\partial A(t)} = 0$$



Critic Network's Training Cycle

The following operations are repeated *N* times:

1. Initialize $t = 0$ and $\Delta Y(0)$
2. Compute output of the critic network at time t, $J(t) = f_C(\Delta Y(t), W_C)$
3. Compute output of the action network at time t,
   $A(t) = f_A(\Delta Y(t), W_A)$
4. Compute output of the model network at time $t+1$,
   $\Delta Y(t+1) = f_M(\Delta Y(t), A(t), W_M)$
5. Compute the output of the critic network at time $t+1$,
   $J(t+1) = f_C(\Delta Y(t+1), W_C)$
6. Compute the critic network error at time t,
   $E_{c1}(t) = J(\Delta Y(t) - \gamma J(\Delta Y(t+1) - U(t))$
   $U(t) = [4\,\Delta V(t) + 4\,\Delta V(t-1) + 16\,\Delta V(t-2)]^2 + [0.4\,\Delta(t) + 0.4\,\Delta(t-1) + 0.16\Delta(t-2)]^2$
7. Update the critic network's weights using the backpropagation algorithm
8. Repeat steps 2 to 7.

Power System Control Application

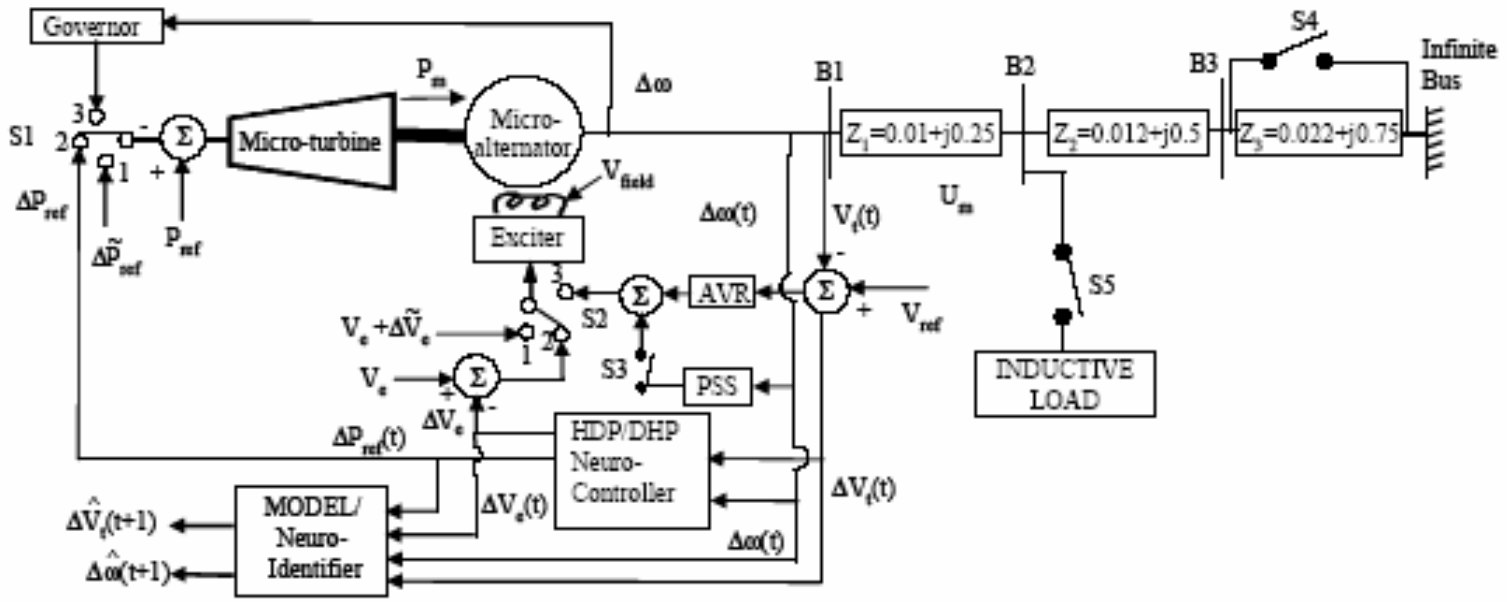


Fig. 19.7 The single machine infinite bus configuration with the conventional AVR and governor controllers, and neurocontroller.

