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
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.
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.
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
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.
Computational Intelligence Paradigms

- Neural & Immune Networks
  - Neuro-Swarm Systems
  - Neuro-Genetic Systems
  - Evolutionary-Swarm Systems

- Evolutionary Computing
  - Fuzzy-GA Systems
  - Neuro-Fuzzy Systems
  - Fuzzy-PSO Systems

- Swarm Intelligence
  - Fuzzy Systems

- Neuro-Swarm Systems
  - Fuzzy Systems
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.**
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.
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
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.
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.
Computational Intelligence

- Evolutionary-Swarm-Neuro Systems
- Swarm Intelligence
- Neural & Immune Networks
- Fuzzy Logic
- Evolutionary Computing
- Evolutionary-Swarm-Fuzzy Systems
- Evolutionary-Swarm-Neuro-Fuzzy Systems
- Evolutionary-Swarm-Fuzzy Systems
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

Activation Functions:
- Input & Context Layer – Linear
- Hidden Layer – Hyperbolic Tangent
- Output Layer – Linear

Inputs

Z\(^{-1}\)

Output

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

Supervised Learning:

- **Environment**
  - Vectors describing state of the environment

- **Teacher**
  - Desired response
  - Actual response
  - Error signal

- **Learning system**
  - \( \Sigma \)
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.
Classification of Learning Algorithms

Neural Network Learning Algorithms

Supervised Learning (Error based)
- Error Correction
  - Gradient descent
- Least Mean Square
- Backpropagation

Unsupervised Learning
- Stochastic
- Hebbian
- Competitive
- PSO
- ES/GA
- Simulated Annealing

Reinforced Learning (Output based)
Gradient Descent Learning Rule – Neuron

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

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.
The weights are updated using:

\[ w_i(t + 1) = w_i(t) + \Delta w_i(t) \]

with \( \Delta w_i(t) = \eta \left( -\frac{\partial E}{\partial w_i} \right) \)

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

where \( \eta \) is the learning rate and \( w_i(t+1) \) is the new weights.
Feedforward Neural Networks

Feedforward Operation

Backpropagation Operation
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

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

$$a = WX$$

Decision vector $d_i$ is given by

$$d_i = \text{sig}(a_i)$$

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

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

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

$$\hat{y} = Vd$$
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.

\[ E(.) \equiv E(e_y) \equiv E(x, W, V, y) \equiv \frac{1}{2} |e_y|^2 \]

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.
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**

**Step 1:** The output error vector is given by

\[ \overline{e_y} = y - \hat{y} \]

**Step 2:** The decision error vector is given by

\[ \overline{e_d} = V^T \overline{e_y} \]

The activation error vector is given by

\[ \overline{e_{ai}} = \frac{d}{da_i} d_i \overline{e_{di}} = \frac{d}{da_i} d_i = d_i (1 - d_i) \]

\[ \overline{e_{ai}} = d_i (1 - d_i) \overline{e_{di}} \]

**Step 3:** The weights changes are given by

\[ \Delta V(k) = \gamma_g \overline{e_y}(k)d^T(k) + \gamma_m \Delta V(k - 1) \]

\[ \Delta W(k) = \gamma_g \overline{e_a}(k)x^T(k) + \gamma_m \Delta w(k - 1) \]

where \( \gamma_g \) and \( \gamma_m \) are learning and momentum gains respectively.
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

\[ \hat{y} = v_{11}\text{sig}(w_{12}x + w_{11}) + v_{12} \]

\[ \text{sig}(.) = \frac{1}{1 + e^{(.)}} \]
Feedforward Neural Networks

\[ \hat{y} = v_{11} \text{sig}(w_{12}x + w_{11}) + v_{12} \]

\[ \text{sig}(.) = \frac{1}{1 + e^{(.)}} \]
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.
Evolutionary Computing

- Genetic Algorithms
  - Genetic Programming
- Evolutionary Programming
  - Evolutionary Strategies
- Differential Evolution
  - Cultural Evolution
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**.
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.
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.
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.

   
   \[
   \text{(Probability of } i^{th} \text{ individual} = \frac{\text{fitness of } i^{th} \text{ individual}}{\text{sum of all individual fitness}})\]

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.
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.
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.
Islands of Population Based Algorithms

Migration

Visitation
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.
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.
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
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 ‘\(p_{best}\)’ 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

\[ V_{id} = w \times V_{id} + c_1 \times \text{rand}_1 \times (P_{\text{bestid}} - X_{id}) + c_2 \times \text{rand}_2 \times (G_{\text{bestid}} - X_{id}) \]

The position vector of the particles is changed as follows

\[ X_{id} = X_{id} + V_{id} \]
1. Initialize a population of \( N \) individuals.

