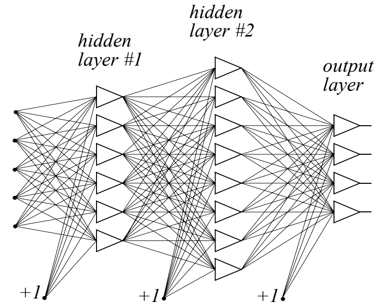
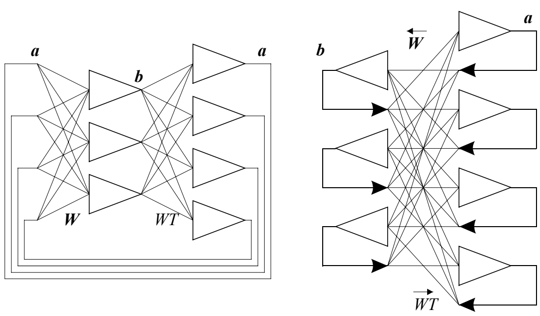
B. M. 'Dan' Wilamowski
University of Idaho
nn.uidaho.edu
wialm@ieee.org

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



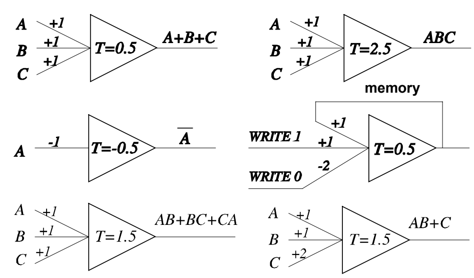
An example of the three layer feedforward neural network, which is sometimes known also as the backpropagation network.



(a) (b)

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.

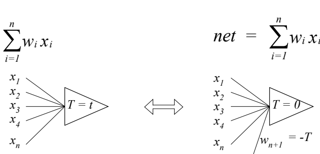
$$net = \sum_{i=1}^{n} w_i x_i \qquad o = \begin{cases} 1 & if \quad net \ge 0 \\ 0 & if \quad net < 0 \end{cases}$$



Several logical operations using networks with McCulloch-Pitts neurons.

Threshold implementation

$$net = \sum_{i=1}^{n} w_i x_i \qquad net = \sum_{i=1}^{n} w_i x_i + w_{n+1}$$

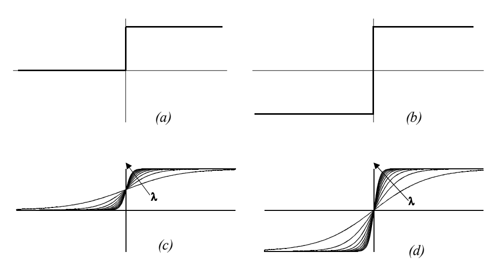


Threshold implementation with an additional weight and constant input with +1 value *: (a)* neuron with threshold *T*, *(b)* modified neuron with threshold T=0 and additional weight equal to *-T*

$$o = f(net) = \frac{\operatorname{sgn}(net)+1}{2} = \begin{cases} 1 & if \quad net \ge 0 \\ 0 & if \quad net < 0 \end{cases} \qquad o = f(net) = \operatorname{sgn}(net) = \begin{cases} 1 & if\, net \ge 0 \\ -1 & if\, net < 0 \end{cases}$$

$$o = f(net) = \frac{1}{1 + \exp(-\lambda net)} \qquad o = f(net) = \tanh(0.5\lambda net) = \frac{2}{1 + \exp(-\lambda net)} - 1$$

$$o = f(net) = \frac{\operatorname{sgn}(net)+1}{2} = \begin{cases} 1 & if \quad net \ge 0 \\ 0 & if \quad net < 0 \end{cases} \qquad o = f(net) = \operatorname{sgn}(net) = \begin{cases} 1 & if\, net \ge 0 \\ -1 & if\, net < 0 \end{cases}$$



$$o = f(net) = \frac{1}{1 + \exp(-\lambda net)} \qquad o = f(net) = \tanh(0.5\lambda net) = \frac{2}{1 + \exp(-\lambda net)} - 1$$

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

Activation functions

bipolar

$$o = f(net) = \frac{2}{1+\exp(-2k\ net)} - 1 = \tanh(k\ net) \qquad f'(o) = k\left(1-o^2\right)$$

unipolar

$$o = f(net) = \frac{1}{1+\exp(-4k\ net)} \qquad f'(o) = 4k\ o(1-o)$$

Learning rules for single neuron

$$\Delta \mathbf{w}_i = \alpha\ \delta\ \mathbf{x}$$

Hebb rule (unsupervised): $\delta = o$

correlation rule (supervised): $\delta = d$

perceptron fixed rule: $\delta = d - o$

perceptron adjustable rule - as above but the learning constant is modified to:

$$\alpha^* = \alpha\ \lambda \frac{\mathbf{x}\,\mathbf{w}^T}{\mathbf{x}\,\mathbf{x}^T} = \alpha\ \lambda \frac{net}{\|\mathbf{x}\|^2}$$

LMS (Widrow-Hoff) rule: $\delta = d - net$

delta rule: $\delta = (d - o) f'$

pseudoinverse rule (for linear system): $\mathbf{w} = \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T d$

iterative pseudoinverse rule (for nonlinear system): $\mathbf{w} = \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T \dfrac{d-o}{f'}$

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:

$$TotalError = \sum_{p=1}^{P}\sum_{j=1}^{J}(d_{pj} - o_{pj})^2$$

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:

$$TotalError = \sum_{p=1}^{P}\sum_{j=1}^{J}(d_{pj} - net_{pj})^2$$

Where $net_{pj} = \sum_{i=1}^{I} w_{ji}\, x_{jip}$

and I is the size of the augmented input vector i.e. x_{jIp}=+1.