G K Venayagamoorthy, et al, Approximate Dynamic Programming for Power Systems Control, in the *Handbook of Learning and Approximate Dynamic Programming, Edited by J Si, A G Barto, W B Powell, D Wunsch, Wiley-IEEE press, July 2004, ISBN: 0-471-66054-X.*

Power System Control Application

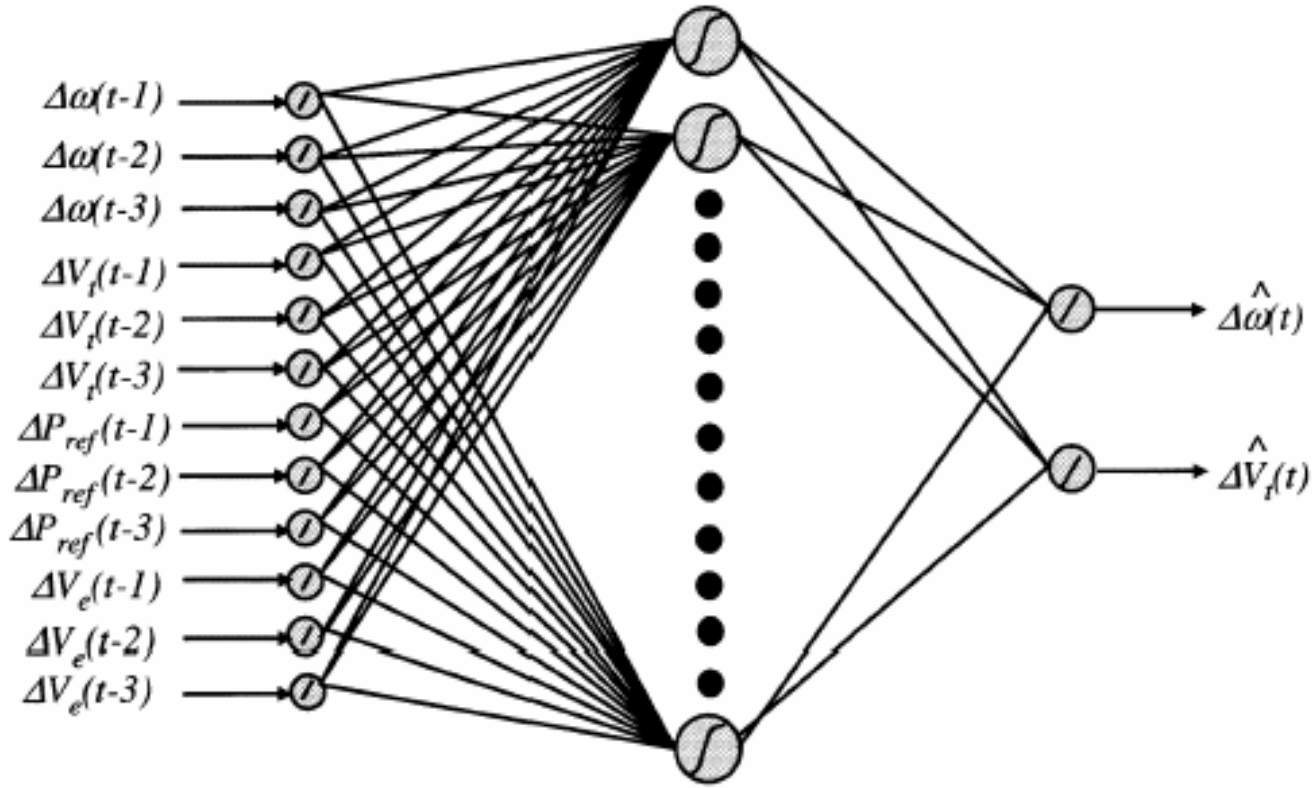
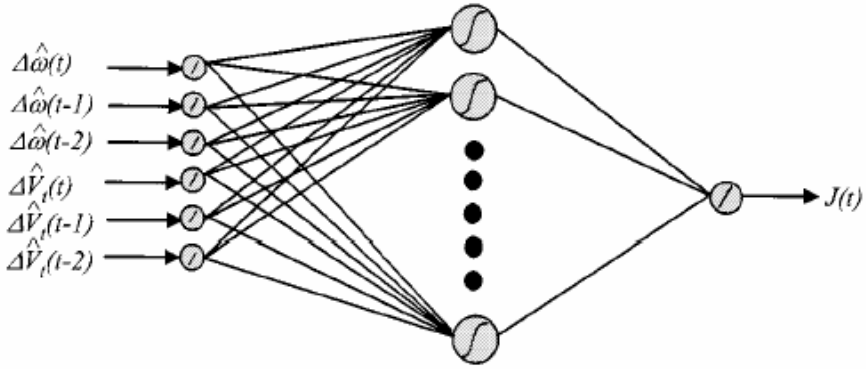


Fig. 12. Model neural-network structure with 12 inputs, 14 sigmoidal hidden layer neurons, and two linear neurons.

Venayagamoorthy GK, Harley RG, Wunsch DC, "Implementation of Adaptive Critic Based Neurocontrollers for Turbogenerators in a Multimachine Power System", *IEEE Transactions on Neural Networks*, vol. 14, no. 5, September 2003, pp. 1047 - 1064.

