

CS5486

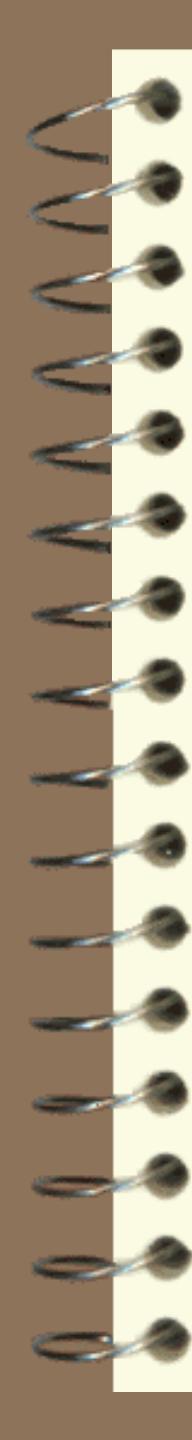
Intelligent Systems

Prof. Jun Wang

Department of Computer Science

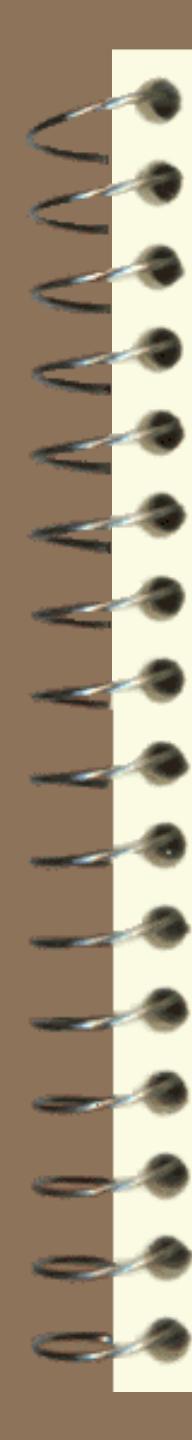
Tel: 3442 9701

Email: jwang.cs@cityu.edu.hk



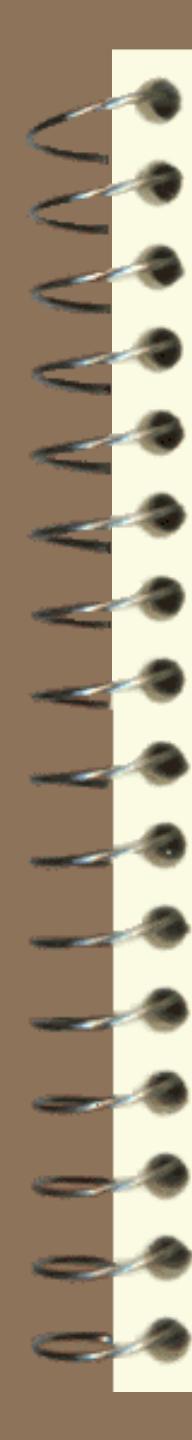
Aims

This course aims to equip students with the skills of problem solving using artificial intelligence (AI) techniques through a demonstrable knowledge in a range problem solving methods and the associated knowledge representation and machine learning techniques.



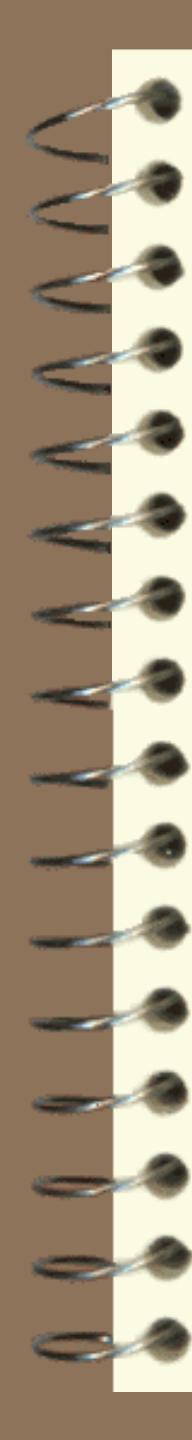
Keywords

Artificial intelligence vs. computational intelligence. Neural networks. Knowledge representations. Machine learning. Rule-based systems. Fuzzy Systems. Evolutionary computation.



Syllabus

- An introduction to the goals and objectives of AI as a discipline and its milestones. Approaches in AI. Major components in intelligent systems.
- Methods of knowledge acquisition and representations. Associative memory. Techniques on machine learning such as supervised learning, unsupervised learning, reinforcement learning, and deep learning. Generalization.

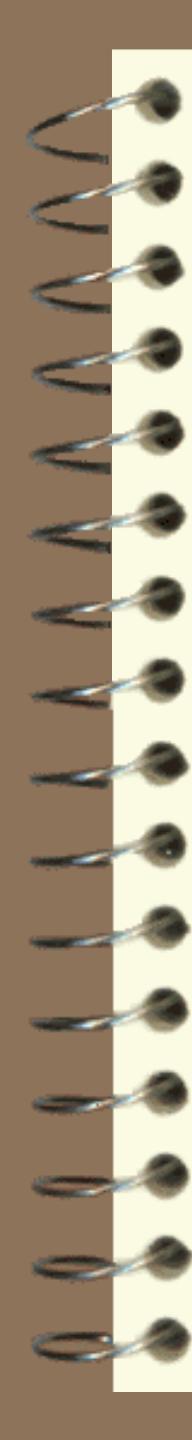


Syllabus (cont'd)

 Basic concepts of graph and tree search.
Optimization methods such as stochastic annealing, neurodynamic optimization, genetic algorithm, particle swarm optimization, ant colony optimization, and differential evolution.

References

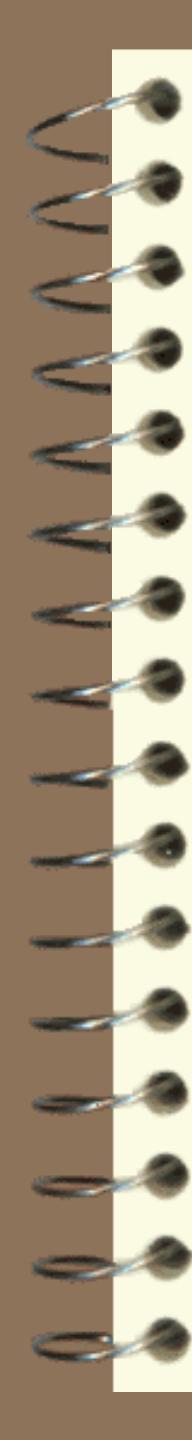
-
- R. Rojas, *Neural Networks: A Systematic Introduction*, Springer, 1996.
 - S. Haykin, *Neural Networks and Learning Machines* (3rd Ed), Prentice-Hall, 2009.
 - S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. 3rd Ed. Prentice-Hall (2009)
 - C.-T. Lin and C.S. G. Lee, *Neural Fuzzy Systems*, Prentice-Hall, 1996.



Intelligence

“Intelligence is a mental quality that consists of the abilities to learn from experience, adapt to new situations, understand and handle abstract concepts, and use knowledge to manipulate one’s environment.”

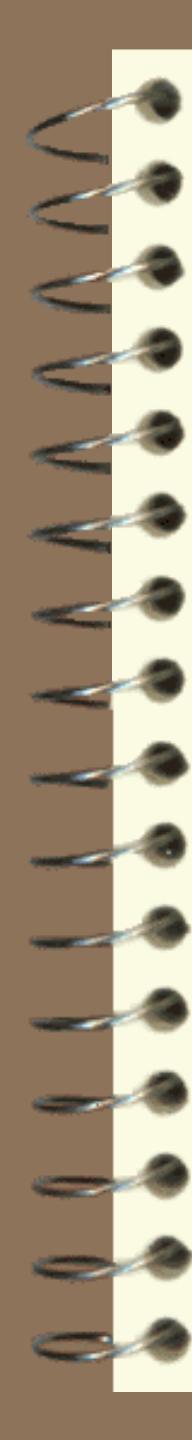
Britannica



Definition of Intelligent Systems

A system is an intelligent system if it exhibits some intelligent behaviors.

For example, neural networks, fuzzy systems, simulated annealing, genetic algorithms, and expert systems.



Intelligent Behaviors

- Inference: Deduction vs. Induction
(generalization); e.g., judgment and pattern recognition
- Learning and adaptation: Evolutionary processes; e.g., learning from examples
- Creativity: e.g., planning and design

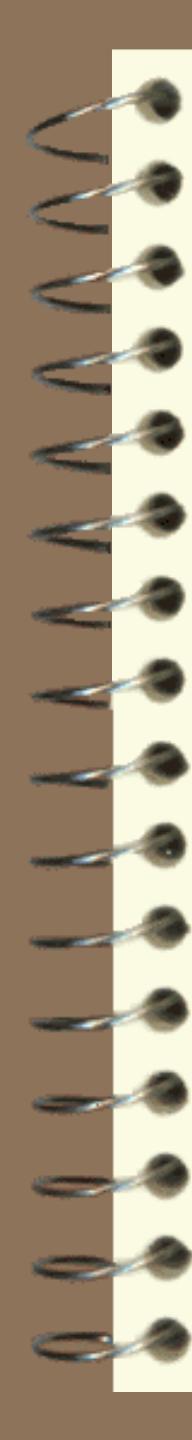
How far have we got?

**An Electric Brain Capable of
Translating Foreign Languages is
Being Built..**

*An headline from the popular press
of 1950*

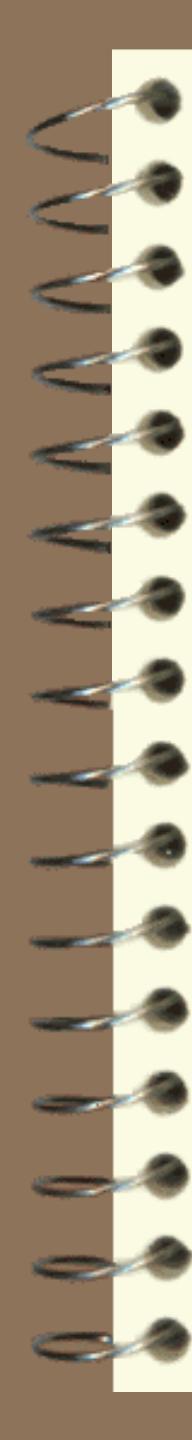


**Today's most intelligent machine is far from being
able to do what a child easily does, such as eating food
with a knife and fork, peeling an orange, etc**



Milestones of Intelligent System Development

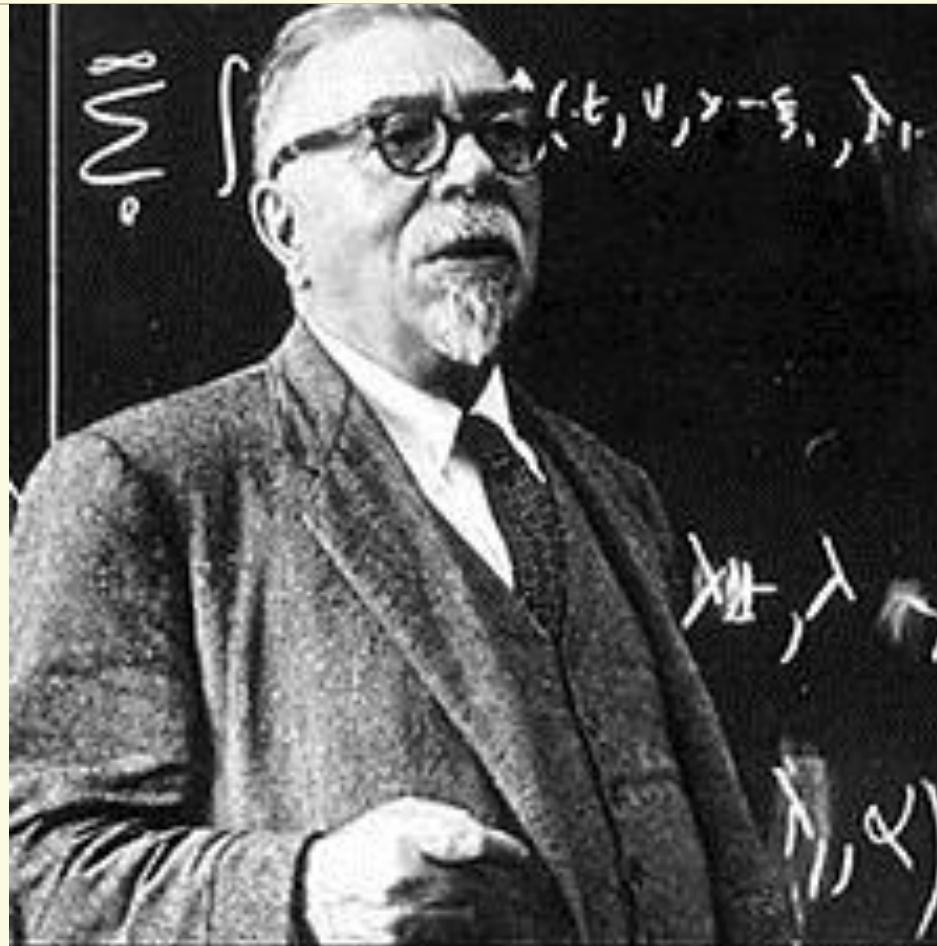
- 1940's: Cybernetics by Wiener
- 1943: Threshold logic networks by McCulloch and Pitts
- 1950's-1960's: Perceptrons by Rosenblatt
- 1960's Adaline by Widrow
- 1970's: AI and Expert systems
- 1970's: Fuzzy logic by L. Zadeh
- 1974: Back propagation algorithm by P. Werbos
- 1970's: Adaptive resonance theory by S. Grossberg
- 1970's: Self-organizing map by T. Kohonen
- 1980's: Hopfield networks by J. Hopfield
- 1980's: Genetic algorithms by J. Holland
- 1980's: Simulated annealing by Kirkpatrick et al.
- 2006-: Deep learning by Hinton et al.



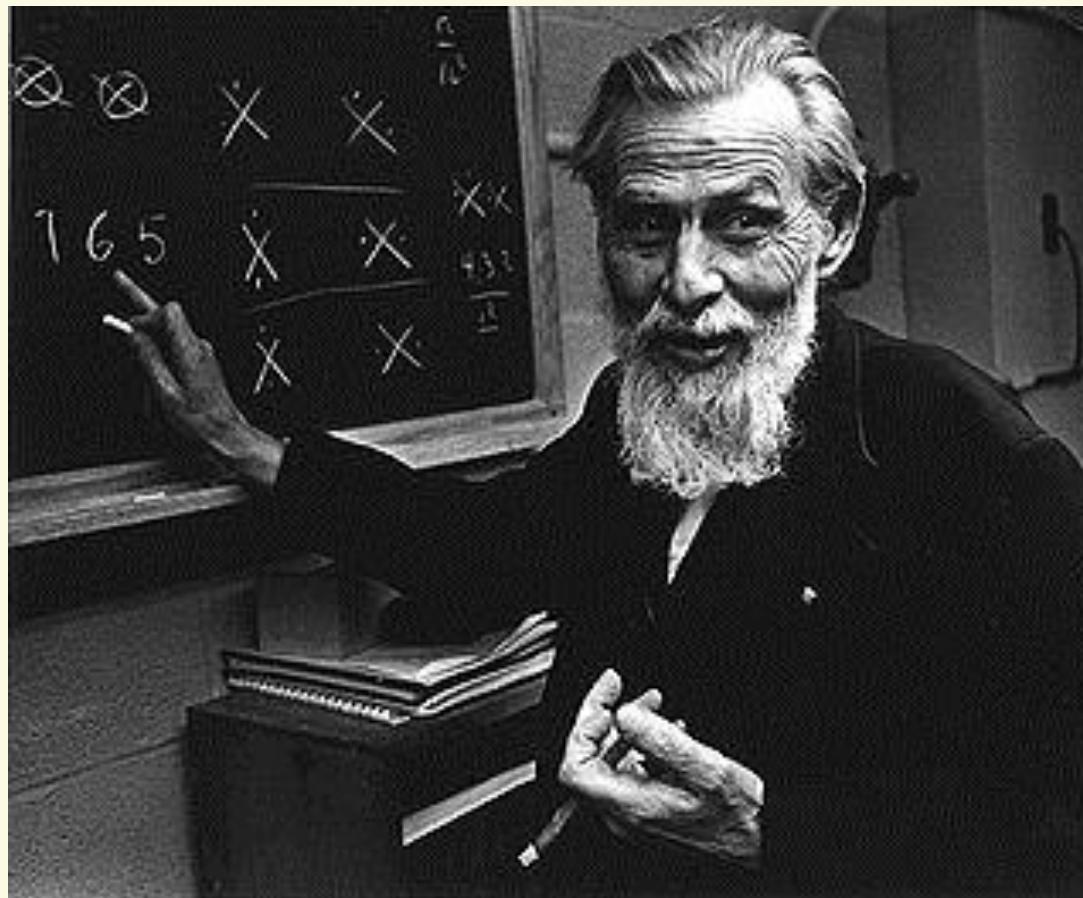
Milestones (cont'd)

- Deep Blue (IBM), 1997
defeated chess world champion
Garry Kasparov
- Alpha Go (Google), 2016
defeated go nine dan Sedol Lee

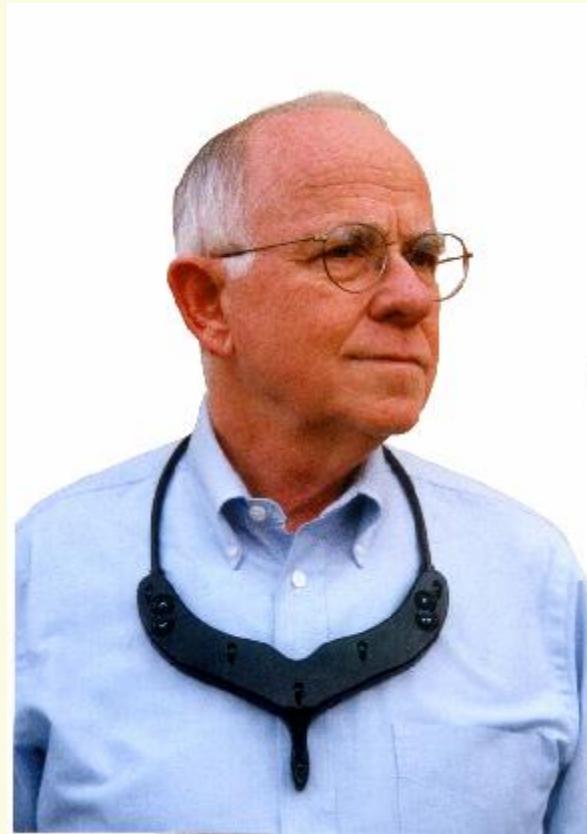
Professor Norbert Wiener



Professor Warren S. McCulloch

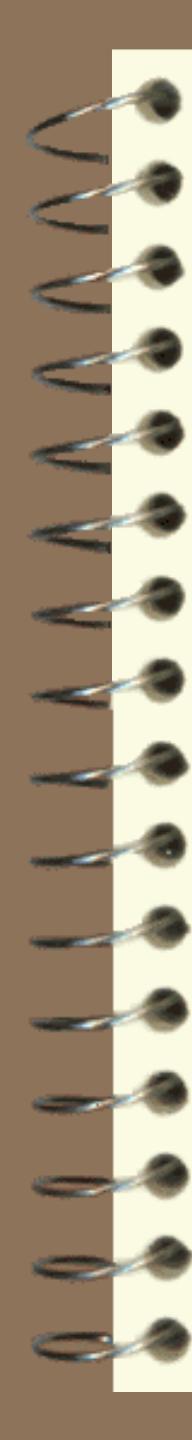


Professor Bernard Widrow



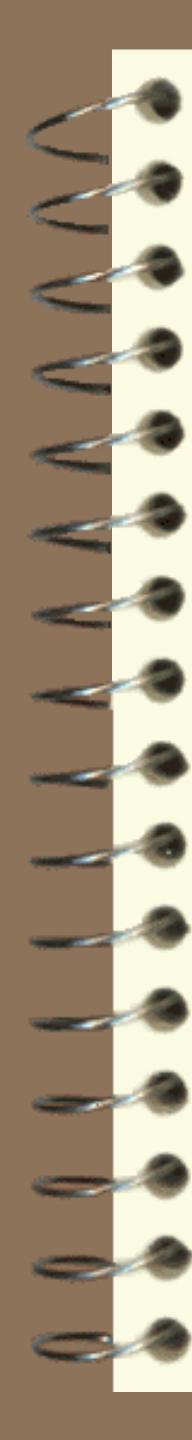
Professor Lofti A. Zadeh





Dr. Paul J. Werbos

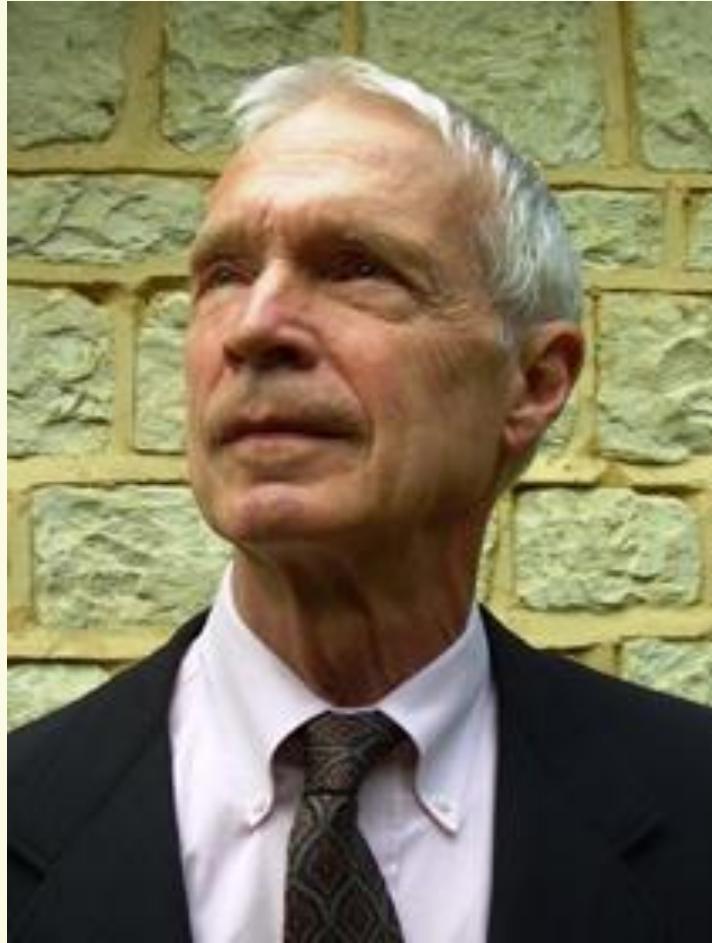




Professor Teuvo Kohonen



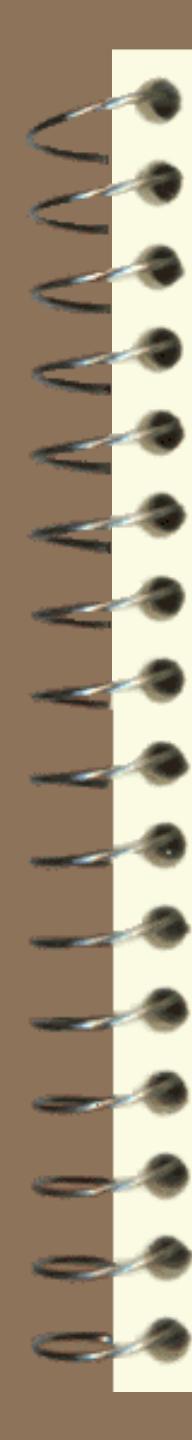
Professor John J. Hopfield





Professor Jeffrey E. Hinton





Engineering Applications of Intelligent Systems

- Pattern recognition: e.g., image processing, pattern analysis, speech recognition, etc.
- Control and robotics: e.g., modeling and estimation
- Associative memory (content-addressable memory)
- Forecasting: e.g., in financial engineering

智能機械猩猩英科學家首研

【明報專訊】據英國《星期日泰晤士報》周日報道，英國一名科學家正在研發一頭人工智能猩猩機械人，如果成功，這將是全球首部有認知能力的機器。

像人類般思考

年初剛獲得OBE勳銜的格蘭德，已經製成這部機械人的手、眼及頭，在一年內會把這些部分與「大腦」連接。到時這頭名為「露西」的機械猩猩，將能夠運用手腳及眼睛，但「牠」與一般機械人不同，不會有程式事先制定手腳的使用方法，而是學習如何移動和對外界刺激作出反應。

靠神經網絡學習作反應

格蘭德說，露西的大腦在開始時會空白一片，牠起初只有求生意志和接收外界刺激的能力，其他東西則要靠其內置的神經網絡學習。該網絡容許程式同一時間及隨機地執行，與生物的機制相似。

如果成功，露西將成為首個能像人類思考的機械人。不少電腦專家認為，格蘭德成功的機會不大，但牛津的生物學家道金斯卻投他信心一票。「如果有人能夠作出如此重大突破，那人就是他了。」道金斯說。

曾淵倉專欄： 電腦神經網絡 的應用

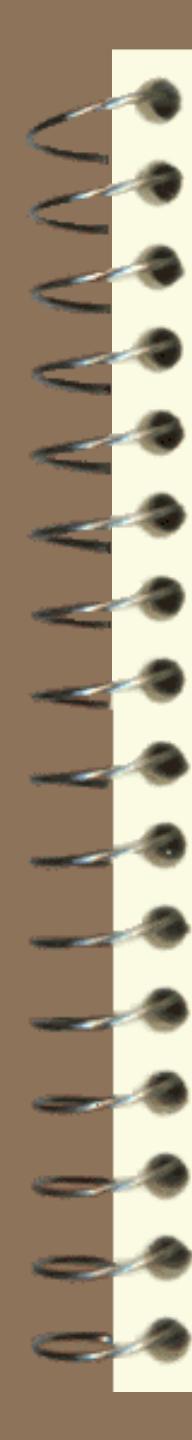
日期：97年11月10日
星期一

我開始認真的研究投資之道是因為遇上十年前的大股災，虧損得厲害，災後痛定思痛，才開始研究股市。

股市投資的研究，在美國非常之盛，美資大行聘用了不少各類專家研究，有學經濟的，有學數學的，有學電腦的……，今年諾貝爾經濟學獎頒發給一位研究出數學模式來計算期權價格的學者，說明了數學模式應用在西方國家之興盛。但數學模式往往深奧難解，得依靠電腦及軟件的協助，因此近年來這類軟件的研究也很盛行。其中發展潛力最强的軟件莫過於電腦神經網絡模擬。甚麼是電腦神經網絡呢？簡單的說是以電腦模擬人腦，能高速的平行思考，電腦思考速度比人腦快，記憶力比人腦強，因此電腦的思考能力比人類強，最近有部名為「深藍」的電腦就擊敗了世界棋王。

電腦神經網絡軟件可以記憶、學習過去各種各樣的股市走勢，然後推斷一個最可能發生的情況，協助決策者做決定。下個月開始，我與城大的另一位教授郝剛博士將合作利用這個最新的技術研究香港股市，已獲城大高層慷慨贊助研究經費。

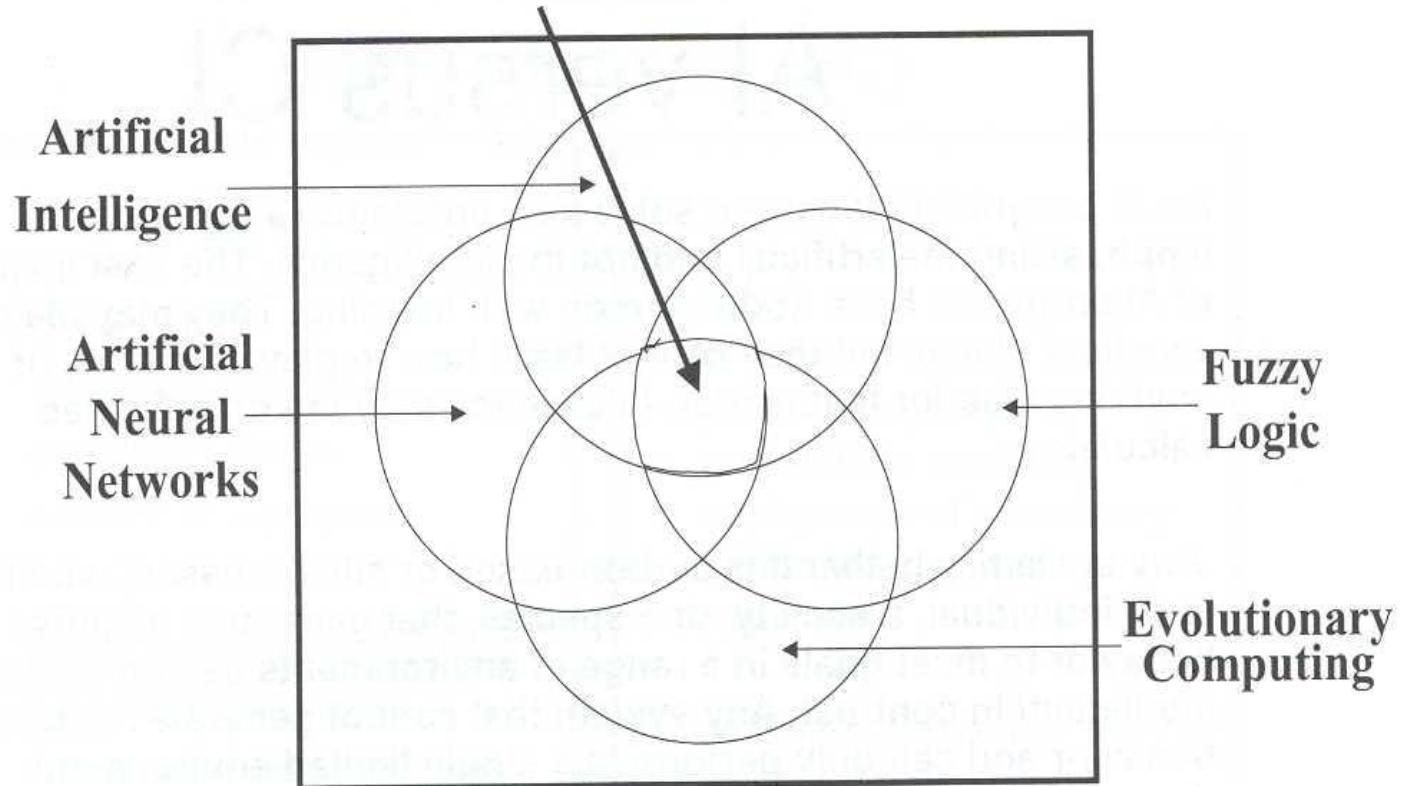
為了讓公眾能了解這個最新的技術，郝剛博士將在十二月十三日下午三時至六時在城大做主題演講，歡迎你們來聽，也像我一樣，在災後發憤研究投資之道。○



Computational Intelligence

- Coined by IEEE Neural Networks Council in 1994.
- Represent a new generation of intelligent systems.
- Consist of Neural Networks, Fuzzy Logic, and evolutionary computing techniques (genetic algorithms).

Intelligent System

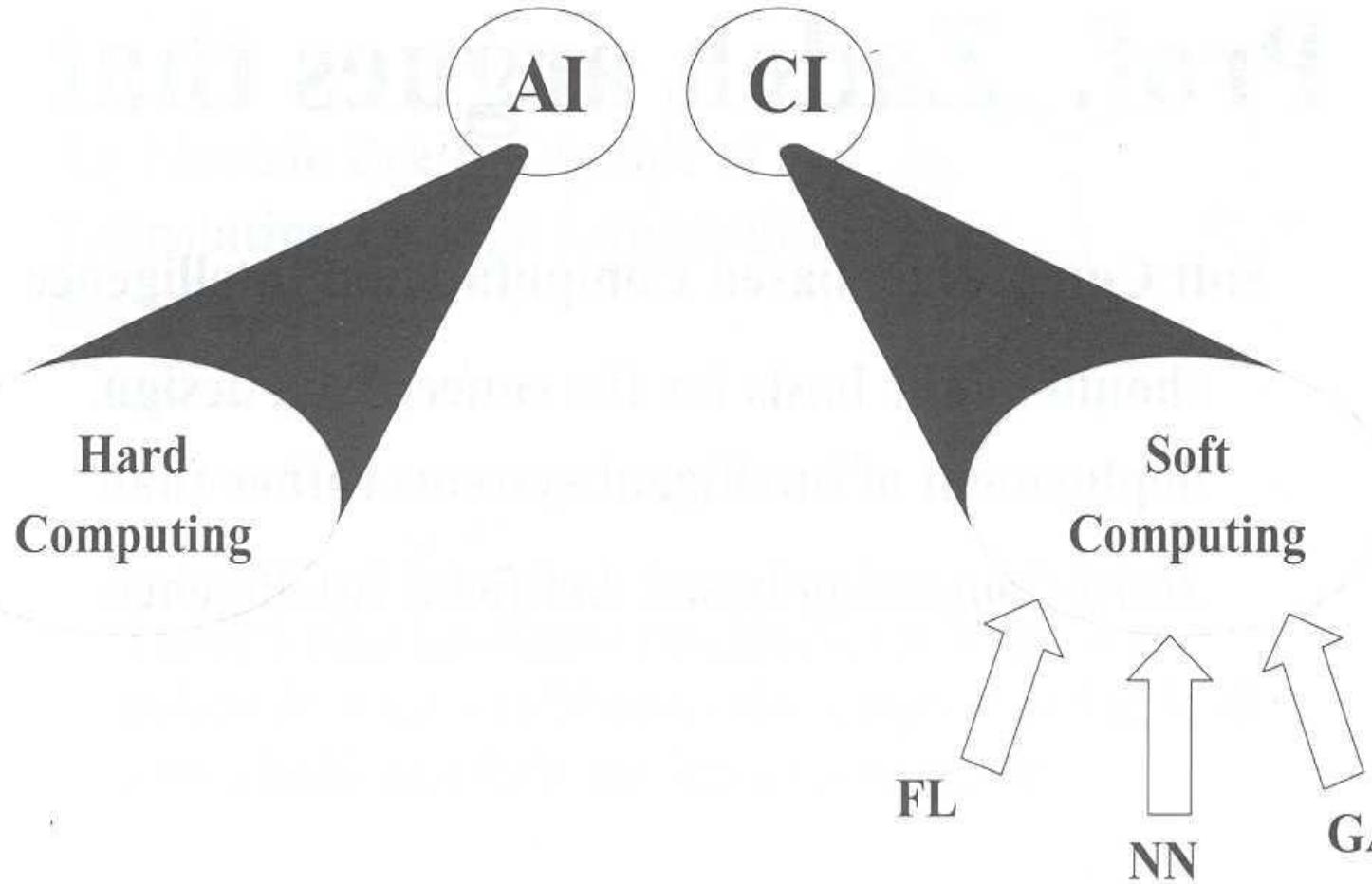


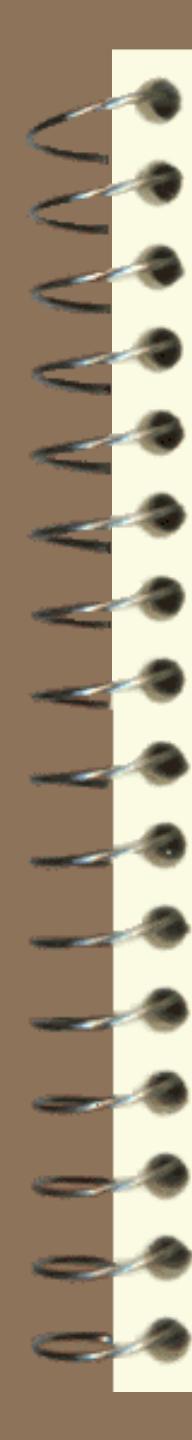
AI versus CI

An AI program that cannot solve new problems in new ways is emphasizing the artificial and not the intelligence. The vast majority of AI programs have nothing to do with learning. They may play excellent chess, but they cannot learn how to play checkers, or anything else for that matter. In essence they are complicated calculators.

Any system, whether it is carbon-based or silicon-based, whether it is an individual, a society, or a species, that generates adaptive behavior to meet goals in a range of environments can be said to be intelligent/ In contrast. Any system that cannot generate adaptive behavior and can only perform in a single limited environment demonstrates no intelligence.

(Folger)





Soft Computing

“Soft computing based on computational intelligence should be the basis for the conception, design, deployment of intelligent systems rather than hard computing based on artificial intelligence.”

Lofti Zadeh

Hard Computing (HC) versus Soft Computing (SC)

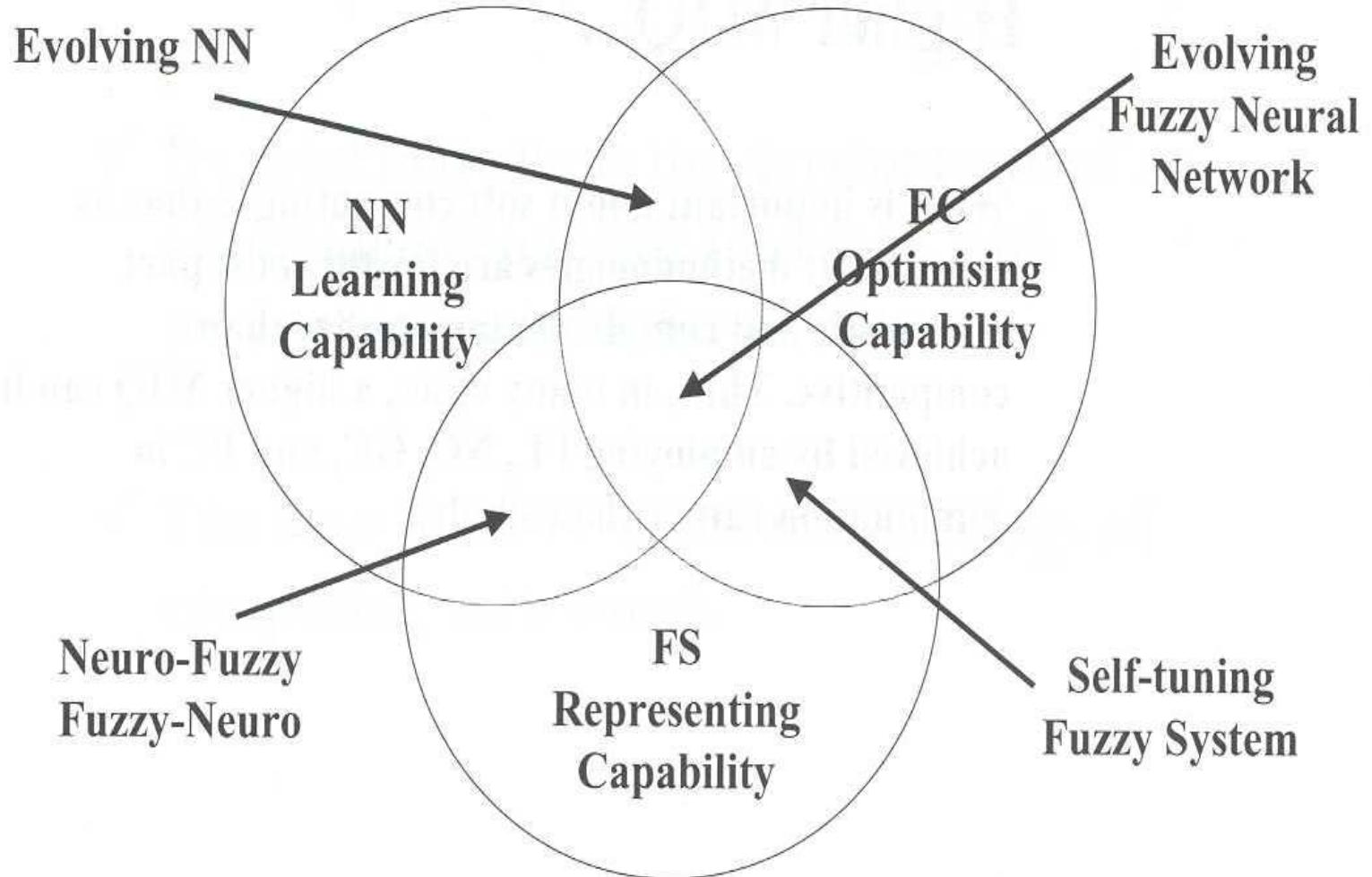
HC

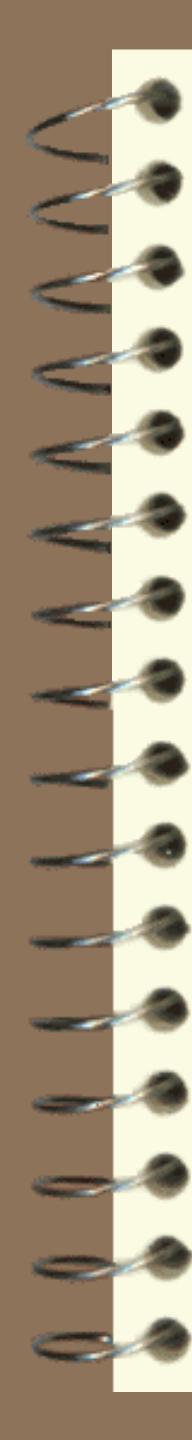
- Bivalent Logic
 - Numerical analysis
 - probability theory
 - differential equations
 - functional analysis
 - Mathematical programming
 - approximation theory
- quantitative, precise, formal*

SC

- fuzzy Logic
- neurocomputing
- genetic computing
- probabilistic reasoning
- management of uncertainty
- evidential reasoning
- rough sets

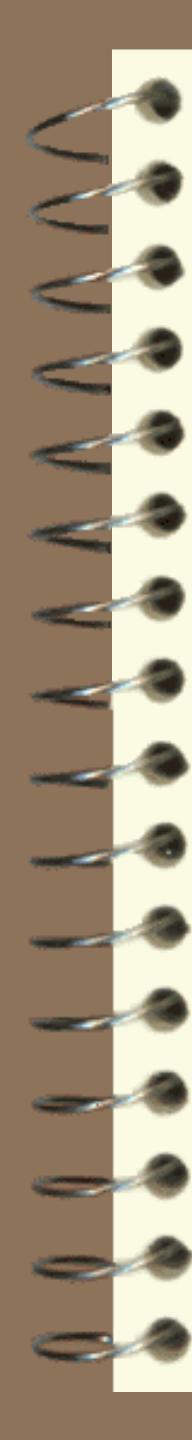
qualitative, imprecise, informal





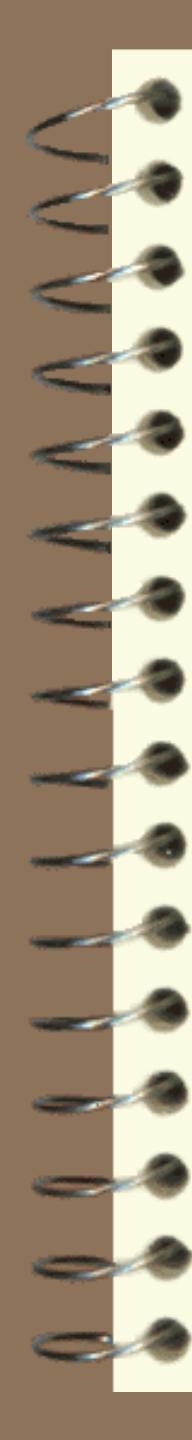
What are Neural Networks?