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

\[
O_{1,j} = P_{4,j} + \gamma \times (P_{2,j} - P_{3,j})
\]

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).
1. Initialize a population of \( N \) individuals.

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

\[
O_{1,j} = P_{4,j} + \gamma \times (P_{2,j} - P_{3,j})
\]

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).
Fuzzy System

Non-Fuzzy Inputs

Fuzzification Process

Fuzzy Logic Controller

Inference Engine

Fuzzy Rules

Fuzzy Sets

Defuzzification Process

Non-Fuzzy Outputs

Non-Fuzzy Inputs
Computing Degree of Membership

System Input Range | Degree of Membership
--- | ---
A | 0
B | \((X - \text{point 1}) \times \text{slope 1}\)
C | \(\text{max}\)
D | \((\text{point 2} - X) \times \text{slope 2}\)
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

$$\mu_C(y) = \min (\mu_{C_1}(y), \mu_{C_2}(y), \mu_{C_3}(y), \ldots, \mu_{C_n}(y))$$
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.
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
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.
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
## Comparison of Number of Computations in Training a Neural Network $2 \times 4 \times 1$ (Bias 1)

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

### With bias 2

<table>
<thead>
<tr>
<th>Error=0.001</th>
<th>PSO</th>
<th>BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patterns (input x)</td>
<td>-1: 0.1:1</td>
<td></td>
</tr>
<tr>
<td>Iterations</td>
<td>194(83)</td>
<td>9836(307)</td>
</tr>
<tr>
<td>Forward path (additions+ multiplications)</td>
<td>348600</td>
<td>148281</td>
</tr>
<tr>
<td>Backward path (additions+ multiplications)</td>
<td>71712</td>
<td>438396</td>
</tr>
<tr>
<td>Total (Forward + Backward)</td>
<td>420312</td>
<td>586677</td>
</tr>
<tr>
<td>Ratio of computations</td>
<td>1.3958</td>
<td></td>
</tr>
</tbody>
</table>
IJCNN04 CATS Problem

Missing data to be predicted
• An Elman RNN is chosen as the predictor
• Novel training algorithm
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

Generation N

Rank

Fitness ranking

Old Population

Winners

Elites

PSO

Enhanced elites

Generation N+1

New Population

Losers

Discard half

EA Mutation

Offspring
Prediction

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

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ F = Z(X \oplus Y) + Y(X \oplus Z) \]

<table>
<thead>
<tr>
<th>HD1</th>
<th>NGA</th>
<th>MGA</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ F = Z(X + Y) \oplus (XY) ]</td>
<td>[ F = Z(X + Y) \oplus XY ]</td>
<td>[ F = (X + Y)Z \oplus XY ]</td>
<td>[ F = XZ \oplus Y(X + Z) ]</td>
</tr>
<tr>
<td>5 gates</td>
<td>4 gates</td>
<td>4 gates</td>
<td>4 gates</td>
</tr>
<tr>
<td>2 AND, 1 OR, 2 XOR</td>
<td>2 AND, 1 OR, 1 XOR</td>
<td>2 AND, 1 OR, 1 XOR,</td>
<td>2 AND, 1 OR, 1 XOR,</td>
</tr>
</tbody>
</table>

\[ F = XZ \oplus Y(X + Z) \]
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.
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

OUTER PSO
- Initialize PSO parameters for the outer swarm \((w_{out}, c_{1out}, c_{2out})\)

INNER SWARM FOR TARGET SEARCHING
- Uses the values \(w_{in}, c_{1in}, c_{2in}\) as passed into it
- The target search application program that uses PSO
- Fitness: lowest number of iterations
- Chooses the values of \(w_{in}, c_{1in}, c_{2in}\) that corresponds to the particle with the least number of iterations

TARGET SEARCHING
- Best values of \(w_{in}, c_{1in}, c_{2in}\)
### Comparison of PSO & Optimal PSO