Power System Control Application

UMR



Venayagamoorthy GK, Harley RG, Wunsch DC, "Comparison of Heuristic Dynamic Programming and Dual Heuristic Programming Adaptive Critics for Neurocontrol of a Turbogenerator", *IEEE Transactions on Neural Networks*, vol. 13, no. 3, May 2002, pp. 764 - 773.

Fig. 6. HDP critic neural network structure with six inputs, ten sigmoidal hidden layer neurons, and one linear output neuron.

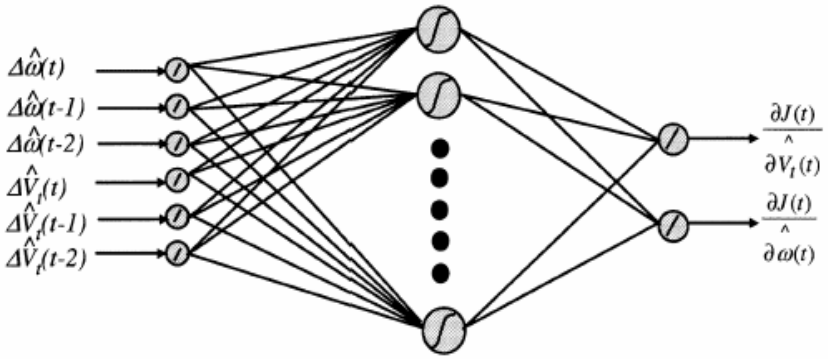


Fig. 13. DHP Critic neural-network structure with six inputs, ten sigmoidal hidden layer neurons, and two linear neurons.

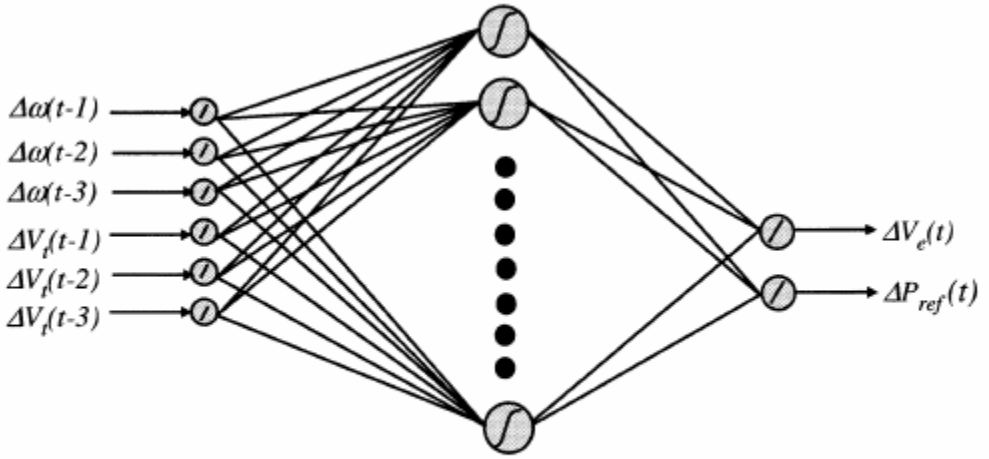


Fig. 15. DHP action neural-network structure with six inputs, ten sigmoidal hidden layer neurons, and two linear output neurons.

Venayagamoorthy GK, Harley RG, Wunsch DC, "Implementation of Adaptive Critic Based Neurocontrollers for Turbogenerators in a Multimachine Power System", *IEEE Transactions on Neural Networks*, vol. 14, no. 5, September 2003, pp. 1047 - 1064.