- Composed of a number of interconnected neurons, resembling the human brain.
- Also known as connectionist models, parallel distributed processing (PDP) models, neural computers, and neuromorphic systems.



Components of Neural Networks

- A number of artificial neurons (also known as nodes, processing units, or computational elements)
- Massive inter-neuron connections with different strengths (also known as synaptic weights).
- Input and output channels

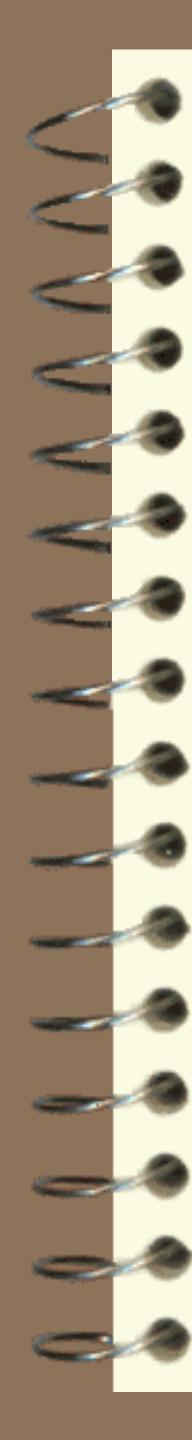


Formalization of Neural Networks

$$\text{ANN} = (\text{ARCH}, \text{RULE})$$

ARCH: architecture, refers to the combination of components

RULE: rules, refers to the set of rules that relate the components



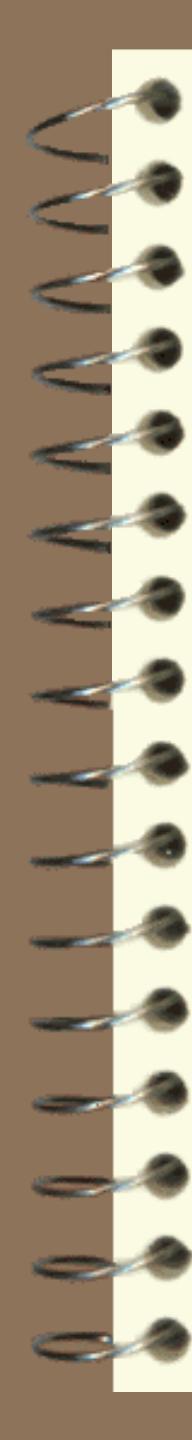
Architecture of Neural Networks

$$\text{ARCH} = (u, v, w, x, y)$$

Simple and alike neurons represented by u and v in N -dimensional space

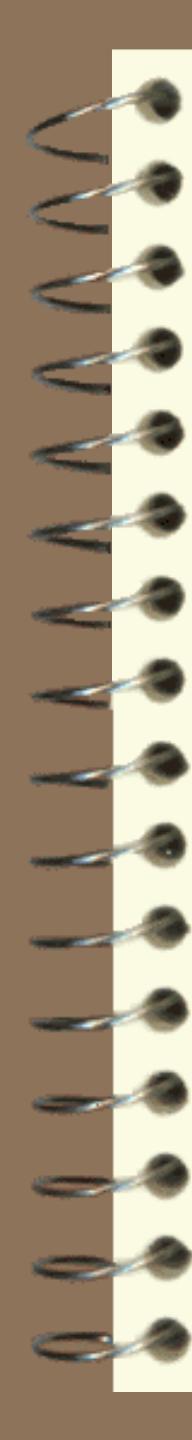
Inter-neuron connection weights represented by w in M -dimensional space

External input and outputs represented respectively by x and y in n and m -dimensional space



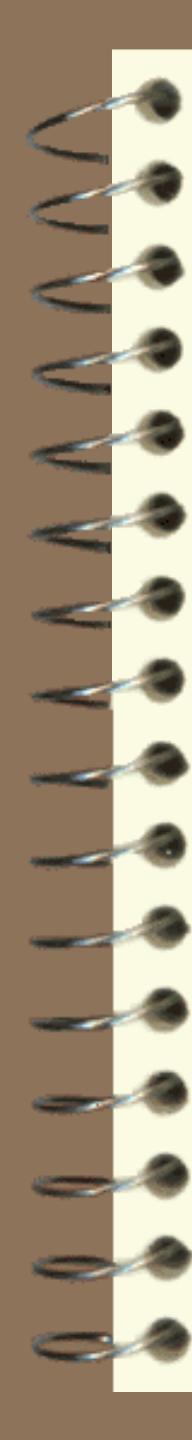
Model of Neurons

- Biological neurons: 10^{10} - 10^{11}
- Highly simplified
- Fire activities are quantified by using state variables (also called activation states)
- Net input to a neuron is usually a weighted sum of state variables from other neurons, input and/or output variables
- Net input to a neuron usually goes thru a nonlinear transformation called activation



Connections between Neurons

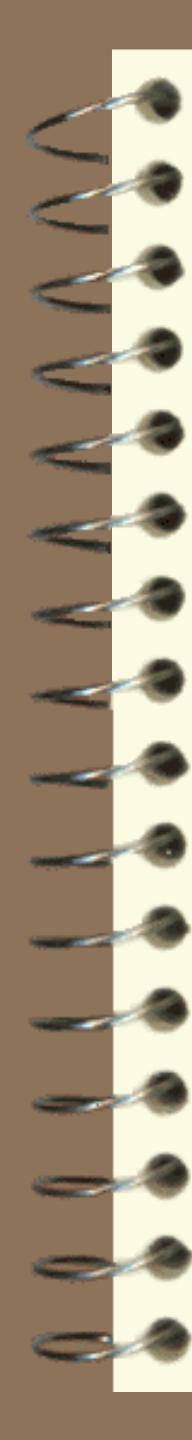
- Adaptive: Synaptic connections with adjustable weights
- Excitatory (positive weight) vs. inhibitory (negative weight)
- Distributed knowledge representation, different from digital computers



Rules of Neural Networks

$$\text{RULE} = (E, F, G, H, L)$$

- E : Evaluation rule mapped from v and/or y to a real line; e.g., error function or energy function
- F : Activation rule mapped from u to v ; e.g., activation function
- G : Aggregation rule mapped from v , w , and/or x to u ; e.g., weighted sum
- H : output rule mapped from v to y , y usually is a subset of v
- L : Learning rule mapped from v , w , and x to w , usually iterative



Learning in Neural Networks

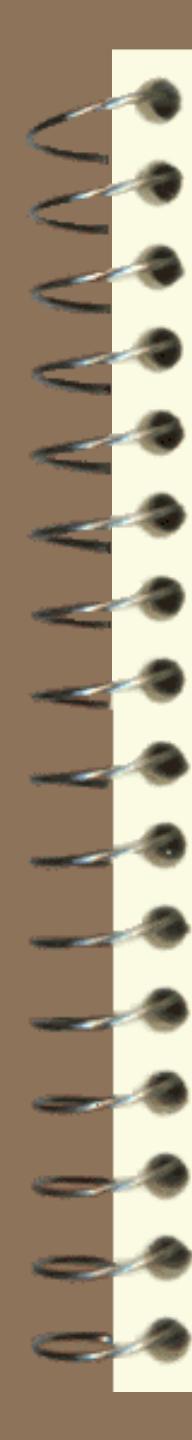
- Goal: To improve performance
- Means: interact with environment
- A process by which the adaptable parameters of an ANN are adjusted thru an iterative process of stimulation by the environment in which the ANN is embedded
- Supervised vs. unsupervised



On Learning

“By three methods we may learn wisdom: First, by reflection, which is the noblest; second, by imitation, which is the easiest, and third by experience, which is the bitterest.”

Confucius 孔子



General Incremental Learning Rule

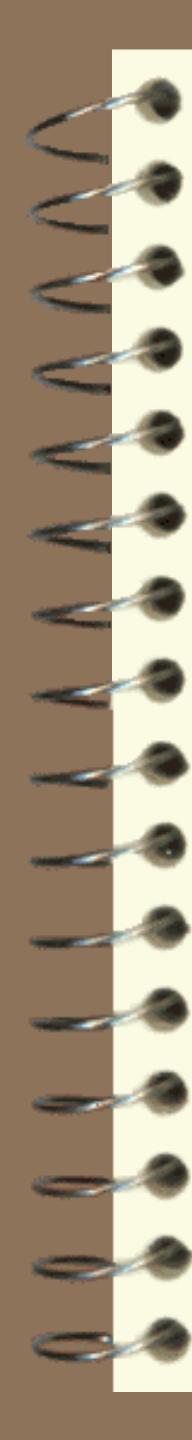
- Discrete-time:

$$w(t+1) = w(t) + L(v, w, x)$$

$$\Delta w(t) = L(v, w, x)$$

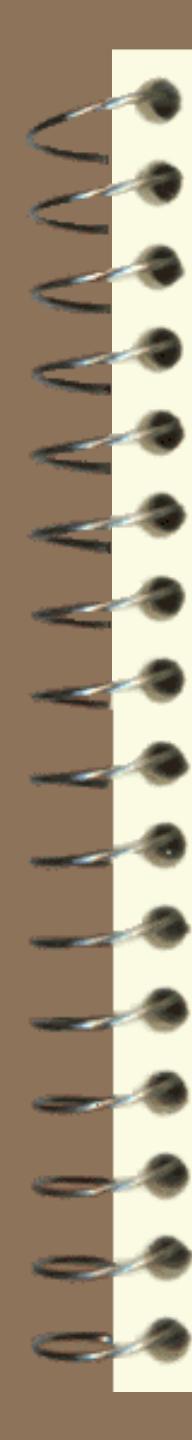
- Continuous-time:

$$\frac{dw(t)}{dt} = L(v, w, x)$$



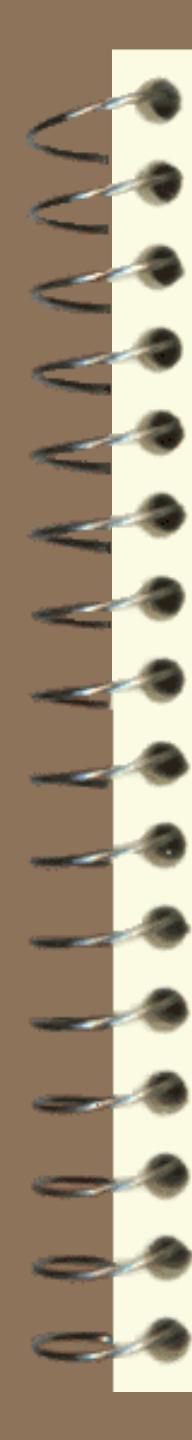
Two-Time Scale Dynamics in Neural Networks

- ▮ Faster dynamics in neuron activities represented by u and v . Also called as short-term memory
- ▮ Slower dynamics in connection weight activities represented by w . Also called as long-term memory



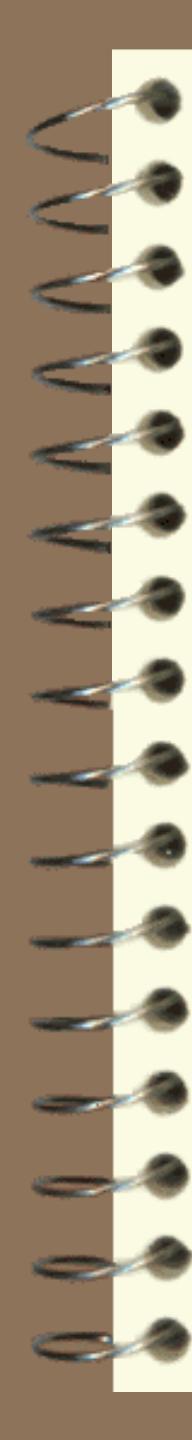
Categories of Neural Networks

- Deterministic vs. stochastic, in terms of F
- Feedforward vs. recurrent, in terms of G and H
- Semilinear vs. higher-order, in terms of G
- Supervised vs. unsupervised, in terms of L



Definition of Neural Networks

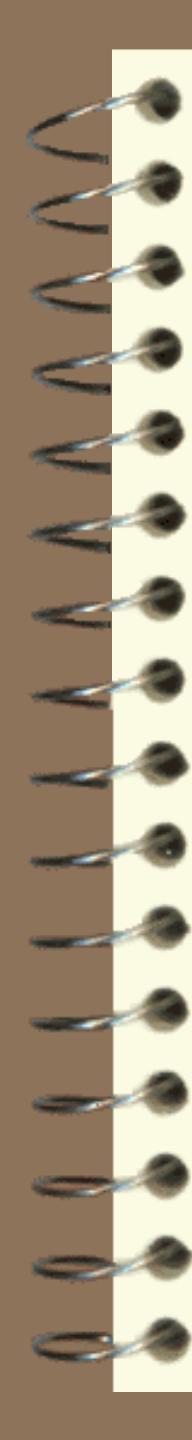
Massive parallel distributed
processors that have a
natural property for storing
experiential knowledge and
making it available for use



Features of Neural Networks

Resemble the brains in two aspects:

1. Knowledge acquisition: knowledge is acquired by neural networks thru learning processes.
2. Knowledge representation: Inter-neuron connections, known as synaptic weights are used to store acquired knowledge



Properties of Neural Networks

- 1 Nonlinearity
- 2 Input-output mapping
- 3 Adaptivity
- 4 Contextual information
- 5 Fault tolerance
- 6 hardware implementability
- 7 Uniformity of analysis and design
- 8 Neurobiological analogy and plausibility

McCulloch-Pitts Neuron

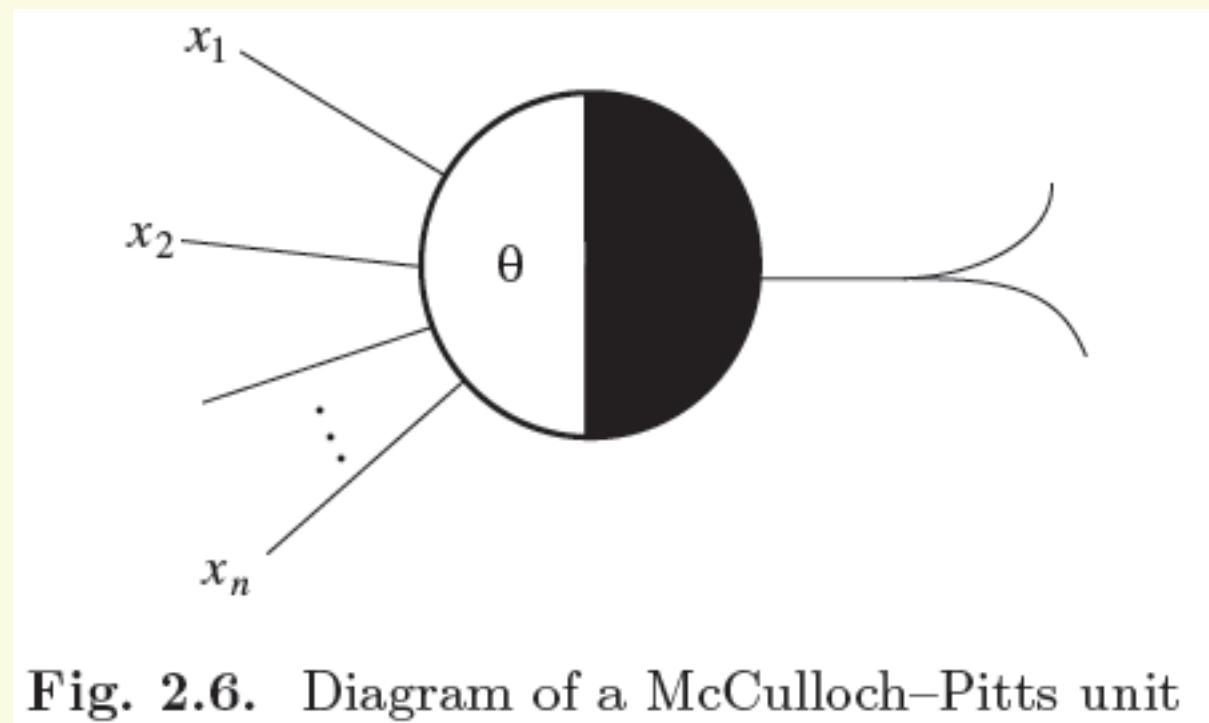
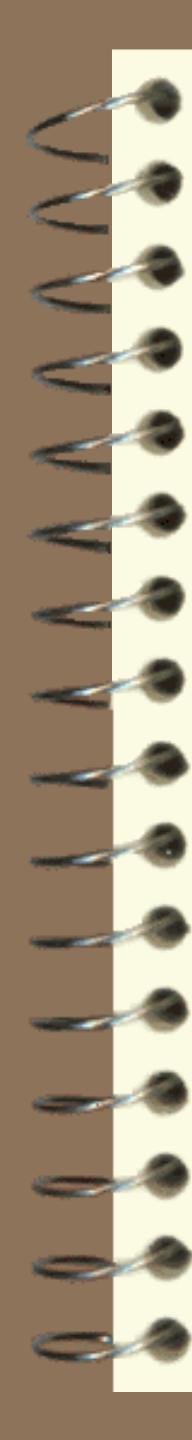


Fig. 2.6. Diagram of a McCulloch-Pitts unit



McCulloch-Pitts Neurons

- Binary values {0, 1}
- Unity connection weights of 1 and -1
- If an input to a neuron is 1 and the associated weight is -1, then the output of the neuron is 0
- Otherwise, if the weighted sum of input is not less than a threshold, then the output is 1; or is less than the threshold, then 0.

Threshold Logic

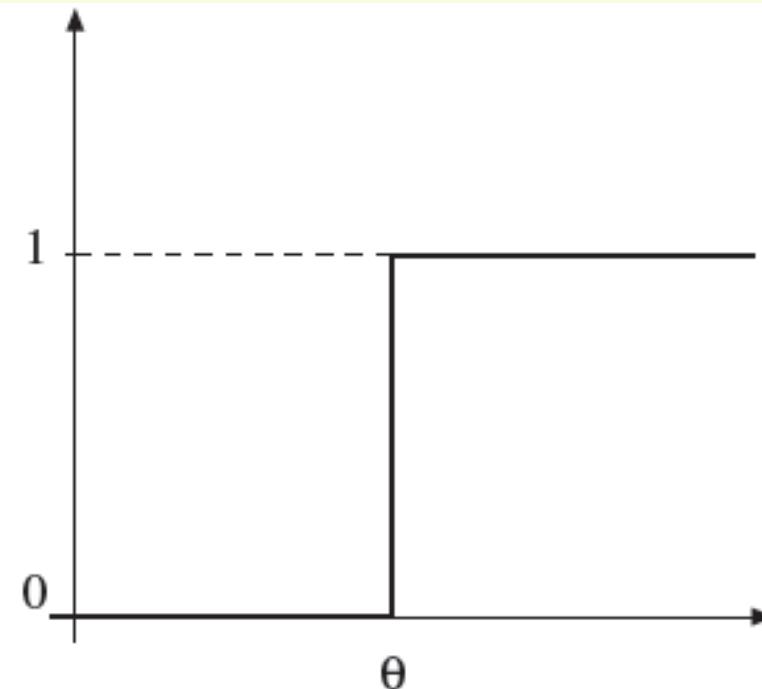


Fig. 2.7. The step function with threshold θ

AND & OR Gates

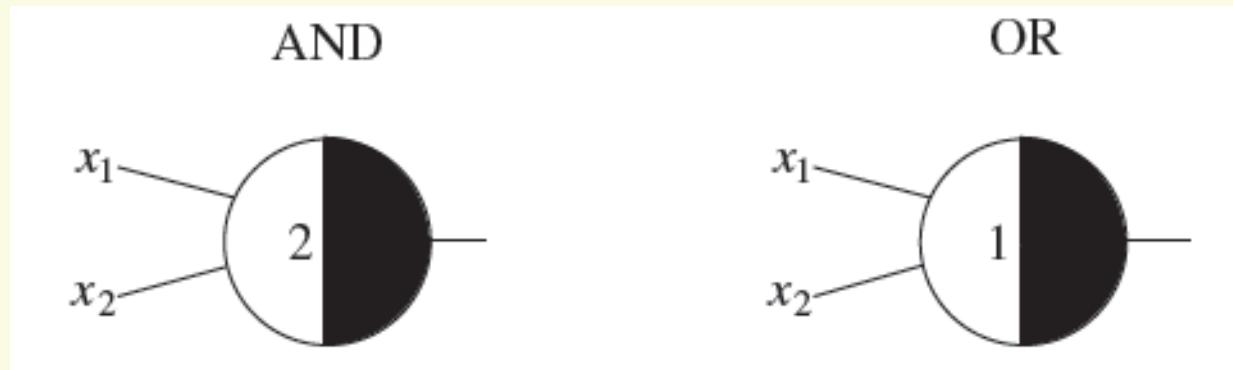
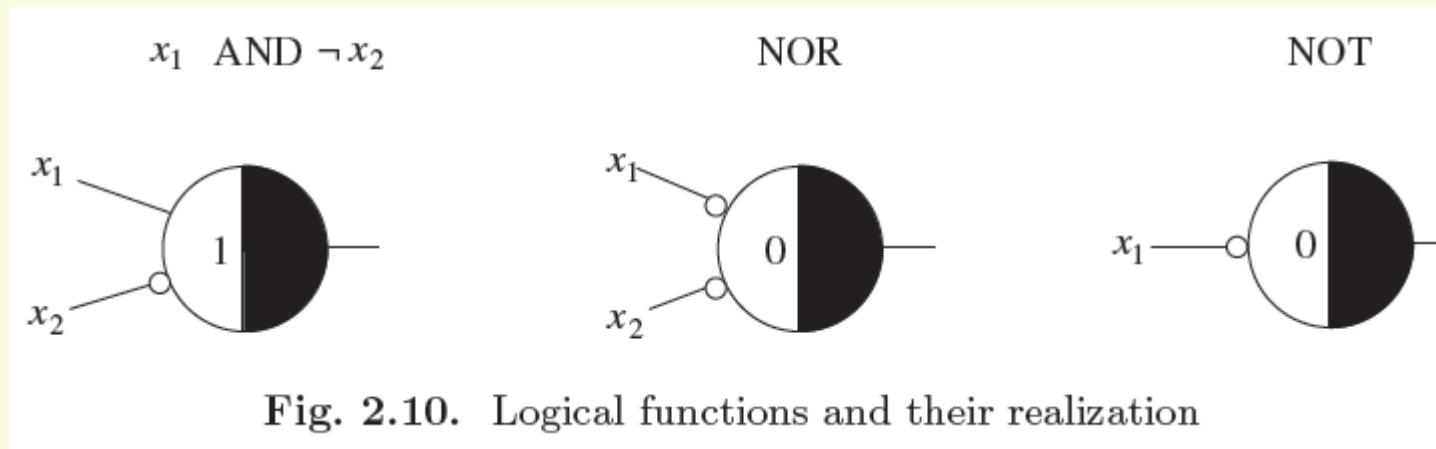


Fig. 2.8. Implementation of AND and OR gates



Fig. 2.9. Generalized AND and OR gates

Other Logic Functions



Decoders

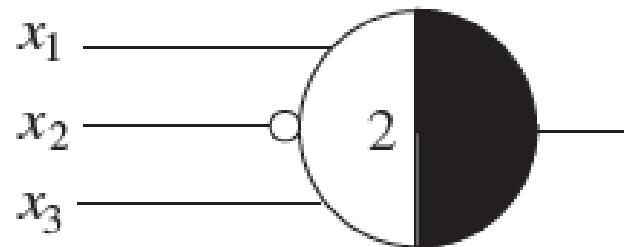


Fig. 2.14. Decoder for the vector $(1, 0, 1)$

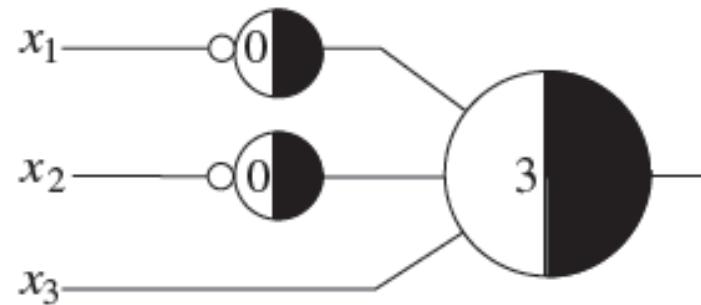
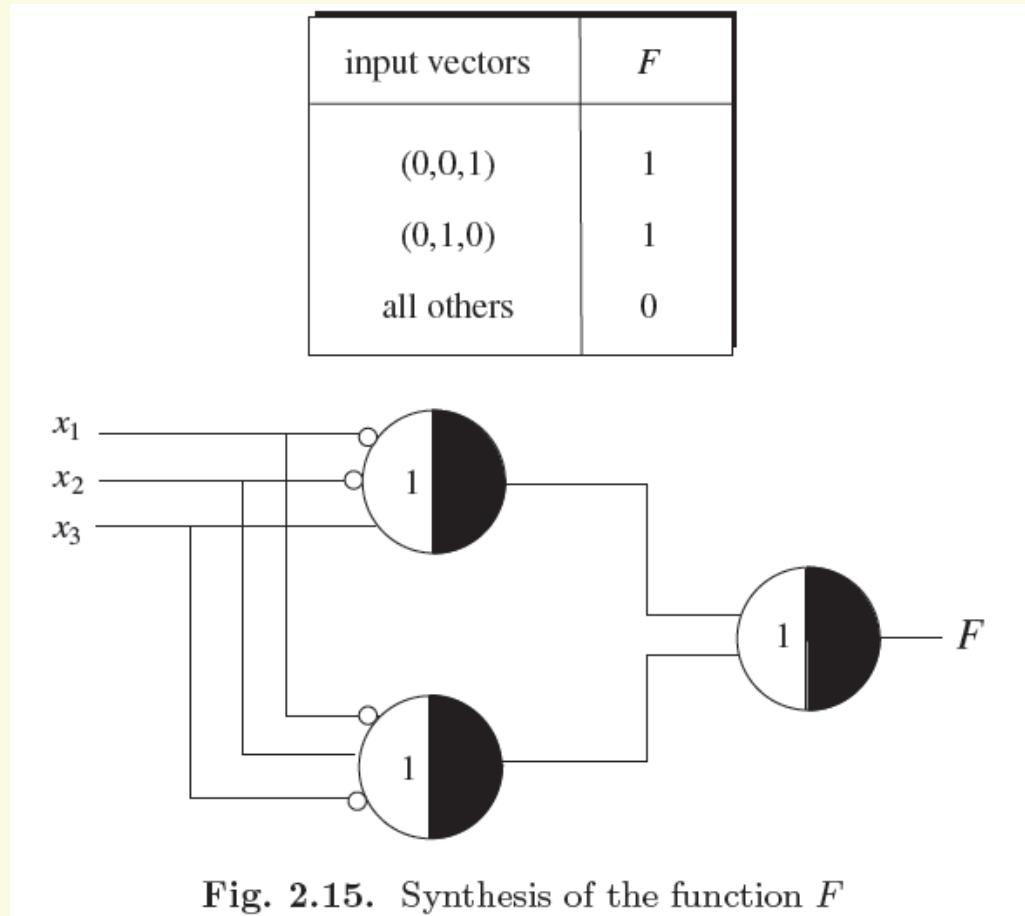
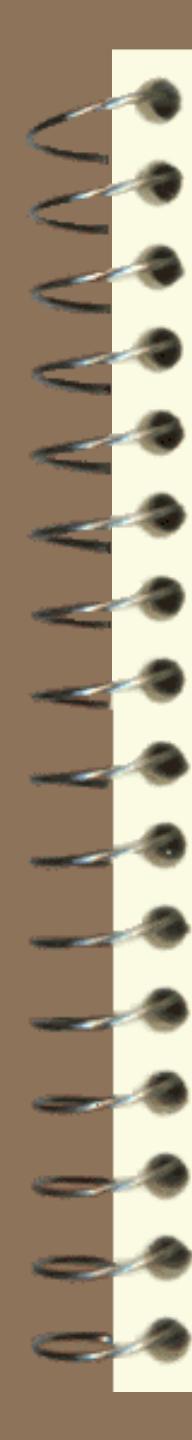


Fig. 2.16. A composite decoder for the vector $(0, 0, 1)$

Another Decoder





Threshold Logic Units

- Proposition: Any logical function $F: \{0, 1\}^n \rightarrow \{0, 1\}$ can be implemented with a two-layer McCulloch-Pitts network.
- Proposition: Uninhibited threshold logic units of McCulloch-Pitts type can only implement monotonic logical functions.

Weighted Connections

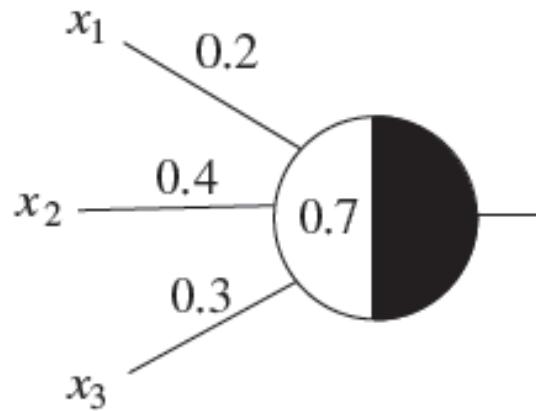


Fig. 2.17. Weighted unit

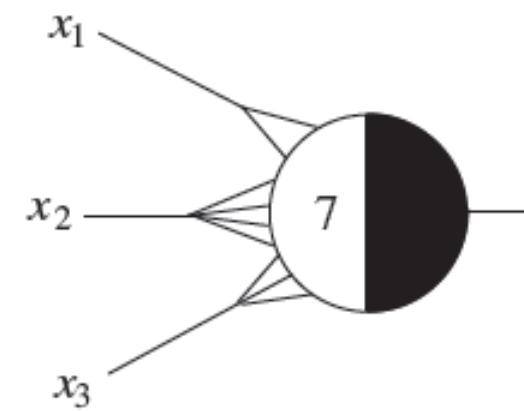


Fig. 2.18. Equivalent computing unit

A Recurrent Network

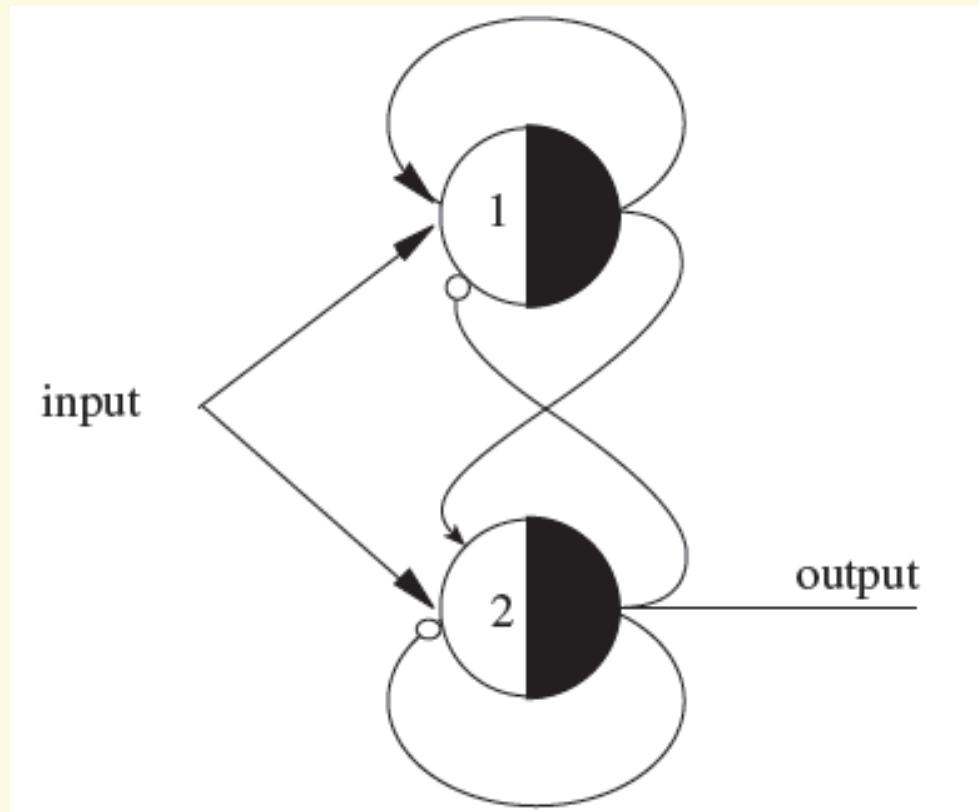
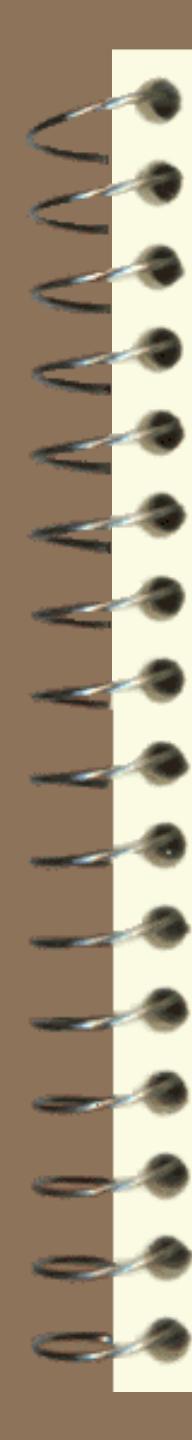


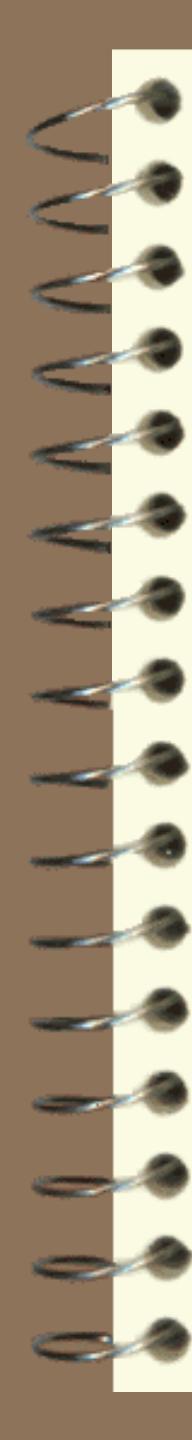
Fig. 2.21. Network for a binary scaler



Finite Automata

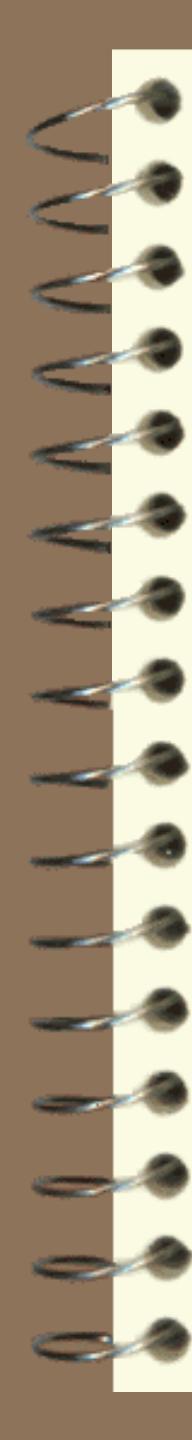
An automaton is an abstract device capable of assuming different states which change according to the received input and previous states.

A finite automaton can take only a finite set of possible states and can react to only a finite set of input signals.



Finite Automata & Recurrent Networks

Proposition: Any finite automaton can be simulated with a recurrent network of McCulloch-Pitts units.



Perceptron

- A single adaptive layer of feedforward network of pure threshold logic units.
- Developed by Rosenblatt at Connell University in late 50's.
- Trained for pattern classification.
- First working model implemented in electronic hardware.

Perceptron

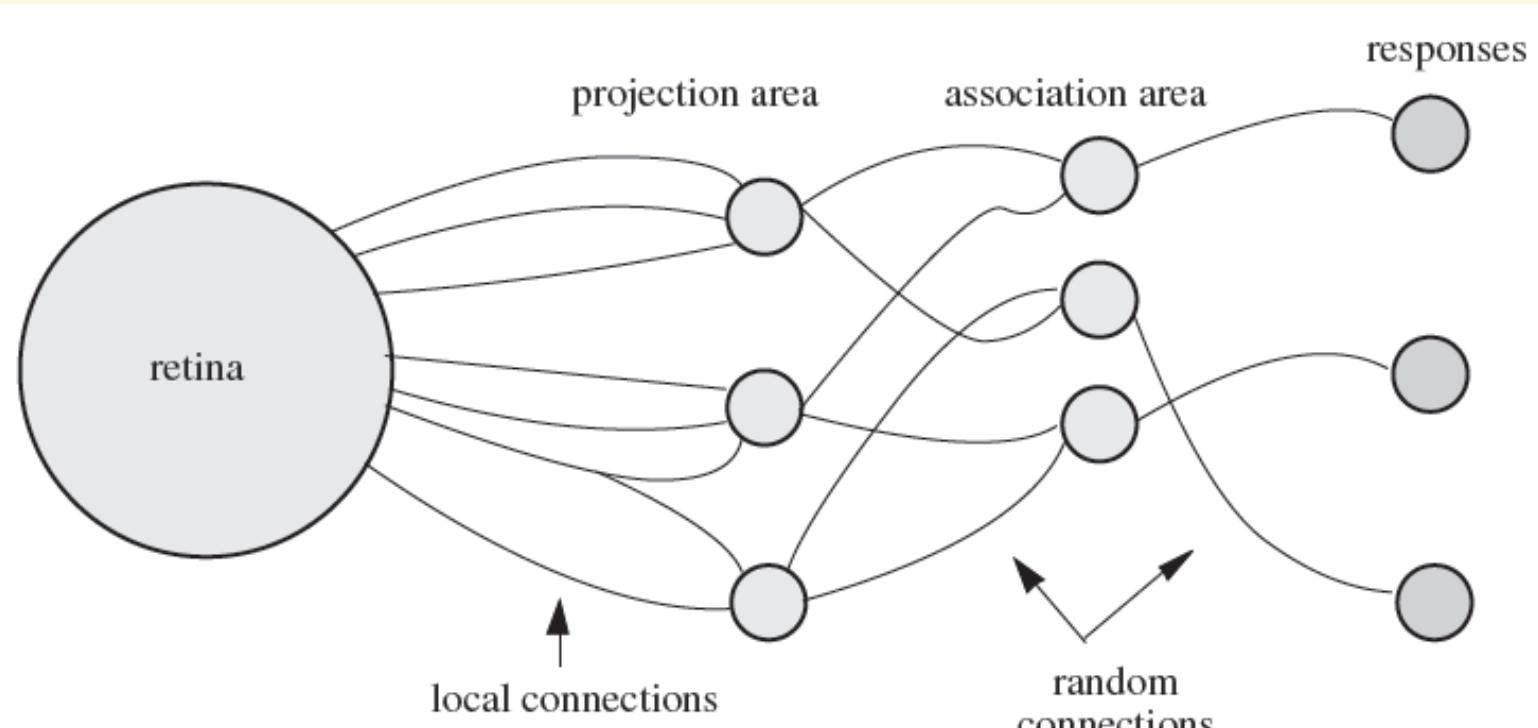


Fig. 3.1. The classical perceptron [after Rosenblatt 1958]

Perceptron

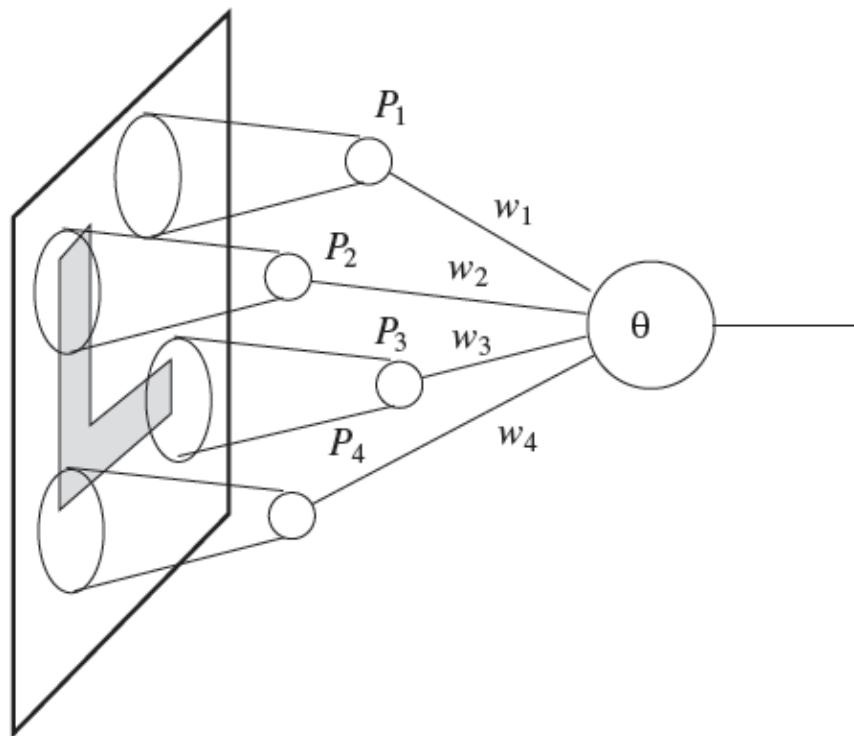
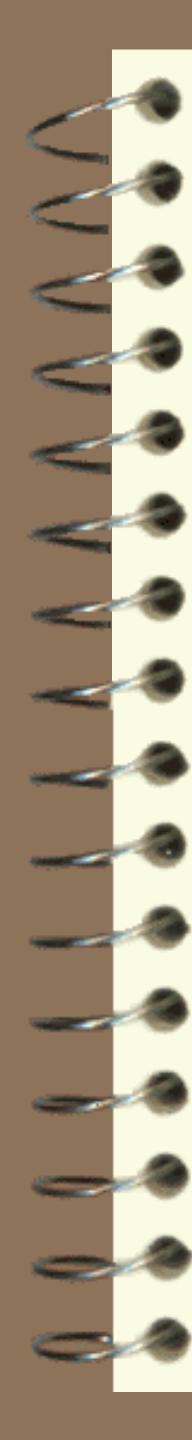
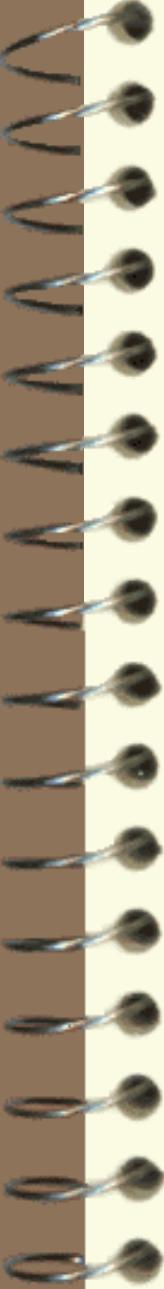


Fig. 3.2. Predicates and weights of a perceptron



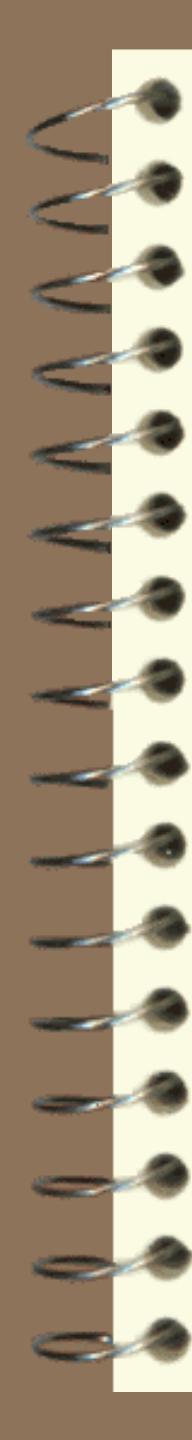
Simple Perceptron

A simple perceptron is a computing device with a threshold logic unit. When receiving n real inputs thru connections with n associated weights, a simple perceptron outputs 1 if the net input of weighted sum is not less than the threshold, and outputs 0 otherwise.