For any given neuron the training data is given in two arrays:

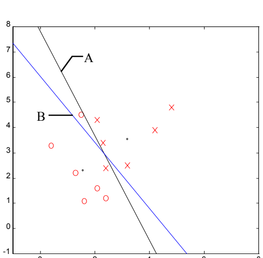
$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \cdots & x_{iI} \\ x_{21} & x_{22} & \cdots\ \cdots & x_{2i} \\ x_{31} & x_{32} & \cdots\ \cdots & x_{3i} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots\ \cdots & x_{PI} \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_I \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_P \end{bmatrix}$$

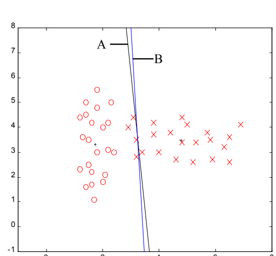
where I is the number of augmented inputs. The over-determinated set of equations can be solved in a least mean square sense:

$$\mathbf{w_j} = \left(\mathbf{x^T\,x}\right)^{-1}\mathbf{x^T\,d_j}$$

where $\mathbf{w_j}$=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.

LMS algorithm





Two examples of non-optimum solutions, where line A is the result of the regression (LMS) algorithm, and line B is the separation generated by the minimum distance classifier.

AW ALGORITHM

The total error for one neuron j and pattern p is now defined by a simple difference:

$$E_{jpo} = D_{jp} - O_{jp}(net)$$

where $net = w_1x_1 + w_2x_2 + \ldots . w_nx_n$. The derivative of this error with respect to the i^{th} weight of the j^{th} neuron can be written as

$$\frac{dE_{jp}}{dw_i} = \frac{dO_{jp}}{dnet}\,\frac{dnet}{dw_i} = -f'_{jp}\,x_{ijp}$$

The error function can then be approximated by the first two terms of the linear approximation around a given point:

$$E_{jp} = E_{jpo} + \frac{dE_{jp}}{dw_1}\Delta w_1 + \frac{dE_{jp}}{dw_2}\Delta w_2 + \ldots\ldots . \frac{dE_{jp}}{dw_n}\Delta w_n$$

Therefore:

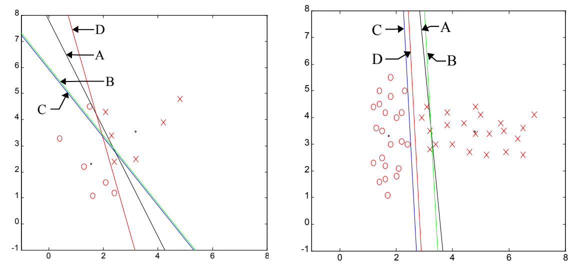
AW ALGORITHM 2

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1I} \\ x_{21} & x_{23} & x_{23} & \cdots & x_{2I} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{p1} & x_{p2} & x_{p3} & \cdots & x_{pI} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{P1} & x_{P1} & x_{P1} & \cdots & x_{PI} \end{bmatrix} \begin{bmatrix} w_1 \\ \nabla w_2 \\ \vdots \\ \nabla w_I \end{bmatrix} = \begin{bmatrix} \frac{D_1 - o_1}{f'_1} \\ \frac{D_2 - o_2}{f'_2} \\ \vdots \\ \frac{D_p - o_p}{f'_p} \\ \vdots \\ \frac{D_P - o_P}{f'_P} \end{bmatrix}$$

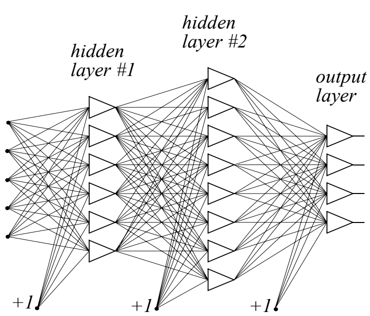
$$\Delta \mathbf{w} = \left(\mathbf{X}^T \quad \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{del} = \mathbf{Y}\ \mathbf{del}$$

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

AW ALGORITHM 3



Comparison of Algorithms where the algorithms can be identified by the labels A-regression, B-minimum distances C-modified minimum distance, and D-modified regression and delta (back propagation) algorithm.



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

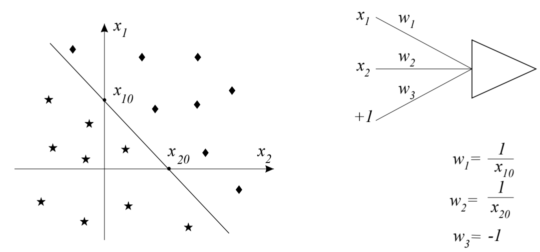
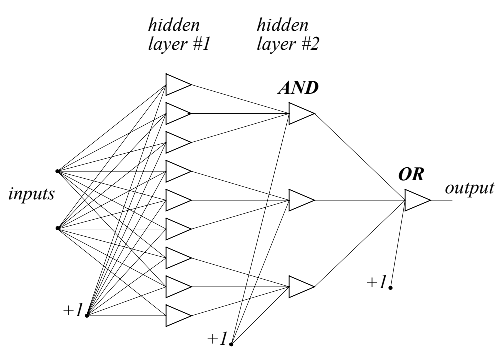
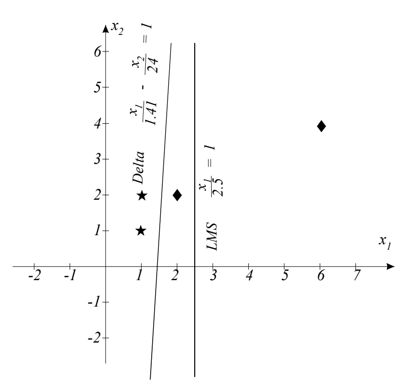


Illustration of the property of linear separation of patterns in the two-dimensional space by a single neuron.