<table>
<thead>
<tr>
<th>Case</th>
<th>Parameter Values</th>
<th># of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Target (Unoptimized)</td>
<td>$w=0.6$</td>
<td>70.7</td>
</tr>
<tr>
<td></td>
<td>$c_1=0.5$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$c_2=2$</td>
<td></td>
</tr>
<tr>
<td>Single Target (Optimized)</td>
<td>$w=0.45$</td>
<td>39.5</td>
</tr>
<tr>
<td></td>
<td>$c_1=0.60$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$c_2=1.50$</td>
<td></td>
</tr>
<tr>
<td>Multiple Targets (Unoptimized)</td>
<td>$w=0.6$</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>$c_1=0.5$</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>$c_2=2$</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td></td>
<td>139</td>
</tr>
<tr>
<td>Multiple Targets (Optimized)</td>
<td>$w=0.55$</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>$c_1=0.55$</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>$c_2=2.23$</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td></td>
<td>124</td>
</tr>
</tbody>
</table>
## Comparison PSO, DE and DEPSO

<table>
<thead>
<tr>
<th>Case</th>
<th>Evaluations</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PSO</strong></td>
<td>11796</td>
<td>100% (84%)</td>
</tr>
<tr>
<td><strong>DE</strong></td>
<td>3624</td>
<td>100% (94%)</td>
</tr>
<tr>
<td><strong>DEPSO</strong></td>
<td>5723</td>
<td>100% (100%)</td>
</tr>
</tbody>
</table>
Fuzzy Logic Control

- **Non-Fuzzy Inputs**
  - Fuzzification Process
  - Knowledge Base (Rules)
  - Inference Engine
  - Defuzzification Process
  - Non-Fuzzy Outputs
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,
- \( Gdx = L_{best}x - P_x \) and \( Gdy = L_{best}y - P_y \)
- \( P_x \) and \( P_y \) 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.
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

<table>
<thead>
<tr>
<th>w</th>
<th>c₁</th>
<th>c₂</th>
<th># of Iterations</th>
<th>Time (sec)</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>2</td>
<td>2</td>
<td>1181.58</td>
<td>6.35</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2</td>
<td>1106.09</td>
<td>5.88</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.5</td>
<td>985.39</td>
<td>4.98</td>
<td>26%</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.5</td>
<td>924.10</td>
<td>3.99</td>
<td>93%</td>
</tr>
<tr>
<td>0.6</td>
<td>2</td>
<td>2</td>
<td>910.60</td>
<td>4.90</td>
<td>94%</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2</td>
<td>859.40</td>
<td>4.79</td>
<td>99%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.5</td>
<td>889.36</td>
<td>5.07</td>
<td>32%</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.5</td>
<td>841.85</td>
<td>5.24</td>
<td>8%</td>
</tr>
</tbody>
</table>
## Results for the Fuzzy Based Navigation (100% Convergence)

<table>
<thead>
<tr>
<th>Simulation case study</th>
<th>Membership Function</th>
<th>Coarse Rule Set</th>
<th>Fine Rule set</th>
<th>Iterations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Original (heuristics)</td>
<td>Original (heuristics)</td>
<td>Original (heuristics)</td>
<td>355.80</td>
<td>349.97</td>
</tr>
<tr>
<td>2</td>
<td>PSO</td>
<td>Original (heuristics)</td>
<td>Original (heuristics)</td>
<td>308.41</td>
<td>328.24</td>
</tr>
<tr>
<td>3</td>
<td>Original (heuristics)</td>
<td>PSO</td>
<td>Original (heuristics)</td>
<td>356.96</td>
<td>356.18</td>
</tr>
<tr>
<td>4</td>
<td>Original (heuristics)</td>
<td>Original (heuristics)</td>
<td>PSO</td>
<td>307.44</td>
<td>325.65</td>
</tr>
<tr>
<td>5</td>
<td>PSO</td>
<td>PSO</td>
<td>PSO</td>
<td>306.15</td>
<td>326.78</td>
</tr>
</tbody>
</table>
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?