Linear Separability

Two sets of data in an n -dimensional space are said to be (absolutely) linearly separable if $n+1$ real weights (including a threshold) exist such that the weighted sum of a datum in one set is always greater than or equal to (greater than but not equal to) the threshold and that in the other set is always less than the threshold.



Absolute Linear Separability

If two finite sets of data are linearly separable, they are also absolutely linearly separable.

Perceptron Convergence Algorithm

- 1) Initialize weights and threshold randomly.
- 2) Calculate actual output of the perceptron:

For all p

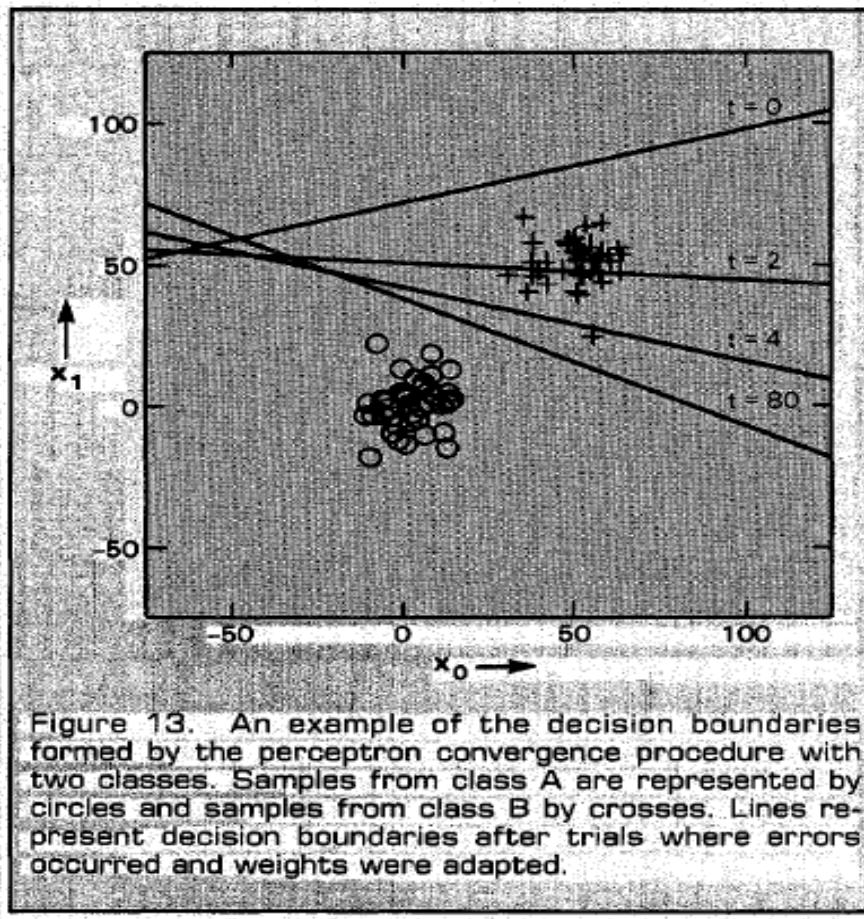
$$y^p = f\left(\sum_{i=1}^n w_i x_i^p - \theta\right)$$

- 1) Adapt weights: for all p

$$w_i(t+1) = w_i(t) + \eta(z^p - y^p)x_i^p, \eta > 0$$

- 2) Repeat until w converges.

Iterative learning



A Simple Case

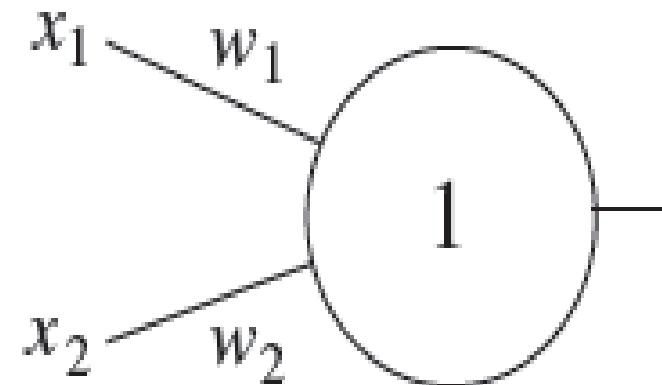


Fig. 4.4. Perceptron with constant threshold

Error Landscape

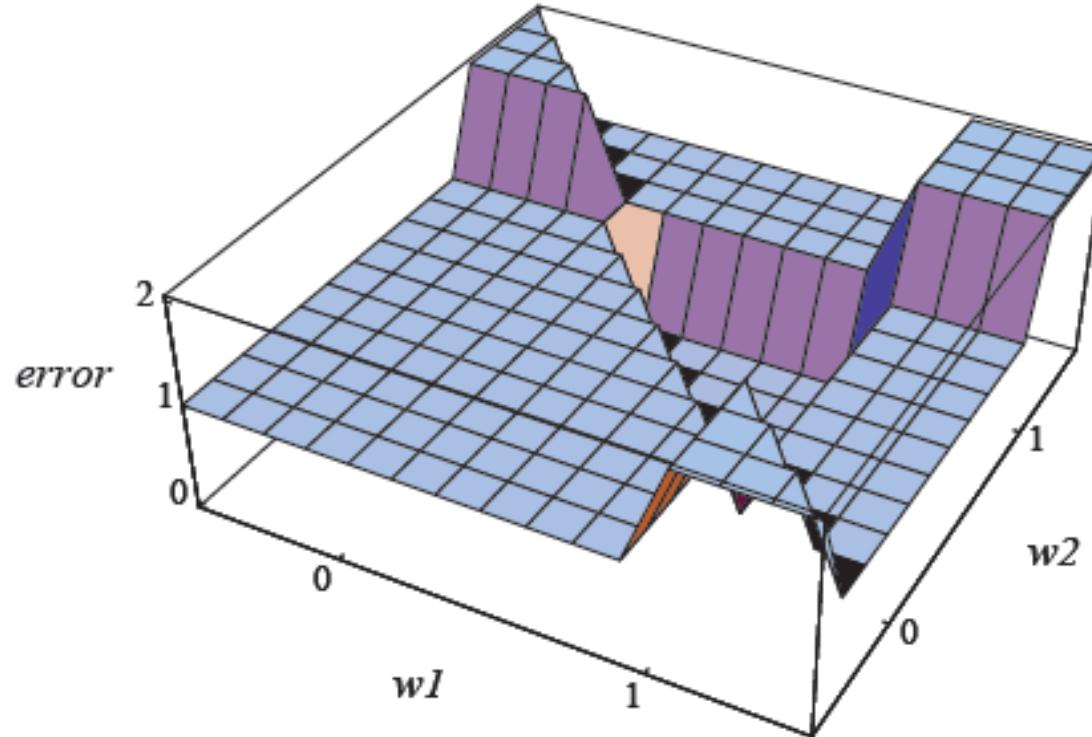


Fig. 4.5. Error function for the AND function

Learning Process

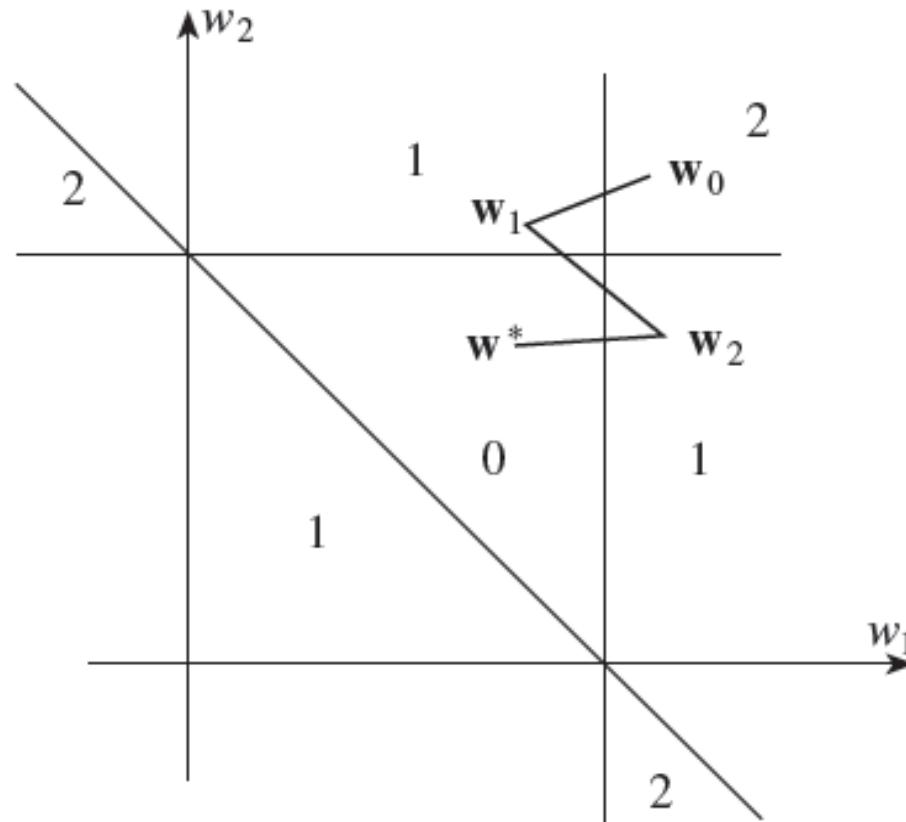


Fig. 4.6. Iteration steps to the region of minimal error

Threshold

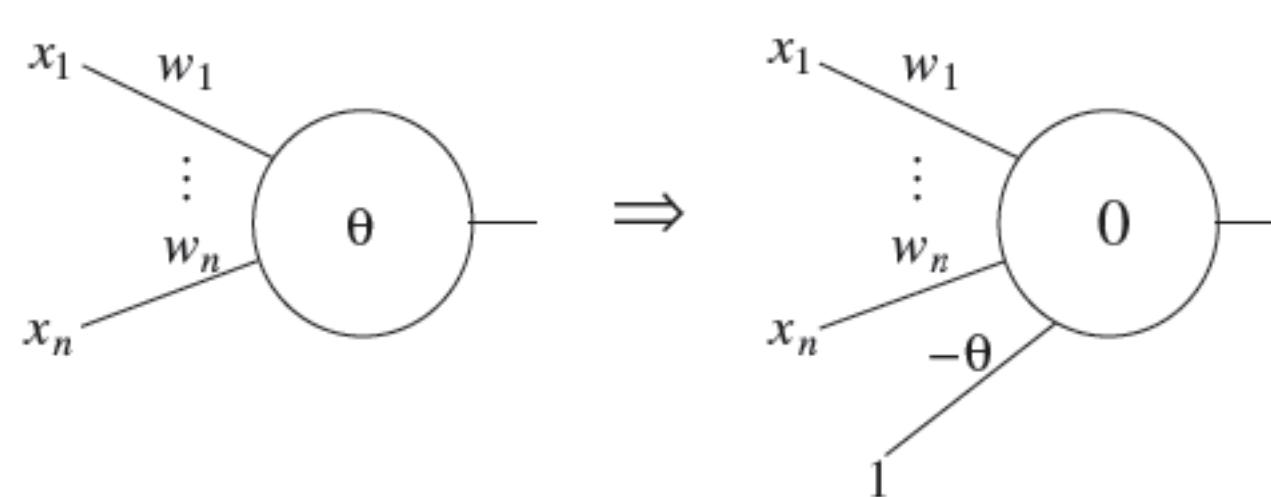


Fig. 3.5. A perceptron with a bias

Perceptron Convergence Theorem

If two sets of data are linearly separable, the perceptron learning algorithm converge to a set of weights and a threshold in a finite steps.

Two-Variable Logic Functions

Table 3.1. The 16 Boolean functions of two variables

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

2D OR and AND

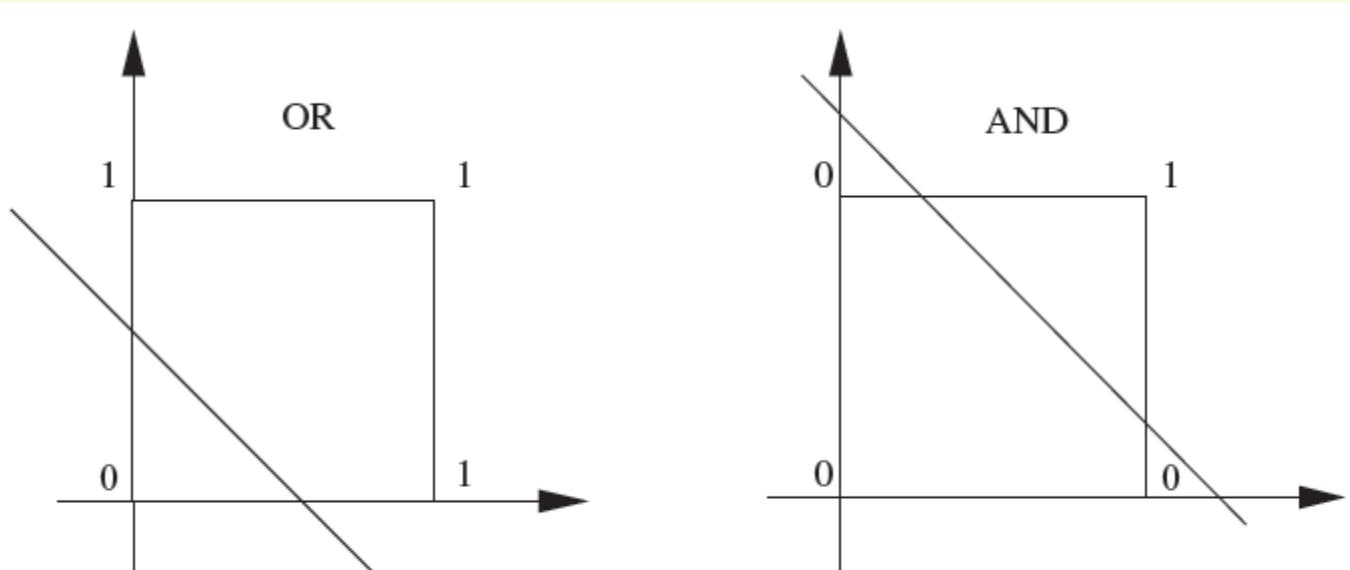


Fig. 3.6. Separations of input space corresponding to OR and AND

3D Logic Function

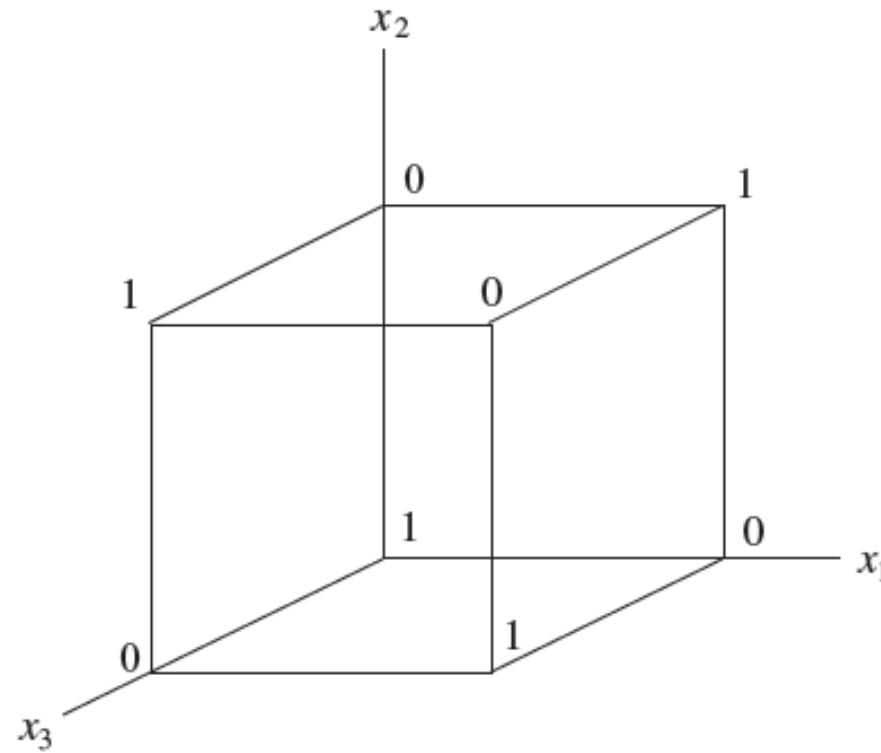


Fig. 2.11. Function values of a logical function of three variables

OR Function

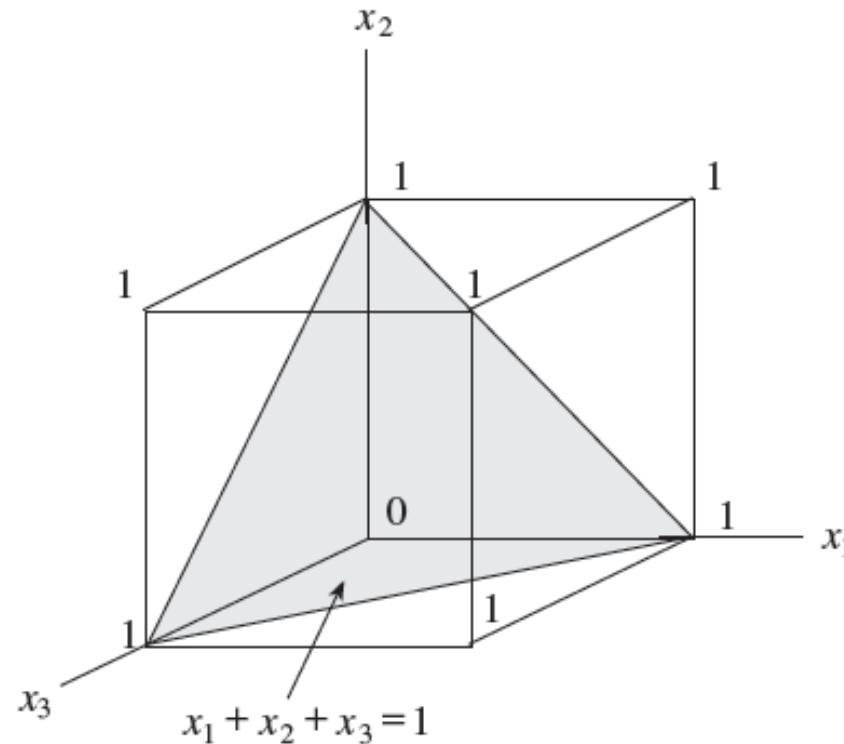


Fig. 2.12. Separation of the input space for the OR function

Majority Function

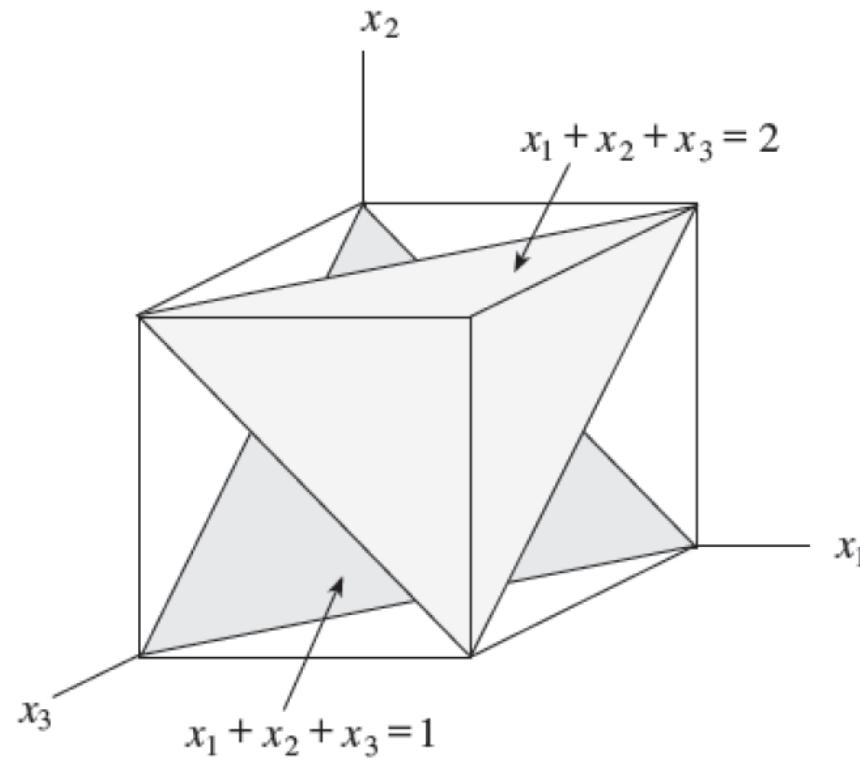


Fig. 2.13. Separating planes of the OR and majority functions

Duality

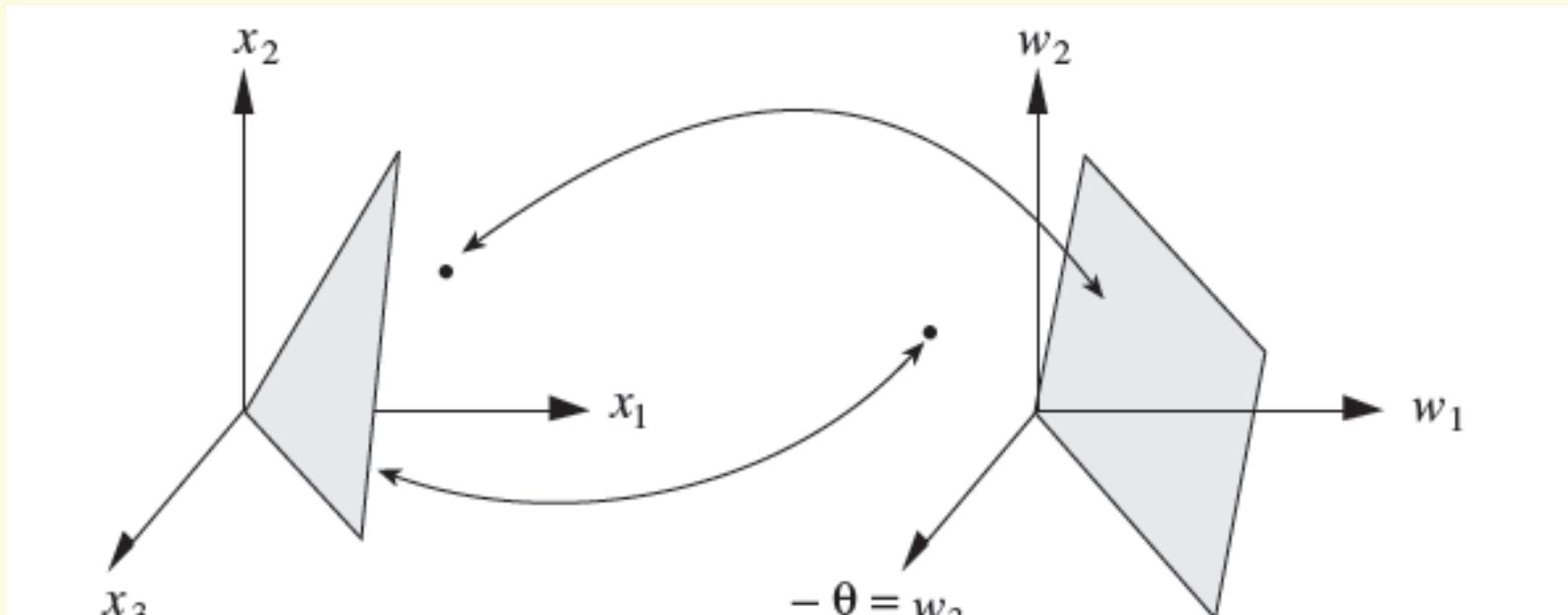
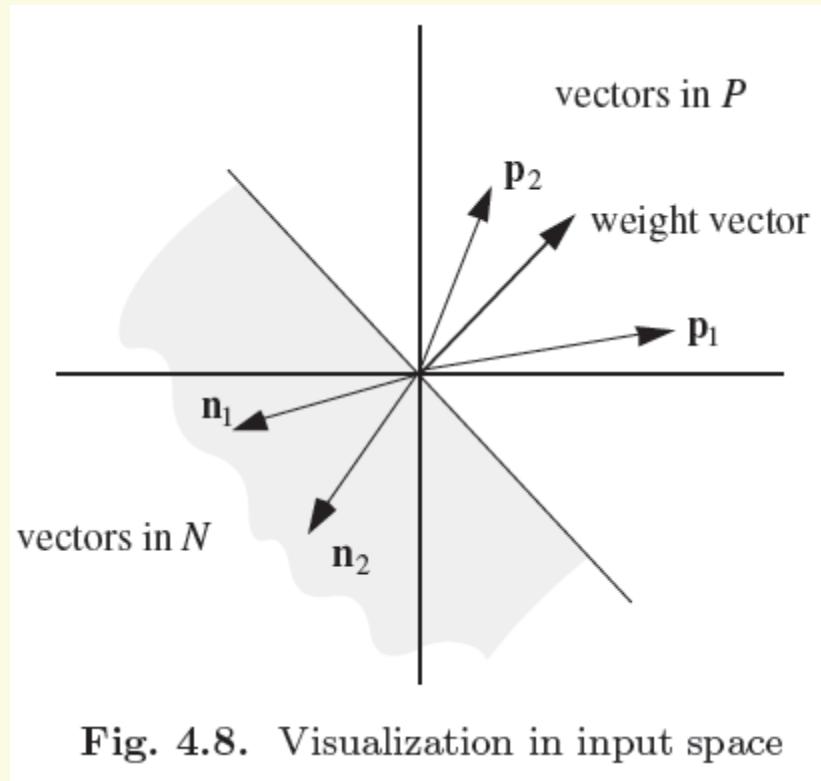


Fig. 3.7. Illustration of the duality of input and weight space

2D Weight Space



2D Weight Space

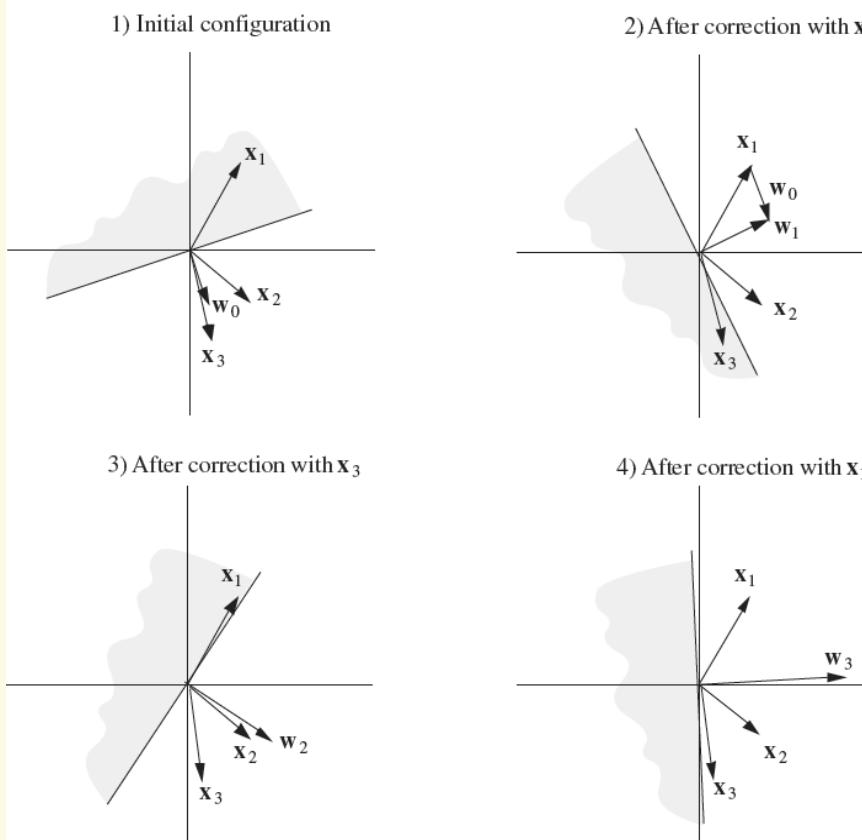


Fig. 4.10. Convergence behavior of the learning algorithm

XOR Problem

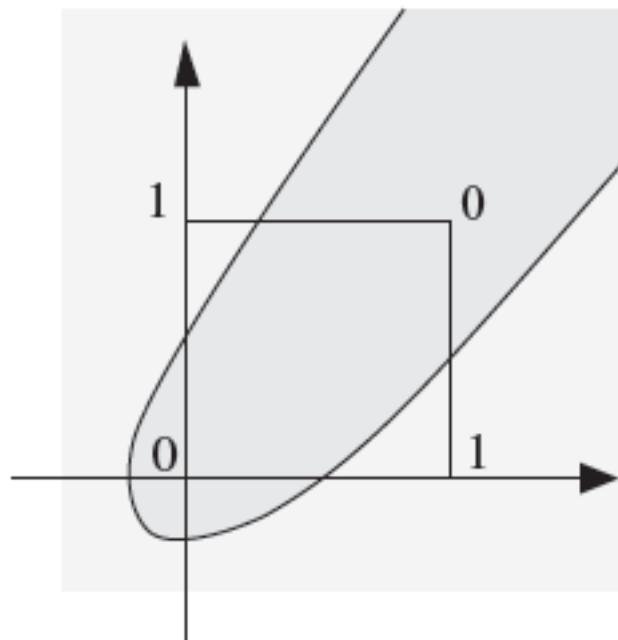
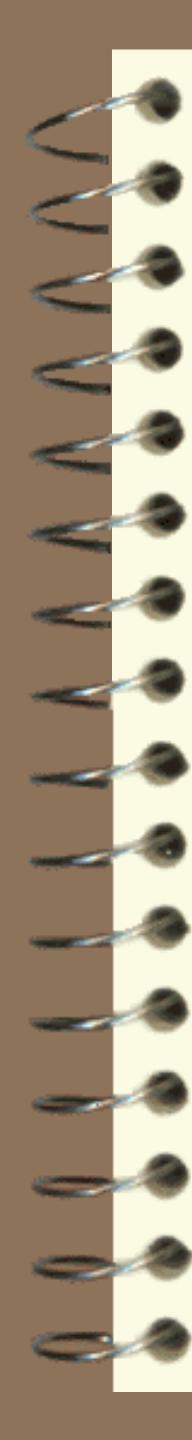


Fig. 3.8. Non-linear separation of input space



Limitations of Perceptrons

- Only linearly separable data can be classified
- The convergence rate may be low for high-dimensional or large number of data.

Bipolar vs. Unipolar State Variables

- Unipolar: $v \in \{0,1\}$
- Bipolar: $v \in \{-1,1\}$

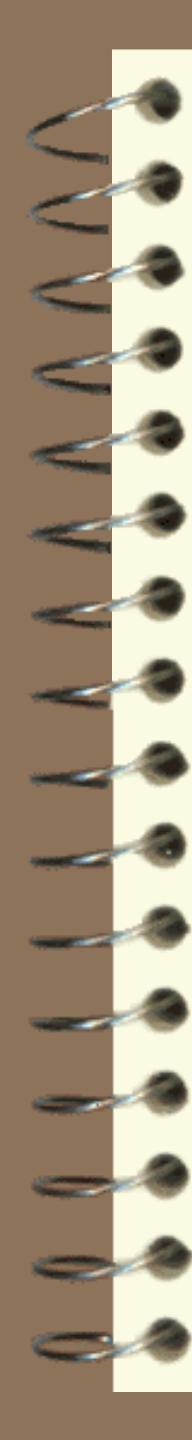
Bipolar coding of state variables is better than unipolar (binary) one in terms of algebraic structure, region proportion in weight space, etc.

Monte Carlo Tests

Table 6.1. Relative sizes of the regions on the Boolean sphere as percentage of the total surface

Coding	Boolean function number							
	0	1	2	3	4	5	6	7
binary	26.83	2.13	4.18	4.13	4.17	4.22	0.00	4.13
bipolar	8.33	6.29	6.26	8.32	6.24	8.36	0.00	6.22

Coding	Boolean function number							
	8	9	10	11	12	13	14	15
binary	4.28	0.00	4.26	4.17	4.17	4.14	2.07	27.12
bipolar	6.16	0.00	8.42	6.33	8.27	6.31	6.25	8.23



ADALINE

- A single adaptive layer of feedforward network of linear elements.
- Full name: Adaptive linear elements.
- Developed by Widrow and Hoff at Stanford University in early 60's.
- Trained using a learning algorithm called Delta Rule or Least Mean Squares (LMS) Algorithm.

LMS Learning Algorithm

- 1) Initialize weights and threshold randomly.
- 2) Calculate actual output of the ADALINE:

$$y = \alpha \left(\sum_{i=1}^n w_i x_i - \theta \right), \alpha > 0$$

- 3) Adapt weights:

$$w_i(t+1) = w_i(t) + \eta(z^p - y^p)x_i^p, \eta > 0$$

- 4) Repeat until w converges

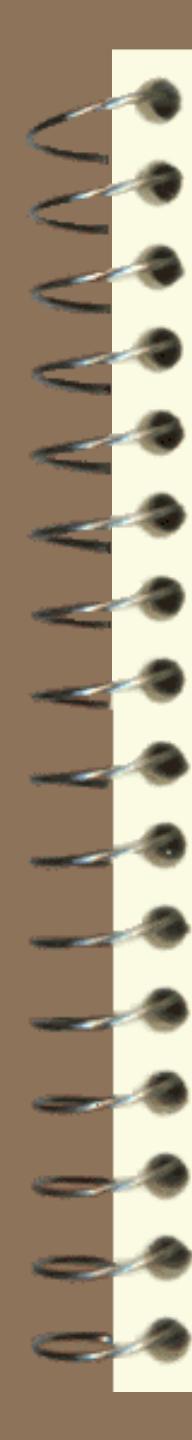
Gradient Descent Learning Algorithms

$$E = \sum_p E_p = \sum_p (z^p - y^p)^2$$

$$w(t+1) = w(t) - \eta \nabla E_w$$

$$\Delta w(t) = -\eta \nabla E_w$$

$$\nabla E_w = (\partial E / \partial w_1, \partial E / \partial w_2, \dots, \partial E / \partial w_M)^T.$$

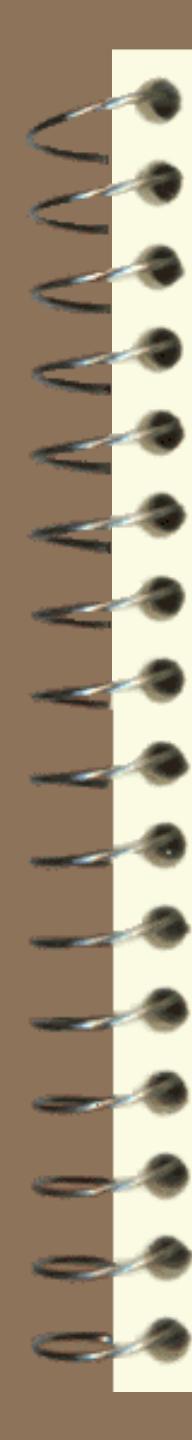


Training Modes

- Sequential mode: input training sample pairs one by one orderly or randomly.
- Batch mode: input training sample pairs in the whole training set at each iteration.
- Perceptron learning: either sequential or batch mode.
- ADALINE training: batch mode only.

Perceptron vs. Adaline

- Architecture: Perceptron uses bipolar or unipolar hardlimiter activation function, Adaline uses linear activation function.
- Learning rule: Perceptron learning algorithm is not gradient-descent and can operate in either sequential or batch training mode, whereas Adaline learning (LMS) algorithm is gradient descent, but can only operate in batch mode.



Weight Space Regions Separated by Hyperplanes

- Each plane is defined by one training sample.
- One plane separates two (2) half-space.
- Two planes separate up to four (4) regions.
- Three planes separate up to eight (8) regions.
- However, four planes separate up to fourteen (14) regions only.

Number of Weight Space Regions

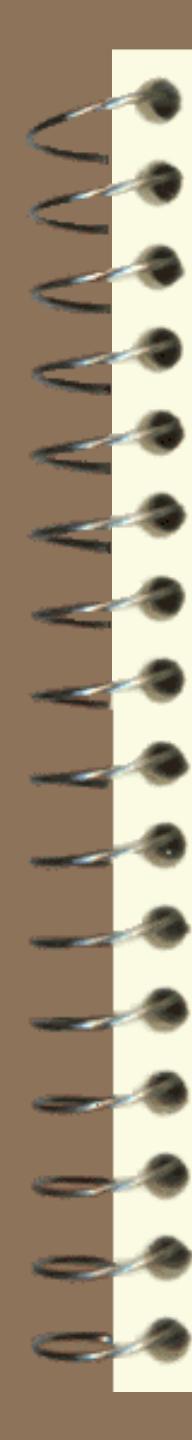
The number of different regions in weight space defined by m separating hyperplanes in n -dimensional weight space is a polynomial of degree $n-1$ on m :

$$R(m, n) = 2 \sum_{i=0}^{n-1} \binom{m-1}{i} < 2 \frac{m^n}{n!}$$

Number of Weight Space Regions

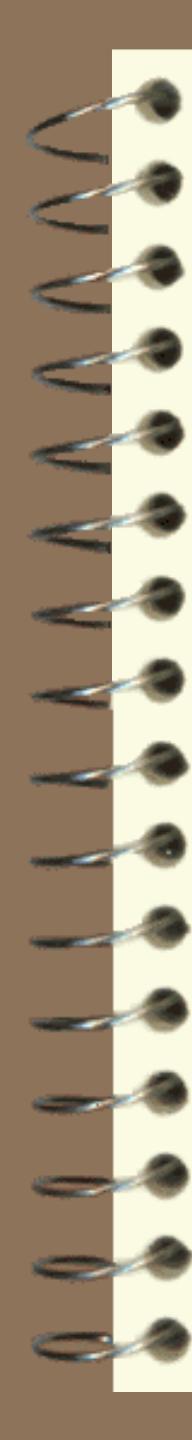
Table 6.2. Comparison of the number of Boolean and threshold functions of n inputs and two different bounds

n	2^{2^n}	$T(2^n, n)$	$R(2^n, n)$	$\lfloor 2^{n^2+1}/n! \rfloor$
1	4	4	4	4
2	16	14	14	16
3	256	104	128	170
4	65,536	1,882	3,882	5,461
5	4.3×10^9	94,572	412,736	559,240



Number of Logic Functions vs. Number of Threshold Functions

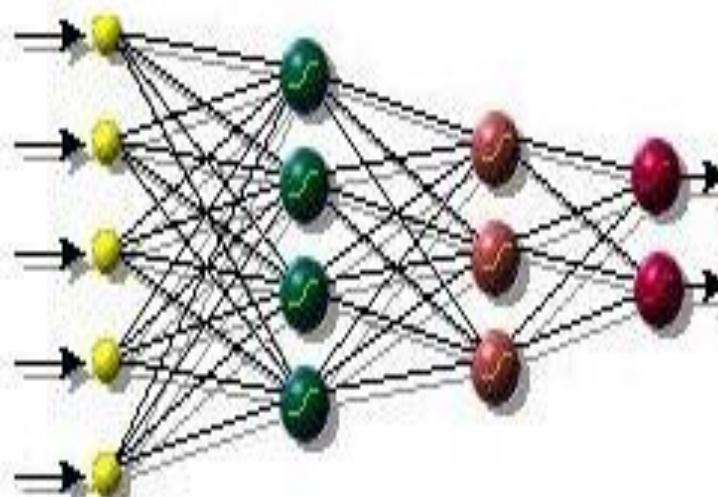
- The number of threshold functions defined by hyperplanes is a function of $2^{n(n-1)}$ whereas that of logical functions is 2^{2^n} .
- The learnability problem: when n is large, there is not enough classification regions in weight space to represent all logical functions.



Learnability Problems

- Solution existence in the weight space? Neither Perceptron nor Adaline can classify patterns with nonlinear distributions such as XOR. But two-layer Perceptron can classify XOR data.
- How to find the solution even though it exists in the weight space? It is known that multilayer Perceptron can classify arbitrary shape of data classes. But how to design learning algorithms to determine the weights?

