Soft Computing and its Application

- Introduction
- Neural networks
- Learning Algorithms
- Advanced Neural Network Architectures
- Pulse Coded Neural Networks
- Fuzzy Systems
- Genetic Algorithms
- Hardware implementation of neuro-fuzzy systems
- Conclusion

An example of the three layer feedforward neural network, which is sometimes known also as the backpropagation network.

An example of the bi-directional autoassociative memory - BAM: (a) drawn as a two layer network with circulating signals (b) drawn as two layer network with bi-directional signal flow.

Several logical operations using networks with McCulloch-Pitts neurons.

Threshold implementation

\[
net = \sum_{i=1}^{n} w_i x_i
\]

Threshold implementation with an additional weight and constant input with +1 value: (a) neuron with threshold \( T = 0 \), (b) modified neuron with threshold \( T = 0 \) and additional weight equal to \(-T\).

\[
a = f(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ 0 & \text{if } net < 0 \end{cases}
\]

\[
a = f(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ -1 & \text{if } net < 0 \end{cases}
\]

Typical activation functions: (a) hard threshold unipolar, (b) hard threshold bipolar, (c) continuous unipolar, (d) continuous bipolar.
Activation functions

bipolar
\[ a = f(\text{net}) = \frac{2}{1 + \exp(-2k \text{net})} - 1 = \tanh(k \text{net}) \]
\[ f'(a) = k \left(1 - a^2\right) \]

unipolar
\[ a = f(\text{net}) = \frac{1}{1 + \exp(-4k \text{net})} \]
\[ f'(a) = 4k \cdot a \cdot (1 - a) \]

Learning rules for single neuron

\[ \Delta w_j = \alpha \delta x \]

Hebb rule (unsupervised):
\[ \delta = 0 \]
correlation rule (supervised):
\[ \delta = d \]
perceptron fixed rule:
\[ \delta = d - o \]
perceptron adjustable rule - as above but the learning constant is modified to:
\[ \alpha^* = \alpha \lambda^T x w^T = \alpha \lambda \frac{\text{net}}{\|x\|} \]

LMS (Widrow-Hoff) rule:
\[ \delta = d - \text{net} \]
delta rule:
\[ \delta = (d - o) f' \]
pseudoinverse rule (for linear system):
\[ w = (x^T x)^{-1} x^T d \]
iterative pseudoinverse rule (for nonlinear system):
\[ w = (x^T x)^{-1} x^T \frac{d - o}{f'} \]

LMS AND REGRESSION ALGORITHMS

If a single layer of neurons is considered, error back propagation type of algorithms minimize global error as shown in equation 1:
\[ \text{TotalError} = \sum_{p=1}^{P} \sum_{j=1}^{J} (d_p - o_p)^2 \]

where \( P \) is the number of patterns and \( J \) is the number of outputs. A similar approach is taken in the Widrow-Hoff (LMS) algorithm:
\[ \text{TotalError} = \sum_{p=1}^{P} \sum_{j=1}^{J} (d_p - \text{net}_p)^2 \]

where \( \text{net}_p = \sum_{i=1}^{I} w_{ij} x_{ip} \)
and \( I \) is the size of the augmented input vector i.e. \( x_{ip}^{I+1} \).

LMS algorithm

For any given neuron the training data is given in two arrays:
\[
\begin{bmatrix}
X_{11} & X_{12} & \ldots & X_{1J} \\
X_{21} & X_{22} & \ldots & X_{2J} \\
\vdots & \vdots & \ddots & \vdots \\
X_{P1} & X_{P2} & \ldots & X_{PJ}
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
\vdots \\
w_J
\end{bmatrix}
= \begin{bmatrix}
d_1 \\
d_2 \\
\vdots \\
d_P
\end{bmatrix}
\]

where \( I \) is the number of augmented inputs. The over-determined set of equations can be solved in a least mean square sense:
\[ w_j = (x^T x)^{-1} x^T d_j \]

where \( w_j \) unknown vector of the weights of the \( j \)th neuron. The matrix \( x \) must be converted only once, and the weights for all the neurons (\( j=1 \) to \( N \)) can be found. Regardless of whether the regression algorithm or the LMS algorithm is used, the outcome will be the same.