Power System Control Application

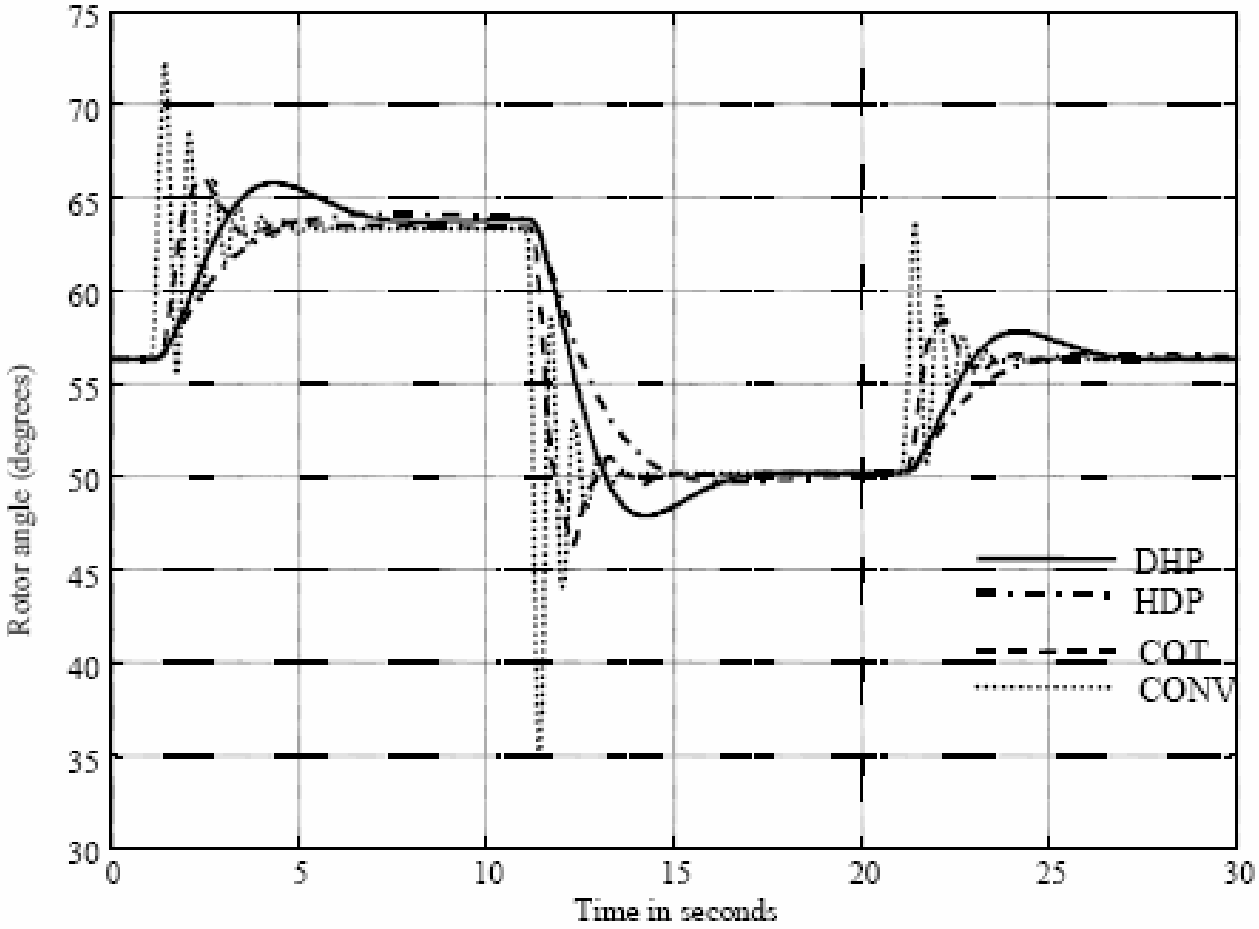


Fig. 19.13 Rotor angle of the micro-alternator for $\pm 5\%$ step changes in the terminal voltage reference (transmission line impedance $Z_1 + Z_2$).

G K Venayagamoorthy, et al, Approximate Dynamic Programming for Power Systems Control, in the *Handbook of Learning and Approximate Dynamic Programming, Edited by J Si, A G Barto, W B Powell, D Wunsch, Wiley-IEEE press, July 2004, ISBN: 0-471-66054-X.*