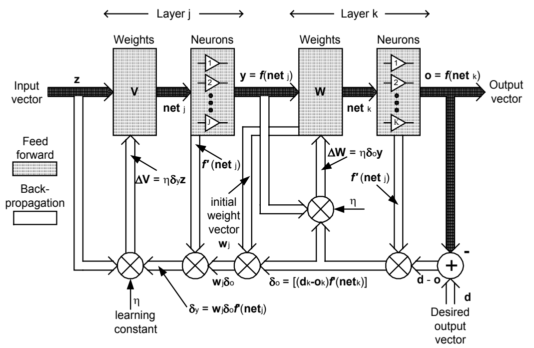


An example of the three layer neural network with two inputs for separation of three clusters into one category. This network can be generalized and can be used for solution all classification problems.



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

Error Backpropagation



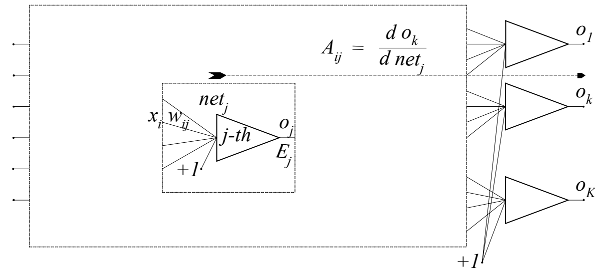


Illustration of the concept of the gain computation in neural networks

- 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
  - stochastic learning
  - artificial enlarging of errors for neurons operating in saturation region
- 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: $\mathbf{A} \approx 2\mathbf{J}^T\mathbf{J}$ and $\mathbf{g} \approx 2\mathbf{J}^T\mathbf{v}$

where **J** is Jacobian and **v** is error vector

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(2\mathbf{J}_k^T\mathbf{J}_k\right)^{-1} 2\mathbf{J}_k^T\mathbf{v} \quad \text{or} \quad \mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T\mathbf{J}_k\right)^{-1}\mathbf{J}_k^T\mathbf{v}$$

Levenberg - Marquardt method: $\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T\mathbf{J}_k + \mu\mathbf{I}\right)^{-1}\mathbf{J}_k^T\mathbf{v}$

$$Hessian \Leftrightarrow \begin{matrix} \frac{\partial^2 v}{\partial x_1^2} & \frac{\partial^2 v}{\partial x_1 \partial x_2} & \frac{\partial^2 v}{\partial x_1 \partial x_3} & . \\ \frac{\partial^2 v}{\partial x_2 \partial x_1} & \frac{\partial^2 v}{\partial x_2^2} & \frac{\partial^2 v}{\partial x_2 \partial x_3} & . \\ \frac{\partial^2 v}{\partial x_3 \partial x_1} & \frac{\partial^2 v}{\partial x_3 \partial x_2} & \frac{\partial^2 v}{\partial x_3^2} & . \\ . & . & . & . \end{matrix} \qquad Jacobian \Leftrightarrow \begin{matrix} \frac{\partial v_1}{\partial x_1} & \frac{\partial v_1}{\partial x_2} & \frac{\partial v_1}{\partial x_3} & . \\ \frac{\partial v_2}{\partial x_1} & \frac{\partial v_2}{\partial x_2} & \frac{\partial v_2}{\partial x_3} & . \\ \frac{\partial v_3}{\partial x_1} & \frac{\partial v_3}{\partial x_2} & \frac{\partial v_3}{\partial x_3} & . \\ . & . & . & . \end{matrix}$$

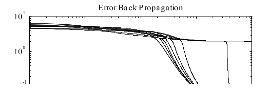
Levenberg - Marquardt 2

LM algorithm combines the speed of the Newton algorithm with the stability of the steepest decent method. The LM algorithm uses the following formula to calculate weights in subsequent iterations:

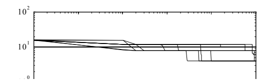
$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T\mathbf{J}_k + \mu\mathbf{I}\right)^{-1}\mathbf{J}_k^T\mathbf{E}$$

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

Error Back Propagation Algorithm

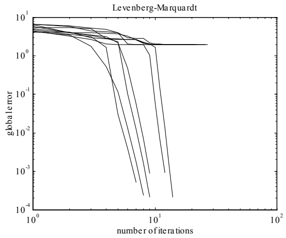


Sum of squared errors as a function of number of iterations for the "XOR" problem using EBP algorithm with Nguyen-Widrow weight initialization



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

Levenberg-Marquardt Algorithm



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

A poor convergence of EBP algorithm is not because of local mimima 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.