AW ALGORITHM

The total error for one neuron \( j \) and pattern \( p \) is now defined by a simple difference:
\[ E_{jpo} = D_{jpo} - O_{jpo}(\text{net}) \]

where \( \text{net} = w_{j1} x_1 + w_{j2} x_2 + \ldots + w_{jN} x_N \). The derivative of this error with respect to the \( i \)th weight of the \( j \)th neuron can be written as
\[ \frac{dE_{jpo}}{dw_i} = \frac{dO_{jpo}}{d\text{net}} \frac{d\text{net}}{dw_i} = -f'(x_{ijp}) \]

The error function can then be approximated by the first two terms of the linear approximation around a given point:
\[ E_{jpo} = E_{jpo} + \frac{dE_{jpo}}{dw_1} \Delta w_1 + \frac{dE_{jpo}}{dw_2} \Delta w_2 + \ldots + \frac{dE_{jpo}}{dw_n} \Delta w_n \]

Therefore:
The Y matrix is composed of input patterns, and must be computed only once!

Illustration of the property of linear separation of patterns in the two-dimensional space by a single neuron.

An example with a comparison of results obtained using LMS and Delta training algorithms. Note that LMS is not able to find the proper solution.
Although the error backpropagation algorithm (EBP) was a significant breakthrough in neural network research, it is known as an algorithm with a very poor convergence rate.

Many attempts have been made to speed up the EBP algorithm:
- heuristics approaches such as momentum,
- variable learning rate
- artificial enlarging of errors for neurons operating in saturation region
- Levenberg-Marquardt (LM) method.  The LM algorithm is now considered as the most efficient one. It combines the speed of the Newton algorithm with the stability of the steepest decent method.

More significant improvement was possible by using various second order approaches:
- Newton,
- conjugate gradient,
- Levenberg-Marquardt (LM) method. The LM algorithm is now considered as the most efficient one. It combines the speed of the Newton algorithm with the stability of the steepest decent method.

Levenberg - Marquardt

Steepest decent method:

\[ \mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \mathbf{g} \]

Newton method:

\[ \mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{A}_k^{-1} \mathbf{g} \]

where \( \mathbf{A}_k \) is Hessian and \( \mathbf{g} \) is gradient vector

Assuming:

\[ \mathbf{A} \approx 2\mathbf{J}^T \mathbf{J} \quad \text{and} \quad \mathbf{g} \approx 2\mathbf{J}^T \mathbf{v} \]

where \( \mathbf{J} \) is Jacobian and \( \mathbf{v} \) is error vector

\[ \mathbf{w}_{k+1} = \mathbf{w}_k - (2\mathbf{J}^T \mathbf{J})^{-1} 2\mathbf{J}^T \mathbf{v} \quad \text{or} \quad \mathbf{w}_{k+1} = \mathbf{w}_k - (\mathbf{J}^T \mathbf{J} + \mu \mathbf{I})^{-1} \mathbf{J}^T \mathbf{v} \]

Levenberg - Marquardt method:

\[ \mathbf{w}_{k+1} = \mathbf{w}_k - (2\mathbf{J}^T \mathbf{J})^{-1} 2\mathbf{J}^T \mathbf{v} \]

\[ \mathbf{w}_{k+1} = \mathbf{w}_k - (\mathbf{J}^T \mathbf{J} + \mu \mathbf{I})^{-1} \mathbf{J}^T \mathbf{v} \]

where \( \mathbf{I} \) is identity matrix, \( \mu \) is a learning parameter and \( \mathbf{J} \) is Jacobian of \( n \) output errors with respect to \( n \) weights of neural network. For \( \mu = 0 \) it becomes the Gauss-Newton method. For very large \( \mu \) the LM algorithm becomes the steepest decent or the EBP algorithm. The \( \mu \) parameter is automatically adjusted at each iteration in order to secure convergence.