Power System Control Application

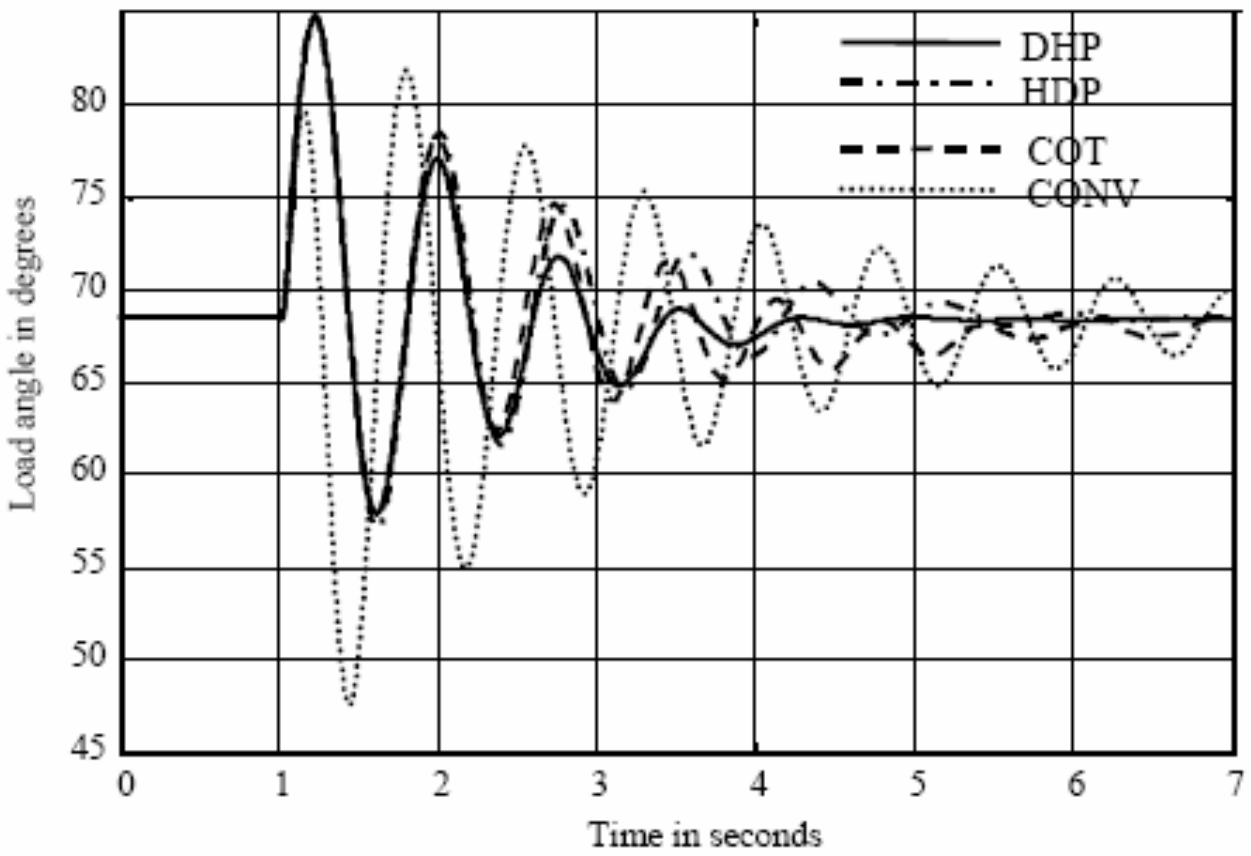
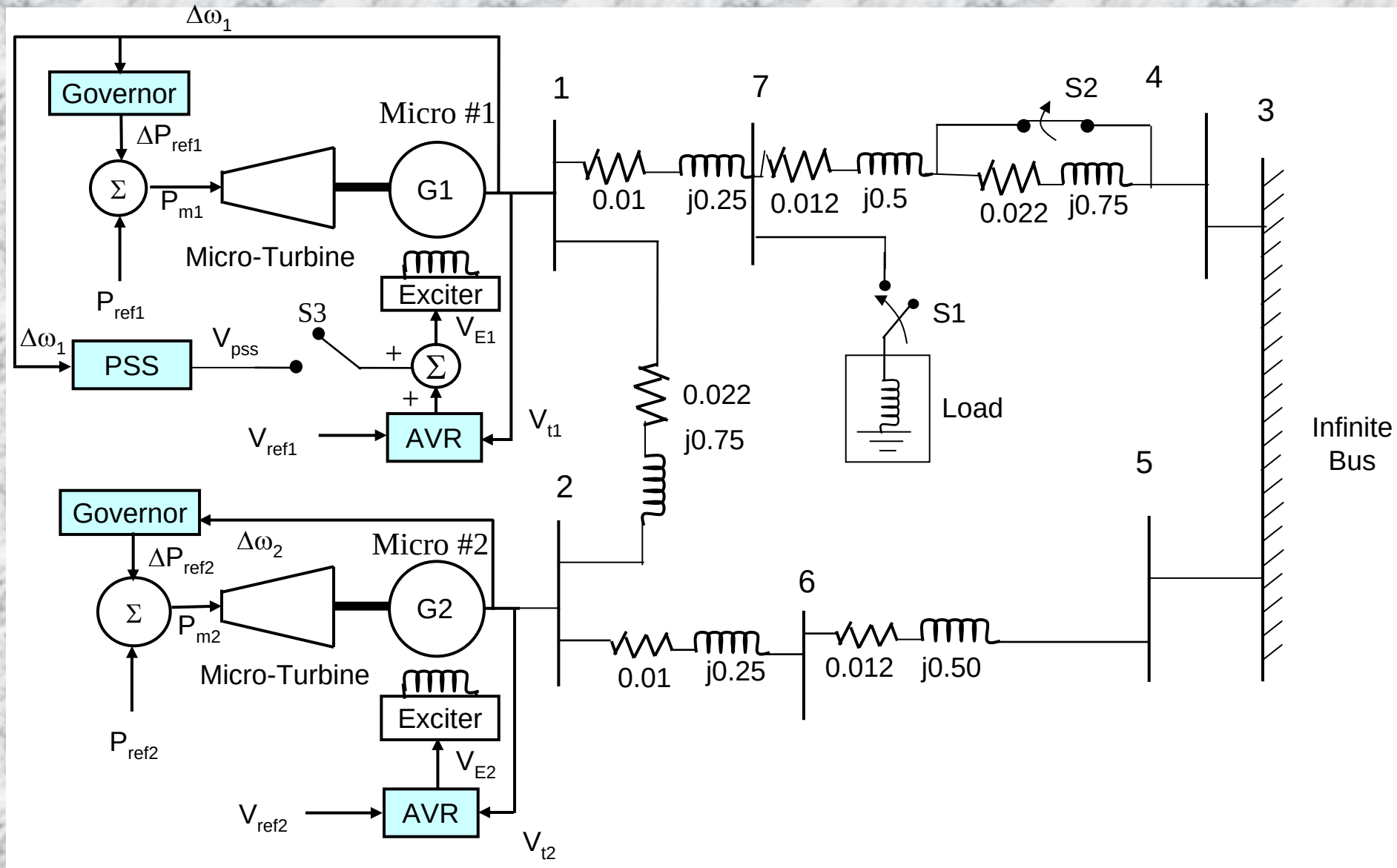