Multilayer Feedforward Network



Multilayer Feedforward Network

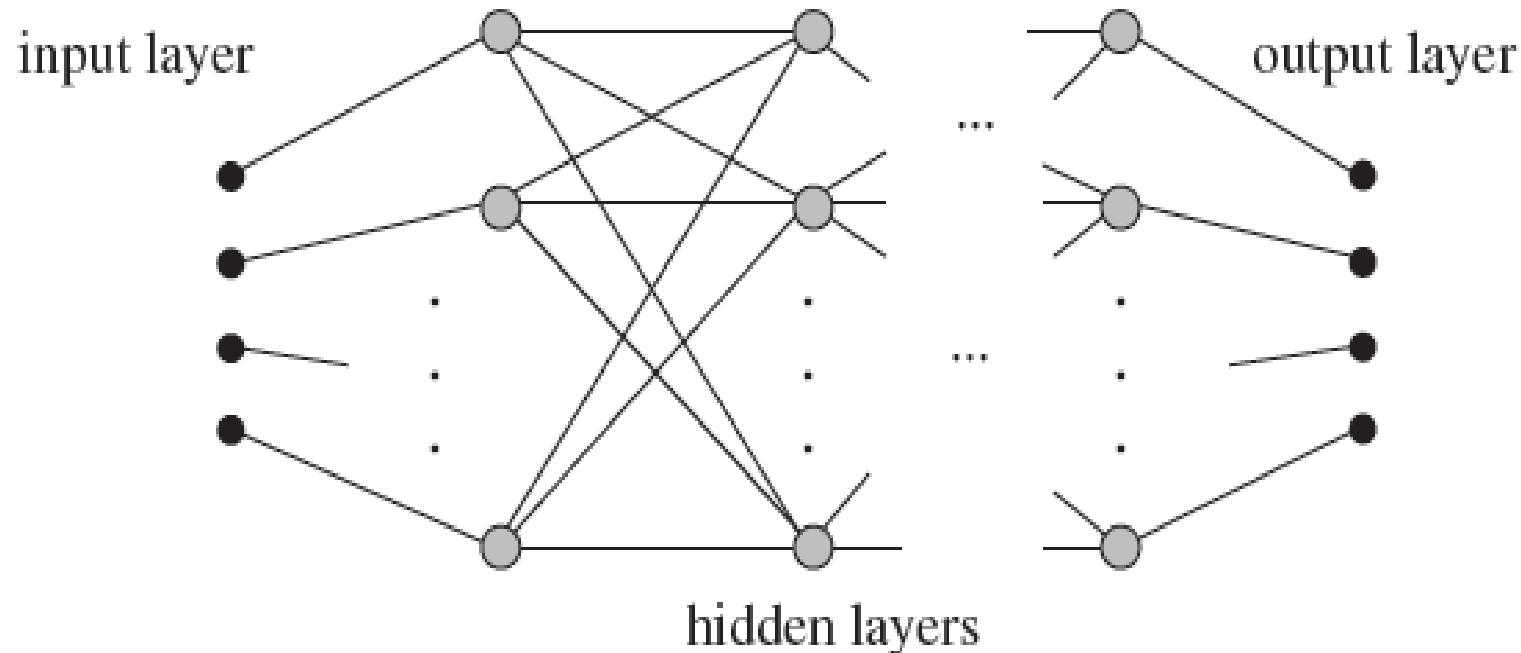


Fig. 6.1. A generic layered architecture

XOR Problem Solved

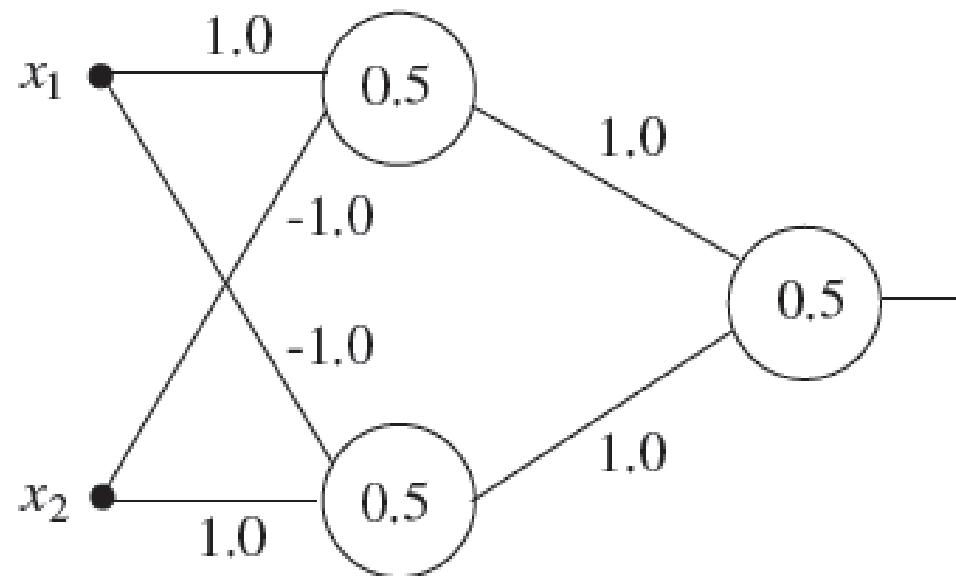


Fig. 6.2. A three-layered network for the computation of XOR

Alternative Networks

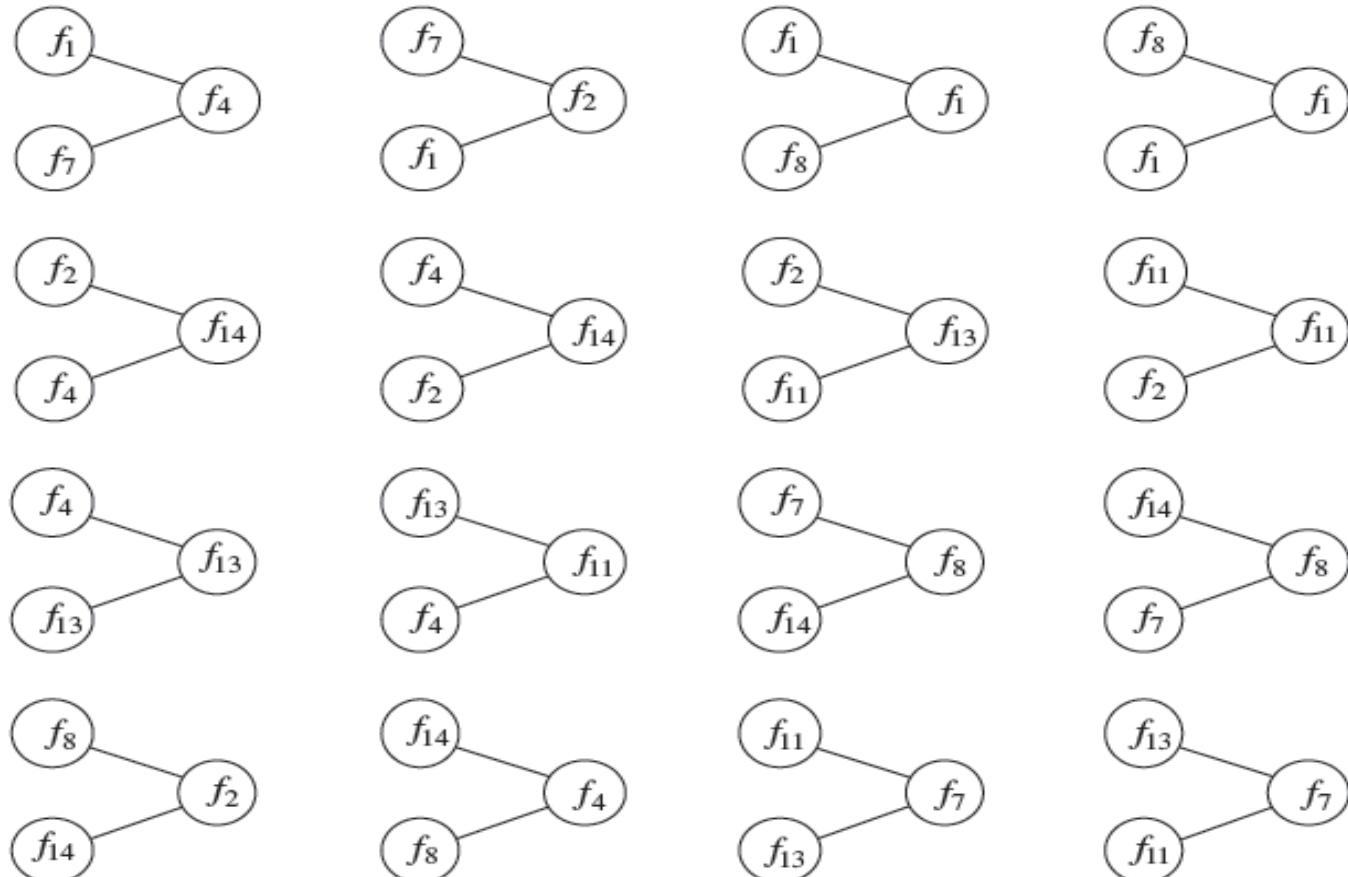
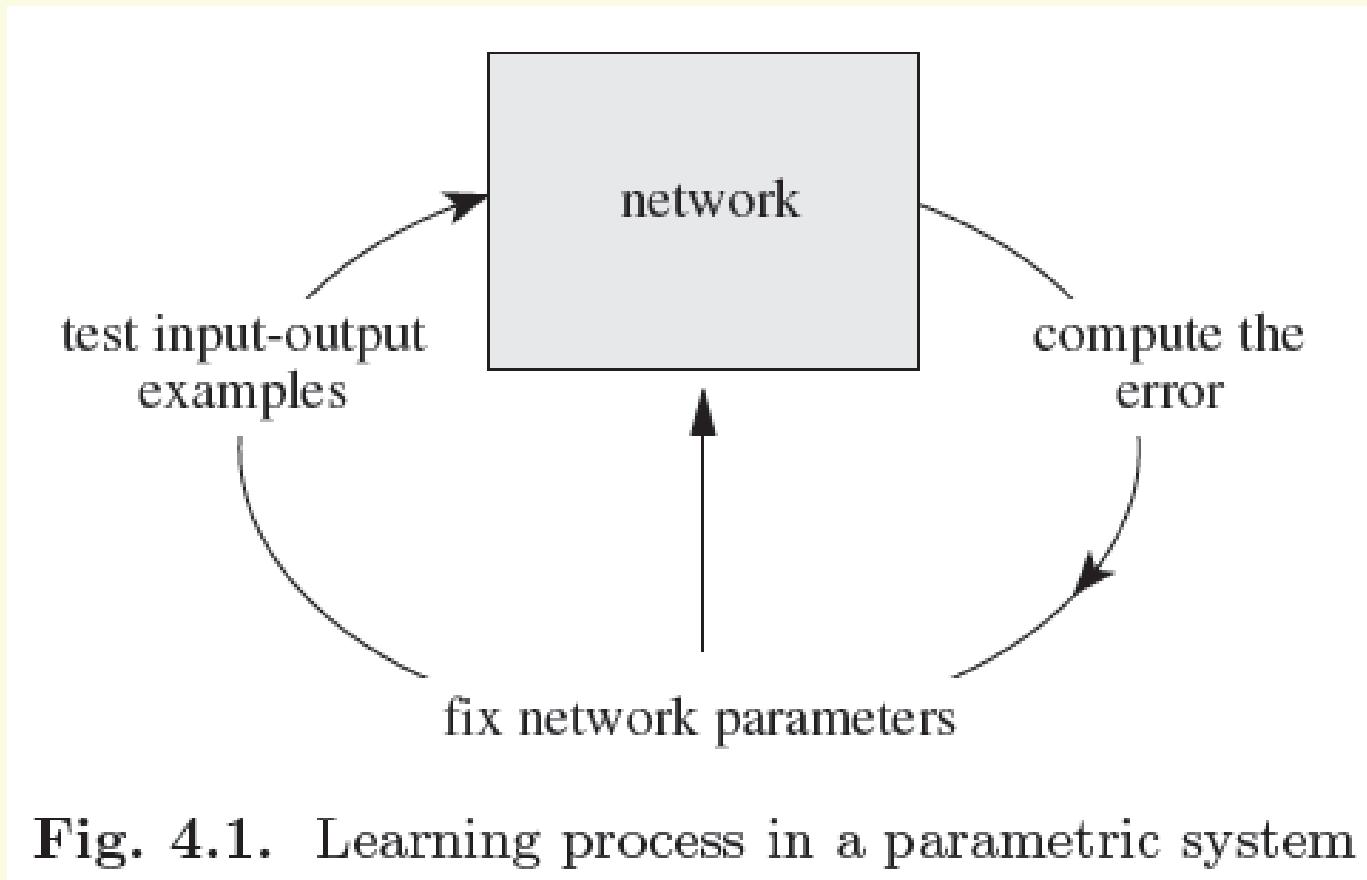


Fig. 6.3. The 16 solutions for the computation of XOR with three computing units
CS5486

Learning Process



Alternative Network

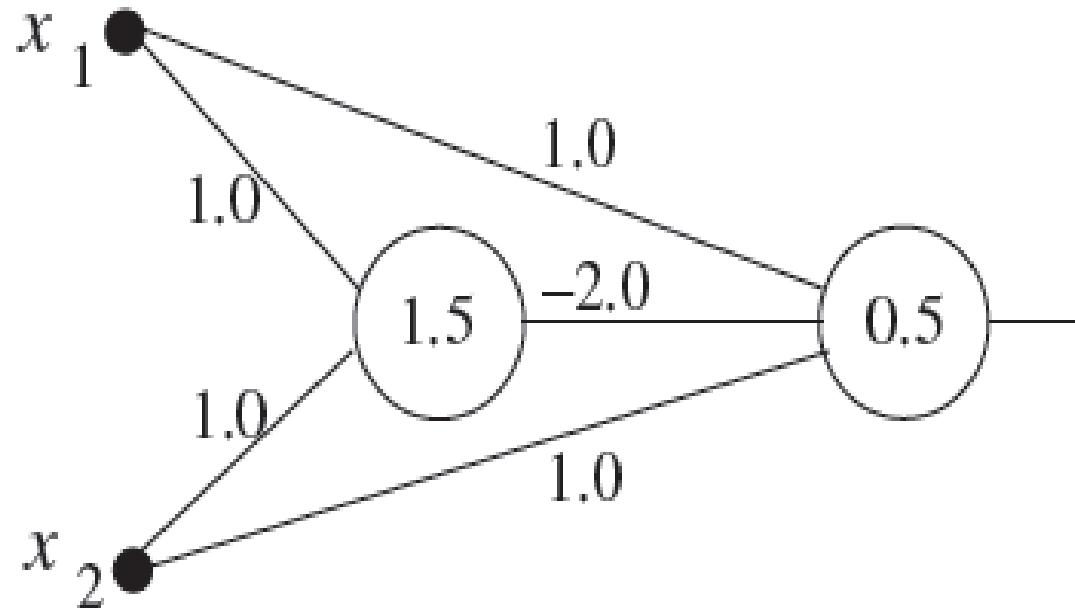


Fig. 6.4. Two unit network for the computation of XOR

Two-layer Separation

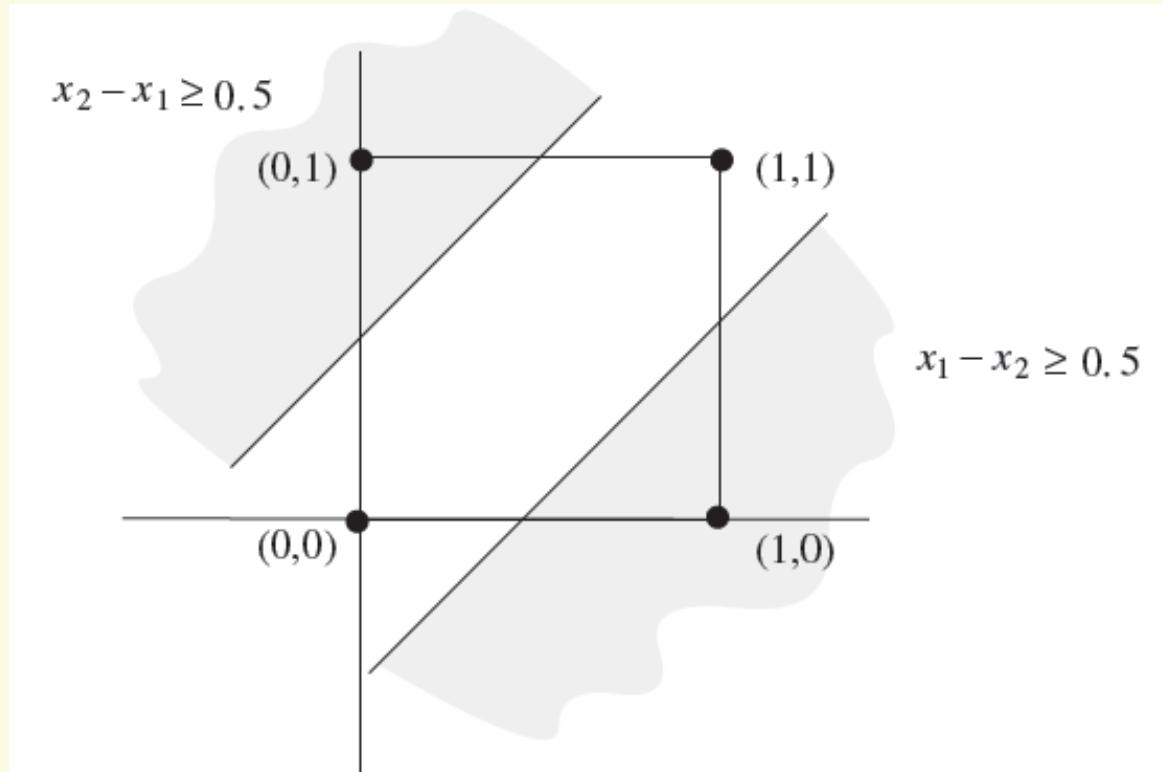


Fig. 6.5. Space separation defined by a two-layered network

Two-layer Separation

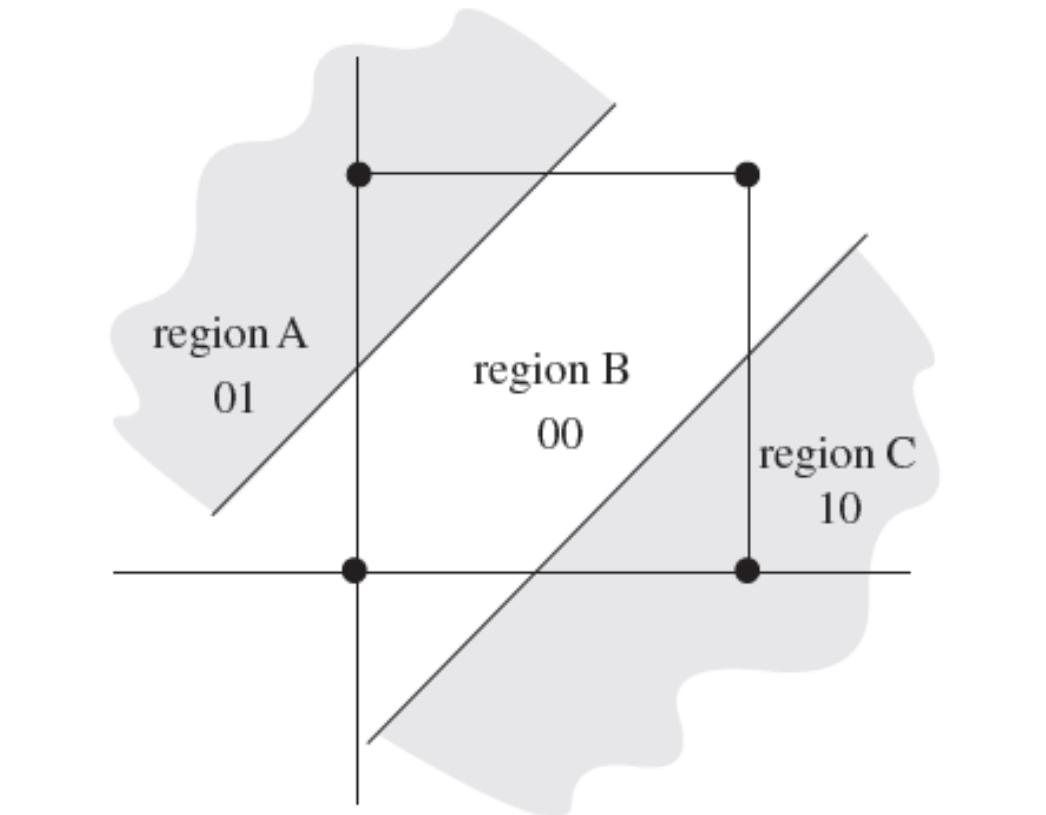
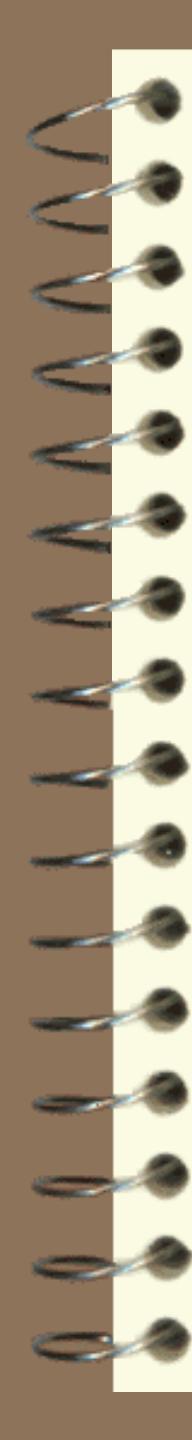


Fig. 6.6. Labeling of the regions in input space



Backpropagation Algorithm

- Also known as generalized delta rule.
- Invented and reinvented by many researchers, popularized by the PDP group at UC San Diego in 1986.
- A recursive gradient-descent learning algorithm for multilayer feedforward networks of sigmoid activation function.
- Compute errors backward from the output layer to input layer.
- Minimize the mean squares error function.

Sigmoid Activation Functions

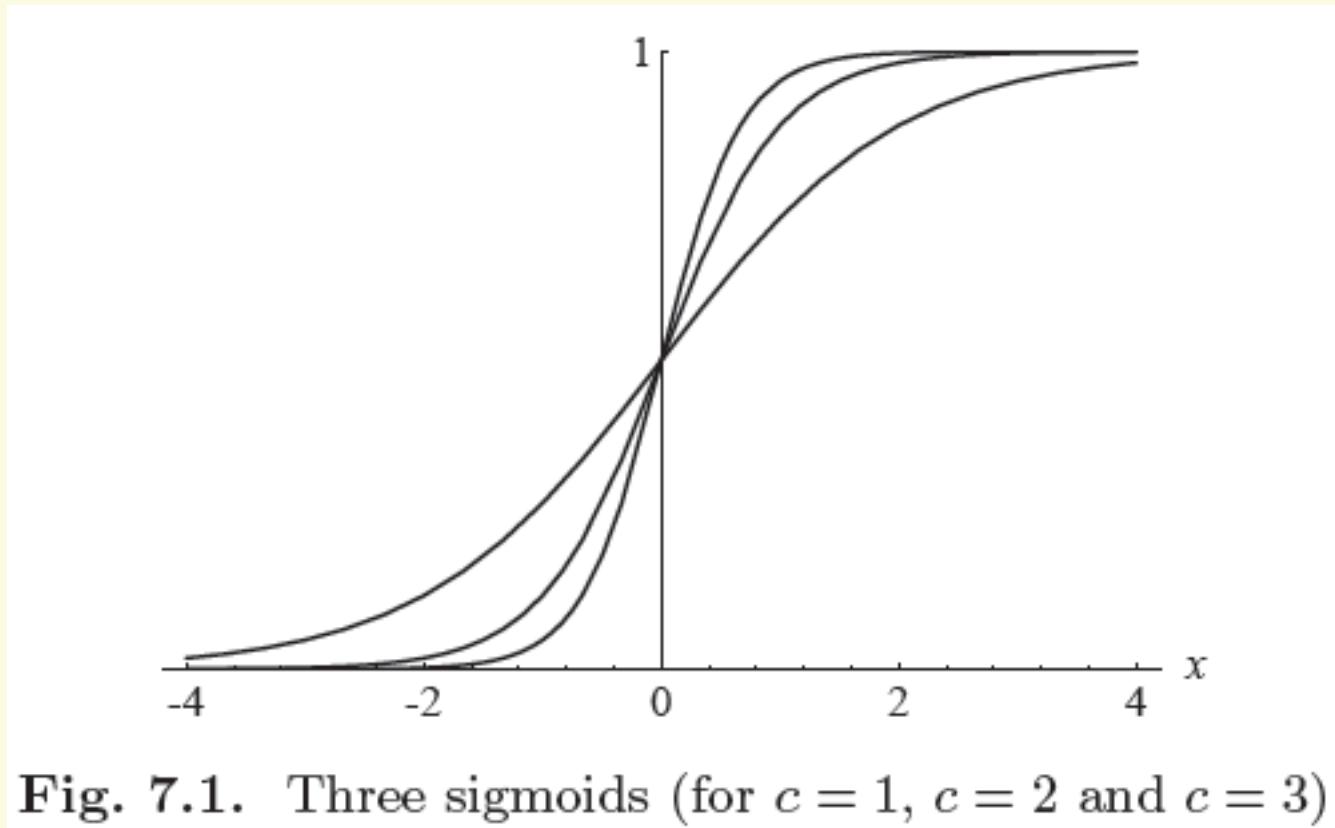


Fig. 7.1. Three sigmoids (for $c = 1$, $c = 2$ and $c = 3$)

Sigmoid Activation Functions

- Unipolar: $f(u) = \frac{1}{1 + \exp(-u)}$
 $\frac{df(u)}{du} = \frac{\exp(-u)}{(1 + \exp(-u))^2} =$
 $\frac{1}{1 + \exp(-u)} \frac{1 + \exp(-u) - 1}{1 + \exp(-u)} = f(u)[1 - f(u)]$
- Bipolar: $f(u) = \tanh(u) = \frac{1 - \exp(-u)}{1 + \exp(-u)}$

Related Activation Functions

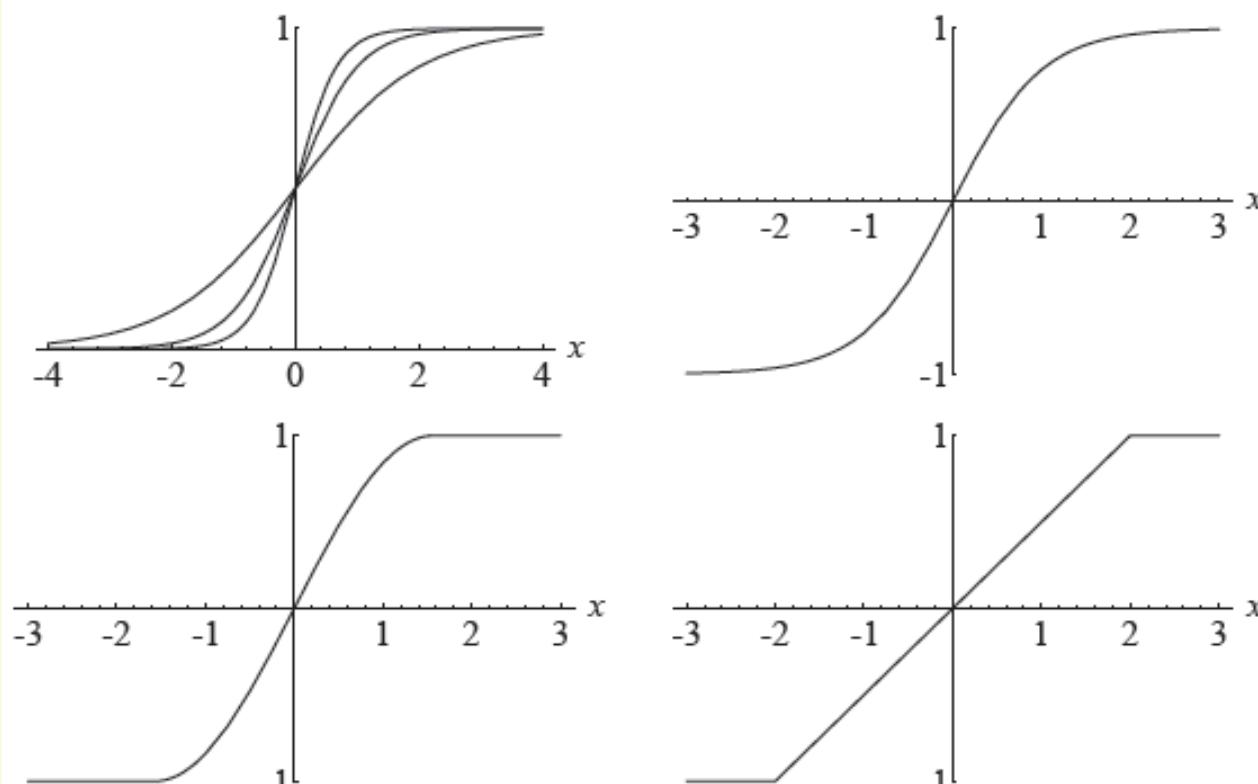


Fig. 7.2. Graphics of some “squashing” functions

Backpropagation Algorithm (cont'd)

Error function:

$$E = \sum_p E_p = \frac{1}{2} \sum_p \sum_{i=1}^m (z_i^p - y_i^p)^2$$

General formula:

$$\Delta w_{ij}(t) = w_{ij}(t+1) - w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \sum_p \frac{\partial E_p}{\partial w_{ij}}$$

Backpropagation Algorithm (cont'd)

Output layer l :

$$\frac{\partial E_p}{\partial w_{ij}^l} = \frac{\partial E_p}{\partial y_i^p} \frac{\partial y_i^p}{\partial u_i^l} \frac{\partial u_i^l}{\partial w_{ij}^l} =$$

$$-(z_i^p - y_i^p) y_i^p (1 - y_i^p) v_j^{l-1} = -\delta_i^l v_j^{l-1}$$

where $\delta_i^l = -\frac{\partial E_p}{\partial u_i^l} = (z_i^p - y_i^p) y_i^p (1 - y_i^p)$.

Backpropagation Algorithm (cont'd)

Hidden layer $l-1$:

$$-\frac{\partial E_p}{\partial v_i^{l-1}} = -\sum_j \frac{\partial E_p}{\partial u_j^l} \frac{\partial u_j^l}{\partial v_i^{l-1}} = \sum_j \left(-\frac{\partial E_p}{\partial u_j^l} \right) \frac{\partial u_j^l}{\partial v_i^{l-1}} = \sum_j \delta_j^l w_{ji}^l$$

$$\delta_i^{l-1} = -\frac{\partial E_p}{\partial u_i^{l-1}} = -\frac{\partial E_p}{\partial v_i^{l-1}} \frac{\partial v_i^{l-1}}{\partial u_i^{l-1}} = \left(\sum_j \delta_j^l w_{ji}^l \right) v_i^{l-1} (1 - v_i^{l-1})$$

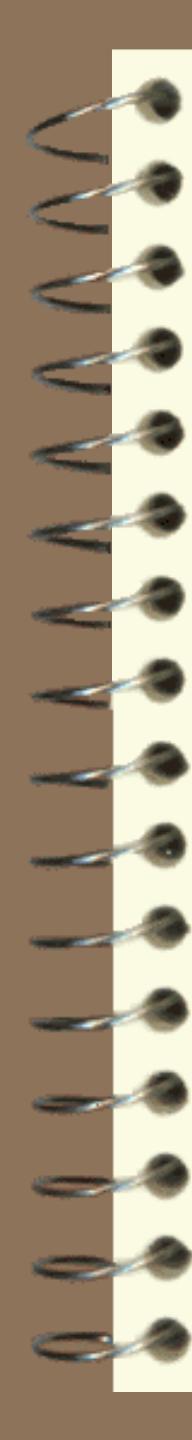
$$\frac{\partial E_p}{\partial w_{ij}^{l-1}} = \frac{\partial E_p}{\partial u_i^{l-1}} \frac{\partial u_i^{l-1}}{\partial w_{ij}^{l-1}} = -\delta_i^{l-1} v_j^{l-2}$$

Backpropagation Algorithm (cont'd)

Input layer 1:

$$\delta_i^1 = \left(\sum_j \delta_j^2 w_{ji}^2 \right) v_i^1 (1 - v_i^1)$$

$$\frac{\partial E_p}{\partial w_{ij}^1} = \frac{\partial E_p}{\partial u_i^1} \frac{\partial u_i^1}{\partial w_{ij}^1} = -\delta_i^1 x_j^p$$



Backpropagation Algorithm (cont'd)

- 1) Initialize weights and threshold randomly.
- 2) Calculate actual output of the MLP:
- 3) Adapt weights for all layers:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \sum_p \frac{\partial E_p}{\partial w_{ij}}$$

- 4) Repeat until w converges

Local Minima

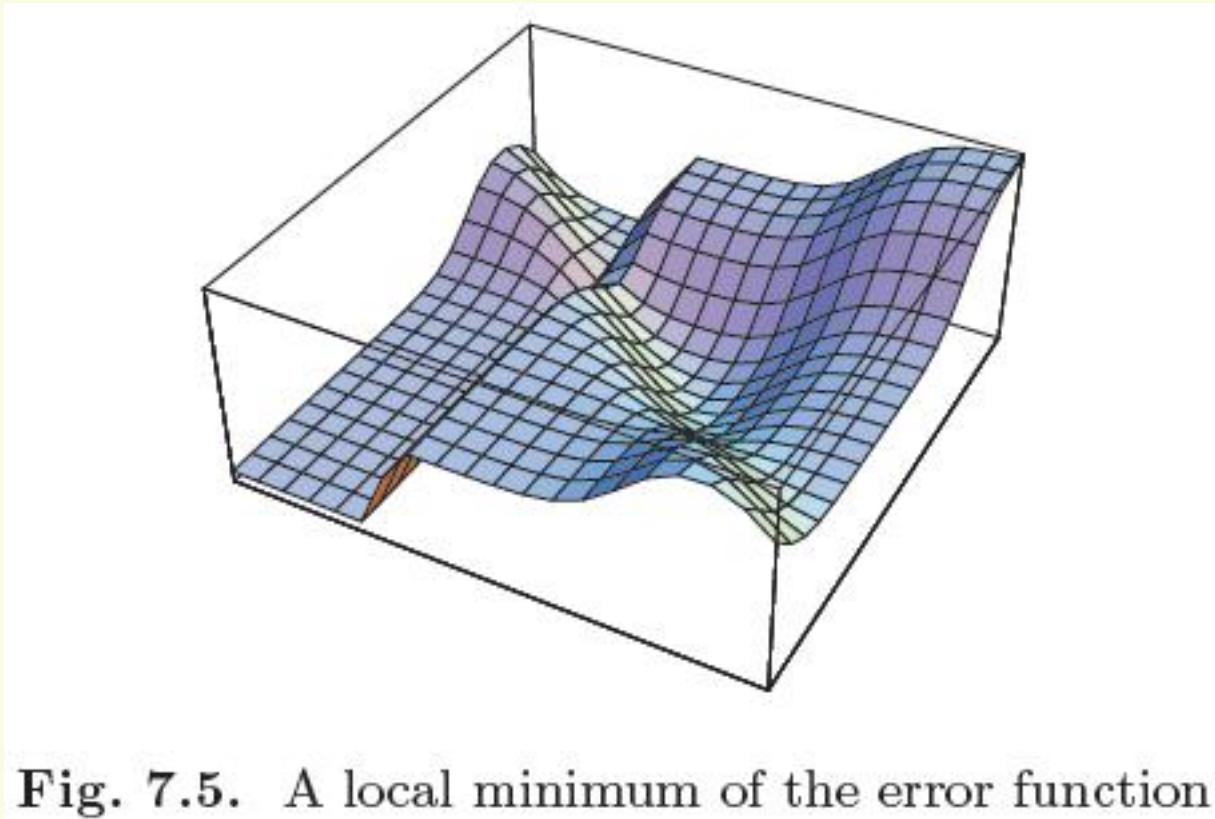


Fig. 7.5. A local minimum of the error function

Momentum Term

To avoid local oscillation, a momentum term is sometimes added:

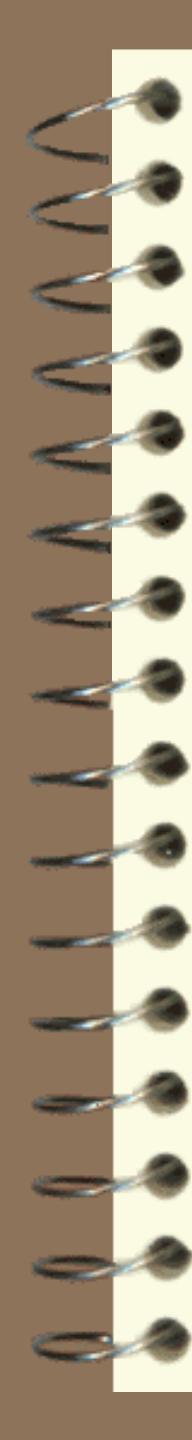
$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-1)$$

$$0 \leq \alpha < 1$$

Kolmogorov Theorem

Let $f: [0, 1]^n \rightarrow [0, 1]$ be a continuous function. There exist functions of one argument g and h_j for $j=1, 2, \dots, 2n+1$ and constant w_i for $i=1, 2, \dots, n$ such that

$$f(x_1, x_2, \dots, x_n) = \sum_{j=1}^{2n+1} g\left[\sum_{i=1}^n w_i h_j(x_i)\right].$$



Universal Approximators

- Multilayer feedforward neural networks are universal approximators of continuous functions.
- A set of weights exist such that the approximation errors can be arbitrarily small.
- However, the BP algorithm is not guaranteed to find such a set of weights.

Overfitting Problem

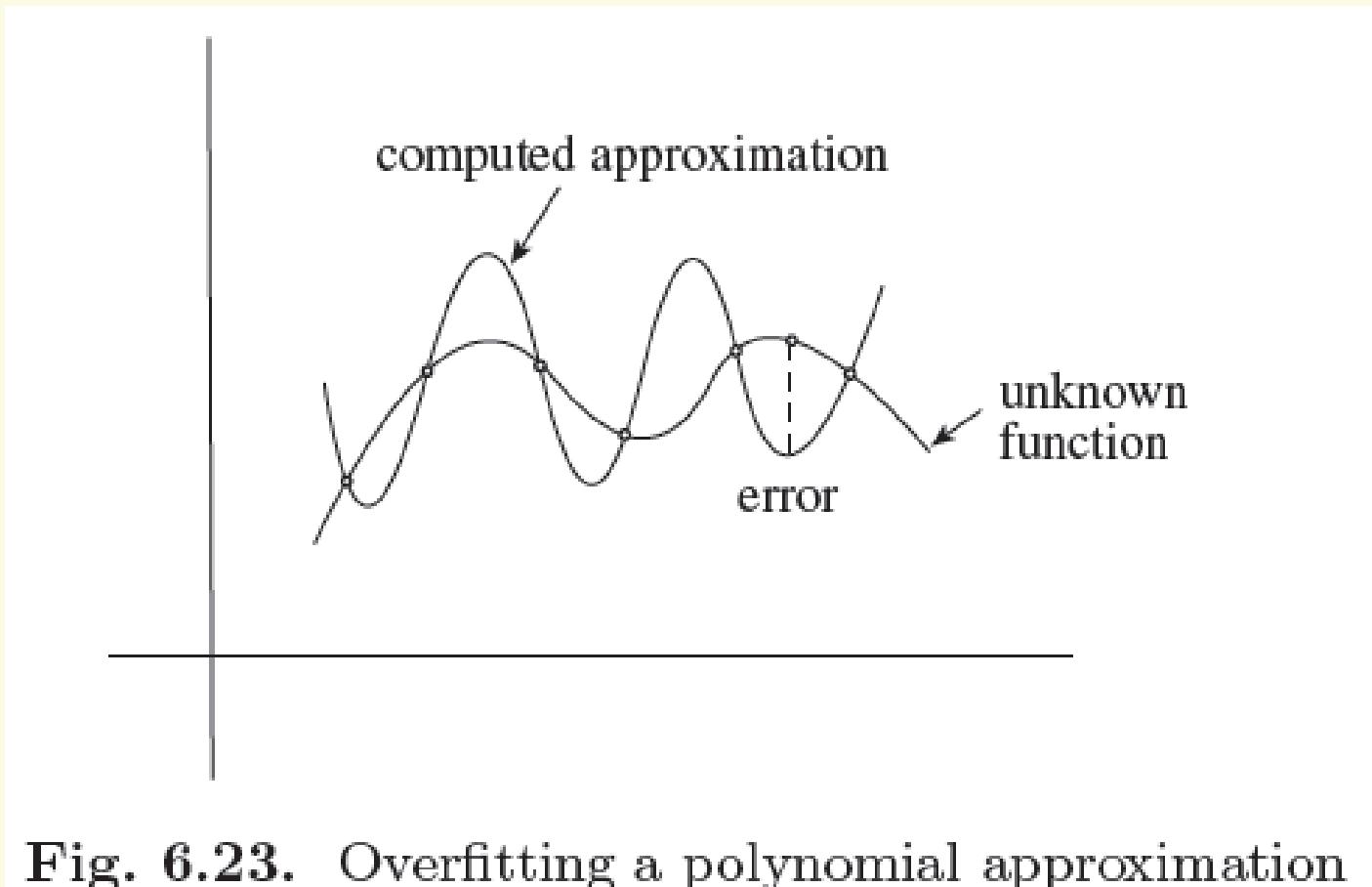
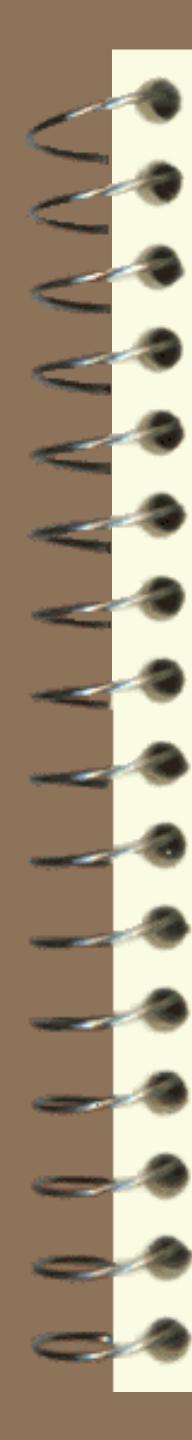


Fig. 6.23. Overfitting a polynomial approximation



Radial Basis Functions

- A radial basis function (RBF) is a real-valued function whose value depends only on the distance from its origin or center.
- Related to kernel theory in statistical learning.
- Used as the means for approximating or interpolating multivariate functions.