- Plant or Environment
- Observables $X(t)$
- Control system
- Control Variables (Actions) $u(t)$
- $t$ 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.
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).
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

\[ u \rightarrow \text{Nonlinear Plant} \rightarrow \text{Output} \]

\[ \Sigma \]

\[ \text{error} \]

\[ \hat{u} \]

\[ \text{Neural Network Inverse Controller} \]

\[ \text{Desired Output} \]

\[ \text{Neural Network Inverse Controller} \]

\[ \hat{u} \]

\[ \text{Control} \]

\[ \text{Nonlinear Plant} \]

\[ \text{Output} \]
Neural Direct Adaptive Control

Setpoint → Nonlinear neural network controller

Control

System state

Backpropagation Algorithm

Error

Σ

Desired output

Output

Adjust weights

© Ganesh Kumar Venayagamoorthy, IEEE Symposium Series on Computational Intelligence, April 1-5, 2007, Honolulu, USA
Neural Indirect Adaptive Control

Neurocontroller \rightarrow \text{Control} \rightarrow \text{Nonlinear Plant} \rightarrow \text{Output} \rightarrow \text{Desired Response Predictor}

\text{Error} \rightarrow \text{Adjust weights} \rightarrow \text{Nonlinear Differential Plant Model} \rightarrow \text{Error} \rightarrow \text{Adjust weights} \rightarrow \text{Neural Network Identifier}
Backpropagated Through Time (BPTT) Based Neurocontrol

\[ X(k) \]
\[ t = k \]

\[ \text{Neurocontroller} \]

\[ \text{Model} \]

\[ \sum \]

\[ \Sigma \]

\[ + \]

\[ - \]

\[ e(T) \]

\[ R(T) \]

\[ R_d(T) \]

\[ t = k + 1 \]

\[ t = k + h - 1 \]

\[ t = k + h = T \]
A CONTINUALLY ONLINE TRAINED NEUROCONTROLLER FOR EXCITATION AND TURBINE CONTROL OF A TURBOGENERATOR

*IEEE Transactions on Energy Conversion*,
Power System Model
Conventional Controllers

Automatic voltage regulator and exciter combination

Micro-turbine and governor combination
Adaptive Neurocontroller

\[ \Delta Y(t) = [\Delta \omega(t), \Delta V_t(t)] \]

\[ \Delta Y(t) = [\Delta \omega(t), \Delta V_t(t)] \]

\[ X(t+1) = [\hat{\Delta} V_t(t+1), \hat{\Delta} \omega(t+1)] \]

\[ \Delta Y(t) = [\Delta \omega(t), \Delta V_t(t)] \]

\[ \Delta Y(t) = [\Delta \omega(t), \Delta V_t(t)] \]

\[ X(t+1) = [\hat{\Delta} V_t(t+1), \hat{\Delta} \omega(t+1)] \]

\[ \Delta Y(t) = [\Delta \omega(t), \Delta V_t(t)] \]

\[ X(t+1) = [\hat{\Delta} V_t(t+1), \hat{\Delta} \omega(t+1)] \]

\[ \Delta Y(t) = [\Delta \omega(t), \Delta V_t(t)] \]

\[ X(t+1) = [\hat{\Delta} V_t(t+1), \hat{\Delta} \omega(t+1)] \]
ANN Identifier/Model

Plant Inputs → Plant → Plant outputs $V_t, \Delta \omega, \Delta \delta$

Time delay line

Plant Inputs $P$ → Plant $U_f$ → Plant Outputs $V_t$ → Time delay line $\Delta \delta$ $\Delta \omega$

$\Delta V_t$ → Time delay line $\Delta \omega$ $\Delta \delta$

$\Delta P$ → Time delay line $\Delta U_f$ → Neural Network Model

Weights update $\Delta P$, $\Delta U_f$ → Neural Network Model $\Delta V_t$, $\Delta \omega$, $\Delta \delta$ → ANN outputs $\sum$

ANN outputs $\sum$ → Training Algorithm $\Delta V_t$, $\Delta \omega$, $\Delta \delta$ → ANN outputs $\sum$

Errors $\Delta V_t$, $\Delta \omega$, $\Delta \delta$ → ANN outputs $\sum$
Series-parallel Nonlinear Auto Regressive Moving Average (NARMA) model

\[
\hat{y}(t + 1) = f \left[ y(t), y(t - 1), \ldots, y(t - n + 1), \\
u(t), u(t - 1), \ldots, u(t - m + 1) \right]
\]