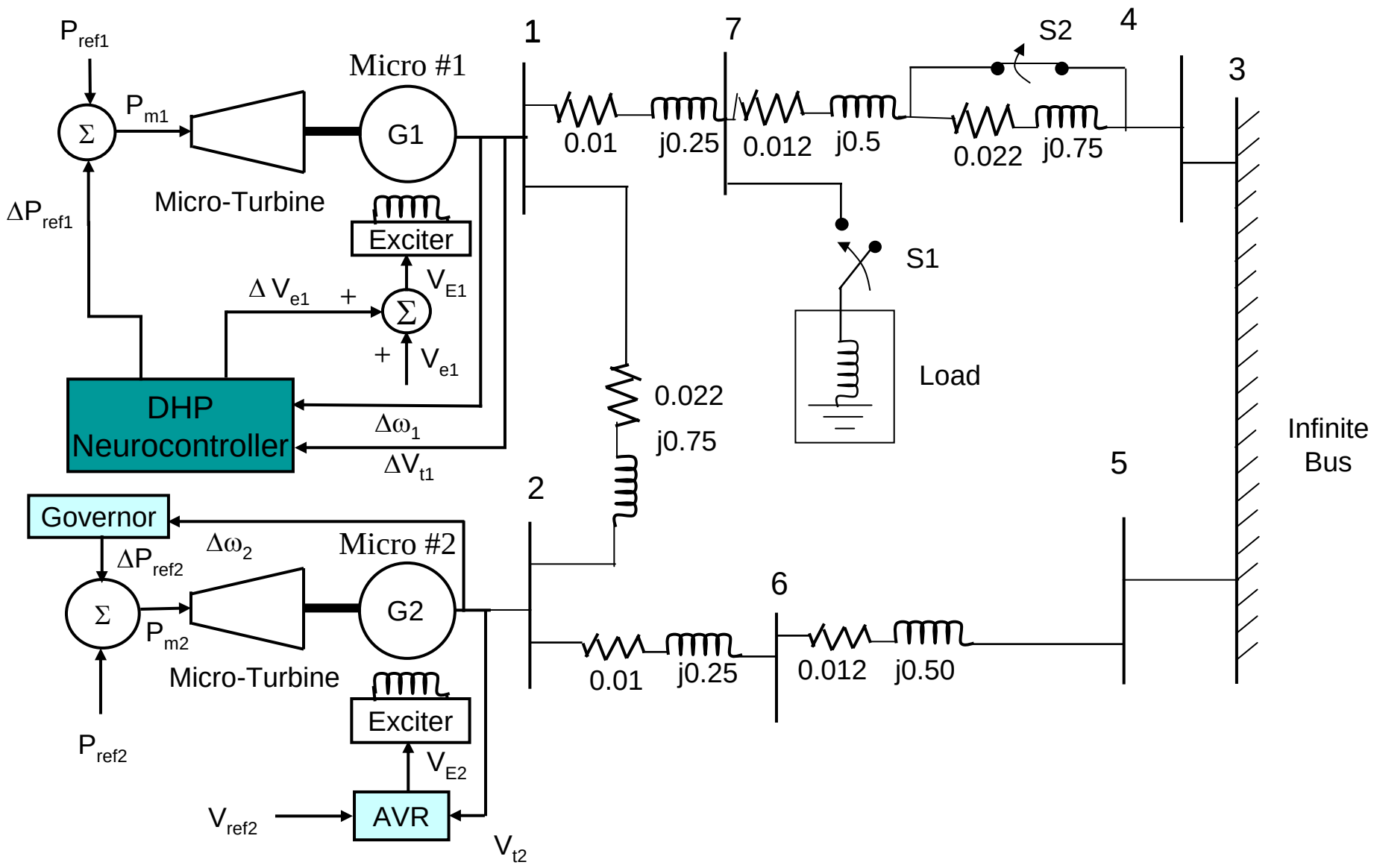
Fig. 19.15 Rotor angle of the micro-alternator for a temporary 50 ms three-phase short circuit (transmission line impedance $Z_1 + Z_2 + Z_3$).