Radial Basis Functions

$$y = \sum_{i=1}^h w_i \Phi(\|x - c\|)$$

Polyharmonic spline $\Phi(r) = r^p$, $p = 1, 3, 5, \dots$

$$\text{Gaussian } \Phi(r) = \exp\left(-\frac{r^2}{\sigma^2}\right)$$

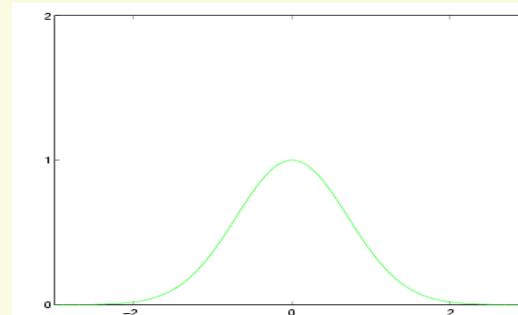
$$\text{Thin plate spline } \Phi(r) = r^2 \log(r)$$

$$\text{Multi-quadric } \Phi(r) = \sqrt{(r^2 + \sigma^2)}$$

$$\text{Inverse multi-quadric } \Phi(r) = \frac{1}{\sqrt{(r^2 + \sigma^2)}}$$

Radial Basis Functions

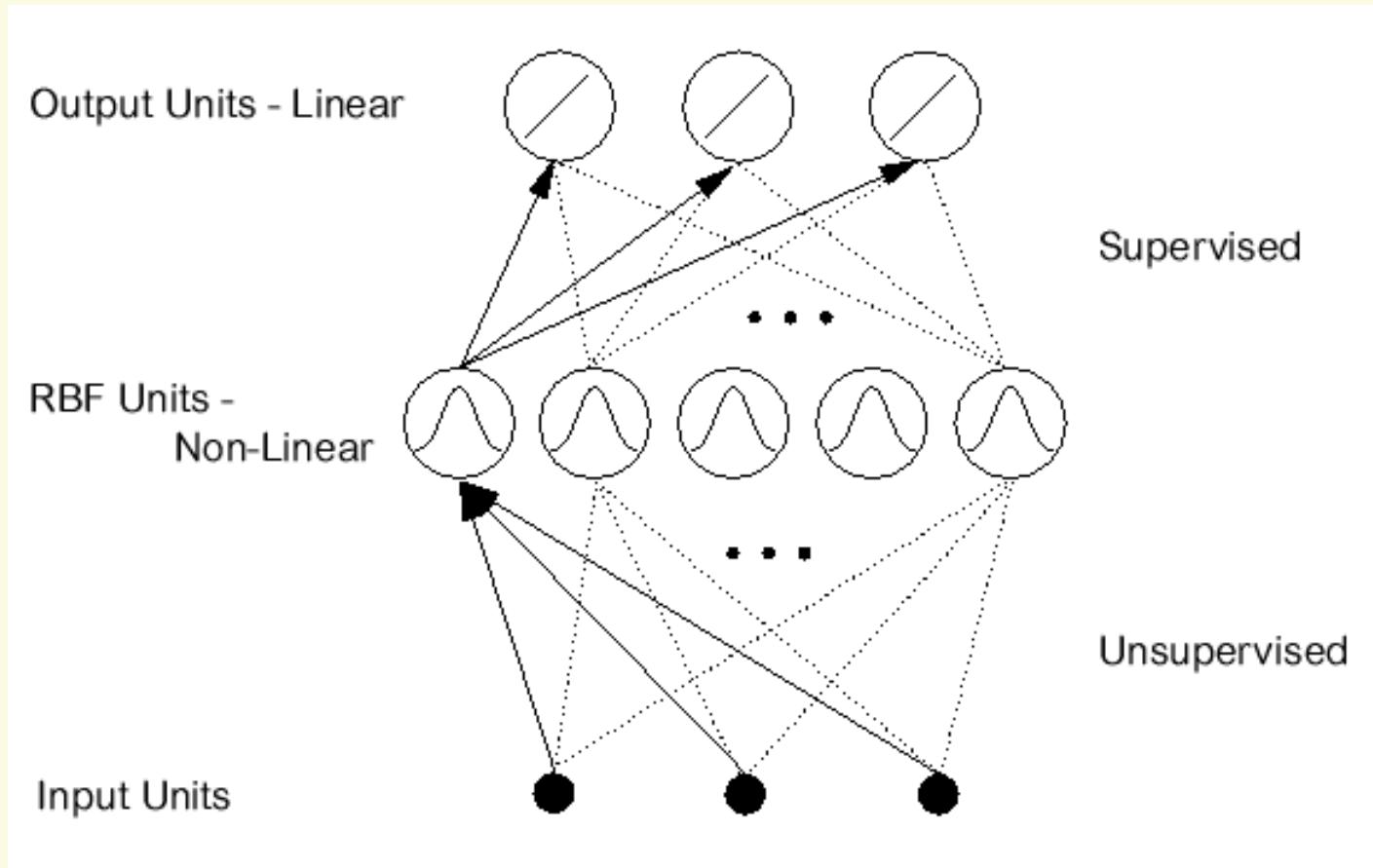
- The most commonly used radial-basis function is a Gaussian function
- In an RBF network, r is the distance from the center.
- Graphically, it is a bell-shaped curve.



Radial Basis Function Networks

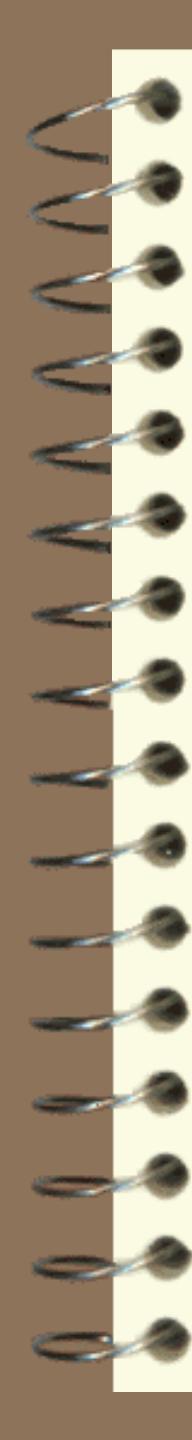
- Proposed first by Broomhead and Lowe in 1988.
- A linear combination of a number radial basis functions that play the role of hidden neurons.
- Two-layer architecture. Its output layer uses a linear activation function as ADALINE. Its hidden layer uses radial basis activation functions.

Radial Basis Function Networks



Cover's Theorem

- Cover's Theorem (1965):
- A dichotomy $\{X^+, X^-\}$ is said to be **φ -separable** if there exist an m -dimensional vector w such that
 - $w^T \varphi(x) \geq 0$, if x in X^+
 - $w^T \varphi(x) < 0$, if x in X^-
 - The hyperplane defined by $w^T \varphi(x) = 0$ is the separating surface between the two classes.



XOR Problem Revisited

An RBF network can transform the linearly inseparable XOR data in the input space to linearly separable data in the hidden state space.

XOR Problem Revisited

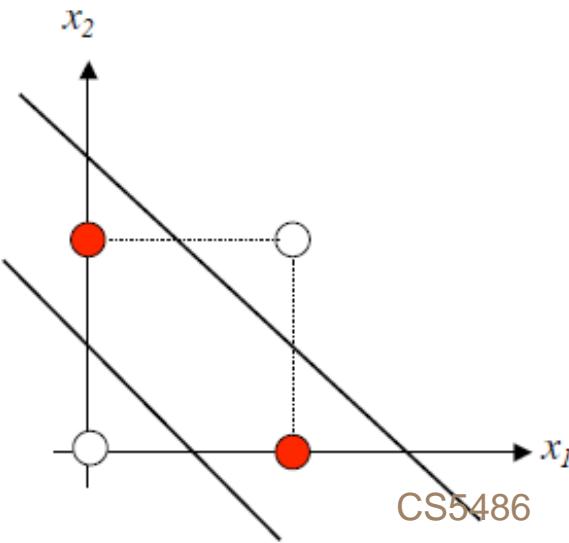
Using a pair of Gaussian RBFs, the input patterns are mapped onto the φ_1 - φ_2 plane and become linearly separable.

$$\phi_1(x) = e^{-\|x - c^1\|^2} \quad \phi_2(x) = e^{-\|x - c^2\|^2}$$

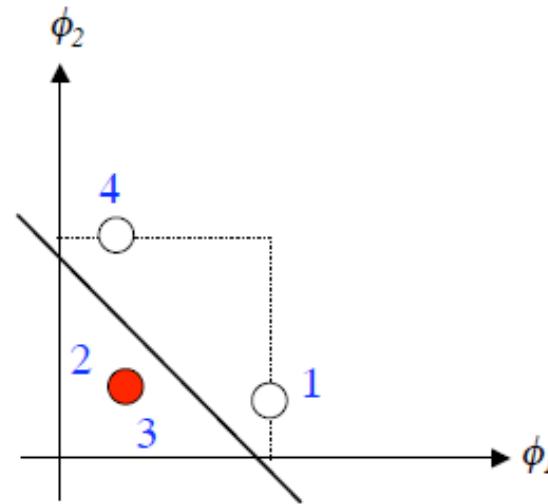
Let $c^1 = [1, 1]^T$ $c^2 = [0, 0]^T$

XOR Problem Revisited

p	x_1	x_2	ϕ_1	ϕ_2
1	0	0	1.0000	0.1353
2	0	1	0.3678	0.3678
3	1	0	0.3678	0.3678
4	1	1	0.1353	1.0000



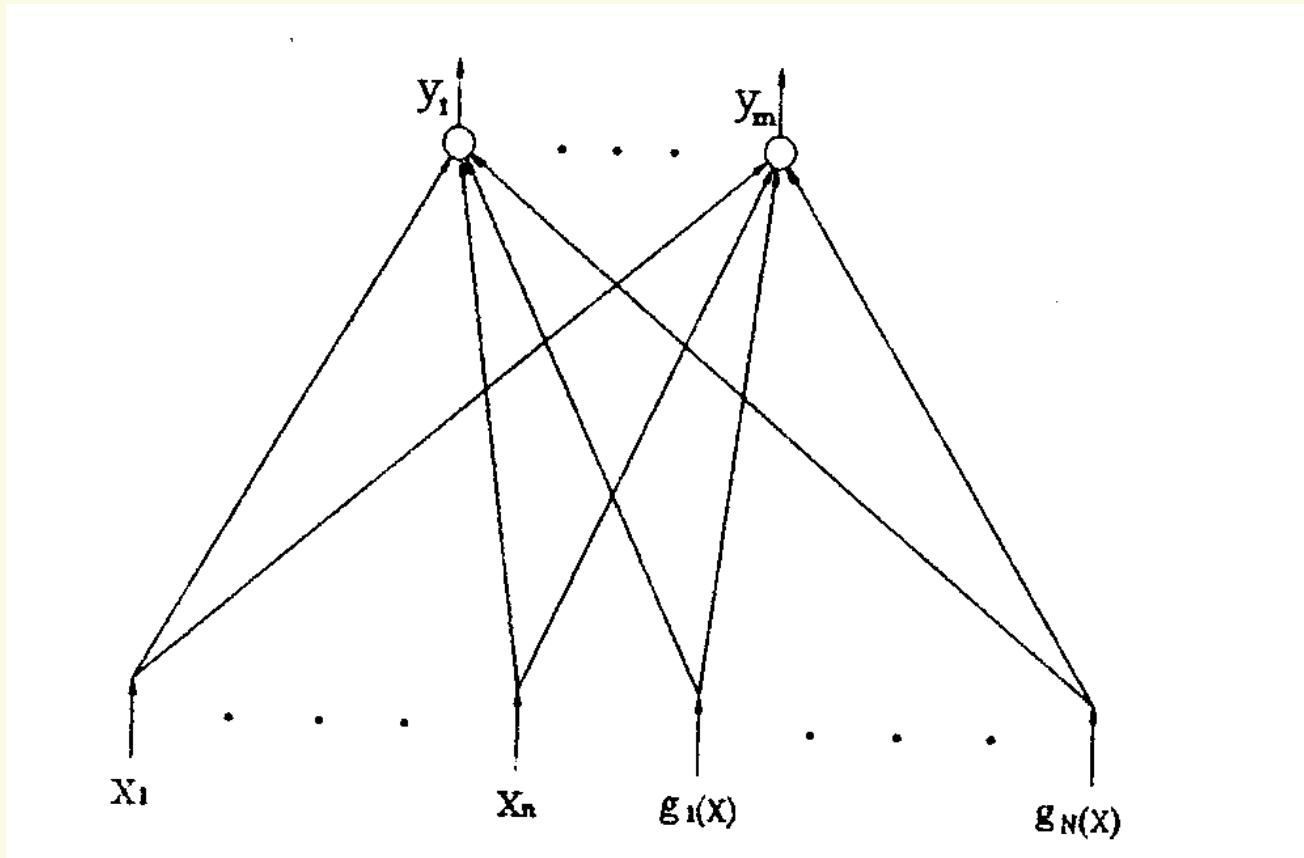
CS5486



Functional Link Network

- Proposed by Yoh-Han Pao at CWRU in late 80's
- One-layer feedforward architecture
- Higher-order aggregation rule
- Fast learning process
- Local minima could be eliminated
- Many successful stories in applications

Functional Link Network



Extreme Learning Machine

- Proposed by Guangbin Huang at NTU in mid 2000's
- One-layer feedforward architecture
- Random connection weights from inputs to hidden neurons
- Fast learning process for weights in output layer.
- Local minima eliminated

Support Vector Machine

- Proposed by Vladimir Vapnik based on statistical learning and kernel theory in early 1990's.
- Minimization of structural risk.
- Maximal generalization power.



V. Vapnik

Linear Discriminant Function

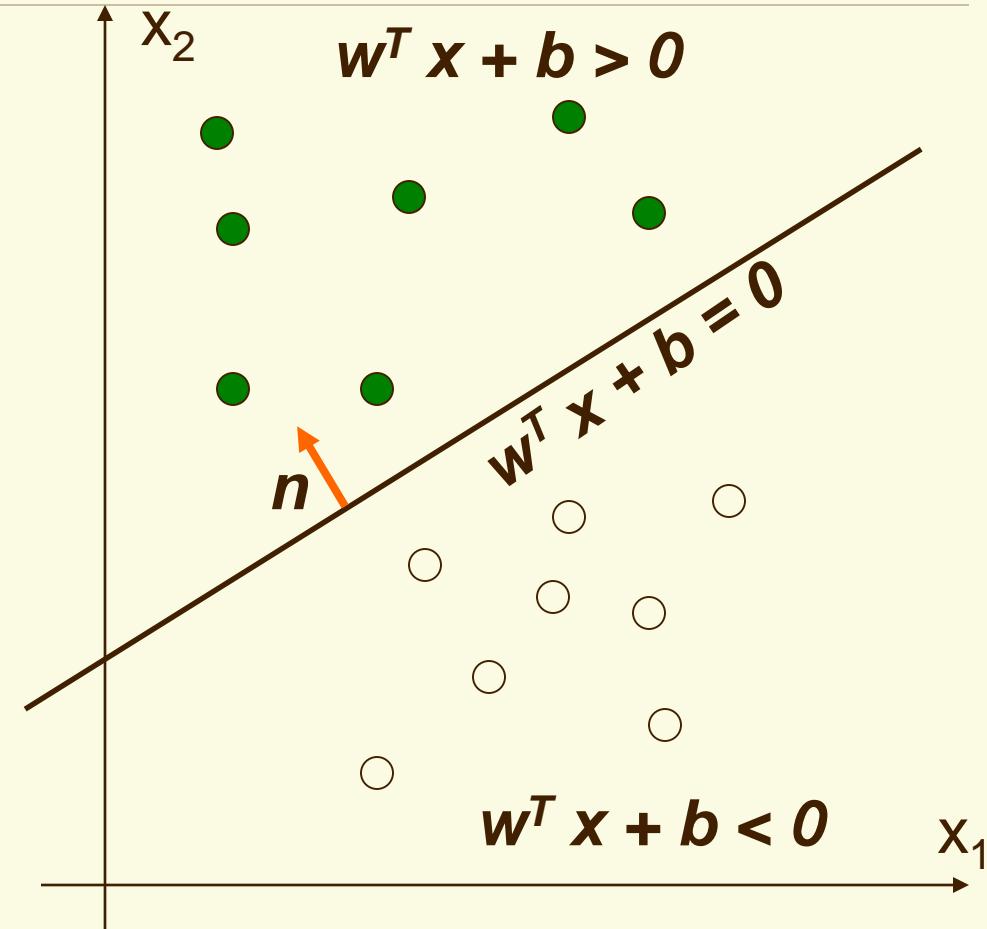
$g(x)$ is a linear function:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

- A hyper-plane in the feature space

(Unit-length) normal vector of the hyper-plane:

$$\mathbf{n} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

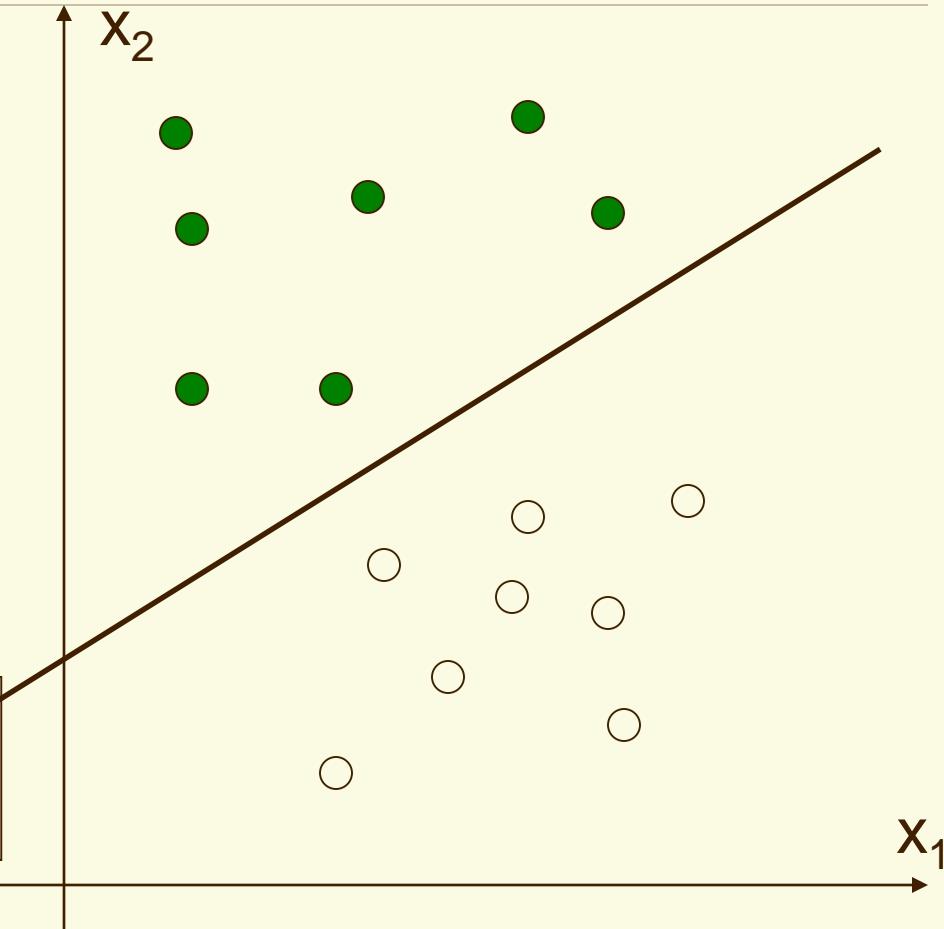


Linear Discriminant Function

How would you classify these data using a linear discriminant function in order to minimize the error?

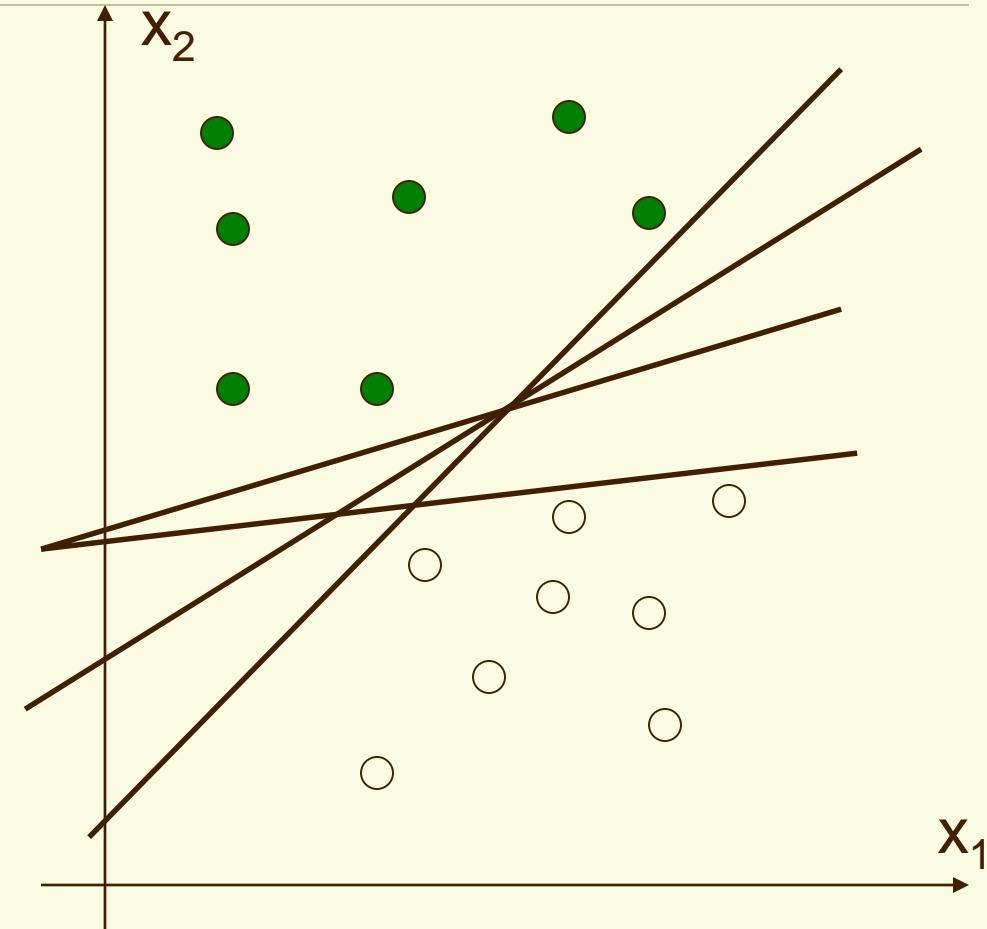
Infinite number of answers!

- denotes +1
- denotes -1



Linear Discriminant Function

Which one is
the best with
maximal
generalization
power?



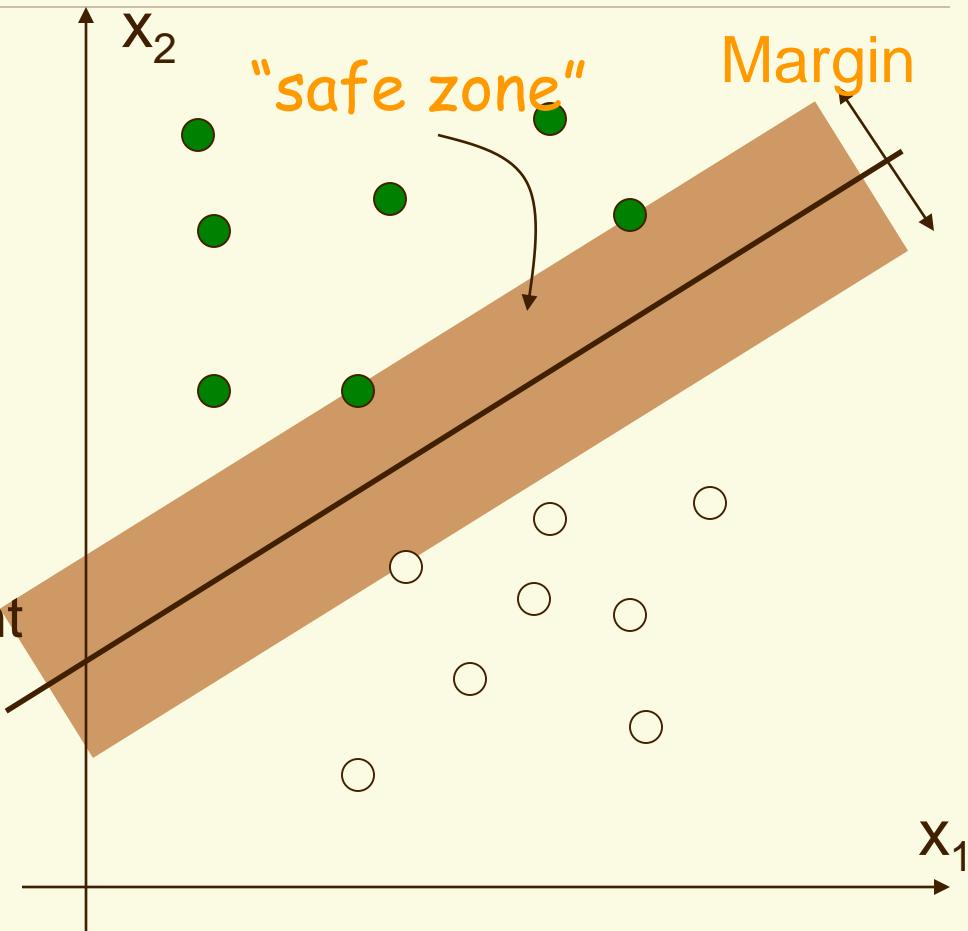
Maximal Margin Classifier

The linear discriminant function (classifier) with the maximum margin is the best.

Margin is defined as the width that the boundary could be increased by before hitting a data point

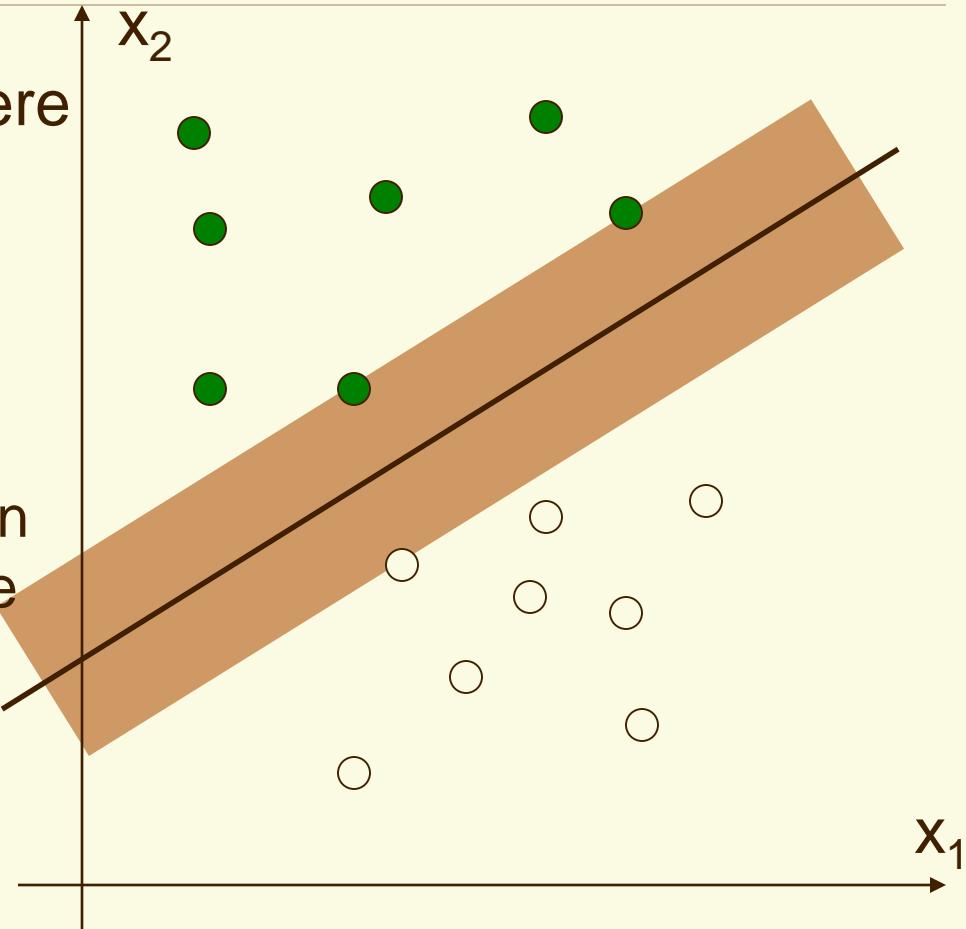
Why it is the best?

Robust to outliers and thus strong generalization ability



Maximal Margin Classifier

- Given a set of data:
 $\{(\mathbf{x}_i, y_i)\}, i = 1, 2, \dots, n$, where
- For $y_i = +1$, $\mathbf{w}^T \mathbf{x}_i + b > 0$
- For $y_i = -1$, $\mathbf{w}^T \mathbf{x}_i + b < 0$
- With a scale transformation
on both w and b , the above
is equivalent to
- For $y_i = +1$, $\mathbf{w}^T \mathbf{x}_i + b \geq 1$
- For $y_i = -1$, $\mathbf{w}^T \mathbf{x}_i + b \leq -1$



Maximal Margin Classifier

Let

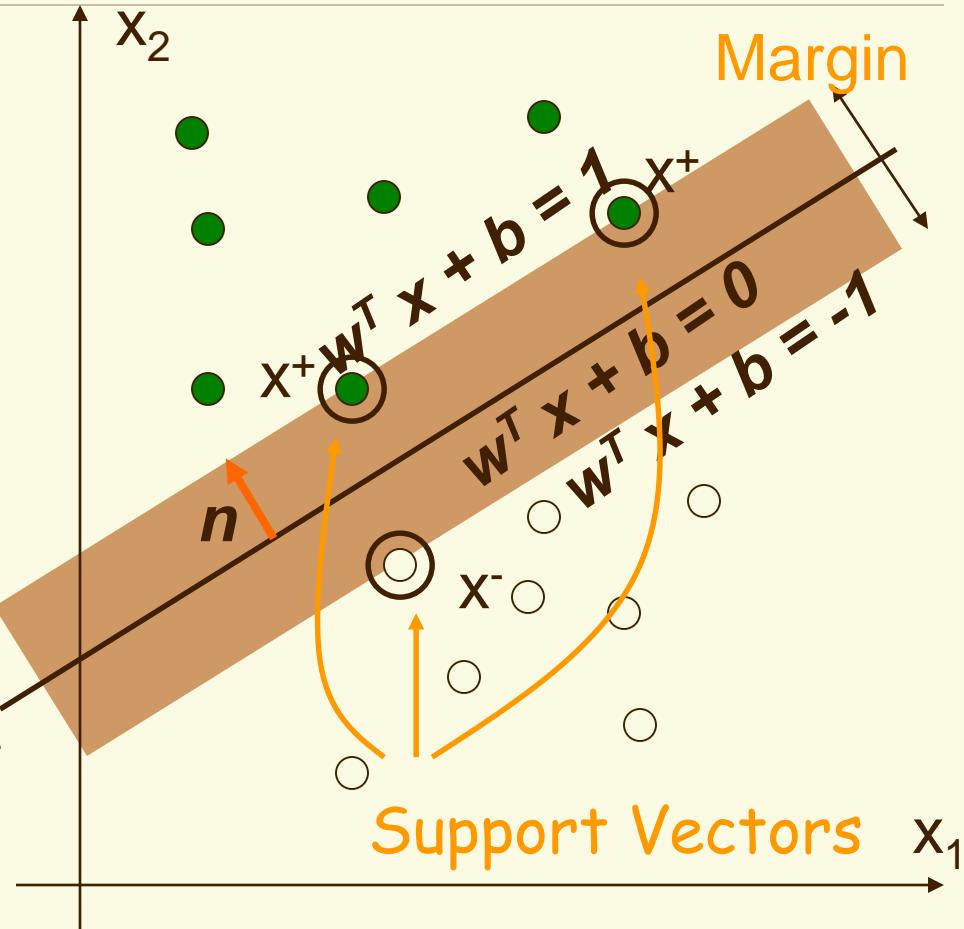
$$\mathbf{w}^T \mathbf{x}^+ + b = 1$$

$$\mathbf{w}^T \mathbf{x}^- + b = -1$$

The margin width is:

$$M = (\mathbf{x}^+ - \mathbf{x}^-) \cdot \mathbf{n}$$

$$= (\mathbf{x}^+ - \mathbf{x}^-) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$



Maximal Margin Classifier

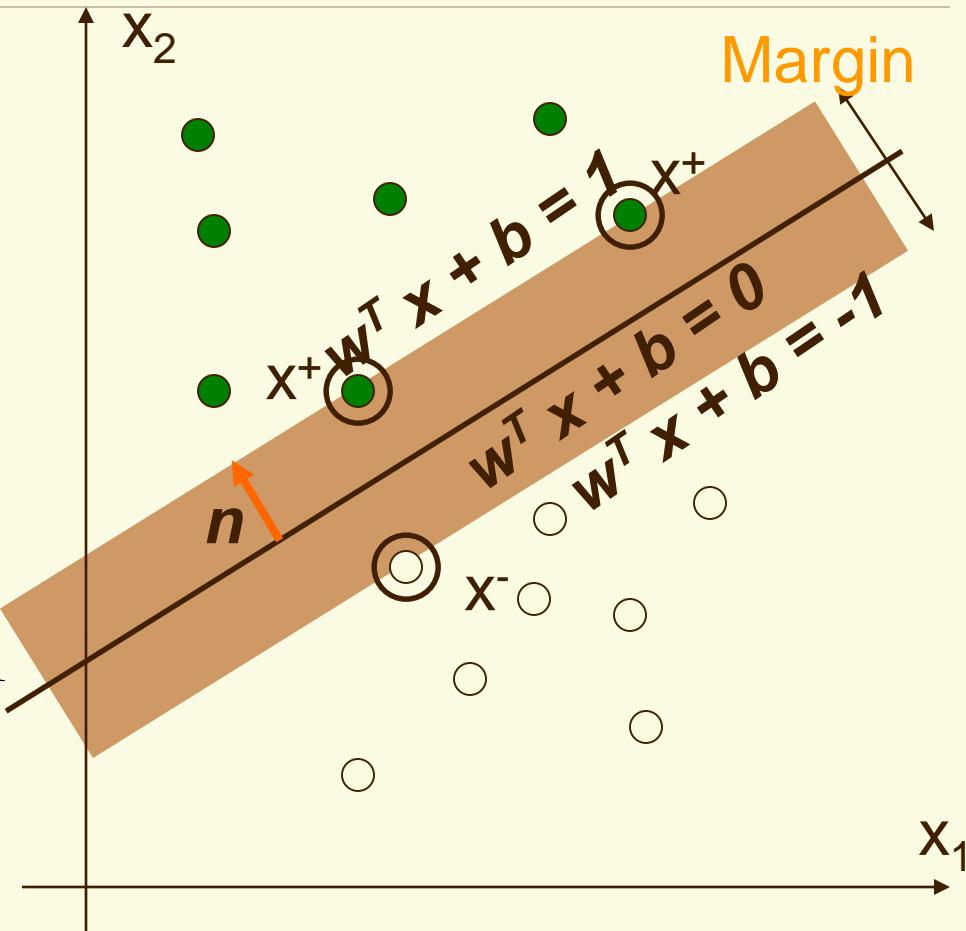
Formulation:

$$\text{maximize } \frac{2}{\|\mathbf{w}\|}$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$



Maximal Margin Classifier

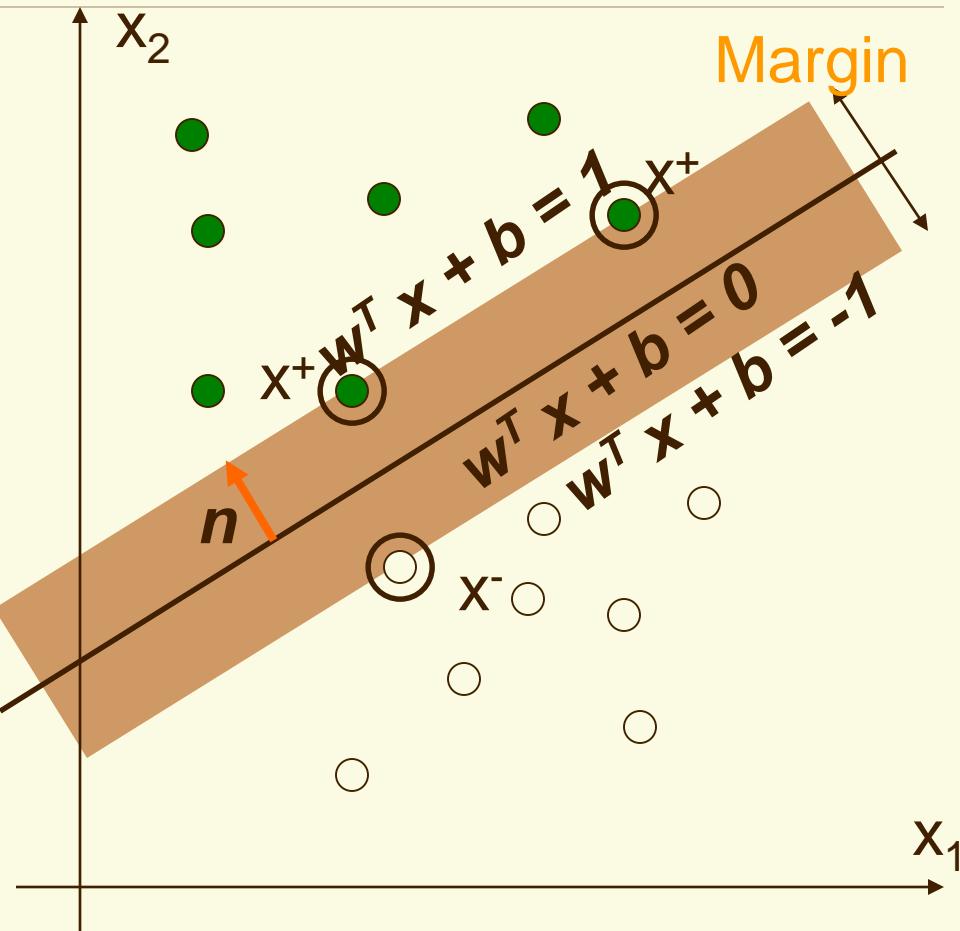
Formulation:

$$\text{minimize } \frac{1}{2} \|\mathbf{w}\|^2$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$



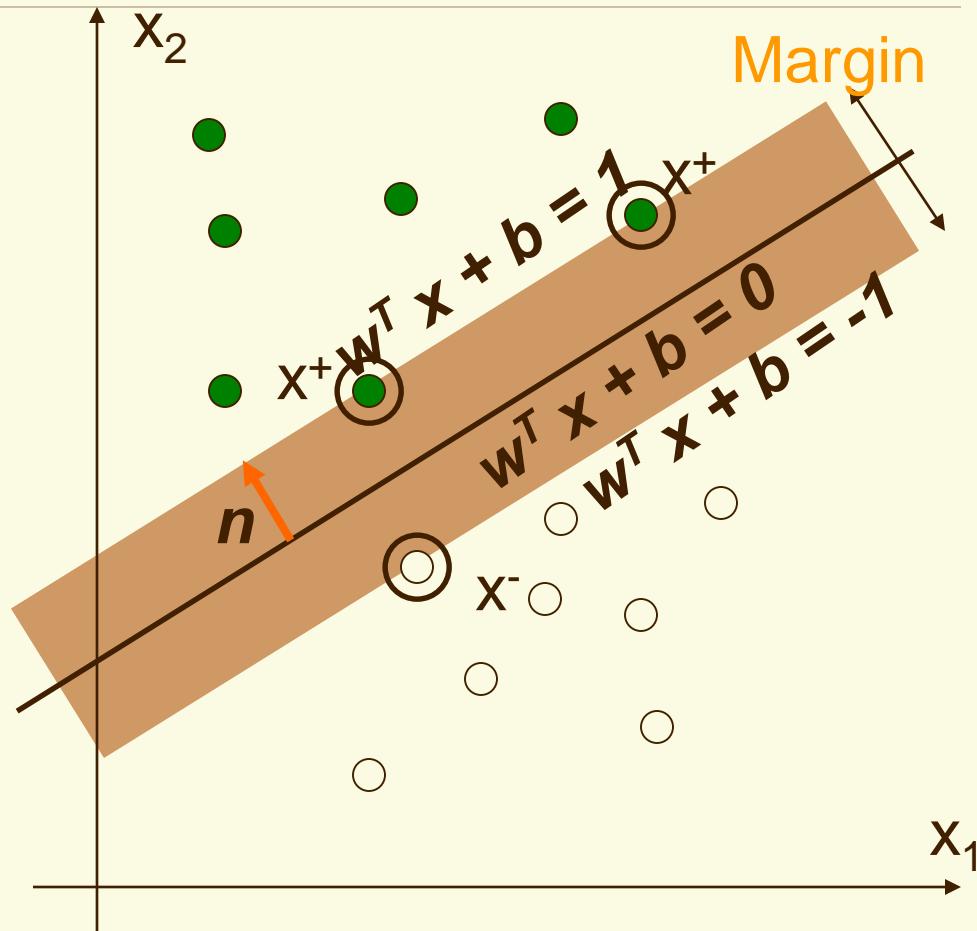
Maximal Margin Classifier

Formulation:

$$\text{minimize } \frac{1}{2} \|\mathbf{w}\|^2$$

subject to

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

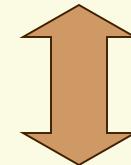


Quadratic Programming Problem

Quadratic
programming
with linear
constraints

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

Lagrangian
Function



$$\begin{aligned} & \text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) \\ \text{s.t.} \quad & \alpha_i \geq 0 \end{aligned}$$

Quadratic Programming Problem

$$\begin{aligned} \text{minimize } L_p(\mathbf{w}, b, \alpha_i) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) \\ \text{s.t. } \alpha_i &\geq 0 \end{aligned}$$

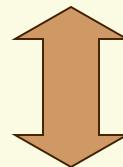
$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \quad \longrightarrow \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \longrightarrow \quad \sum_{i=1}^n \alpha_i y_i = 0$$

Quadratic Programming Problem

$$\begin{aligned} \text{minimize } L_p(\mathbf{w}, b, \alpha_i) = & \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) \\ \text{s.t. } & \alpha_i \geq 0 \end{aligned}$$

Lagrangian dual
problem



$$\begin{aligned} \text{maximize } & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{s.t. } & \alpha_i \geq 0, \text{ and } \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

Quadratic Programming Problem

From KKT condition, we know:

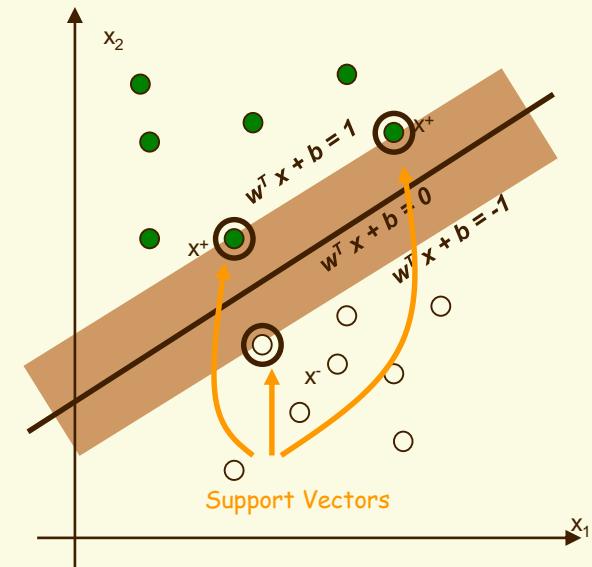
$$\alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0$$

Thus, only support vectors $\alpha_i \neq 0$

The solution has the form:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i$$

get b from $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$,
where \mathbf{x}_i is support vector



Linear Discriminant Function

The linear discriminant function is:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i \in SV} \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

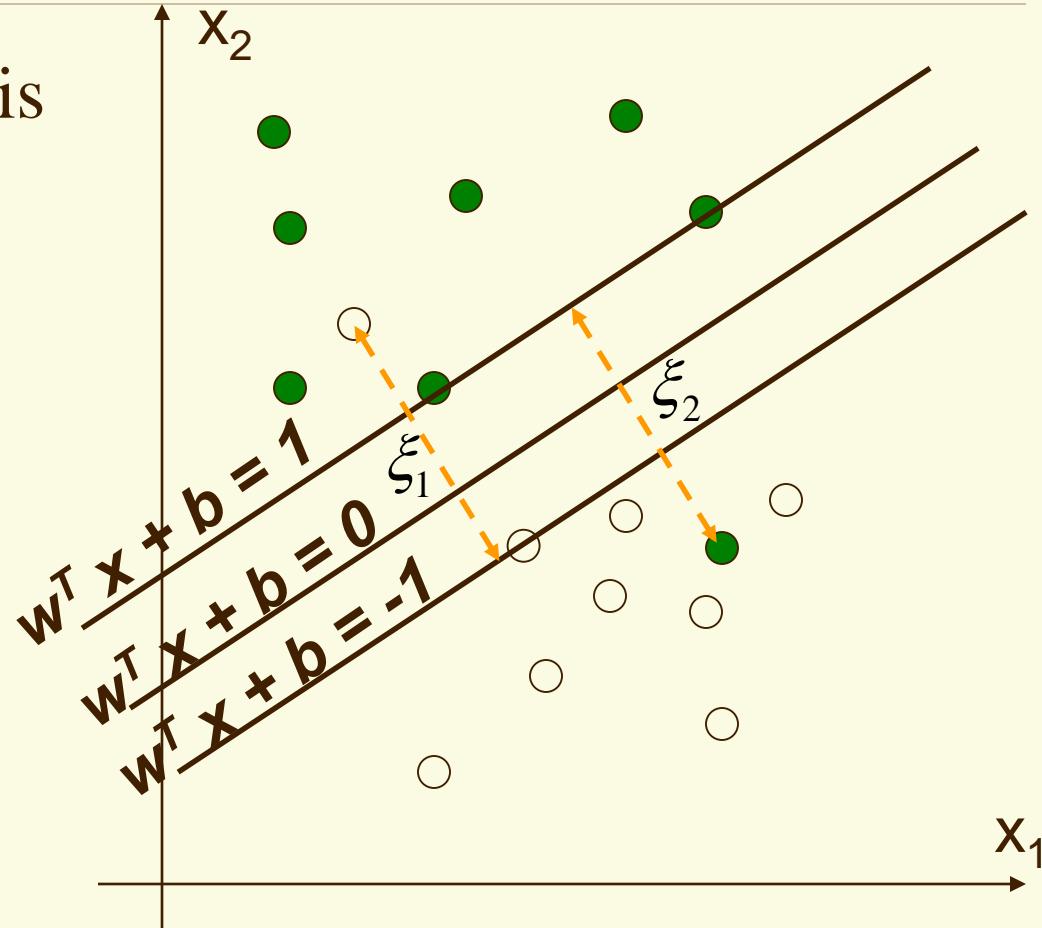
It is a weighted *dot product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i

Solving the optimization problem involved computing the *dot products* $\mathbf{x}_i^T \mathbf{x}_j$ between all pairs of training samples

Soft Constraints

What to do if data is not linearly separable? (noisy data, outliers, etc.)