\[
\mathbf{W}_{k+1} = \mathbf{W}_k - \left( \mathbf{J}_k^T \mathbf{J}_k + \mu \mathbf{I} \right)^{-1} \mathbf{J}_k^T \mathbf{E}
\]

where \( \mathbf{E} \) is the cumulative (for all patterns) error vector \( \mathbf{I} \) is identity unit matrix, \( \mu \) is a learning parameter and \( \mathbf{J} \) is Jacobian of \( n \) output errors with respect to \( n \) weights of neural network. For \( \mu = 0 \) it becomes the Gauss-Newton method. For very large \( \mu \) the LM algorithm becomes the steepest decent or the EBP algorithm. The \( \mu \) parameter is automatically adjusted at each iteration in order to secure convergence.
**Levenberg-Marquardt Algorithm**

![Graph showing Levenberg-Marquardt Algorithm](image)

Sum of squared errors as a function of number of iterations for the “XOR” problem using LM algorithm with Nguyen-Widrow weight initialization. Algorithm failed in 15% to 25% cases.

When initial weight were chosen purposely very far from the solution the LM algorithm failed in 100% cases.

**Results of flat spot elimination**

![Graph showing Sum of squared errors](image)

Sum of squared errors as a function of number of iterations for the “XOR” problem using modified EBP algorithm with unfavorable weight initialization.

Illustration of the modified derivative calculation for faster convergence of the error backpropagation algorithm.

A poor convergence of EBP algorithm is not because of local minima but it is due to plateaus on the error surface. This problem is also known as “flat spot” problem. The prime reason for the plateau formations is a characteristic shape of the sigmoidal activation functions.

- **Kohonen layer**
  - The counterpropagation network.
  - Find number and location of clusters in 4-dim. space

- **Grossberg layer**
  - **unipolar neurons**
  - **summing circuits**

A winner take all - WTA architecture for cluster extracting in the unsupervised training mode: (a) network connections, (b) single layer network arranged into a hexagonal shape

<table>
<thead>
<tr>
<th>Find number and location of clusters in 4-dim. space</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Clusters in 4-dim space" /></td>
</tr>
<tr>
<td>(-2 3 -3 4) (4 -4 6 -6) (2 -2 4 2)</td>
</tr>
</tbody>
</table>
The cascade correlation architecture

A typical structure of the radial basis function network.

A Hopfield network or autoassociative memory

An example of the bi-directional autoassociative memory - BAM: (a) drawn as a two layer network with circulating signals (b) drawn as two layer network with bi-directional signal flow.

The functional link network

Functional link networks for solution of the XOR problem: (a) using unipolar signals, (b) using bipolar signals.
Sarajedini and Hecht-Nielsen network

Let us consider stored vector \( w \) and input pattern \( x \). Both input and stored patterns have the same dimension \( n \). The square Euclidean distance between \( x \) and \( w \) is:

\[
||x - w||^2 = (x_1 - w_1)^2 + (x_2 - w_2)^2 + \cdots + (x_n - w_n)^2
\]

After defactorization

\[
||x - w||^2 = x^T x + w^T w - 2x^T w = ||x||^2 + ||w||^2 - 2x^T w
\]

finally

\[
||x - w||^2 = ||x||^2 + ||w||^2 - 2x^T w = 2nc\theta
\]

Input pattern transformation on a sphere

Network with two neurons capable of separating crescent shape of patterns (a) input-output mapping, (b) network diagram

Spiral problem solved with sigmoidal type neurons (a) network diagram, (b) input-output mapping.

Pulse Coded Neural Networks

Pulse Coded Neural Networks 2
Pulse Coded Neural Networks

Fuzzy systems

- Inputs can be any value from 0 to 1.
- The basic fuzzy principle is similar to Boolean logic.
- Max and min operators are used instead of AND and OR. The NOT operator also becomes $1 - $. 