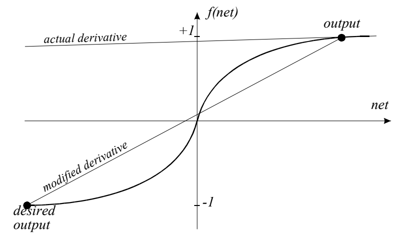
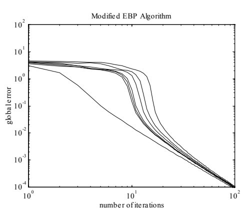
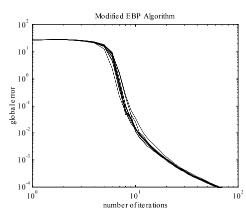


Illustration of the modified derivative calculation for faster convergence of the error backpropagation algorithm

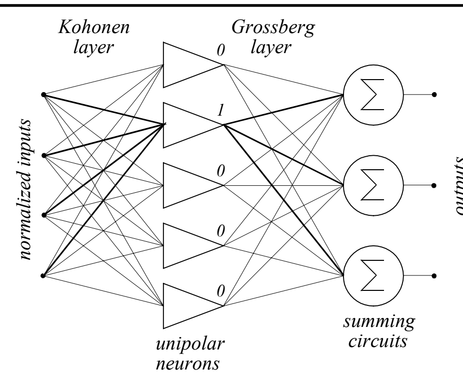
Results of flat spot elimination



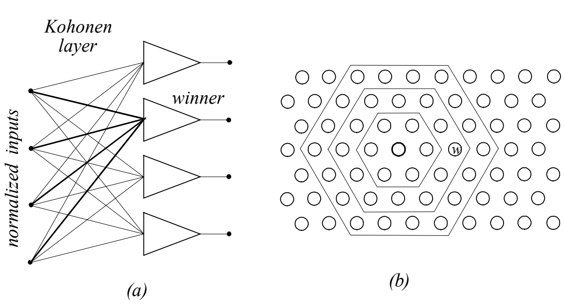
Sum of squared errors as a function of number of iterations for the "XOR" problem using modified EBP algorithm with Nguyen-Widrow weight initialization



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



The counterpropagation network.

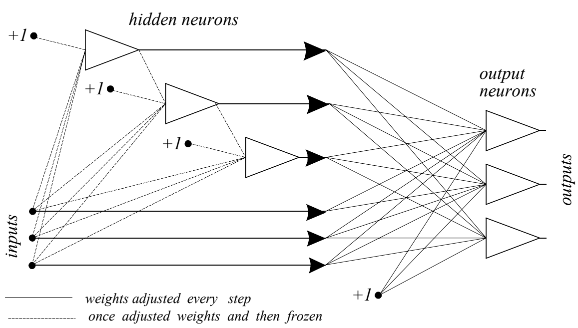


A winner take all - WTA architecture for cluster extracting in the unsupervised training mode: *(a)* network connections, *(b)* single layer network arranged into a hexagonal shape.

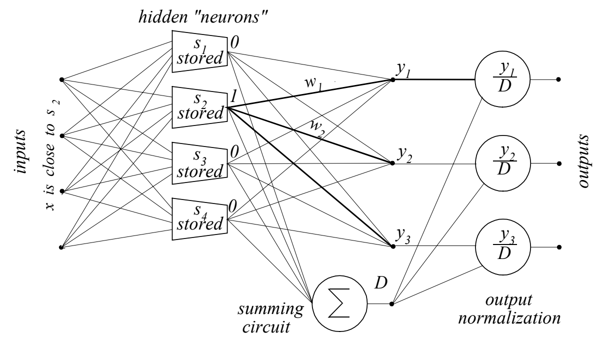
Find number and location of clusters in 4-dim. space

-2 2 -3 6	-2 2 -2 5	-5 7 -5 8	-4 4 -1 4
4 -4 4 3	-5 4 -5 8	-3 8 -8 9	-1 1 -2 2
-2 3 -4 3	0 1 -3 3	2 -2 2 2	-4 3 -3 2
-2 6 -8 8	-2 1 -5 3	-2 6 -8 9	0 3 -4 6
0 2 -5 4	-2 6 -7 6	2 -2 3 2	-4 5 -5 7
2 -4 5 2	-6 6 -7 8	-2 1 -5 3	-4 3 -2 4
-4 8 -4 9	-2 2 -2 6	2 -1 4 1	4 0 2 4
0 -2 6 3	-3 6 -5 7	-6 4 -6 6	-2 3 -3 4
4 -1 4 1	-4 6 -6 6	-2 3 -2 5	-2 1 -3 6
-3 7 -6 7	-6 7 -5 9	-1 2 -3 2	-2 1 -3 4
-2 3 -2 4	-2 5 -6 8	-2 5 -2 2	4 -3 5 0
3 -2 6 2	0 3 -2 6	0 -2 4 0	2 -4 4 0
0 2 -3 4	3 -2 4 4	-6 4 -6 8	-3 6 -6 7
-6 4 -8 7	-6 6 -6 6	-2 3 -4 4	4 -4 3 2
-3 5 -3 6	-3 6 -7 7	1 0 6 2	-1 4 -3 5
-3 5 -4 4	-3 8 -5 10	-4 4 -6 10	-2 5 -1 5
-3 2 -5 2	-2 7 -6 6	-1 3 -4 6	-3 6 -6 9
-1 3 -3 3	2 -2 4 4	-4 5 -8 6	-4 4 -2 4

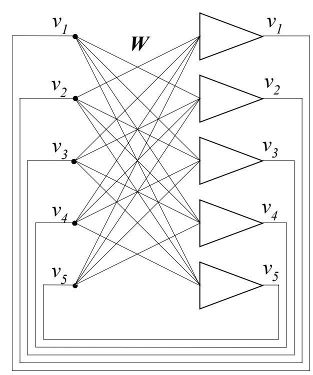
(-2 3 –3 4)   (4 –4 6 –6)   (2 –2 4 2)



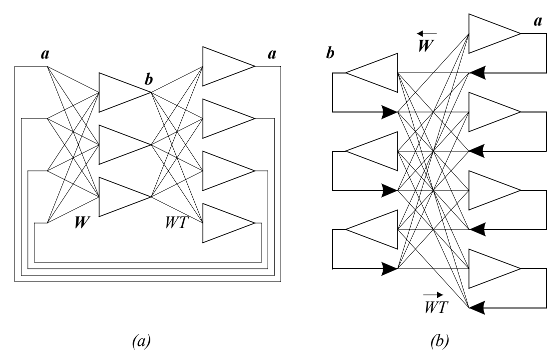
The cascade correlation architecture



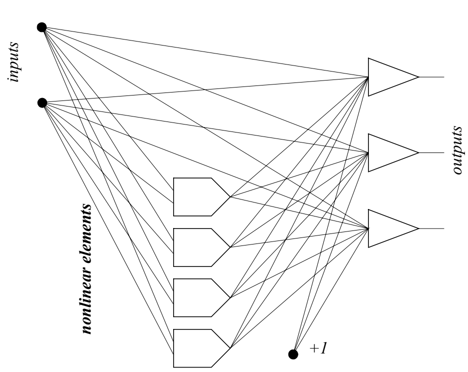
A typical structure of the radial basis function network.