Slack variables ξ_i can be added to allow misclassification of nonlinearly distributed or noisy data



Soft Constraints

Problem formulation:

$$\text{minimize} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

subject to

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

Parameter C is a weight between marginal maximization and misclassification minimization.

Maximal Margin Classifier

Problem reformulation: (Lagrangian dual problem)

$$\text{maximize} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

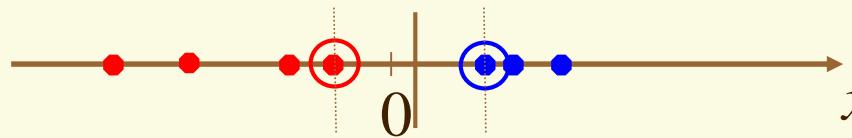
Subject to

$$0 \leq \alpha_i \leq C$$

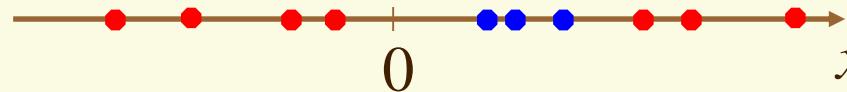
$$\sum_{i=1}^n \alpha_i y_i = 0$$

Nonlinear SVM

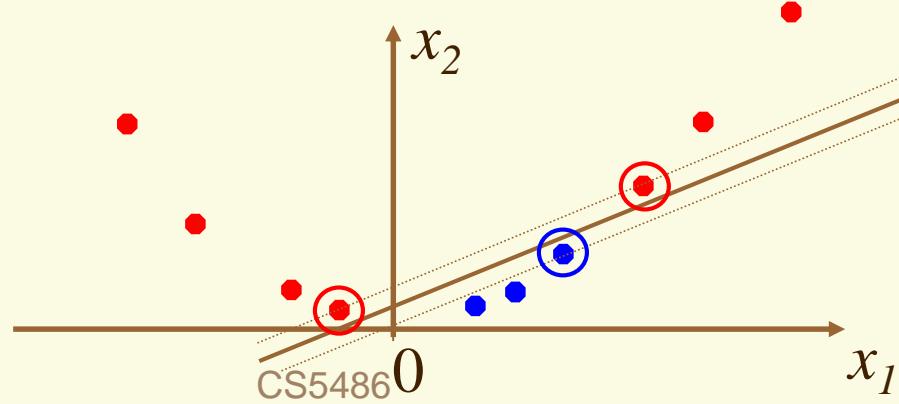
Datasets that are linearly separable with noise work out great:



But what to do if the dataset is nonlinearly distributed?

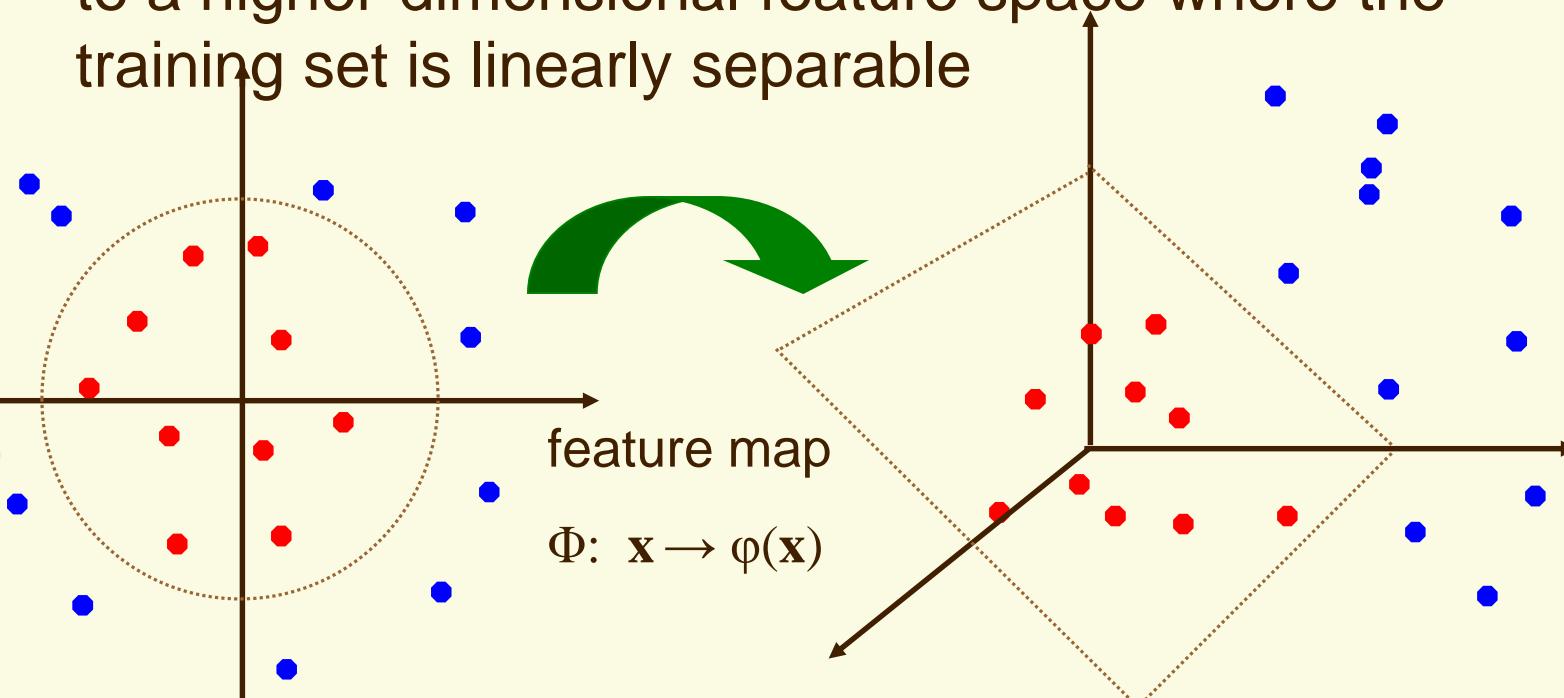


How about... mapping data to a higher-dimensional space:



Feature Space

General idea: the original input space can be mapped to a higher-dimensional feature space where the training set is linearly separable



The Kernel Trick

With this mapping, the discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in SV} \alpha_i \boxed{\phi(\mathbf{x}_i)^T \phi(\mathbf{x})} + b$$

No need to know this mapping explicitly, because we only use the **dot product** of feature vectors in both the training and test.

A ***kernel function*** is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

CS5486

An Example

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$;

$$\text{let } K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{1} + \mathbf{x}_i^T \mathbf{x}_j)^2,$$

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j)$:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{1} + \mathbf{x}_i^T \mathbf{x}_j)^2, \\ &= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \\ &= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}] \\ &= \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j), \quad \text{where } \boldsymbol{\varphi}(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2] \end{aligned}$$

Common Kernel Functions

- Linear kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

- Polynomial kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$$

- Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

Kernel Functions

For some functions $K(\mathbf{x}_i, \mathbf{x}_j)$ checking that

$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ can be cumbersome.

Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_3)$...	$K(\mathbf{x}_1, \mathbf{x}_N)$
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_3)$		$K(\mathbf{x}_2, \mathbf{x}_N)$
...
$K(\mathbf{x}_P, \mathbf{x}_1)$	$K(\mathbf{x}_N, \mathbf{x}_2)$	$K(\mathbf{x}_N, \mathbf{x}_3)$...	$K(\mathbf{x}_N, \mathbf{x}_N)$

Nonlinear SVM: Optimization

Problem reformulation: (Lagrangian dual problem)

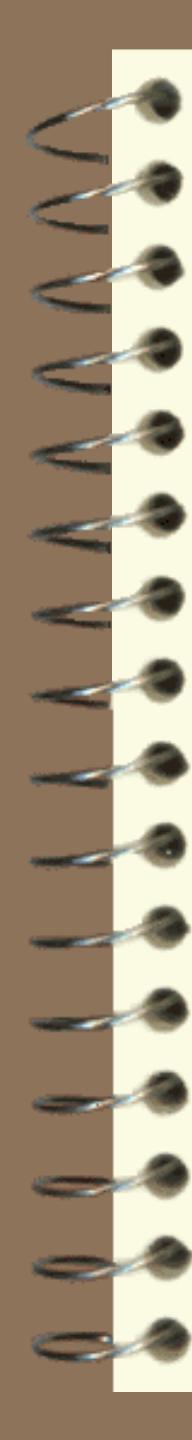
$$\text{maximize} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\begin{aligned} \text{subject to} \quad & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i \in SV} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

The optimization technique is the same.



SVM Learning

1. Choose a kernel function
2. Choose a value for C
3. Solve the quadratic programming problem (many software packages available)
4. Construct the discriminant function from the support vectors

Design Issues

¶ Choice of kernel

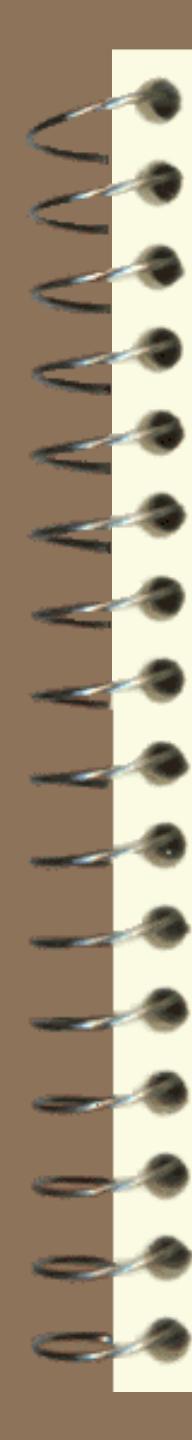
- Gaussian or polynomial kernel is default
- if ineffective, more elaborate kernels are needed
- domain experts can give assistance in formulating appropriate similarity measures

¶ Choice of kernel parameters

- e.g. σ in Gaussian kernel
- σ is the distance between closest points with different classifications
- In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

¶ Optimization criterion – Hard margin v.s. Soft margin

- a lengthy series of experiments in which various parameters are tested



SVM Summary

1. Maximal Margin Classifier

- Better generalization ability & less over-fitting

2. The Kernel Trick

- Map data points to a higher dimensional space to make them linearly separable.
- Since only dot product is used, we do not need to represent the mapping explicitly.

Support Vector Regression

Problem formulation

$$\text{minimize} \frac{1}{2} \|w\|^2$$

$$\text{subject to } \|y_i - (w \cdot x_i - b)\| \leq \varepsilon$$

Support Vector Regression

Problem reformulation

$$\text{minimize} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^l (\xi_i + \xi_i^*)$$

subject to

$$\begin{cases} y_i - w \cdot x_i - b \leq \varepsilon + \xi_i \\ w \cdot x_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

Support Vector Regression

$$\begin{aligned} L = & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^l (\xi_i + \xi_i^*) - \sum_{i=1}^l (\eta_i \xi_i + \eta_i^* \xi_i^*) \\ & - \sum_{i=1}^l \alpha_i (\varepsilon + \xi_i - y_i + w \cdot x_i + b) \\ & - \sum_{i=1}^l \alpha_i^* (\varepsilon + \xi_i^* + y_i - w \cdot x_i - b) \end{aligned}$$

$$\min_{w,b,\xi} \max_{\alpha,\eta} L$$

subject to $\alpha, \eta \geq 0$

Support Vector Regression

$$\frac{\partial L}{\partial b} = \sum_{i=1}^l (\alpha_i^* - \alpha_i) = 0$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^l (\alpha_i - \alpha_i^*) x_i = 0$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0$$

$$\max_{\alpha, \eta} -\frac{1}{2} \sum_{i,j=1}^l (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) x_i \cdot x_j - \varepsilon \sum_{i=1}^l (\alpha_i + \alpha_i^*) + \sum_{i=1}^l y_i (\alpha_i - \alpha_i^*)$$

subject to $\sum_{i=1}^l (\alpha_i - \alpha_i^*) = 0$ and $\alpha_i, \alpha_i^* \in [0, C]$

Least-squares SVM

Proposed by Suykens and Vandewalle at KUL in 1999.

Let $y_i = \mathbf{w}^T \varphi(\mathbf{x}_i) + b + e_i$

Primal problem formulation

$$\min_{\mathbf{w}, b, e_i} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + \gamma \frac{1}{2} \sum_{i=1}^N e_i^2$$

$$y_i = \mathbf{w}^T \varphi(\mathbf{x}_i) + b + e_i, \quad i = 1, \dots, N.$$

Least-squares SVM

Lagrangian

$$\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \gamma \frac{1}{2} \sum_{i=1}^N e_i^2 - \sum_{i=1}^N \alpha_i (\mathbf{w}^T \varphi(\mathbf{x}_i) + b + e_i - y_i)$$

$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0, \frac{\partial \mathcal{L}}{\partial b} = 0, \frac{\partial \mathcal{L}}{\partial e_i} = 0, \frac{\partial \mathcal{L}}{\partial \alpha_i} = 0$ gives

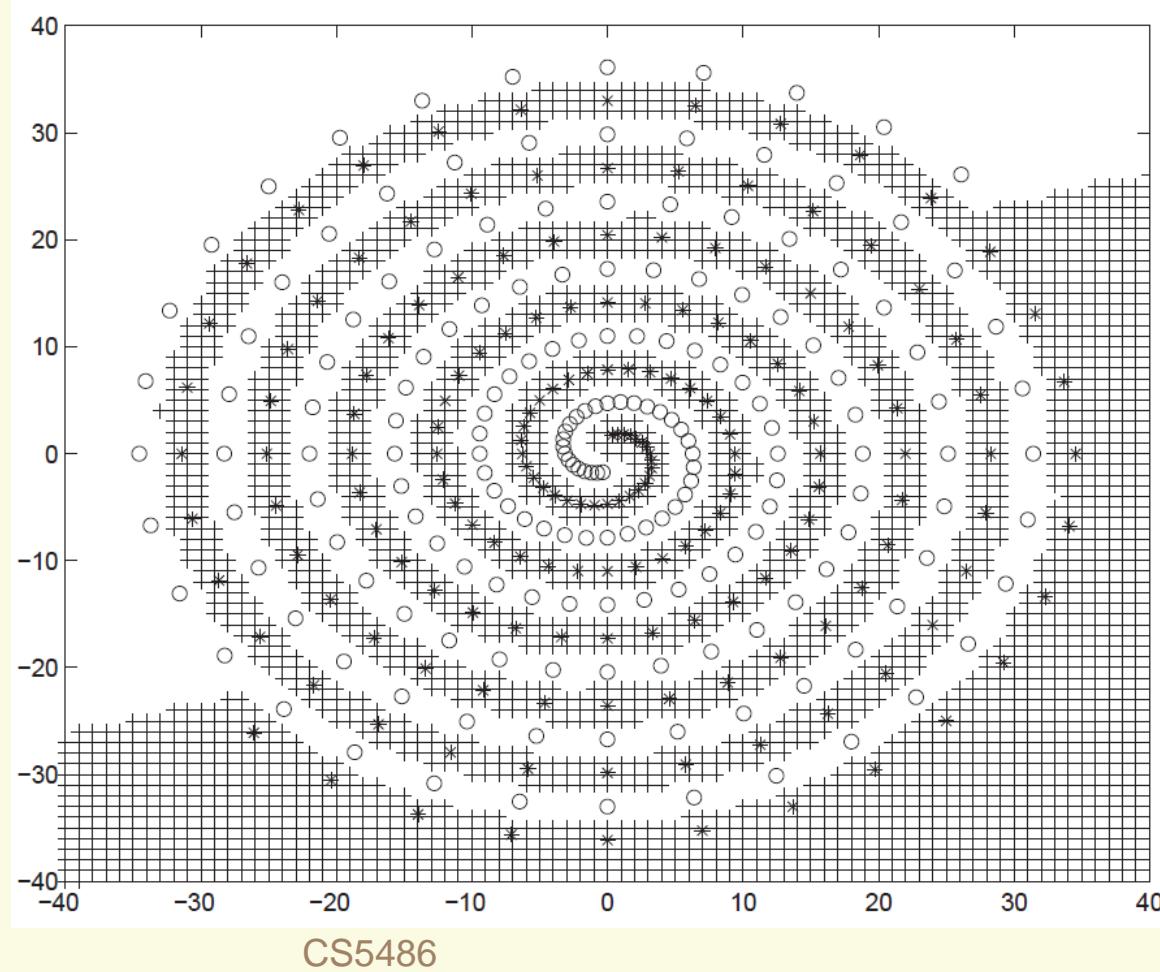
$$\left[\begin{array}{c|c} \Omega + \gamma^{-1} \mathbf{I} & \mathbf{1} \\ \hline \mathbf{1}^T & 0 \end{array} \right] \left[\begin{array}{c} \boldsymbol{\alpha} \\ b \end{array} \right] = \left[\begin{array}{c} \mathbf{y} \\ 0 \end{array} \right]$$

Least-squares SVM

Final model

$$y(\mathbf{x}) = \sum_{i=1}^N \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

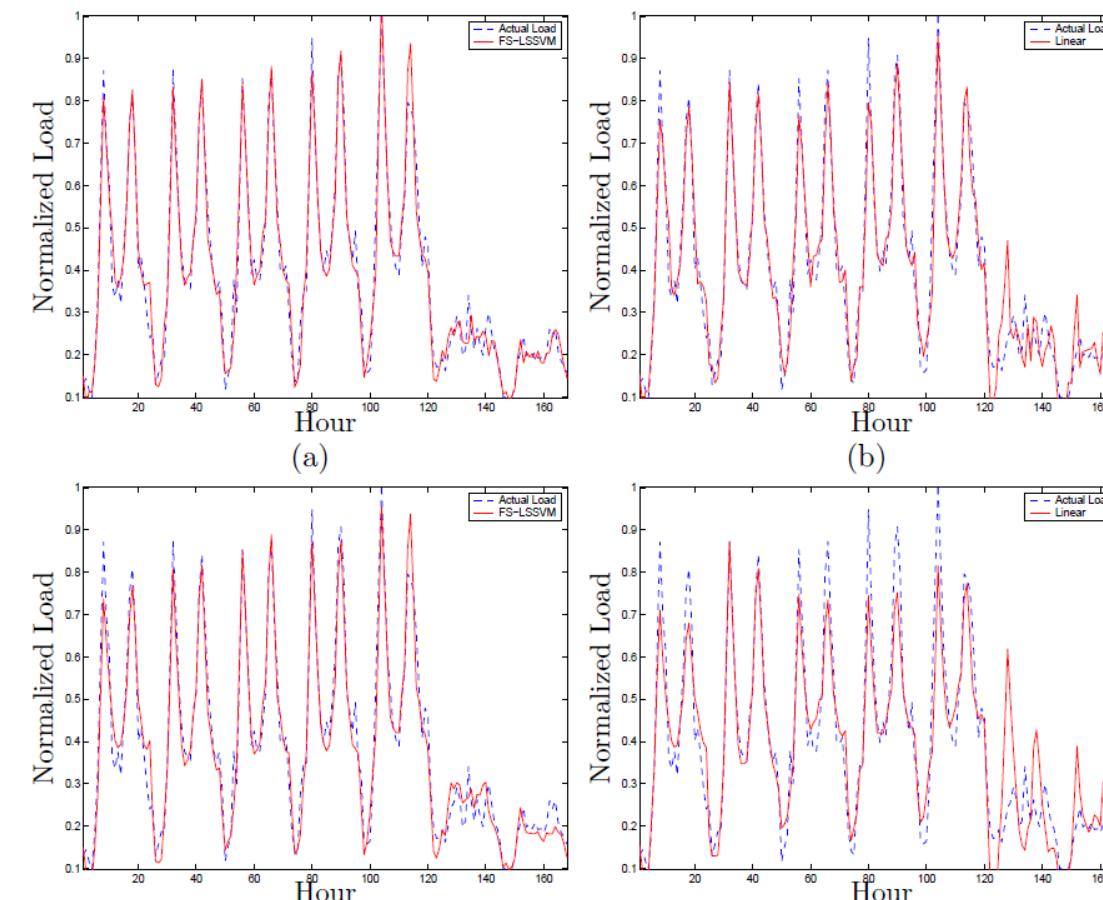
Two-spirial Classification

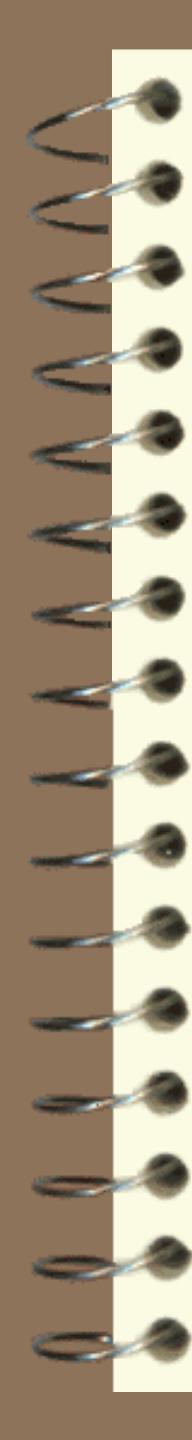


CS5486

162

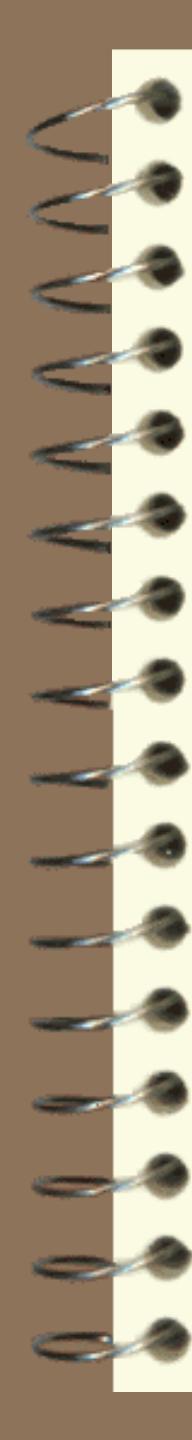
Short-term Prediction





MAXNET

- A sub-network for selecting the input with maximum value - winner takes all.
- By means of mutually prohibition, a MAXNET keeps the maximal input and presses down the rest.
- It is often used as the output layer in some existing neural networks



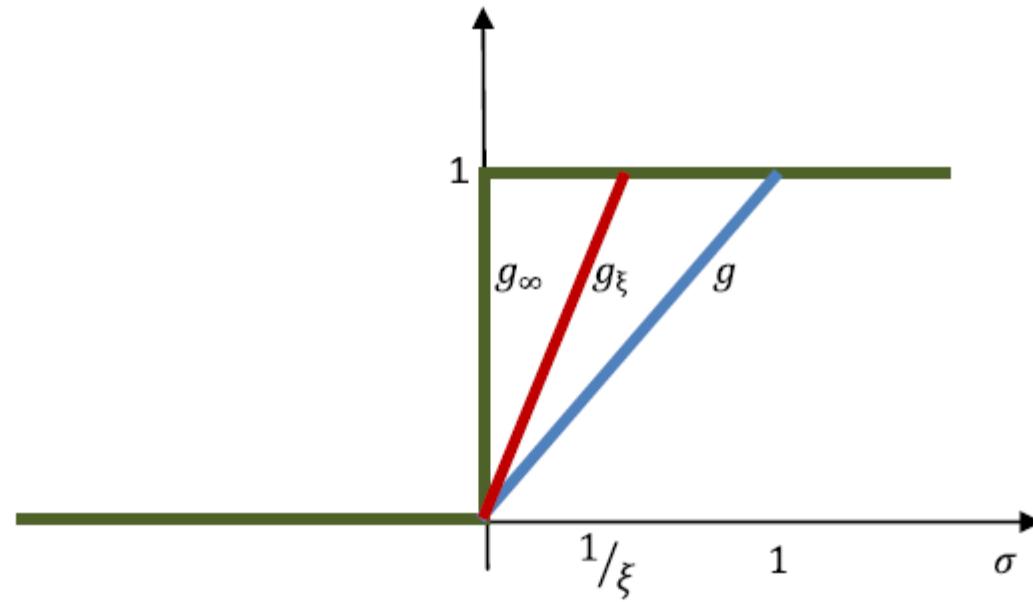
MAXNET

- A recurrent neural network with self excitatory connections and laterally inhibitory connections.
- The weight of self excitatory connections is 1.
- The weight of self inhibitory connections is $-w$ where $w < 1/m$, and m is the number of output neurons.

kWTA Model

state equation $\epsilon \frac{dy}{dt} = \sum_{i=1}^n x_i - k,$

output equation $x_i = g_\infty(u_i - y), i = 1, \dots, n;$



Desirable Properties

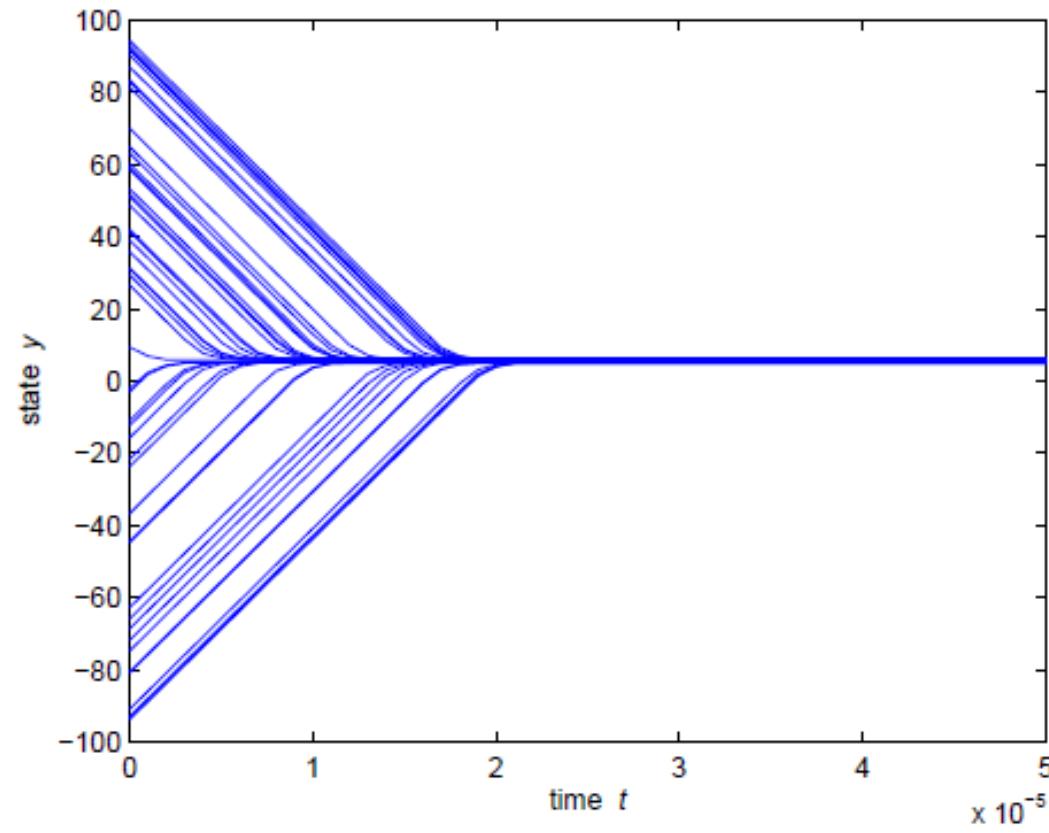
- The k WTA model with Heaviside activation function has been proven to be globally stable and globally convergent to the k WTA solutions in *finite time*.
- Derived lower and upper bounds of convergence time are respectively

$$\bar{t} \geq \frac{\epsilon |\bar{y} - y_0|}{\max\{n - k, k\}}$$

$$\bar{t} \leq \epsilon |\bar{y} - y_0|.$$

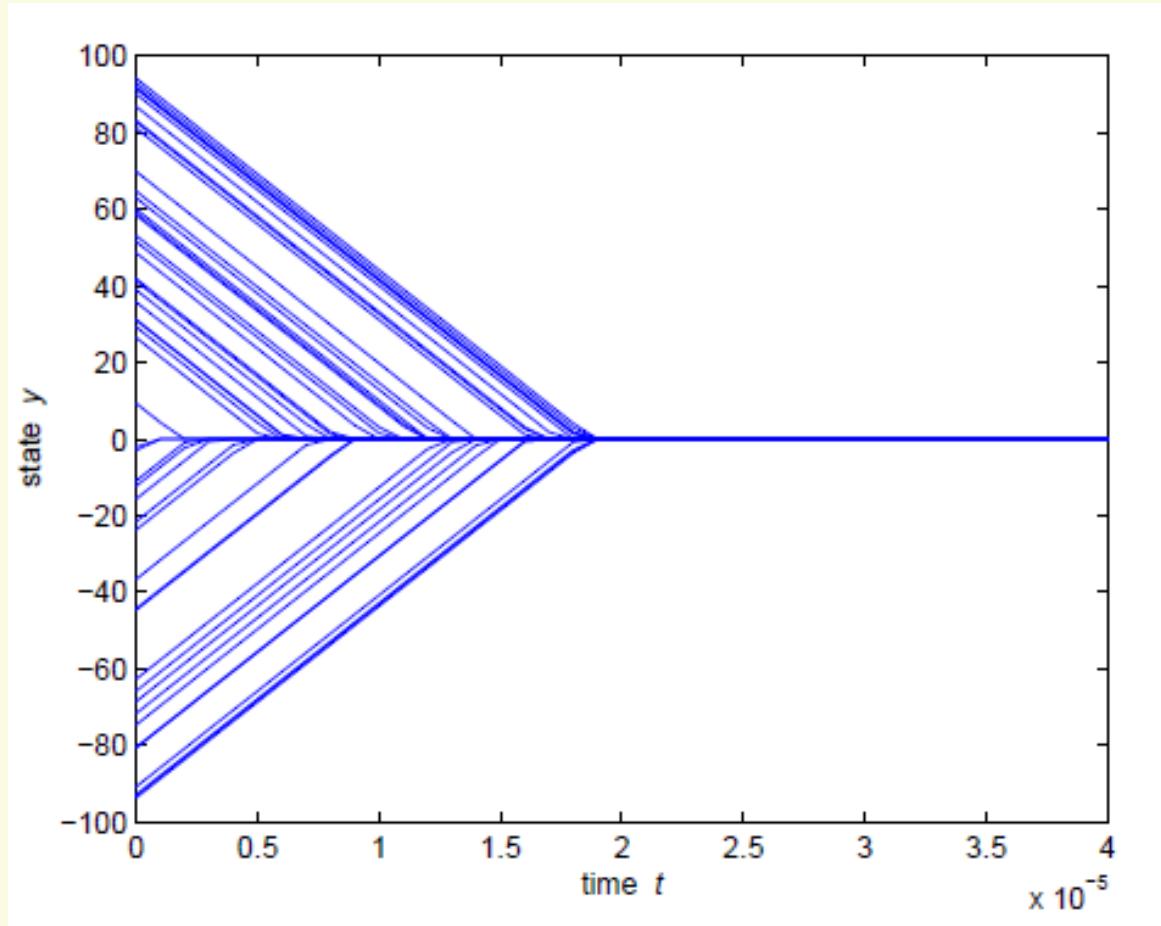
Simulation Results

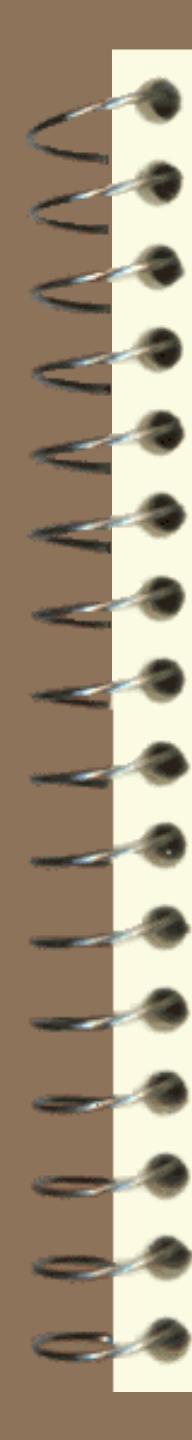
Randomized Integer Inputs



Simulation Results

Low-Resolution Inputs



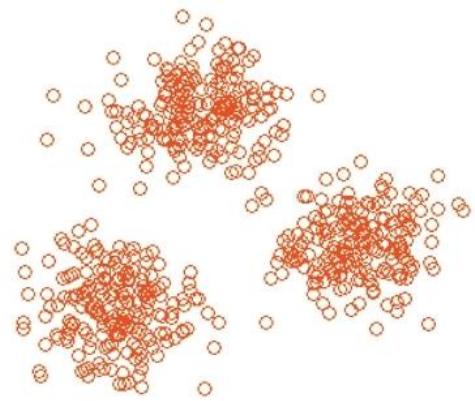


Clustering

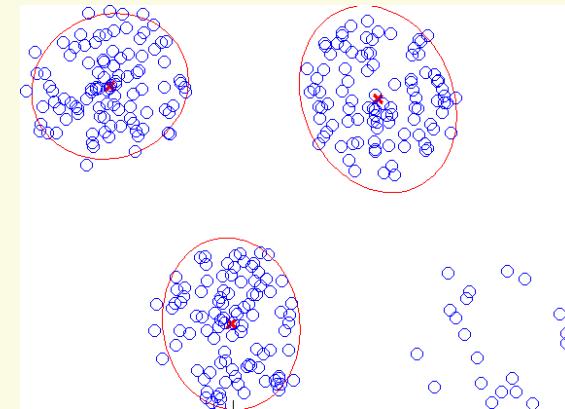
物以类聚
人以群分

Clustering

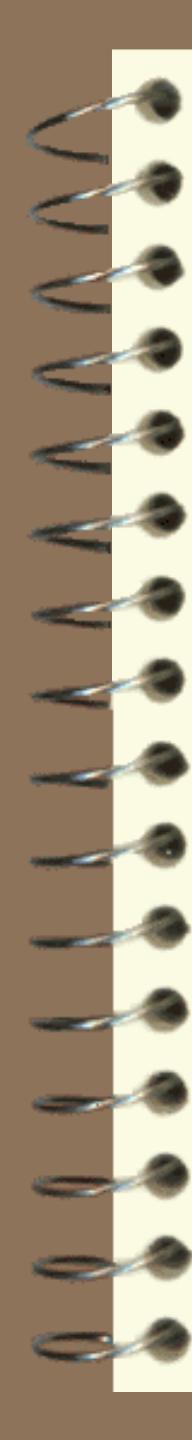
- Clustering or cluster analysis is to group similar data based on a given similarity measure.
- It is subjective without unique solutions.
- It is done via unsupervised learning.



... .86



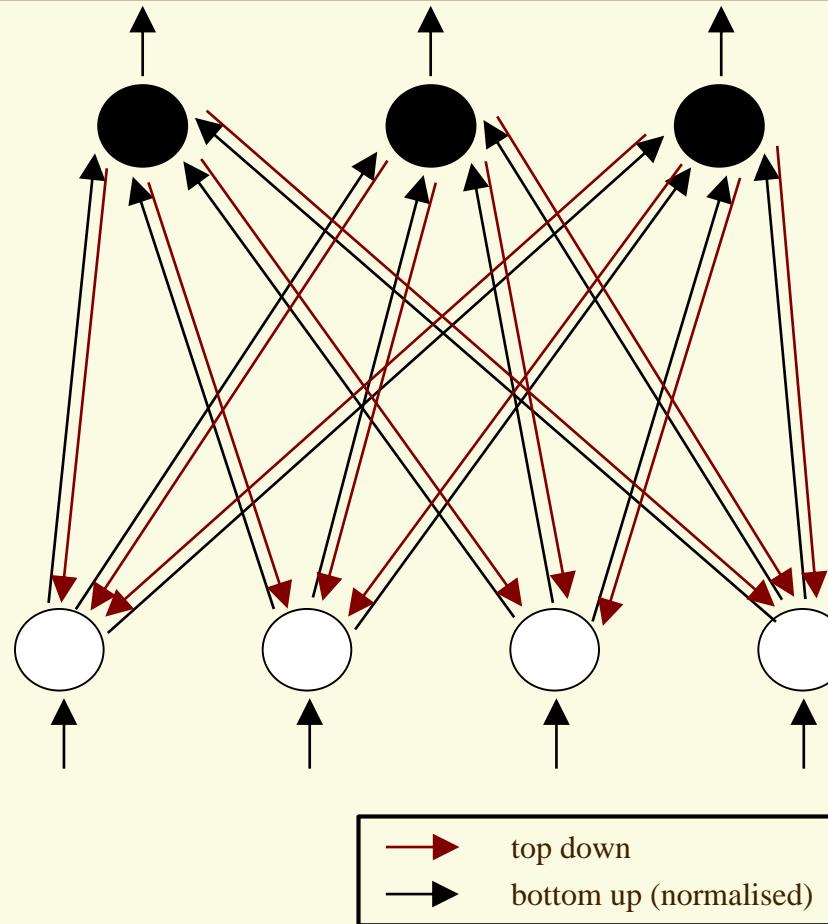
175



ART1 Network

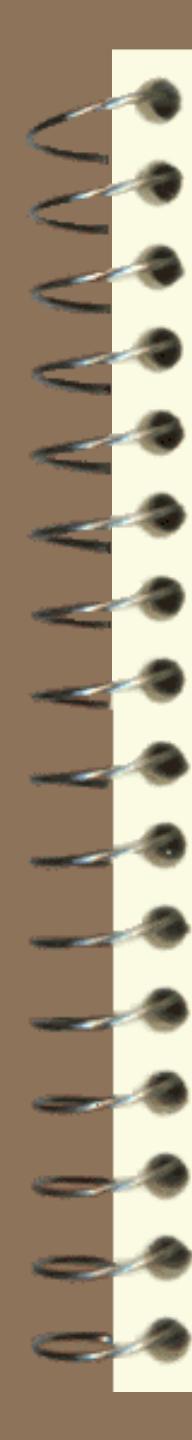
- Invented by Stephen Grossberg at Boston University in 1970's.
- Used to cluster binary data w/ unknown cluster number.
- A two-layer recurrent neural network.
- MAXNET serves as its output layer.
- Bidirectional adaptive connections called bottom-up and top-down connections.