G K Venayagamoorthy, et al, Approximate Dynamic Programming for Power Systems Control, in the *Handbook of Learning and Approximate Dynamic Programming, Edited by J Si, A G Barto, W B Powell, D Wunsch, Wiley-IEEE press, July 2004, ISBN: 0-471-66054-X.*

Multimachine Power System

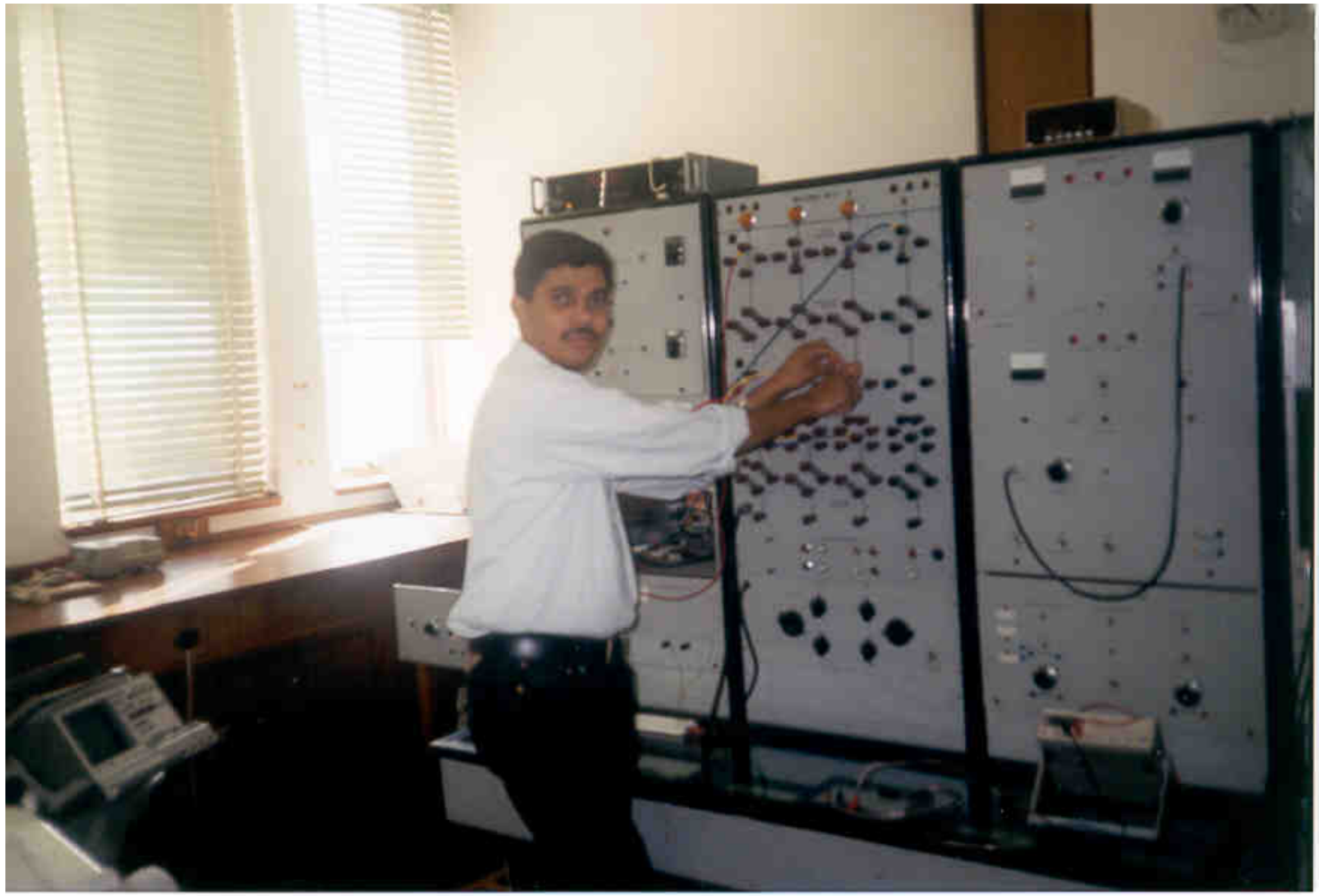


Multimachine Power System