$A \cap B \cap C \Rightarrow \min\{A, B, C\}$ – smallest value of $A, B$ or $C$

$A \cup B \cup C \Rightarrow \max\{A, B, C\}$ – largest value of $A, B$ or $C$

$\overline{A} \Rightarrow 1 - A$ – one minus $A$

<table>
<thead>
<tr>
<th>Boolean</th>
<th>Fuzzy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \cap B$</td>
<td>$0 \ 0 \ 0$</td>
</tr>
<tr>
<td>$0 \ 1 \ 0$</td>
<td>$0 \ 1 \ 0$</td>
</tr>
<tr>
<td>$1 \ 0 \ 0$</td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$1 \ 1 \ 1$</td>
<td>$1 \ 1 \ 1$</td>
</tr>
</tbody>
</table>

**Fuzzification**

- There are three major types of membership functions
  - Gaussian, Triangular and Trapezoidal
- Three basic membership function rules
  1. Each point of an input should belong to one membership function
  2. The sum of two overlapping functions should never be greater than 1.
  3. For higher accuracy, more membership functions can be used, but this can lead to system instability and will require a larger fuzzy table.
Fuzzification

Trapezoidal Membership Function

Results from Fuzzification

Defuzzification

- The equation to describe the defuzzification process.
  - \( n \) – Number of membership functions
  - \( z_k \) – Fuzzy output variables
  - \( z_{ck} \) – analog values from table
- Outputs:
  - Zadeh
    \[
    \text{Output} = \frac{\sum z_{ck}}{\sum z_k}
    \]
    \[
    = 0.2A + 0.7B + 0.3C
    \]
    \[
    = 0.2 + 0.7 + 0.3
    \]
  - Tagagi-Sugeno
    \[
    \text{Output} = \frac{0.2O3 + 0.7O4 + 0.2O8 + 0.3O9}{0.2 + 0.7 + 0.3 + 0.2}
    \]

Fuzzy systems VLSI implementation

Block diagrams of the fuzzy VLSI chip

Control surfaces: (a) desired control surface, (b) information stored in defuzzifier as weights, and (c) measured control surface of VLSI chip

Fuzzy systems VLSI implementation 2

Fuzzifier (a) circuit diagram of fuzzifier, (b) example of the SPICE simulation

Fuzzy systems VLSI implementation 3

Defuzzifier using normalization and weighted sum

Selection circuits (a) MIN circuit in voltage mode (b) neuron circuit with threshold in the current mode
Fuzzy systems VLSI implementation 4

MAX operators (a) concept diagram and (b) simulation results for MAX1 and for the proposed MAX2.

Fuzzy systems VLSI implementation 5

The cluster cell with rule selection (transistors M1-M4) and defuzzification (source I0 and transistors M4-M6)

Six bit programmable current sources

Fuzzy systems VLSI implementation 6

Normalization circuit (a) circuit diagram and (b) characteristics

Fuzzy systems - microprocessor implementation

Required control surface
Zadeh with trapezoidal membership functions
Zadeh with triangular membership functions
Zadeh with Gaussian membership functions
Tagagi-Sugeno with trapezoidal membership functions
Tagagi-Sugeno with triangular membership functions

Fuzzy systems - microprocessor implementation 2

<table>
<thead>
<tr>
<th>Approach used</th>
<th>Error SSE</th>
<th>Error MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Zadeh fuzzy controller with trapezoidal function</td>
<td>908.4</td>
<td>0.945</td>
</tr>
<tr>
<td>2. Zadeh fuzzy controller with triangular function</td>
<td>644.4</td>
<td>0.671</td>
</tr>
<tr>
<td>3. Zadeh fuzzy controller with Gaussian function</td>
<td>562.0</td>
<td>0.585</td>
</tr>
<tr>
<td>4. Tagagi-Sugeno fuzzy controller with trapezoidal function</td>
<td>296.5</td>
<td>0.309</td>
</tr>
<tr>
<td>5. Tagagi-Sugeno fuzzy controller with triangular function</td>
<td>210.8</td>
<td>0.219</td>
</tr>
<tr>
<td>6. Tagagi-Sugeno fuzzy controller with Gaussian function</td>
<td>294.2</td>
<td>0.306</td>
</tr>
</tbody>
</table>

Neural systems - microprocessor implementation

Required control surface
Genetic Algorithms 1
The genetic algorithms follow the evolution process in the nature to find the better solutions of some complicated problems. Foundations of genetic algorithms are given in Holland (1975) and Goldberg (1989) books.

Genetic algorithms consist the following steps:

- Initialization
- Selection
- Reproduction with crossover and mutation

Selection and reproduction are repeated for each generation until a solution is reached.