ART1 Architecture



ART1 for Clustering

-
- 1) Initialize weights: $w_{ij}^{td}(0) = 1, w_{ij}^{bu}(0) = \frac{1}{1+n}$
 - 2) Compute net input for an input pattern x^p :
$$u_i^p = \sum_{j=1}^n w_{ij}^{bu}(t)x_j^p, i = 1, 2, \dots, m$$
 - 3) Select the best match using the MAXNET $u_k^p = \max_i\{u_i^p\}$
 - 4) Vigilance test: If $\frac{\sum_{j=1}^n w_{kj}^{td}(t)x_j^p}{\sum_{j=1}^n x_j^p} \geq \rho$, then next; otherwise, disable neuron k and go to step 2).
 - 5) Adapt weights:
$$w_{kj}^{td}(t+1) = w_{kj}^{td}(t)x_j^p, w_{kj}^{bu}(t+1) = \frac{w_{kj}^{td}(t)x_j^p}{0.5 + \sum_{j=1}^n w_{kj}^{td}(t)x_j^p}$$



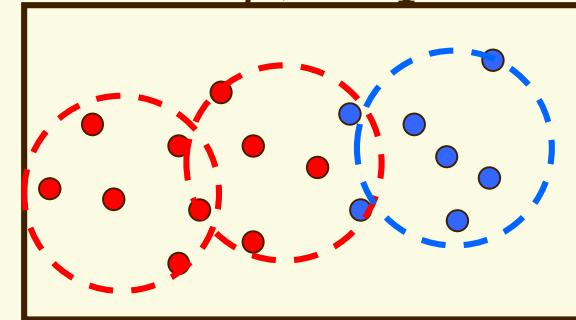
Vigilance Parameter in ART1 Network

- Value ranges between 0 and 1.
- A user-chosen design parameter to control the sensitivity of the clustering.
- The larger its value is, the more homogenous the data are in each cluster.
- Determine in an *ad hoc* way.

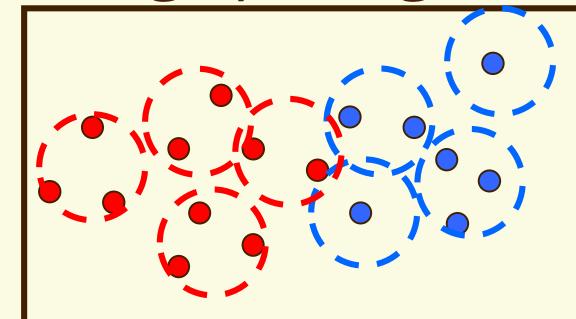
Vigilance Parameter in ART1 Network

- Vigilance sets granularity of clustering
- It defines basin of attraction of each cluster
- Low threshold
 - Large mismatch accepted
 - Few large clusters
- High threshold
 - Small mismatch accepted
 - Many small clusters
 - Higher precision

Small ρ , imprecise



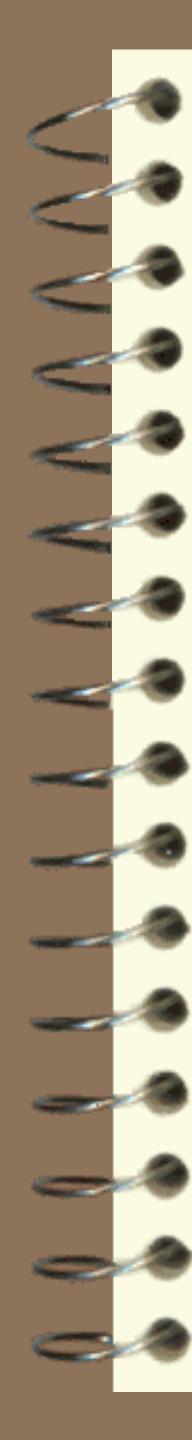
Large ρ , fragmented



Illustrative Example

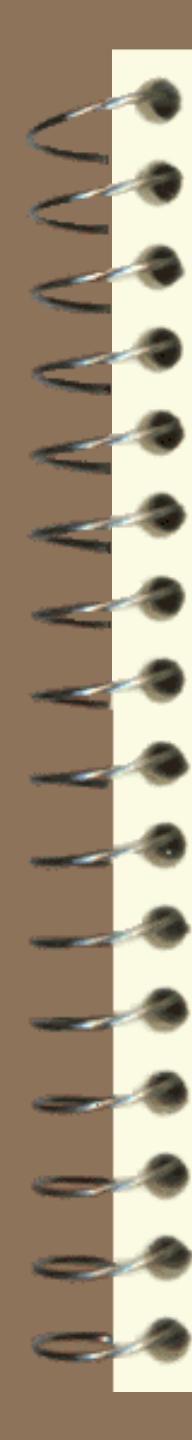
INPUT	EXEMPLARS AFTER EACH INPUT
C	C
E	CE
F	CEF
F	CEF
F	CEFF

Figure 11. An example of the behavior of the Carpenter Grossberg net for letter patterns. Binary input patterns on the left were applied sequentially starting with the upper "C" pattern. Exemplars formed by top-down connection weights after each input was presented are shown at the right.



Hopfield Networks

- Invented by John Hopfield at Princeton University in 1980's.
- Used as associative memories or optimization models.
- Single-layer recurrent neural networks.
- The discrete-time model uses bipolar threshold logic units and the continuous-time model uses unipolar sigmoid activation function.



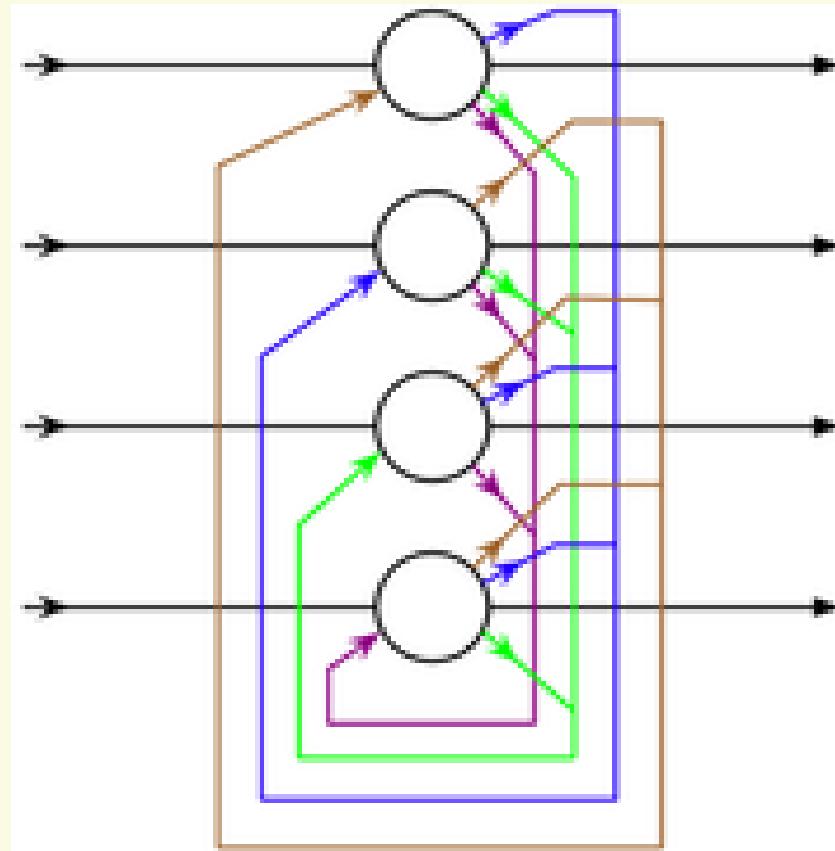
Hopfield Networks

The Hopfield networks are the classical recurrent neural networks.

John Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *PNAS*, USA, vol. 79, 1982.

“As far as public visibility goes, the modern era in neural networks dates from the publication of this paper.” – J. A. Anderson and E. Rosenfeld (1988).

Hopfield Networks



Discrete-Time Hopfield Network

$$u(t+1) = Wv(t) + x$$

$$u_i(t+1) = \sum_j w_{ij} v_j(t) + x_i, \forall i$$

$$v_i(t) = \text{sgn}(u_i(t)) \in \{-1, 1\}$$

Stability Analysis

$$E[v(t)] = -\frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} w_{ij} v_i(t) v_j(t) - \sum_{i=1}^n x_i v_i(t)$$

$$E[v(t+1)] - E[v(t)] = -\frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} w_{ij} v_i(t+1) v_j(t+1) -$$

$$\sum_{i=1}^n x_i v_i(t+1) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} w_{ij} v_i(t) v_j(t) + \sum_{i=1}^n x_i v_i(t) =$$

$$-\frac{1}{2} \sum_{i=1}^n (w_{ik} + w_{ki}) v_i(t+1) v_k(t+1) - x_k v_k(t+1) +$$

$$\frac{1}{2} \sum_{i=1}^n (w_{ik} + w_{ki}) v_i(t) v_k(t) + x_k v_k(t) =$$

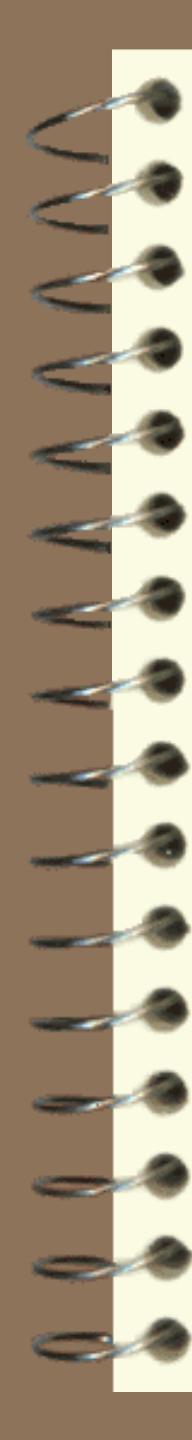
$$-[v_k(t+1) - v_k(t)] [\sum_{i=1}^n w_{ik} v_i(t) + x_k] = -\Delta v_k(t) u_k \leq 0$$

Stability Conditions

 Stability: $\lim_{t \rightarrow \infty} v(t) = \bar{v}$

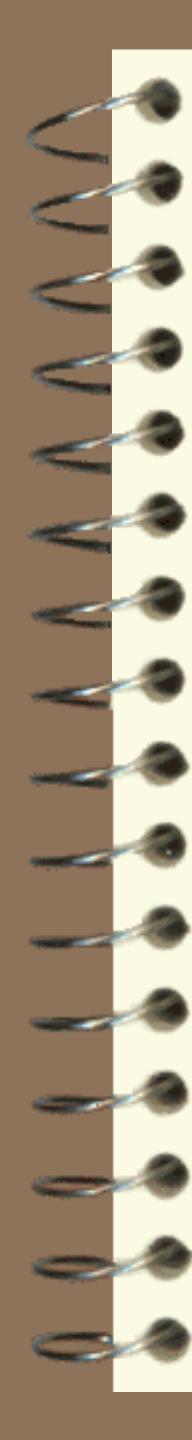
 Sufficient conditions:

1. $w_{ij} = w_{ji}, w_{ii} = 0; i, j = 1, 2, \dots, n$
2. Activation is conducted asynchronously;
i.e., the state updating from $v(t)$ to $v(t+1)$ is
performed for one neuron each iteration.



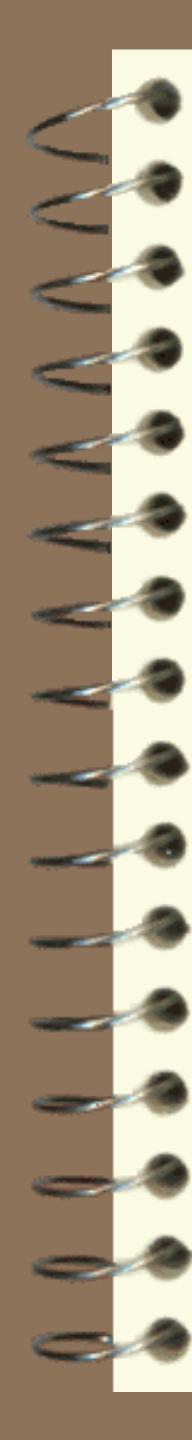
Stability Properties

- If W is symmetric with zero diagonal elements and the activation is conducted asynchronously (i.e., one neuron at one time), then the discrete-time Hopfield network is stable (a sufficient condition).
- If W is symmetric with zero diagonal elements and the activation is conducted synchronously, then the discrete-time Hopfield network is either stable or oscillates in a limit cycle of two states.



Associative Memory

- Learning and memory are the two most important cognitive functions in brain-like intelligence
- As important as learning
- Also known as *content-addressable memory*
- Fundamentally different from the existing computer memory



Associative Memory

Memory is a process of acquiring and storing information such that it will be available when needed.

It is well known that human memory is far more robust and fault tolerant than existing computer memory.

Human Brain and Memory

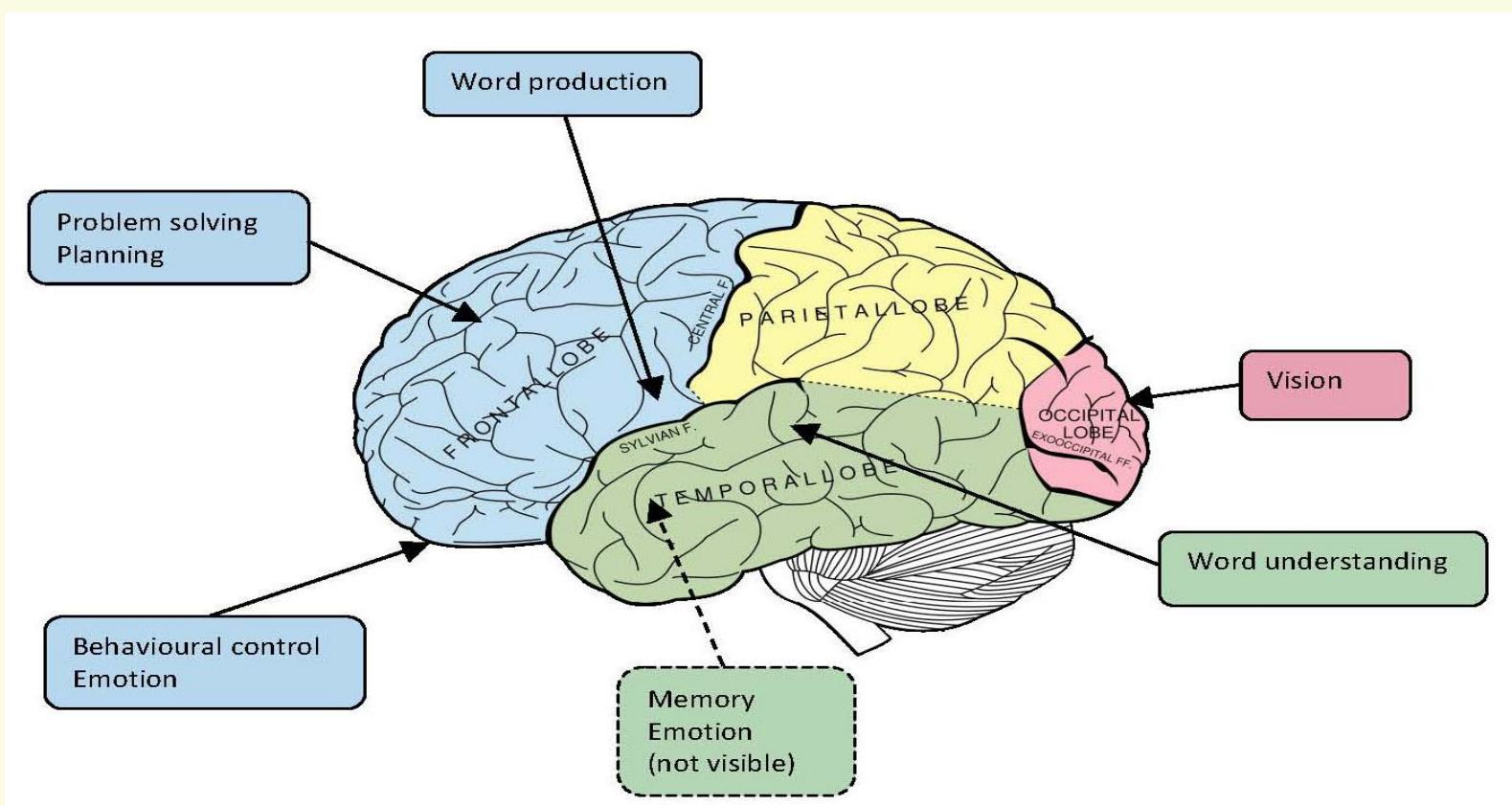
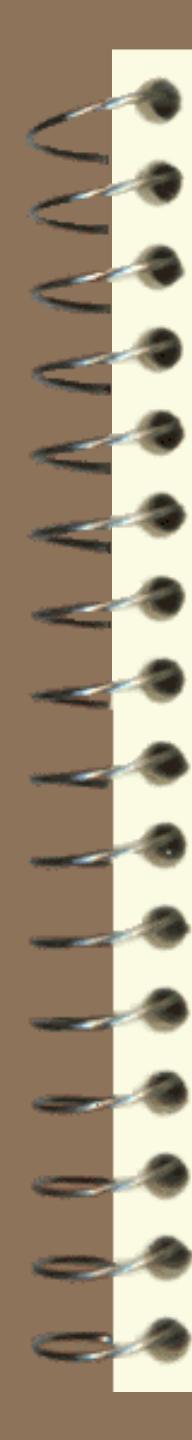


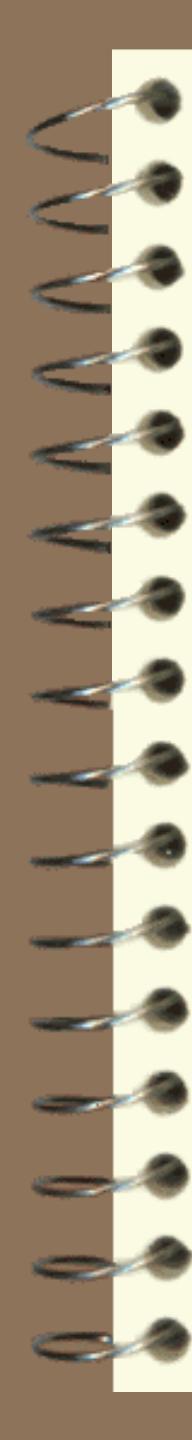
Figure courtesy of the Frontotemporal dementia group (FRONTIER), Prince of Wales Medical Research Institute, Sydney NSW 2031 Australia



Associative Memory

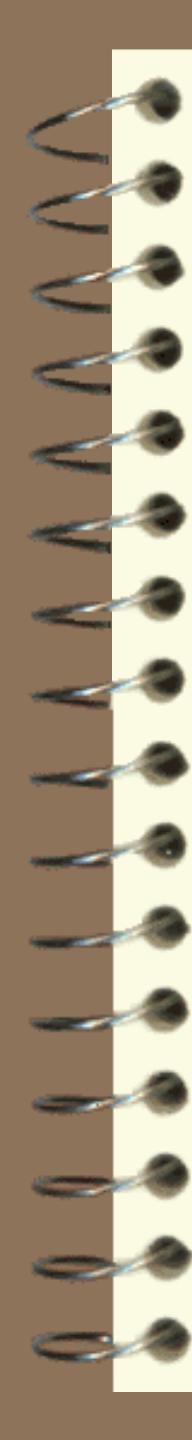
As the original source of human intelligence, the brain has a powerful ability of association.

Associative memories are content-addressable mechanisms for storing prototype patterns such that the stored patterns can be retrieved with the recalling probes (cues).



Associative Memory

- When given a probe (e.g., noisy or corrupted version of a prototype pattern), the retrieval dynamics of an associate memory should converge to an equilibrium representing the prototype pattern.
- In associative memories, the stored patterns are associated with their retrieval probes internally in a robust and fault-tolerant way.

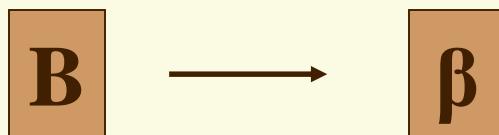


Auto-association vs. Hetero-association

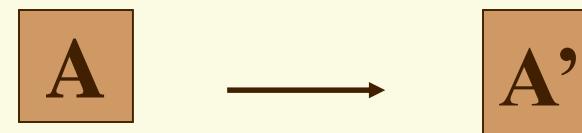
- There are two types of associative memories:
auto-associative and hetero-associative
memories.
- An auto-associative memory retrieves a
previously stored pattern that closely
resembles the recalling probe.
- In a hetero-associative memory, the retrieved
pattern is generally different from the probe in
content or format.

Auto-association vs. Hetero-association

Hetero associative



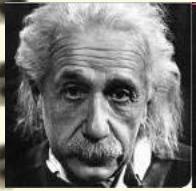
Auto associative



For example,

ID → Name → A

Hetero-Associations



Albert
Einstein



Charles
Kao



Images

Names

⋮

⋮

Input pattern

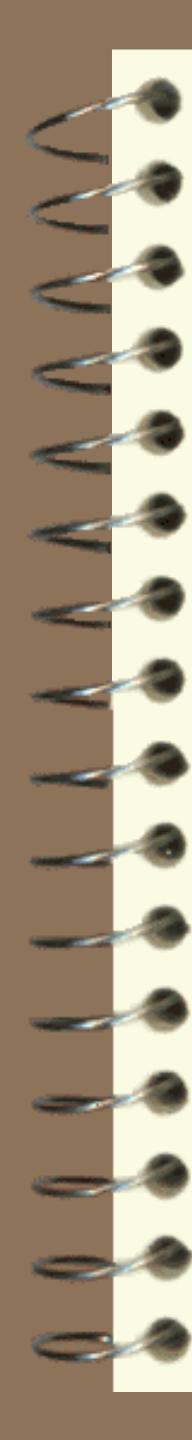
$$\begin{bmatrix} x_1^1 & x_1^2 & x_1^3 & x_1^4 \\ x_2^1 & x_2^2 & x_2^3 & x_2^4 \\ x_3^1 & x_3^2 & x_3^3 & x_3^4 \\ x_4^1 & x_4^2 & x_4^3 & x_4^4 \end{bmatrix}$$



desired output

$$\begin{bmatrix} y_1^1 & y_1^2 & y_1^3 & y_1^4 \\ y_2^1 & y_2^2 & y_2^3 & y_2^4 \\ y_3^1 & y_3^2 & y_3^3 & y_3^4 \\ y_4^1 & y_4^2 & y_4^3 & y_4^4 \end{bmatrix}$$

1. Feed forward neural networks
2. Recurrent neural networks



Memory Processes

 Storage
(Information
encoding)

 Given a set of
prototype patterns
to be memorized,
place them into
the memory
indefinitely.

 Retrieval
(Information
decoding)

 Given any probe (key
or cue), recall the
corresponding
prototype patterns in
the memory.

Discrete-Time Hopfield Network as an Associative Memory

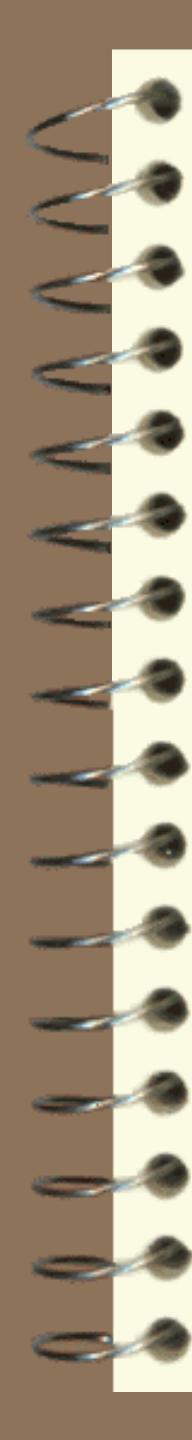
- Storage: Outer product weight matrix

$$w_{ij} = \sum_{p=1}^P s_i^p s_j^p - P\delta_{ij}, W = \sum_{p=1}^P s^p (s^p)^T - PI, s^p \in \{-1,1\}^n$$

- Retrieval (recall):

$$v(0) = s^q, u(1) = Wv(0) = \sum_{p=1}^P s^p (s^p)^T s^q - Ps^q$$

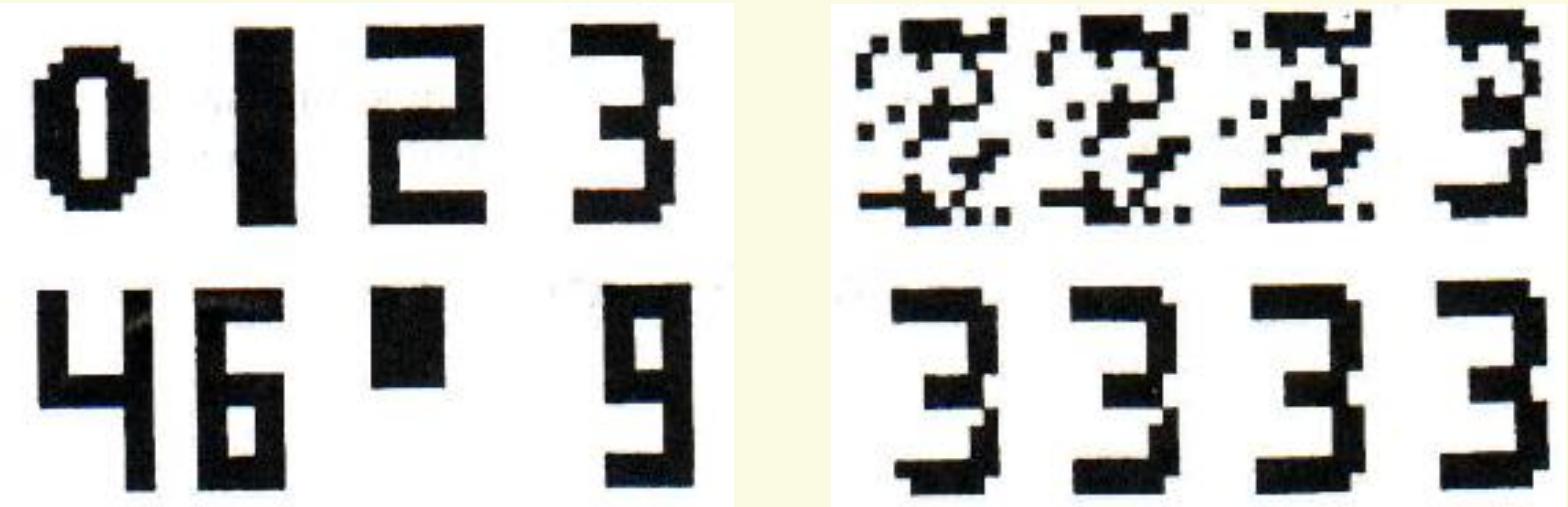
$$= s^q (s^q)^T s^q + \sum_{p \neq q} s^p (s^p)^T s^q - Ps^q$$



Discrete-Time Hopfield Network as an Associative Memory

- If s^p is orthonormal; i.e., $(s^p)^T s^q = 0, \forall p \neq q$. then the second term in recall formula (cross-talk or noise) is zero.
- If $(s^q)^T s^q = \|s^q\|_2^2 = n > P$, then $v(1) = s^q$
- If s^p is not orthonormal, for a small variation of probe patterns, the Hopfield network can still recall the correct patterns.

Illustrative Example

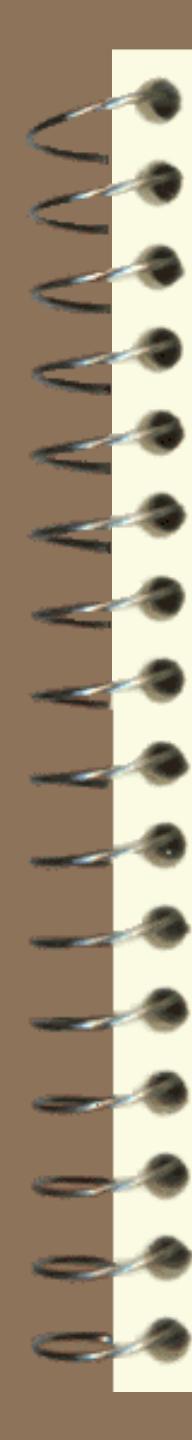


Stored
Patterns
CS5486

Memory
Recall

Limitations

- Very limited capacity: $\frac{n}{2 \log n}$, where n is the memory length
- Many spurious states; e.g., $-s^q$

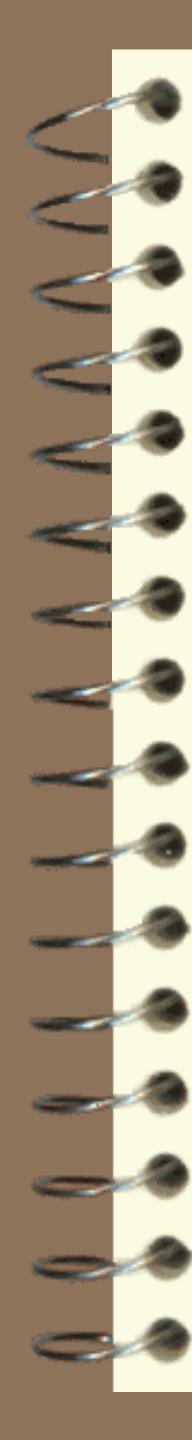


Discrete-Time Hopfield Network as an Optimization Model

- Formulate the energy function according to the objective function and constraints of a given optimization problem.

$$E(v) = -\frac{1}{2} v^T W v - x^T v, v \in \{-1,1\}^n$$

- Form a Hopfield network, then update the states asynchronously until convergence.
- Shortcoming: slow convergence due to asynchrony.



Bidirectional Associative Memories (BAM)

- Also known as hetero-associative memories and resonance networks.
- A generalization of auto-associative memories.
- Proposed by Bart Kosko of University of Southern California in 1988.
- Using bipolar signum activation functions.

BAM Architecture

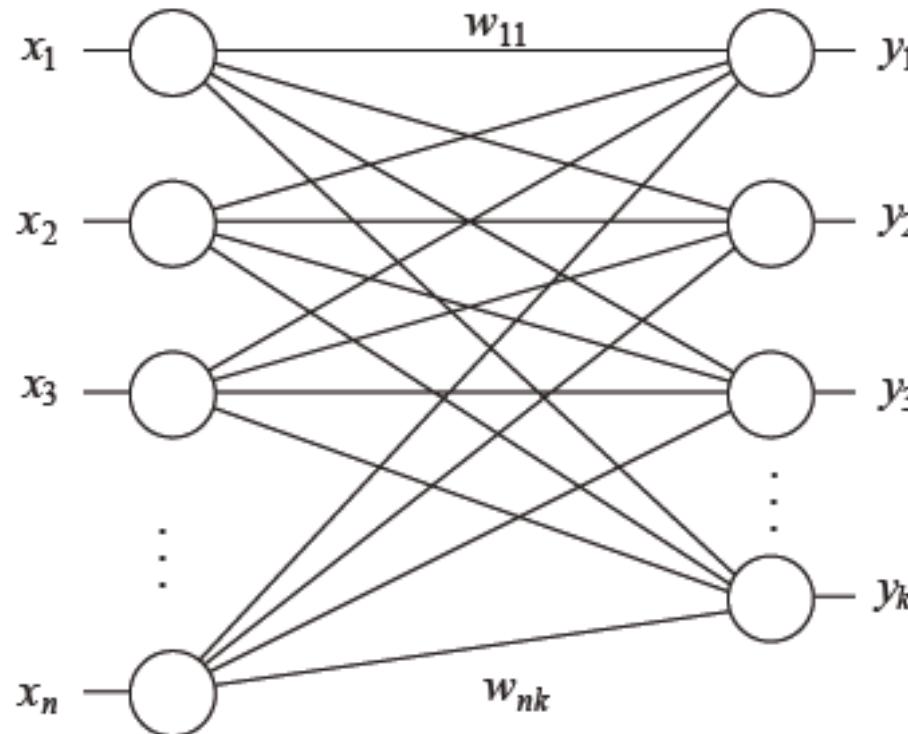


Fig. 13.1. Example of a resonance network (BAM)

Bidirectional Associative Memories (BAM)

$$x \in \mathbb{R}^n, y(t) = \text{sgn}(Wx(t))$$

$$y \in \mathbb{R}^m, x(t+1) = \text{sgn}(W^T y(t))$$

$$W = \sum_{p=1}^P \bar{y}^p (\bar{x}^p)^T$$

$$\text{sgn}(W\bar{x}^q) = \text{sgn}\left(\sum_{p=1}^P \bar{y}^p (\bar{x}^p)^T \bar{x}^q\right)$$

$$= \text{sgn}(\|\bar{y}^q\| \|\bar{x}^q\|_2^2 + \sum_{p \neq q} \bar{y}^p (\bar{x}^p)^T \bar{x}^q) = \bar{y}^q,$$

$$\text{sgn}(W^T \bar{y}^q) = \text{sgn}\left(\sum_{p=1}^P \bar{x}^p (\bar{y}^p)^T \bar{y}^q\right)$$

$$= \text{sgn}(\|\bar{x}^q\| \|\bar{y}^q\|_2^2 + \sum_{p \neq q} \bar{x}^p (\bar{y}^p)^T \bar{y}^q) = \bar{x}^q,$$

Continuous-time Hopfield Network

$$\frac{du}{dt} = -\frac{u}{\rho} + Wv + x$$

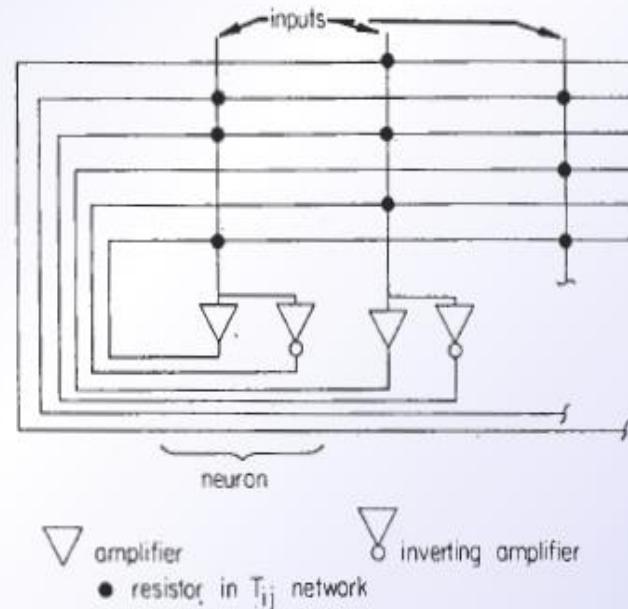
$$\frac{du_i}{dt} = -\frac{u_i}{\rho} + \sum_j w_{ij} v_i + x_i, \forall i$$

$$v_i = \frac{1}{1 + \exp(-u_i)}$$

Continuous-time Hopfield Network

Continuous-time
model [Hopfield,
PNAS, vol. 81,
1984]

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_j W_{ij} v_j + I_i$$
$$v_i = g_i(u_i)$$



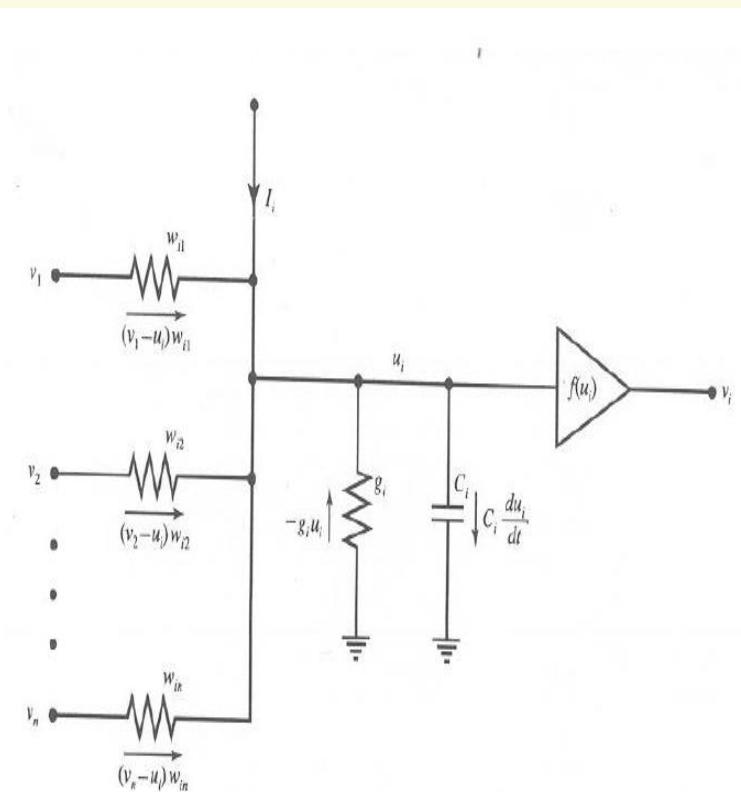
Continuous-time Hopfield Network

$$C_i \frac{du_i}{dt} = \sum_{j=1}^n w_{ij}(v_j - u_i) - u_i g_i + i_i$$

$$C_i \frac{du_i}{dt} = \sum_{j=1}^n w_{ij} v_j - u_i G_i + i_i$$

$$G_i = \sum_{j=1}^n w_{ij} + g_i$$

$$L(v) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} v_i v_j + \sum_{i=1}^n G_i \int_0^{v_i} f_i^{-1}(z) dz - \sum_{i=1}^n i_i v_i$$



Stability Analysis

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij} v_i v_j + \sum_j \frac{1}{\rho} \int_0^{v_j} f^{-1}(v) dv - \sum_j x_j v_j$$

$$\frac{dE}{dt} = \sum_i \frac{\partial E}{\partial v_i} \frac{dv_i}{dt} = -\sum_i \left(\sum_j w_{ij} v_j - \frac{u_i}{\rho} + x_i \right) \frac{dv_i}{dt} =$$

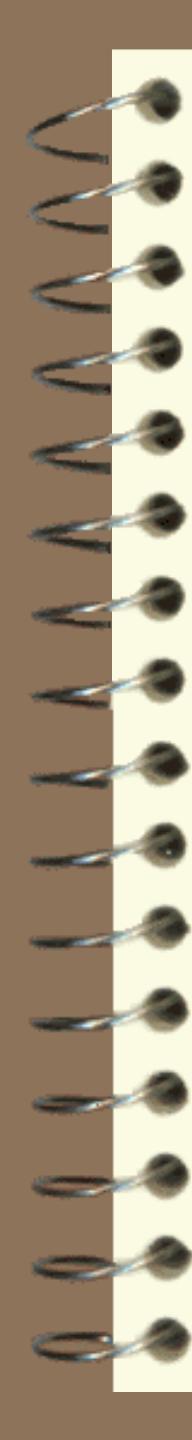
$$-\sum_i \frac{du_i}{dt} \frac{dv_i}{dt} = -\sum_i \frac{df(u_i)}{du_i} \left(\frac{du_i}{dt} \right)^2 < 0, \forall \frac{du_i}{dt} \neq 0.$$

High Gain Unipolar Sigmoid Activation Function

$$f(u) = \frac{1}{1 + \exp(-\xi u)}, u = f^{-1}(v) = \frac{1}{\xi} \ln \frac{v}{1-v}$$

$$\int_0^v f^{-1}(x) dx = \frac{1}{\xi} [v \ln v + (1-v) \ln(1-v)]$$

If $\xi \rightarrow +\infty$, then $\int_0^v f^{-1}(x) dx \rightarrow 0$



Continuous-Time Hopfield Network as an Optimization Model

- Formulate the energy function according to the objective function and constraints of a given optimization problem.

$$E(v) = -\frac{1}{2} v^T W v - x^T v, v \in [0,1]^n$$

- Synthesize a continuous-time Hopfield network, then an equilibrium state is a local minimum of the energy function. .

Traveling Salesman Problem

Find a complete route by visiting each city once and only once.

Checking out all possible routes: $(N - 1)!/2$

For example, $N = 30 \Rightarrow 4.4 \times 10^{30}$ routes

If we compute 10^{12} routes per second \Rightarrow we need 10^{18} seconds or 31,709,791,984 years!

In most practical problems $N \gg 30$

A continuous Hopfield network can compute a good solution to the TSP in a parallel and distributed manner.

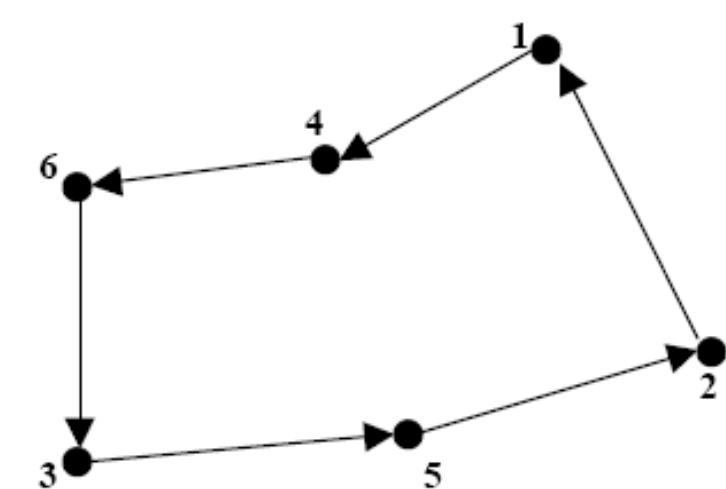
Vertex Path Representation

$$x_{ij} = \begin{cases} 1, & \text{if the } j\text{th vertex is the } i\text{th stop in the path;} \\ 0, & \text{otherwise.} \end{cases}$$

The Hopfield network approach to the TSP involves arranging the neurons in such a way that they represent the decision variable for an N -city problem we would require N^2 neurons

N of them will be turned ON with the remainder turned OFF

Vertex Path Representation



		Route Order					
		1	2	3	4	5	6
C	1	1	0	0	0	0	0
C	2	0	0	0	0	0	1
I	3	0	0	0	1	0	0
T	4	0	1	0	0	0	0
Y	5	0	0	0	0	1	0
	6	0	0	1	0	0	0

Objective Function and Constraints

Constraints (permutation matrix):

- One neuron should be on in each row
- One neuron should be on in each column

Objective function:

- Total distance should be minimized

Determine the network weights and bias so that the Lyapunov function is minimized when constraints are met and objective function is minimum

Objective Function to be Minimized

V_{ia} : binary state variable of the neuron in position (i,a) in the matrix or 2D array

d_{ij} : distance between city i and city j

Total distance of a route:

$$L = \frac{1}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1})$$

Lyapunov Function to be Minimized

$$L = \frac{D}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1}) + \frac{A}{2} \sum_{i,a,b}^{a \neq b} V_{ia} V_{ib} + \frac{B}{2} \sum_{i,j,a}^{i \neq j} V_{ia} V_{ja} + \frac{C}{2} [\sum_{i,a} V_{ia} - N]^2$$

: $\frac{D}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1})$: for minimizing tour length

$\frac{A}{2} \sum_{i,a,b}^{a \neq b} V_{ia} V_{ib}$: is non-zero if more than one neuron is ON in each row

$\frac{B}{2} \sum_{i,j,a}^{i \neq j} V_{ia} V_{ja}$: is non-zero if more than one neuron is ON in each column

$\frac{C}{2} [\sum_{i,a} V_{ia} - N]^2$: ensure that there is a total of N neurons ON

Constructing the Hopfield Network

We should select weights and currents so that two following equations become equal

$$L = \frac{D}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1}) + \frac{A}{2} \sum_{i,a,b}^{a \neq b} V_{ia} V_{ib} + \frac{B}{2} \sum_{i,j,a}^{i \neq j} V_{ia} V_{ja} + \frac{C}{2} \left[\sum_{i,a} V_{ia} - N \right]^2$$

$$L(v) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} v_i v_j - \sum_{i=1}^n i_i v_i$$

First we make output voltages double subscripts:

$$L(v) = -\frac{1}{2} \sum_{i,j,a,b}^{CS5486} w_{ia,jb} V_{ia} V_{jb} - \sum_{i,a} i_{ia} V_{ia}$$

Constructing the Hopfield Network

Note first that $W_{ia,jb}$ multiply the second-order terms $V_{ia}V_{jb}$ and i_{ia} multiply first-order terms V_{ia}

.