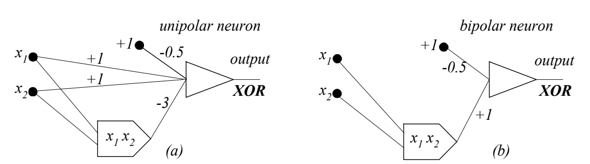
A Hopfield network or autoassociative memory



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:

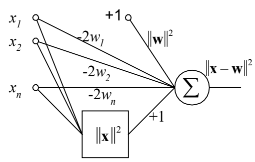
$$\|\mathbf{x}-\mathbf{w}\|^2 = (x_1 - w_1)^2 + (x_2 - w_2)^2 + \cdots + (x_n - w_n)^2$$

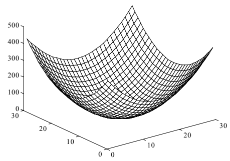
After defactorization

$$\|\mathbf{x}-\mathbf{w}\|^2 = x_1^2 + x_2^2 + \cdots + x_n^2 + w_1^2 + w_2^2 + \cdots + w_n^2 - 2(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n)$$

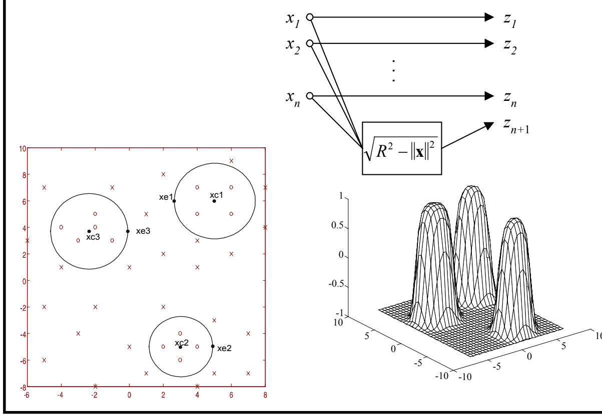
finally

$$\|\mathbf{x}-\mathbf{w}\|^2 = \mathbf{x}^T\mathbf{x} + \mathbf{w}^T\mathbf{w} - 2\mathbf{x}^T\mathbf{w} = \|\mathbf{x}\|^2 + \|\mathbf{w}\|^2 - 2net$$

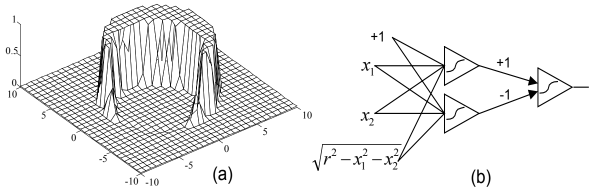




Input pattern transformation on a sphere

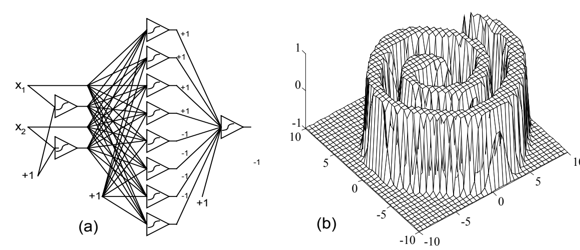


Input pattern transformation on a sphere 2



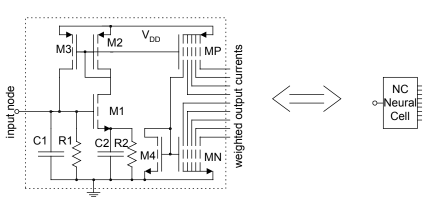
Network with two neurons capable of separating crescent shape of patterns (a) input-output mapping, (b) network diagram

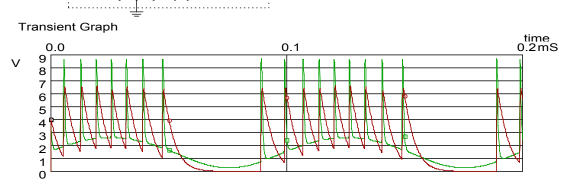
Input pattern transformation on a sphere 3



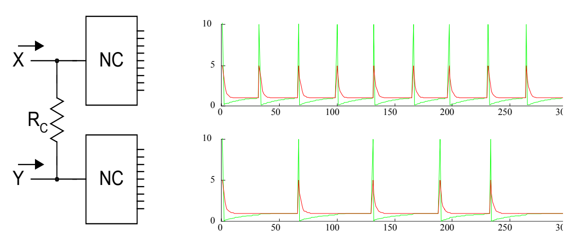
Spiral problem solved with sigmoidal type neurons (a) network diagram, (b) input-output mapping.