During this procedure a certain strings of symbols, known as chromosomes, evaluate toward better solution.

Genetic Algorithms 2
All significant steps of the genetic algorithm will be explained using a simple example of finding a maximum of the function $(\sin(x)-0.5*x)^2$ with the range of $x$ from 0 to 1.6. Note, that in this range the function has global maximum at $x=1.309$, and local maximum at $x=0.262$.

Coding and initialization
At first, the variable $x$ has to be represented as a string of symbols. With longer strings process converges usually faster, so less symbols for one string field are used it is the better. While this string may be the sequence of any symbols, the binary symbols "0" and "1" are usually used. In our example, let us use for coding six bit binary numbers having a decimal value of 40x.

Process starts with a random generation of the initial population given in Table

<table>
<thead>
<tr>
<th>Type of controller</th>
<th>length of code</th>
<th>processing time (ms)</th>
<th>Error MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zadeh with trapezoidal</td>
<td>2324</td>
<td>1.95</td>
<td>0.945</td>
</tr>
<tr>
<td>Zadeh with triangular</td>
<td>2324</td>
<td>1.95</td>
<td>0.671</td>
</tr>
<tr>
<td>Zadeh with Gaussian</td>
<td>3245</td>
<td>39.8</td>
<td>0.585</td>
</tr>
<tr>
<td>Tagagi-Sugeno with trapezoidal</td>
<td>1502</td>
<td>28.5</td>
<td>0.309</td>
</tr>
<tr>
<td>Tagagi-Sugeno with triangular</td>
<td>1502</td>
<td>28.5</td>
<td>0.219</td>
</tr>
<tr>
<td>Tagagi-Sugeno with Gaussian</td>
<td>2845</td>
<td>52.3</td>
<td>0.306</td>
</tr>
<tr>
<td>Neural network with 3 neurons in cascade</td>
<td>680</td>
<td>1.72</td>
<td>0.00057</td>
</tr>
<tr>
<td>Neural network with 5 neurons in cascade</td>
<td>1070</td>
<td>3.3</td>
<td>0.00009</td>
</tr>
<tr>
<td>Neural network with 6 neurons in one hidden layer</td>
<td>660</td>
<td>3.8</td>
<td>0.00030</td>
</tr>
</tbody>
</table>

Genetic Algorithms 3

<table>
<thead>
<tr>
<th>Initial Population</th>
<th>string</th>
<th>decimal value</th>
<th>variable value</th>
<th>function value</th>
<th>fraction of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101101</td>
<td>45</td>
<td>1.125</td>
<td>0.0633</td>
<td>0.2465</td>
</tr>
<tr>
<td>2</td>
<td>101000</td>
<td>40</td>
<td>1.000</td>
<td>0.0433</td>
<td>0.1686</td>
</tr>
<tr>
<td>3</td>
<td>010100</td>
<td>20</td>
<td>0.500</td>
<td>0.0004</td>
<td>0.0016</td>
</tr>
<tr>
<td>4</td>
<td>100101</td>
<td>37</td>
<td>0.925</td>
<td>0.0307</td>
<td>0.1197</td>
</tr>
<tr>
<td>5</td>
<td>001010</td>
<td>10</td>
<td>0.250</td>
<td>0.0041</td>
<td>0.0158</td>
</tr>
<tr>
<td>6</td>
<td>110001</td>
<td>49</td>
<td>1.225</td>
<td>0.0743</td>
<td>0.2895</td>
</tr>
<tr>
<td>7</td>
<td>100111</td>
<td>39</td>
<td>0.975</td>
<td>0.0390</td>
<td>0.1521</td>
</tr>
<tr>
<td>8</td>
<td>000100</td>
<td>4</td>
<td>0.100</td>
<td>0.0016</td>
<td>0.0062</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>0.2568</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Genetic Algorithms 4

Selection and reproduction
Selection of the best members of the population is an important step in the genetic algorithm. Many different approaches can be used to rank individuals. In our example the ranking function is given. Highest rank has member number 6 and lowest rank has member number 3.