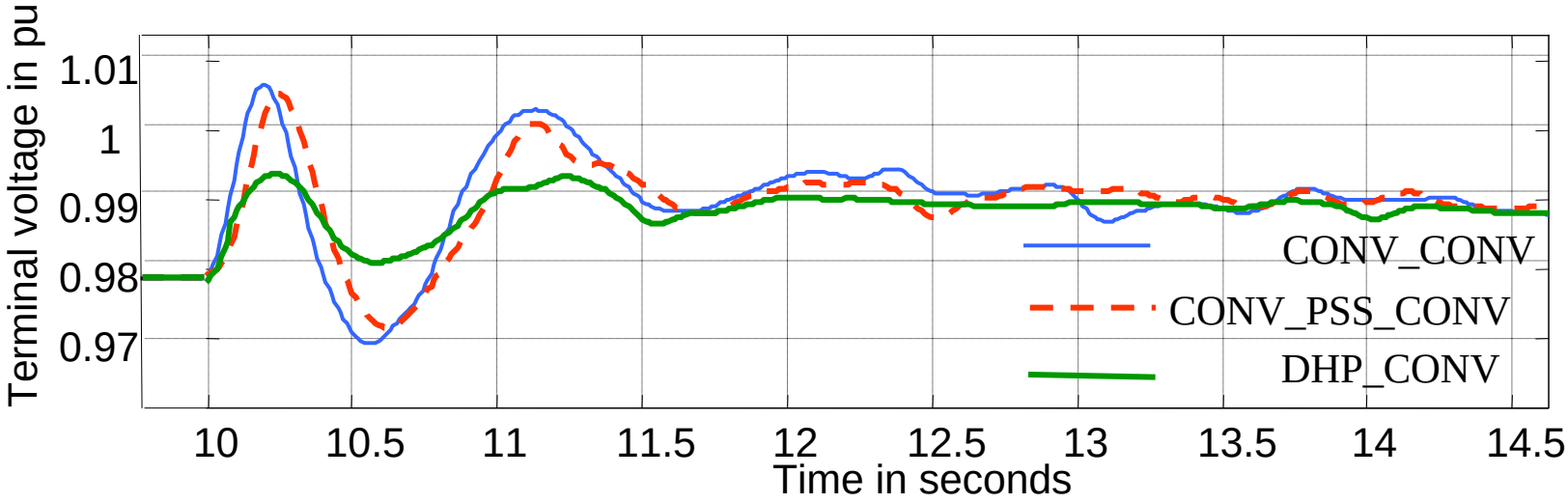
Micro-Machine Research Laboratory at the University of Natal, Durban, South Africa

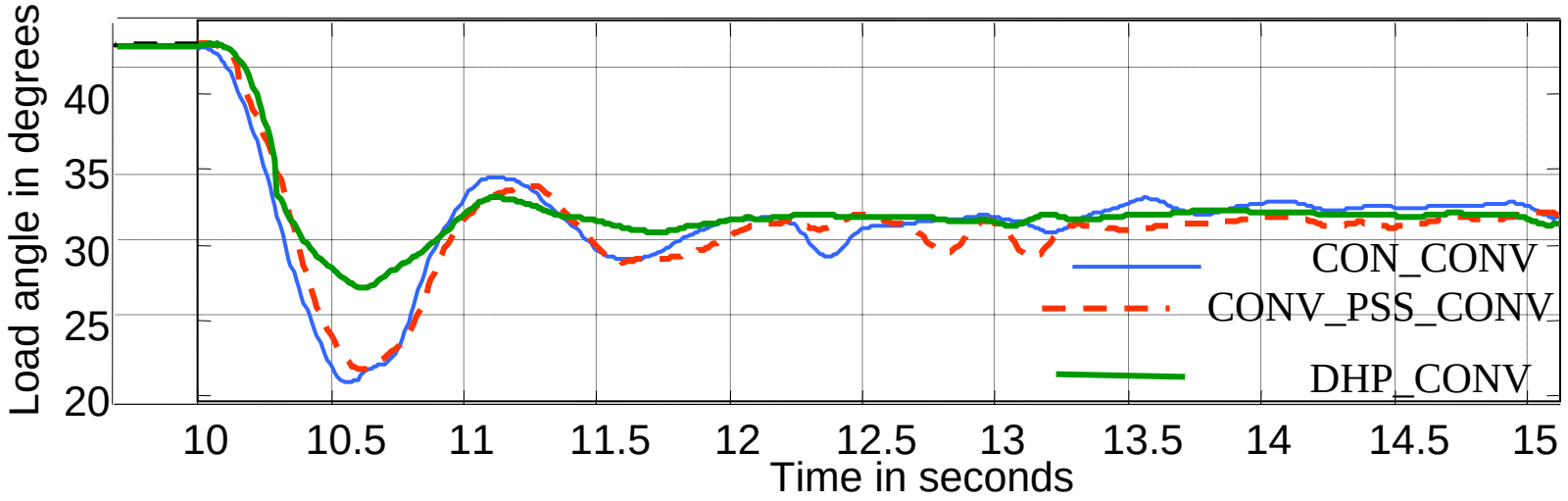
Micro-Machine Research Laboratory at the University of Natal, Durban, South Africa



December 2000/ January 2001

Machine #1: Trans. Line Impedance Increase





Machine #2: Trans. Line Impedance Increase