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).
ANN Identifier/Model

\[ \Delta \omega(t-1) \rightarrow \Delta \omega(t-2) \rightarrow \Delta \omega(t-3) \rightarrow \Delta V_t(t-1) \rightarrow \Delta V_t(t-2) \rightarrow \Delta V_t(t-3) \rightarrow \Delta P_{ref}(t-1) \rightarrow \Delta P_{ref}(t-2) \rightarrow \Delta P_{ref}(t-3) \rightarrow \Delta V_e(t-1) \rightarrow \Delta V_e(t-2) \rightarrow \Delta V_e(t-3) \longrightarrow \hat{\Delta \omega}(t) \]

\[ \Delta \omega(t-1) \rightarrow \Delta \omega(t-2) \rightarrow \Delta \omega(t-3) \rightarrow \Delta V_t(t-1) \rightarrow \Delta V_t(t-2) \rightarrow \Delta V_t(t-3) \rightarrow \Delta P_{ref}(t-1) \rightarrow \Delta P_{ref}(t-2) \rightarrow \Delta P_{ref}(t-3) \rightarrow \Delta V_e(t-1) \rightarrow \Delta V_e(t-2) \rightarrow \Delta V_e(t-3) \longrightarrow \hat{\Delta V_t}(t) \]
Training Signals

Power deviations at inputs to the turbine and the ANN

![Graph showing power deviations](image)

- Time in seconds: 0, 5, 10, 15
- Turbine power input deviation in pu: -0.1 to 0.1
Plant and ANN Outputs

Speed deviations at outputs of the PLANT and the ANN

![Graph showing speed deviations over time for PLANT and ANN outputs with three operating points: first operating point (trained), second operating point (untrained), and third operating point (untrained).]
Voltage deviations at outputs of the PLANT and the ANN

Time in seconds

Terminal voltage deviation in pu

First operating point

Second operating point (untrained)

Third operating point (untrained)

PLANT

ANN
Pre-training of the Neurocontroller

- **Inputs to the Neurocontroller**
  - Delayed values of Terminal voltage deviation $\Delta V_t$
  - Delayed values of Speed deviation $\Delta \omega$
  - Outputs of the Neurocontroller
  - Deviation in Field voltage $\Delta V_{field}$
  - Deviation in Turbine Power $\Delta P_{ref}$

- **Neuroidentifier weights**
  - Fixed
  - Backpropagation of errors at H

- **Desired Response Predictor**
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 $\Delta u$ 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
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.
The basic concept of all forms of dynamic programming can be summarized as follows:

Model of Reality ($F$) → Utility Function ($U$) → Dynamic Programming → Secondary Utility ($J$)
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).
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)
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

ΔY(t) → TDL

ACTION Neural Network

ΔY(t) → CRITIC Neural Network

ΔY(t) → MODEL Neural Network

Y(t) → PLANT

Y_ref → PLANT

A(t) → CRITIC Neural Network

J(t) → CRITIC Neural Network

ΔY(t) → CRITIC Neural Network

[A(t-1), A(t-2), A(t-3)] → MODEL Neural Network

[ΔY(t-1), ΔY(t-2), ΔY(t-3)] → MODEL Neural Network

∂J(t) / ∂A(t) → CRITIC Neural Network

ΔY(t) → CRITIC Neural Network

[A(t-1), A(t-2), A(t-3)] → MODEL Neural Network

[ΔY(t-1), ΔY(t-2), ΔY(t-3)] → MODEL Neural Network

∂J(t) / ∂ΔY(t) → CRITIC Neural Network

1
\[ J(Y(t)) = \sum_{k=0}^{\infty} \gamma^k U(Y(t + k)) \]

Critic Neural Network

\[ \text{Target} = \gamma J(\Delta Y(t+1)) + U(\Delta Y(t)) \]

\[ J(\Delta Y(t)) \]

\[ \sum \text{error} \]
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.
Fig. 19.7 The single machine infinite bus configuration with the conventional AVR and governor controllers, and neurocontroller.

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


Power System Control Application

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

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

Multimachine Power System
Multimachine Power System

- **Micro-Turbine**
  - Micro #1
  - Micro #2

- **Exciter**
  - Ve1
  - Ve2

- **Governor**
  - Δω1
  - Δω2

- **DHP Neurocontroller**

- **AVR**

- **Load**

- **Infinite Bus**
  - S1
  - S2
Micro-Machine Research Laboratory at the University of Natal, Durban, South Africa
Micro-Machine Research Laboratory at the University of Natal, Durban, South Africa
Machine #1: Trans. Line Impedance Increase

- Terminal voltage in pu
- Load angle in degrees

Graphs showing the impact of increased line impedance on terminal voltage and load angle over time.
Machine #2: Trans. Line Impedance Increase

![Graph showing load angle in degrees over time for different scenarios: CON_CONV, CONV_PSS_CONV, DHP_CONV. The graph indicates the impact of different transmission line impedance increases on the load angle over time.](image)