Pulse Coded Neural Networks

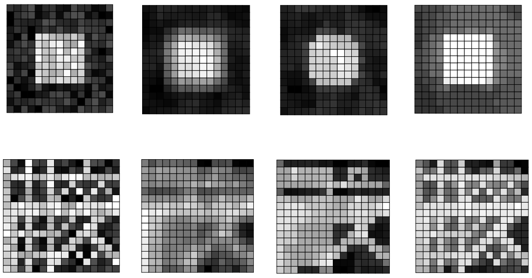




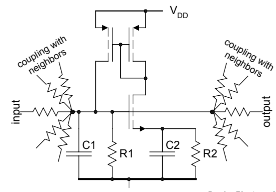
Pulse Coded Neural Networks 2

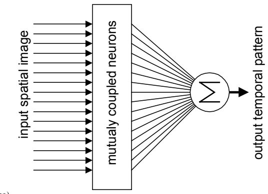


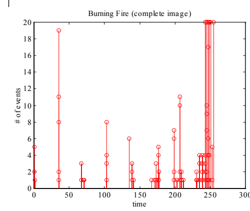
Pulse Coded Neural Networks 3



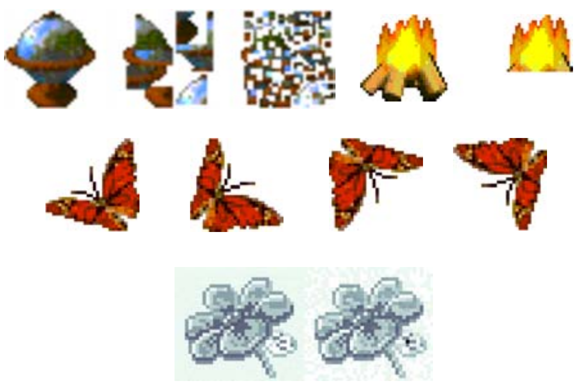
Pulse Coded Neural Networks 4







Pulse Coded Neural Networks 5



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 - #.

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

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

$$\overline{A} \quad \Rightarrow 1 - A \quad - \text{one minus } A$$

Boolean

$A \cap B$

0	0	0
0	1	0
1	0	0
1	1	1

$A \cup B$

0	0	0
0	1	0
1	0	0
1	1	1

Fuzzy

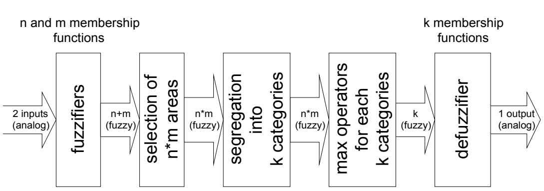
$A \cap B$

0.2	0.3	0.2
0.2	0.8	0.2
0.7	0.3	0.3
0.7	0.8	0.7

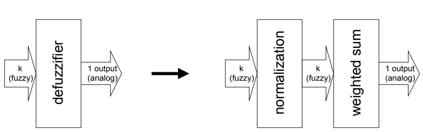
$A \cup B$

0.2	0.3	0.3
0.2	0.8	0.8
0.7	0.3	0.7
0.7	0.8	0.8

Fuzzy systems 2



Block diagram for Zadeh fuzzy controller

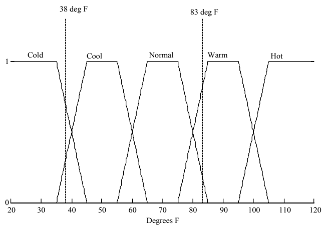


Takagi-Sugeno type defuzzifier

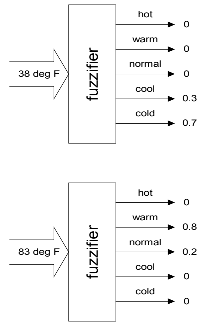
Fuzzification

- There are three major types of membership functions
  – Gausian, 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

Rule Evaluation

	Input 2				
Input 1	→				
↓	cold	cool	normal	warm	hot
cold	A	A	A	B	A
cool	A	A	B	C	B
norma	A	B	C	C	C
warm	A	B	C	D	D
hot	B	C	D	E	E

	Input 2				
Input 1	→				
↓	0 cold	0 cool	0.2 normal	0.8 warm	0 hot
0.7 cold	A	A	A	B	A
0.3 cool	A	A	B	C	B
0 normal	A	B	C	C	C
0 warm	A	B	C	D	D
0 hot	B	C	D	E	E

	Input 2				
Input 1	→				
↓	0 cold	0 cool	0.2 normal	0.8 warm	0 hot
0.7 cold	0	0	0.2 A	0.7B	0
0.3 cool	0	0	0.2 B	0.3 C	0
0 normal	0	0	0	0	0
0 warm	0	0	0	0	0
0 hot	0	0	0	0	0

Zadeh fuzzy tables

	Input 2				
Input 1	→				
↓	cold	cool	normal	warm	hot
cold	O1	O2	O3	O4	O5
cool	O6	O7	O8	O9	O10
normal	O11	O12	O13	O14	O15
warm	O16	O17	O18	O19	O20
hot	O21	O22	O23	O24	O25

	Input 2				
Input 1	→				
↓	0 cold	0 cool	0.2 normal	0.8 warm	0 hot
0.7 cold	O1	O2	O3	O4	O5
0.3 cool	O6	O7	O8	O9	O10
0 normal	O11	O12	O13	O14	O15
0 warm	O16	O17	O18	O19	O20
0 hot	O21	O22	O23	O24	O25

	Input 2				
Input 1	→				
↓	0 cold	0 cool	0.2 normal	0.8 warm	0 hot
0.7 cold	0	0	0.2*O3	0.7*O4	0
0.3 cool	0	0	0.2*O8	0.3*O9	0
0 normal	0	0	0	0	0
0 warm	0	0	0	0	0
0 hot	0	0	0	0	0