Members with higher rank should have higher chances to reproduce. The probability of reproduction for each member can be obtained as fraction of the sum of all objective function values. This fraction is shown in the last column of the Table.

Using a random reproduction process the following population arranged in pairs could be generated:

Genetic Algorithms 5

Reproduction

If size of the population from one generation to another is the same, therefore two parents should generate two children. By combining two strings two another strings should be generated. The simplest way to do it is to split in half each of the parent string and exchange substrings between parents. For example from parent strings 010100 and 100111 the following child strings will be generated 010111 and 100100. This process is known as the crossover and resultant children are shown below

010111 -> 47 110101 -> 53 100001 -> 33 110000 -> 48
100100 -> 37 101001 -> 41 110101 -> 53 101000 -> 40

Genetic Algorithms 6

Mutation

On the top of properties inherited from parents they are acquiring some new random properties. This process is known as mutation. In most cases mutation generates low ranked children, which are eliminated in reproduction process. Sometimes however, the mutation may introduce a better individual with a new property into. This prevents process of reproduction from degeneration. In genetic algorithms mutation plays usually secondary role. Mutation rate is usually assumed on the level much below 1%. In our example mutation is equivalent to the random bit change of a given pattern. In this simple example with short strings and small population with a typical mutation rate of 0.1%, our patterns remain practically unchanged by the mutation process. The second generation for our example is shown in Table

Genetic Algorithms 7

Population of Second Generation

<table>
<thead>
<tr>
<th>string number</th>
<th>string</th>
<th>decimal value</th>
<th>variable value</th>
<th>function value</th>
<th>fraction of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>010111</td>
<td>47</td>
<td>1.175</td>
<td>0.0696</td>
<td>0.1587</td>
</tr>
<tr>
<td>2</td>
<td>100100</td>
<td>37</td>
<td>0.925</td>
<td>0.0307</td>
<td>0.0696</td>
</tr>
<tr>
<td>3</td>
<td>110101</td>
<td>53</td>
<td>1.325</td>
<td>0.0774</td>
<td>0.1766</td>
</tr>
<tr>
<td>4</td>
<td>010001</td>
<td>41</td>
<td>1.025</td>
<td>0.0475</td>
<td>0.0951</td>
</tr>
<tr>
<td>5</td>
<td>100000</td>
<td>33</td>
<td>0.825</td>
<td>0.0161</td>
<td>0.0322</td>
</tr>
<tr>
<td>6</td>
<td>110101</td>
<td>53</td>
<td>1.325</td>
<td>0.0774</td>
<td>0.1766</td>
</tr>
<tr>
<td>7</td>
<td>110000</td>
<td>48</td>
<td>1.200</td>
<td>0.0722</td>
<td>0.1444</td>
</tr>
<tr>
<td>8</td>
<td>101001</td>
<td>41</td>
<td>1.025</td>
<td>0.0475</td>
<td>0.0951</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4387</td>
</tr>
</tbody>
</table>

Note, that two identical highest ranking members of the second generation are very close to the solution x=1.309. The randomly chosen parents for third generation are

010111 -> 47 110101 -> 53 100000 -> 48 101001 -> 40
110101 -> 53 110000 -> 48 101001 -> 41 110101 -> 53

which produces following children:

010101 -> 21 110000 -> 48 101001 -> 41 110101 -> 45
110111 -> 55 110101 -> 53 101000 -> 40 110001 -> 49

The best result in the third population is the same as in the second one. By careful inspection of all strings from second or third generation one may conclude that using crossover, where strings are always split into half, the best solution 110100 -> 52 will never be reached no matter how many generations are created.

The genetic algorithm is very rapid and it leads to a good solution within a few generations. This solution is usually close to global maximum, but not the best.

Genetic Algorithms 8

\[ x_{i+1} = x_i + r \sin(\phi_i) \]
\[ y_{i+1} = y_i + r \cos(\phi_i) \]
\[ \phi_{i+1} = \phi_i + \Theta_i \]

\[ E = w_x x^2 + w_y y^2 + w_\phi \phi^2 \]

Genetic Algorithms 9

Genetic Algorithms 10
Soft Computing and its Application

nn.uidaho.edu

wialm@ieee.org