Tagagi-Sugeno fuzzy tables

Defuzzification

- The equation to describe the defuzzification process.
  - n – Number of membership functions
  - zk – Fuzzy output variables
  - zck – analog values from table

$$\text{Output} = \frac{\sum_{k=1}^{n} z_k zc_k}{\sum_{k=1}^{n} z_k}$$

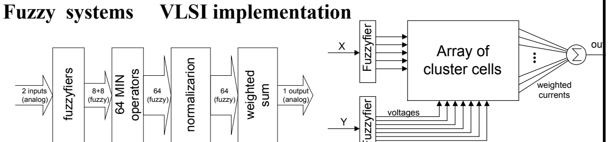
- Outputs:
  - Zadeh

$$\text{Output} = \frac{0.2*A + 0.7*B + 0.3*C}{0.2 + 0.7 + 0.3}$$

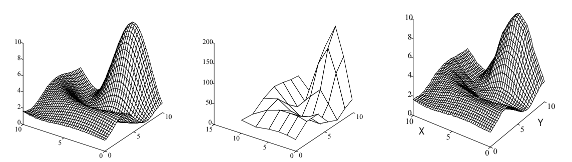
  - Tagagi-Sugeno

$$\text{Output} = \frac{0.2*O3 + 0.7*O4 + 0.2*O8 + 0.3*O9}{0.2 + 0.7 + 0.3 + 0.2}$$

Fuzzy systems VLSI implementation

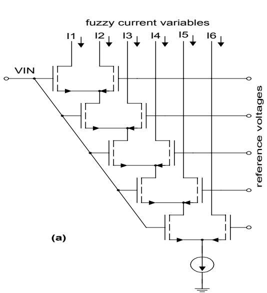


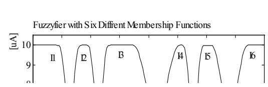
Block diagrams of the fuzzy VLSI chip



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

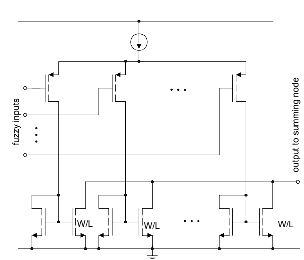
Fuzzy systems VLSI implementation 2



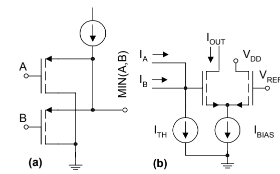


Fuzzifier (a) circuit diagram of fuzzifier, (b) example of the SPICE simulation

Fuzzy systems VLSI implementation 3

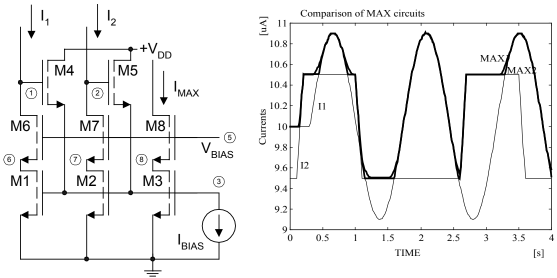


Defuzzifier using normalization and weighted sum



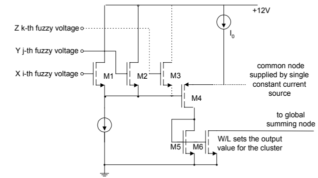
Selection circuits (a) MIN circuit in voltage mode (b) neuron circuit with threshold in the current mode

Fuzzy systems VLSI implementation 4

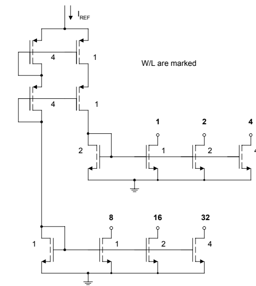


MAX operators (a) concept diagram and (b) simulation results for MAX1 and for the proposed MAX2.

Fuzzy systems VLSI implementation 5

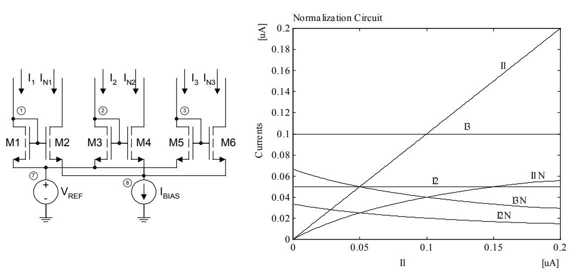


The cluster cell with rule selection (transistors M1-M4) and defuzzification (source I_0 and transistors M4-M6)



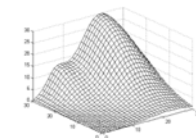
Six bit programmable current sources

Fuzzy systems VLSI implementation 6

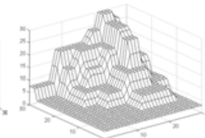


Normalization circuit (a) circuit diagram and (b) characteristics

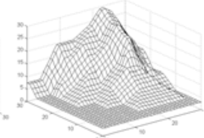
Fuzzy systems - microprocessor implementation



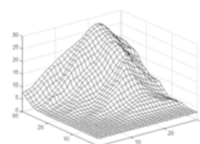
Required control surface



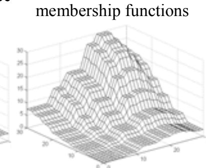
Zadeh with trapezoidal membership functions



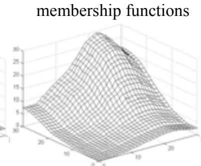
Zadeh with triangular membership functions



Zadeh with Gaussian membership functions



Tagagi-Sugeno with trapezoidal membership functions

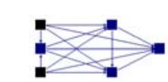


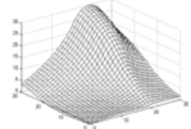
Tagagi-Sugeno with triangular membership functions

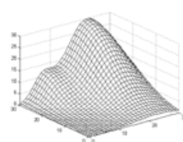
Fuzzy systems - microprocessor implementation 2

	Approach used	Error SSE	Error MSE
1	Zadeh fuzzy controller with trapezoidal membership function	908.4	0.945
2	Zadeh fuzzy controller with triangular membership function	644.4	0.671
3	Zadeh fuzzy controller with Gaussian membership function	562.0	0.585
4	Tagagi-Sugeno fuzzy controller with trapezoidal membership function	296.5	0.309
5	Tagagi-Sugeno fuzzy controller with triangular membership function	210.8	0.219
6	Tagagi-Sugeno fuzzy controller with Gaussian membership function	294.2	0.306

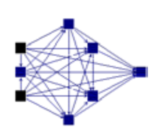
Neural systems - microprocessor implementation

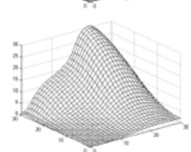


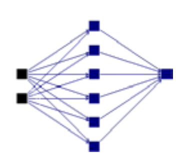


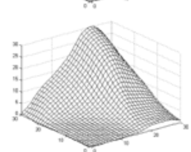


Required control surface









Neural systems - microprocessor implementation 2

	Approach used	Error SSE	Error MSE
1	Neural network with 3 neurons in cascade	0.5559	0.000578
2	Neural network with 5 neurons in cascade	0.0895	0.000093
3	Neural network with 6 neurons in one hidden layer	0.2902	0.000302

Comparison of various fuzzy and neural controllers

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

Genetic Algorithms

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 $(\sin^2(x)-0.5*x)^2$ 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

Genetic Algorithms 3

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

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:

101101 -> 45 110001 -> 49 100101 -> 37 110001 -> 49
100111 -> 39 101101 -> 45 110001 -> 49 101000 -> 40

Genetic Algorithms 5

Reproduction

101101 -> 45 110001 -> 49 100101 -> 37 110001 -> 49
100111 -> 39 101101 -> 45 110001 -> 49 101000 -> 40

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 simples 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

101111 -> 47 110101 -> 53 100001 -> 33 110000 -> 48
100101 -> 37 101001 -> 41 110101 -> 53 101001 -> 41

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

string number	string	decimal value	variable value	function value	fraction of total
1	*010111*	*47*	*1.175*	*0.0696*	*0.1587*
2	*100100*	*37*	*0.925*	*0.0307*	*0.0701*
3	*110101*	*53*	*1.325*	*0.0774*	*0.1766*
4	*010001*	*41*	*1.025*	*0.0475*	*0.1084*
5	*100001*	*33*	*0.825*	*0.0161*	*0.0368*
6	*110101*	*53*	*1.325*	*0.0774*	*0.1766*
7	*110000*	*48*	*1.200*	*0.0722*	*0.1646*
8	*101001*	*41*	*1.025*	*0.0475*	*0.1084*
Total				*0.4387*	*1.0000*

Genetic Algorithms 8

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 110000 -> 48 101001 -> 41
110101 -> 53 110000 -> 48 101001 -> 41 110101 -> 53

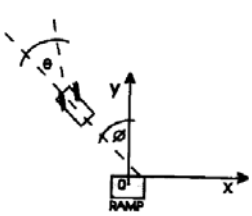
which produces following children:

010101 -> 21 110000 -> 48 110001 -> 49 101101 -> 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 9

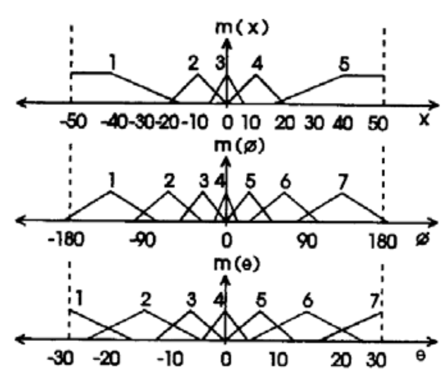


$$x_{i+1} = x_i - r\sin(\phi_i)$$
$$y_{i+1} = y_i + r\cos(\phi_i)$$
$$\phi_{i+1} = \phi_i + \Theta_i$$

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

$$E = w_x x^2 + w_\phi \phi^2$$

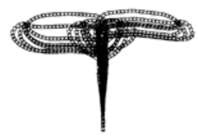
Genetic Algorithms 10



Genetic Algorithms 11

φ \ x	1	2	3	4	5
1	7	7	7	7	7
2	4	6	7	7	7
3	1	3	5	6	7
4	1	2	4	6	7
5	1	2	3	5	7
6	1	1	1	2	4
7	1	1	1	1	1

φ \ x	1	2	3	4	5
1	5	6	6	7	7
2	3	3	7	7	7
3	2	3	5	7	6
4	3	3	4	5	7
5	1	1	3	5	7
6	1	1	1	5	6
7	1	1	1	2	3



Genetic Algorithms 12

φ \ x	1	2	3	4	5
1	5	6	6	7	7
2	3	5	6	7	7
3	2	3	5	6	7
4	2	3	4	5	6
5	1	2	3	5	6
6	1	1	2	3	5
7	1	1	2	2	3

(a)

(b)

φ \ x	1	2	3	4	5
1	7	7	7	1	1
2	2	4	7	7	7
3	1	1	6	7	7
4	1	1	6	7	7
5	1	1	2	7	7
6	1	1	1	4	6
7	7	7	1	1	1

B. M. 'Dan' Wilamowski

University of Idaho

Soft Computing and its Application

nn.uidaho.edu

wialm@ieee.org