First order terms should be equal:

$$-\sum_{i,a} i_{ia} V_{ia} = -CN \sum_{i,a} V_{ia}$$

$$i_{ia} = CN \text{ for all } i,a$$

Constructing the Hopfield Network

Let's treat the four sets of second-order terms separately

$$w_{ia, jb} = w_{ia, jb}^A + w_{ia, jb}^B + w_{ia, jb}^C + w_{ia, jb}^D$$

The second-order C terms are given by

$$\frac{C}{2} \left[\sum_{i,a} V_{ia} \right]^2 = \frac{C}{2} \left[\sum_{i,a} V_{ia}^2 + 2 \sum_{ia \neq jb} V_{ia} V_{jb} \right]$$

→ $w_{ia, jb}^C = -C$

Constructing the Hopfield Network

If $w_{ia, jb}^A = -A(1 - \delta_{ab})\delta_{ij}$ then

corresponding term is $\frac{A}{2} \sum_{i,a,b}^{a \neq b} V_{ia}V_{ib}$

where Kronecker delta function

$$\delta_{ij} = 1 \text{ if } i = j \text{ else } 0$$

Similarly, $w_{ia, jb}^B = -B(1 - \delta_{ij})\delta_{ab}$

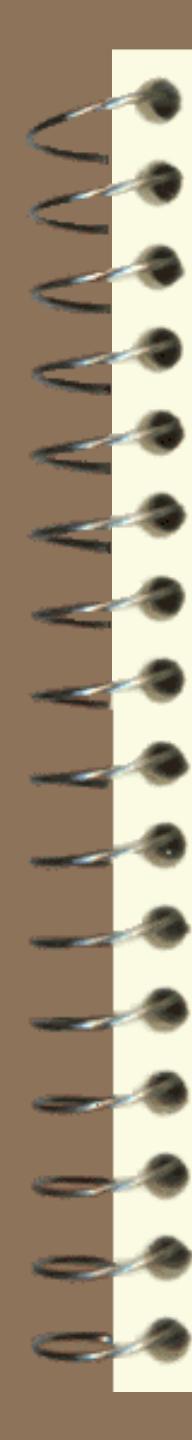
Constructing the Hopfield Network

D term contributes an amount d_{ij} to the Lyapunov function only when $V_{jb} = V_{j,a+1}$ or $V_{jb} = V_{j,a-1}$
So

$$w_{ia, jb}^D = -D d_{ij} (1 - \delta_{ij}) (\delta_{a-1, b} + \delta_{a+1, b})$$

Bringing together all four components of the weights, we have

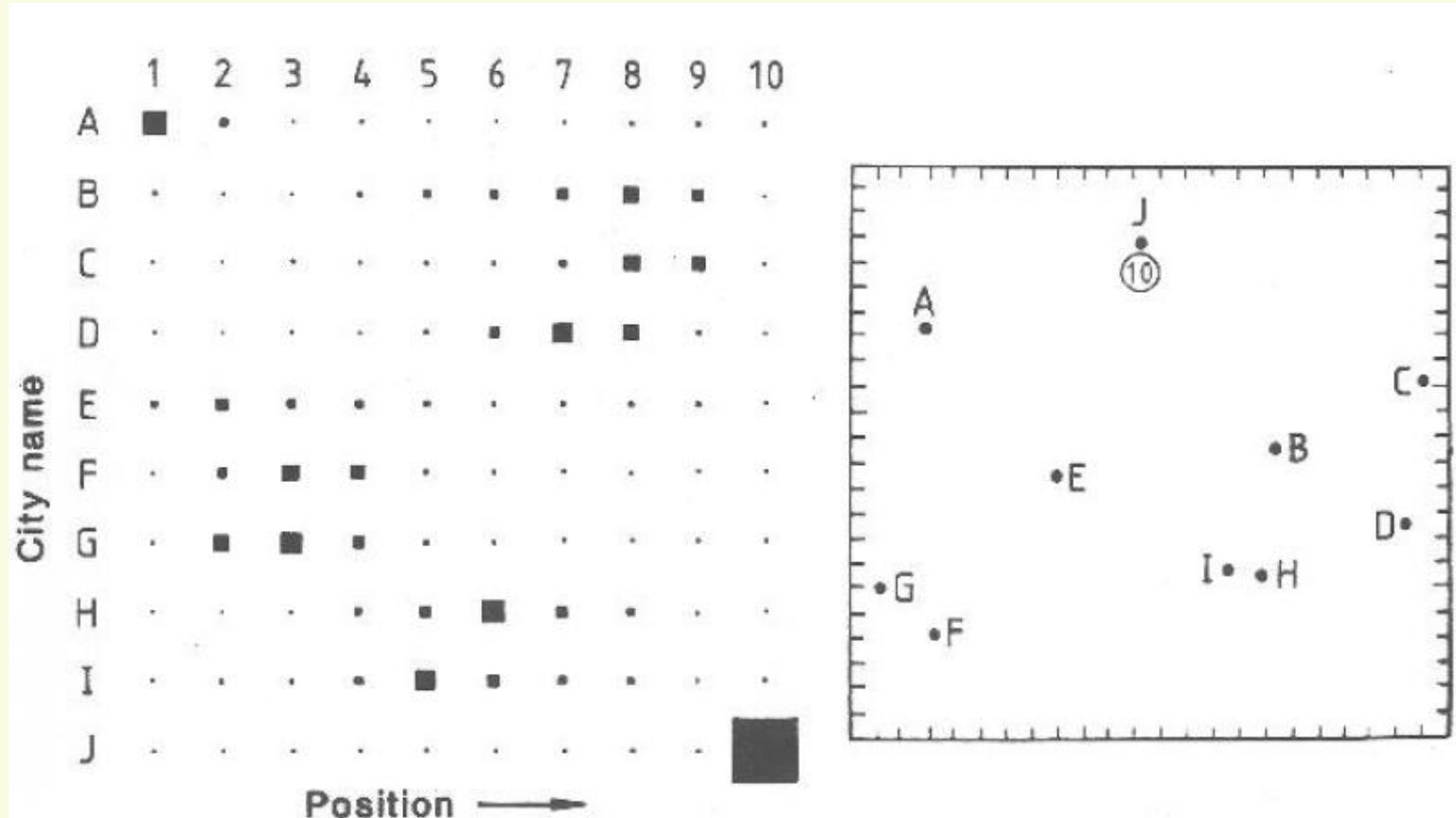
$$w_{ia, jb} = -D d_{ij} (1 - \delta_{ij}) (\delta_{a-1, b} + \delta_{a+1, b}) - A(1 - \delta_{ab})\delta_{ij} - B(1 - \delta_{ij})\delta_{ab} - C$$



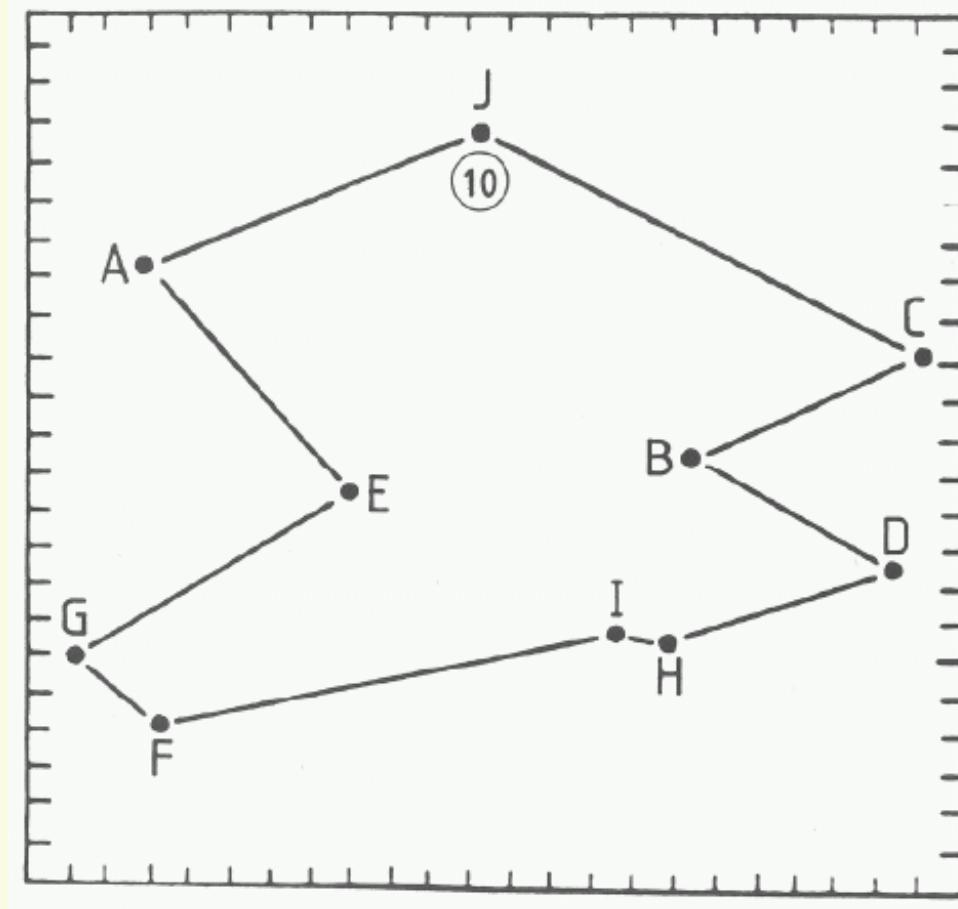
Parameter Selection

- ❖ Appropriate values for the parameters A, B, C, D and ξ must be determined
- ❖ Tank and Hopfield used $A = B = D = 250$, $C = 1000$ and $\xi = 50$
- ❖ Tank and Hopfield applied it to random 10-city maps and found that, overall, in about 50% of cases, the method found the optimum route from among the 181,440 distinct paths.

Ten-City Example



Optimal Route



Local Minima

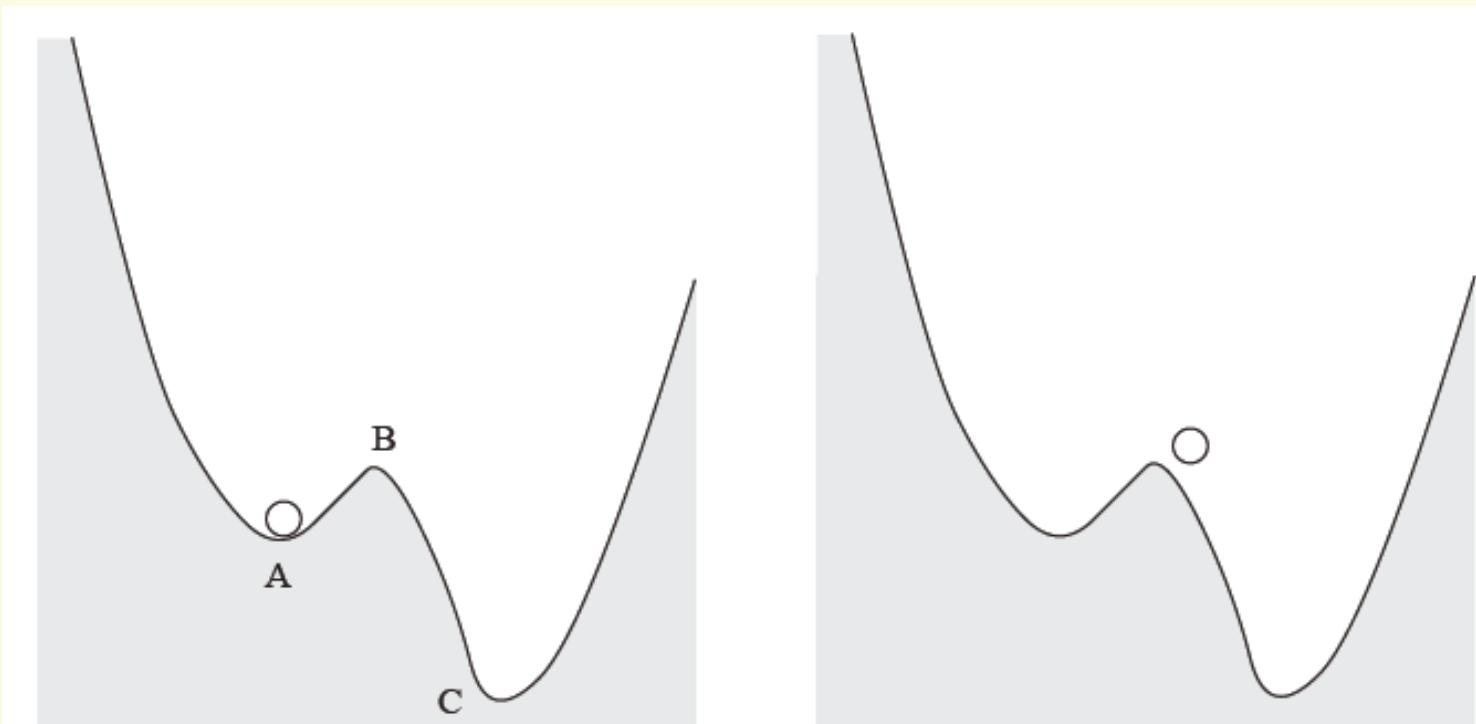
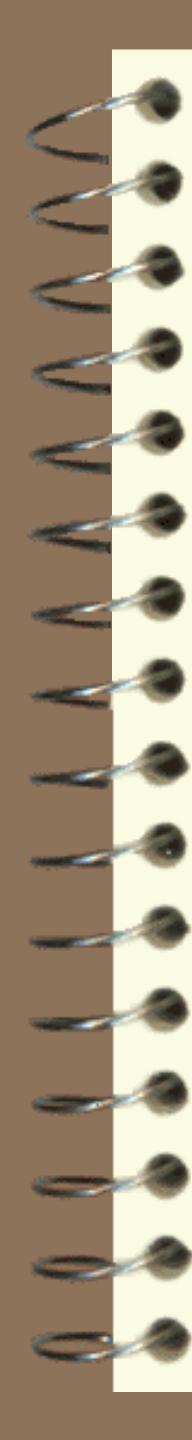


Fig. 14.1. The effect of thermal noise



Simulated Annealing

- Annealing is a metallurgical process in which a material is heated and then slowly brought to a lower temperature to let molecules to assume optimal positions.
- Simulated annealing simulates the physical annealing process mathematically for global optimization of nonconvex objective function.

Updating Probability

$$P_{\Delta E} = \exp\left(-\frac{\Delta E}{T}\right), \text{ if } \Delta E > 0$$

$$P_{\Delta E} = 1, \text{ if } \Delta E \leq 0$$

where $T \geq 0, \Delta T < 0, \lim_{t \rightarrow \infty} T = 0$

The tangent of the probability function intersects with the horizontal axis at T

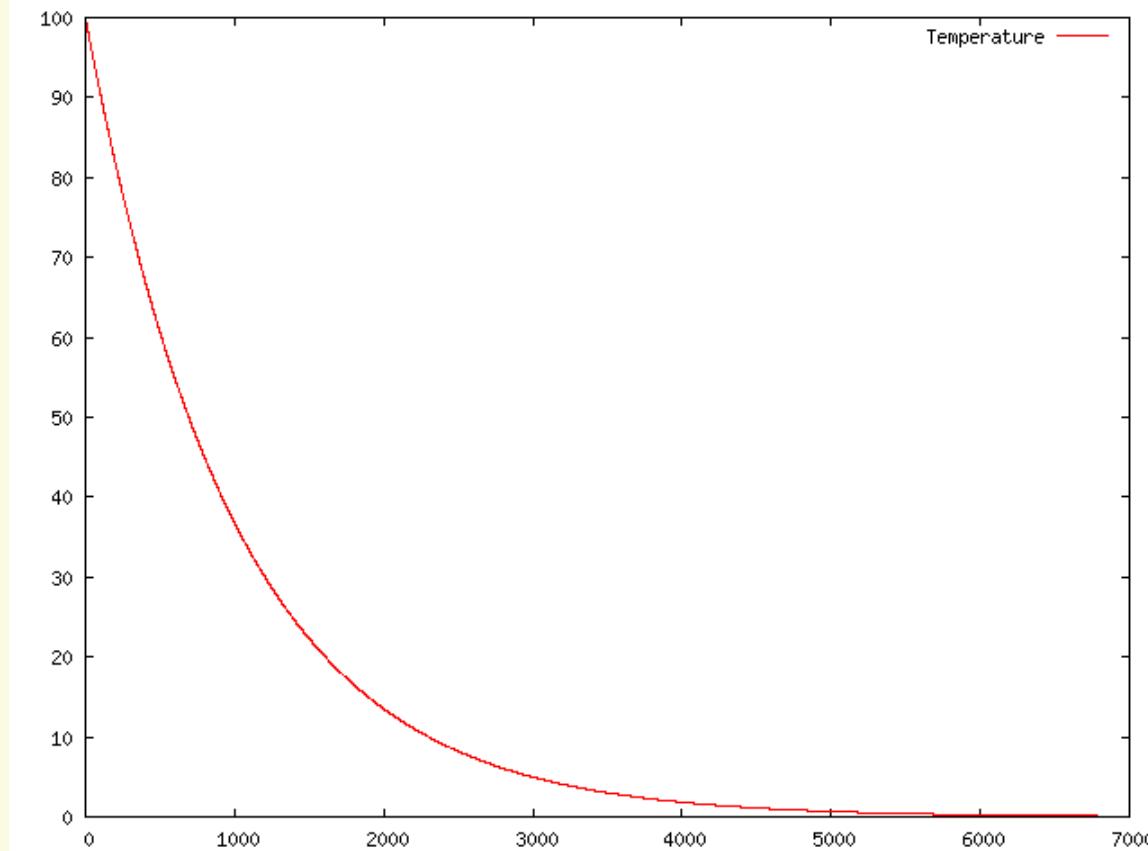
Updating Probability

$$P_{\Delta E} = \frac{1}{1 + \exp\left(\frac{\Delta E}{T}\right)}$$

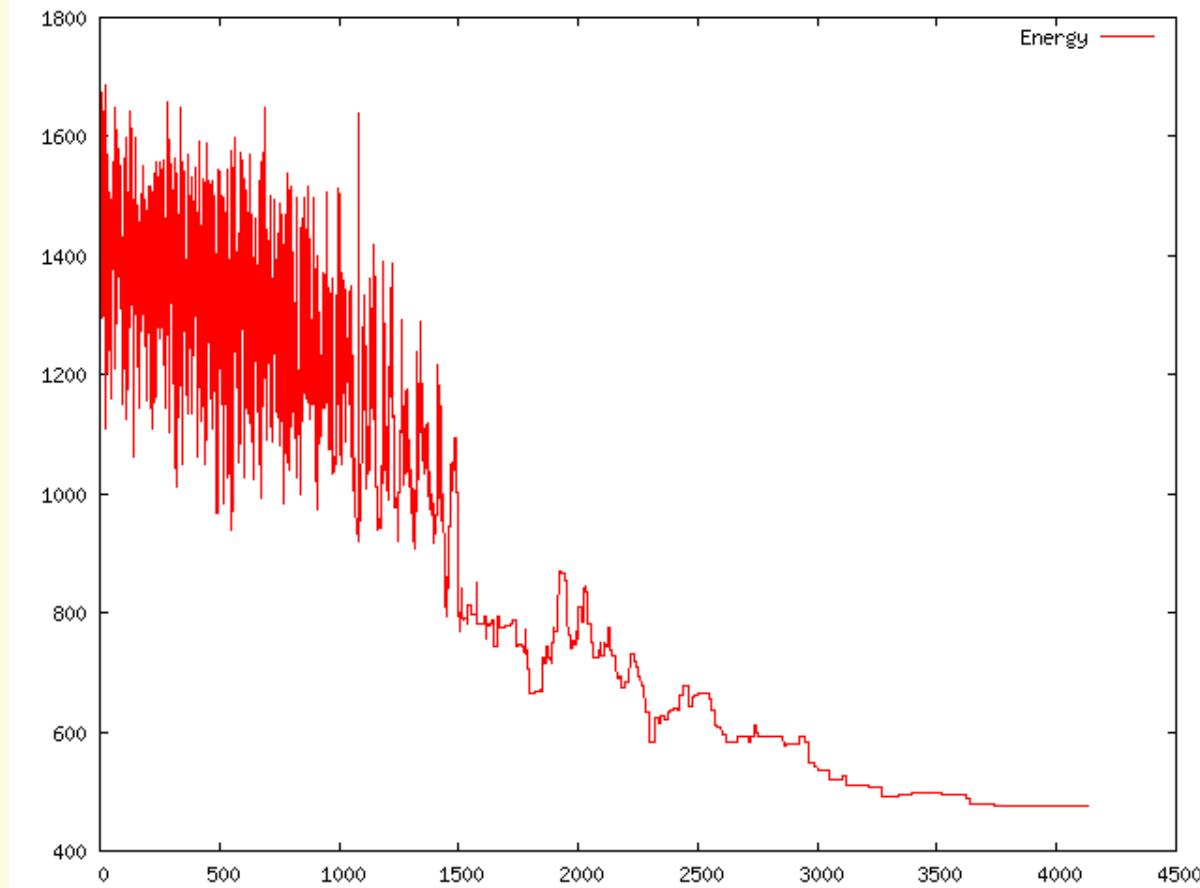
where $T \geq 0, \Delta T < 0, \lim_{t \rightarrow \infty} T = 0$

The tangent of the probability function intersects with the horizontal axis at $2T$.

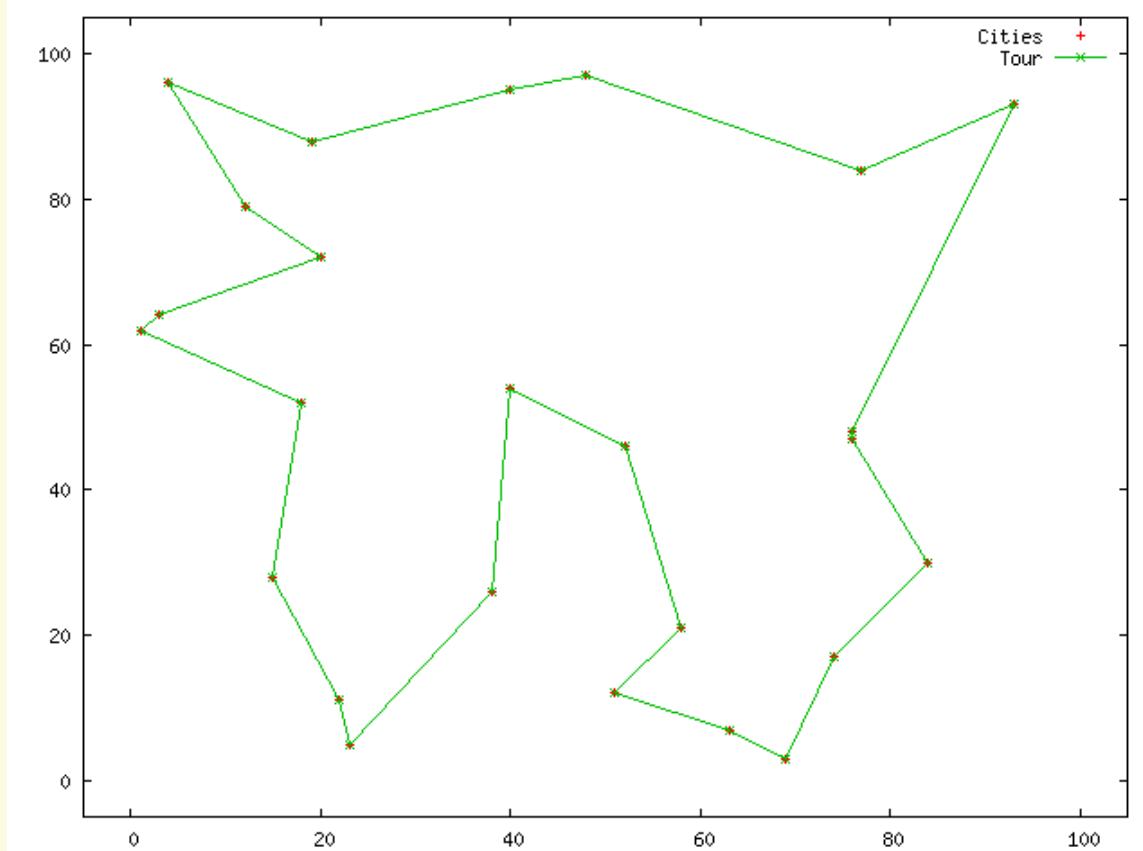
Decreasing Temperature

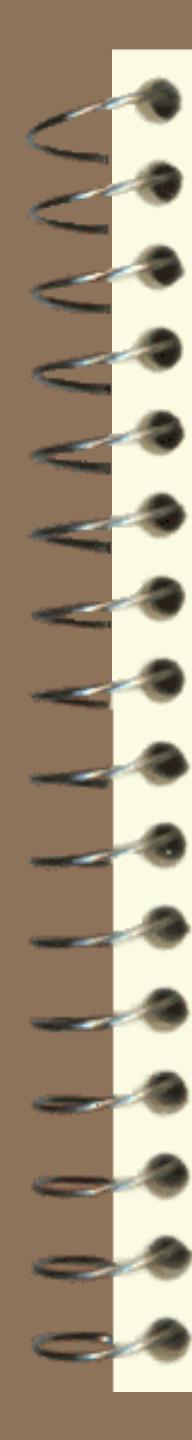


Descending Energy



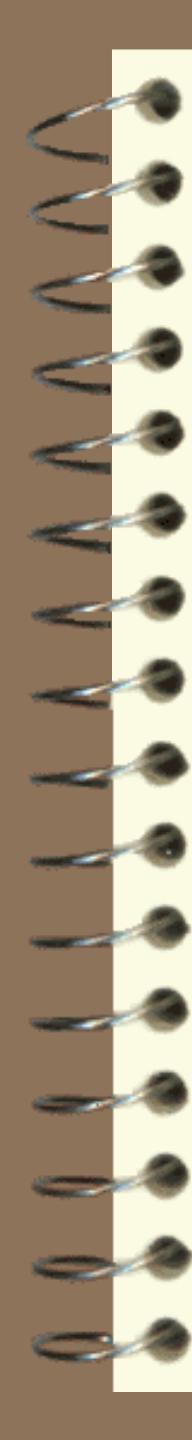
Sample TSP Solution





Characteristics of Simulated Annealing

- The higher the temperature, the higher the probability of an energy increase.
- As the temperature approaches to zero, the simulated annealing procedure becomes an iterative improvement one.
- The temperature parameter has to be lower gradually to avoid prematurity.



Boltzmann Machine

- A stochastic recurrent neural network invented by G. Hinton (Univ. of Toronto) and T. Sejnowski (Salk Institute) in 1983.
- It has binary state variables $\{-1, 1\}^n$ with a probabilistic activation function.
- A parallel implementation of simulated annealing procedure.
- It can be seen as a stochastic, generative counterpart of the Hopfield networks.

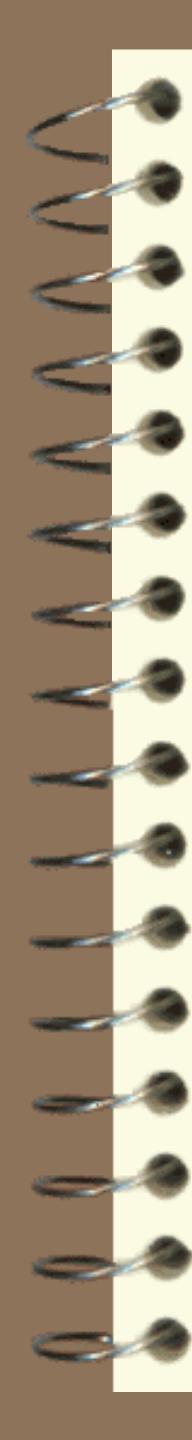
Boltzmann Machine

$$v_i \in \{-1,1\}^n, u_i = \sum_j w_{ij} v_j + x_i, w_{ij} = w_{ji}, E(v) = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} v_i v_j - \sum_i x_i v_i$$

$$\Delta E(v_i) := E(-v_i) - E(v_i) = \frac{\partial E}{\partial v_i} \Delta v_i = (\sum_j w_{ij} v_j + x_i) 2v_i = 2u_i v_i$$

$$P(-v_i \rightarrow v_i) = \frac{1}{1 + \exp(\frac{\Delta E(v_i)}{T})} = \frac{1}{1 + \exp(-\frac{2u_i v_i}{T})}$$

$$P(v_i = -1 \rightarrow 1) = \frac{1}{1 + \exp(\frac{2u_i}{T})}, P(v_i = 1 \rightarrow -1) = \frac{1}{1 + \exp(-\frac{2u_i}{T})}$$



Mean Field Annealing Network

- A deterministic recurrent neural network.
- Based on mean-field theory.
- Continuous state variables on $[-1, 1]^n$.
- use a bipolar sigmoid activation function.
- Use a gradual decreasing temperature parameter like simulated annealing.
- Used for combinatorial optimization.

Mean Field Annealing Network

$$u_i = \sum_j w_{ij} v_j + x_i, v_i = \frac{1 - \exp(-\frac{2u_i}{T})}{1 + \exp(-\frac{2u_i}{T})} = \tanh(\frac{u_i}{T})$$

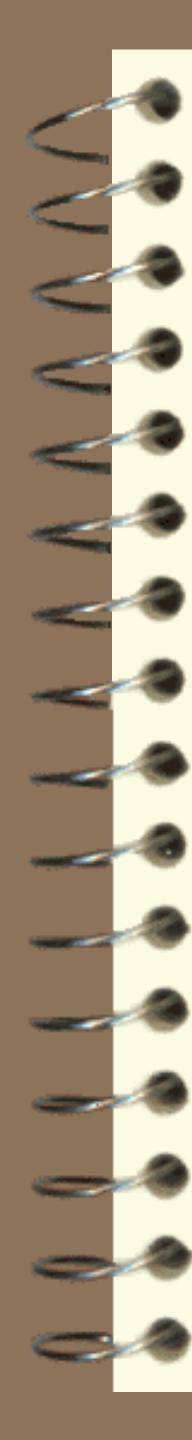
$$T \geq 0, \frac{dT}{dt} < 0, \lim_{t \rightarrow \infty} T = 0. \text{ As } T \rightarrow 0, v_i \in \{-1, 1\}$$

$$E(v) = -\frac{1}{2} \sum_i \sum_j w_{ij} v_i v_j - \sum_i x_i v_i, v_i \in [-1, 1],$$

$$\text{Exp}(v_i) = P(v_i = 1) - P(v_i = -1) = P(v_i = 1) - [1 - P(v_i = 1)] =$$

$$2P(v_i = 1) - 1 = \frac{2}{1 + \exp(-\frac{2u_i}{T})} - 1 = \frac{1 - \exp(-\frac{2u_i}{T})}{1 + \exp(-\frac{2u_i}{T})} = \tanh(\frac{u_i}{T})$$

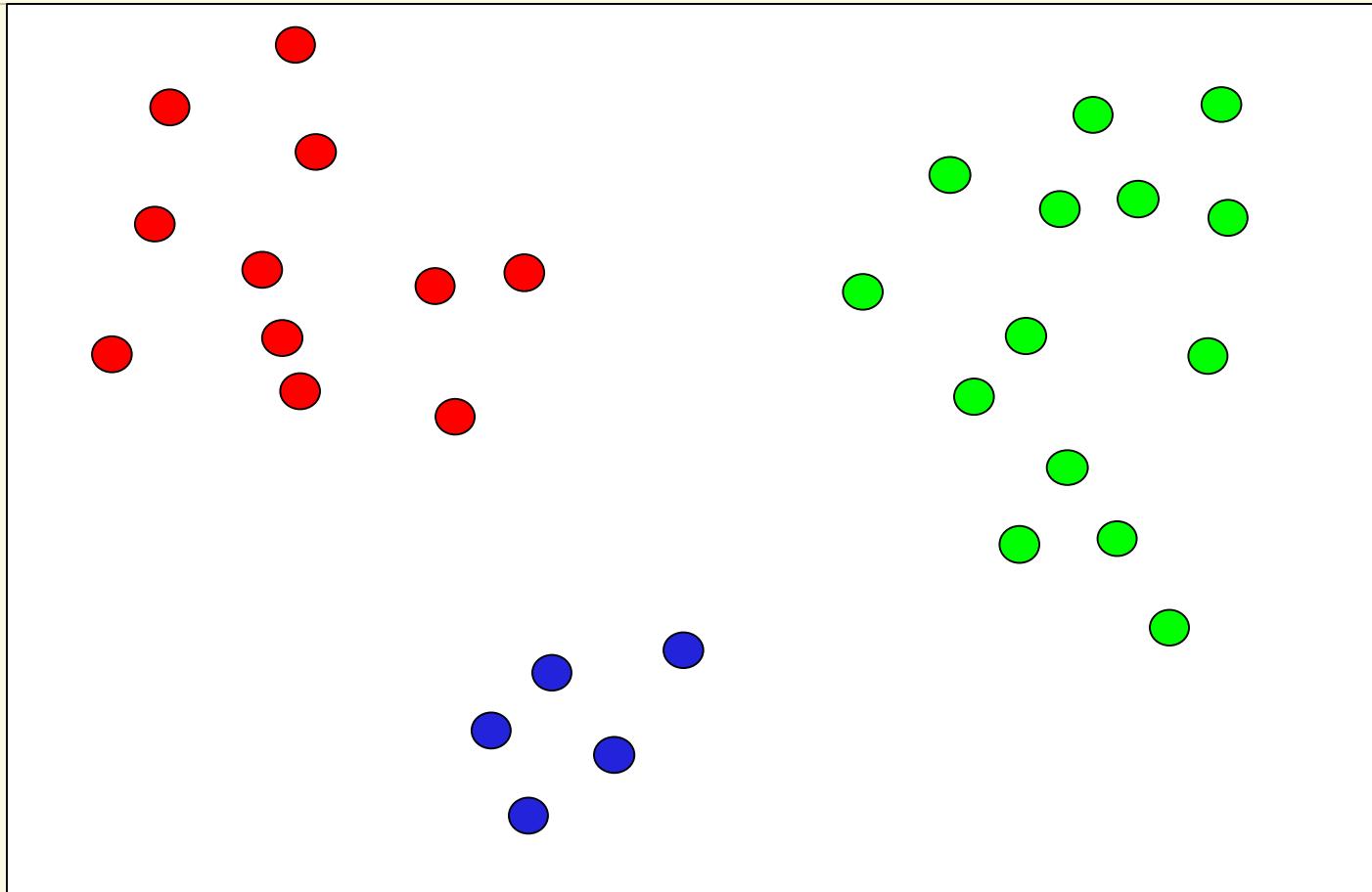
$$v_i = \tanh(-\frac{1}{T} \frac{\partial E(v)}{\partial v_i})$$



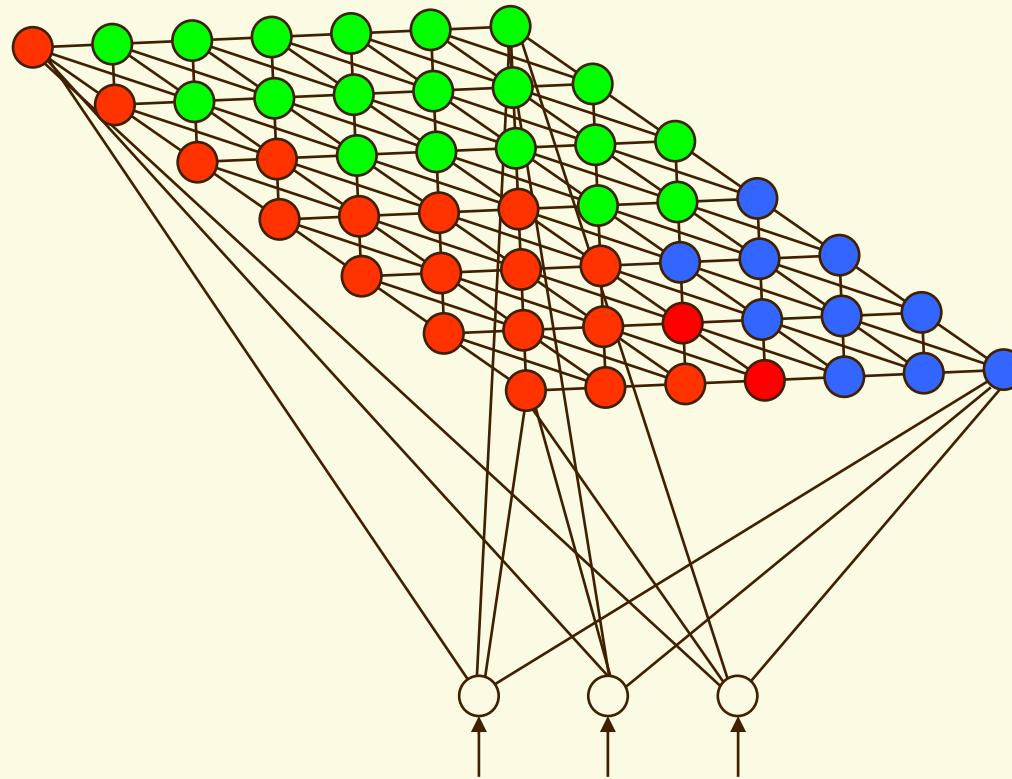
Self-Organizing Maps (SOMs)

- Developed by Prof. T. Kohonen at Helsinki University of Technology in Finland in 1970's.
- A single-layer network with a winner-take-all layer using a unsupervised learning algorithm.
- Formation of topographic map through self-organization.
- Map high-dimensional data to one or two dimensional feature maps.

Data Clusters



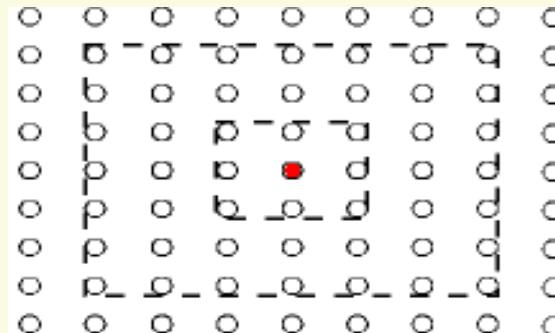
SOM Architecture



Kohonen's Learning Algorithm

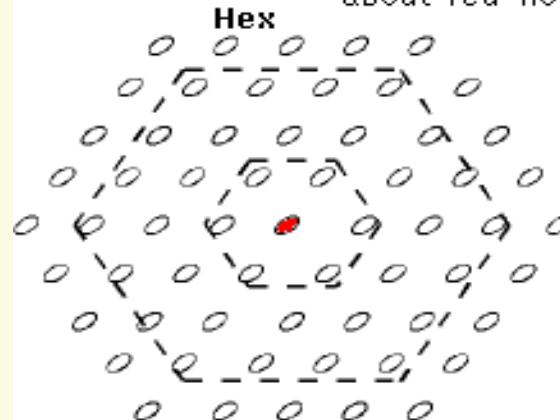
1. (Initialization) Randomize $w_{ij}(0)$ for $i = 1, 2, \dots, n; j = 1, 2, \dots, m; p = 1, t = 0$.
2. (Distance) for datum x^p , $d_j = \sum_i^n [x_i^p - w_{ij}(t)]^2$
3. (Minimization) Find k such that
$$d_k = \min_j d_j$$
4. (Adaptation)
$$\forall j \in N_k(t), i = 1, 2, \dots, n, \Delta w_{ij}(t) = \eta(t)[x_i^p - w_{ij}(t)]$$
$$0 \leq \eta < 1, d\eta/dt < 0, p \leftarrow p + 1, \text{ goto Distance.}$$

Neighborhood in SOMs



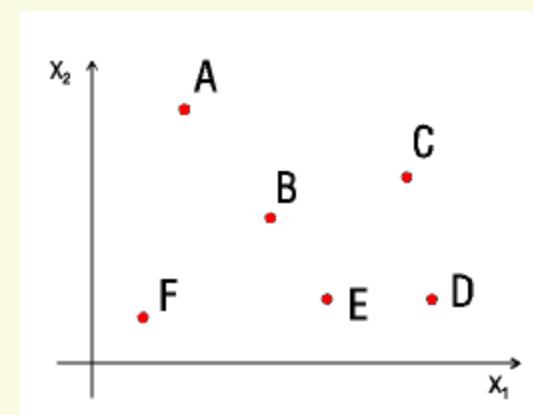
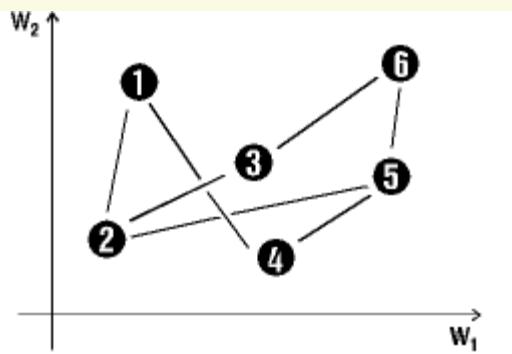
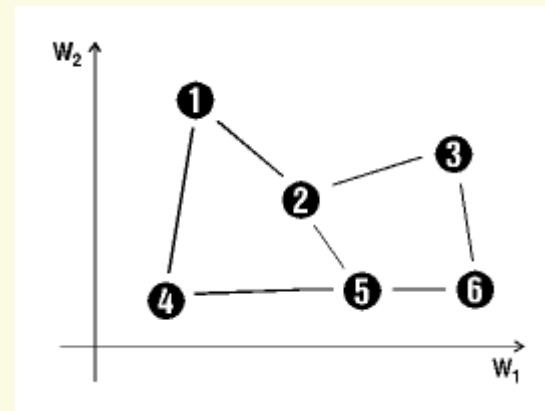
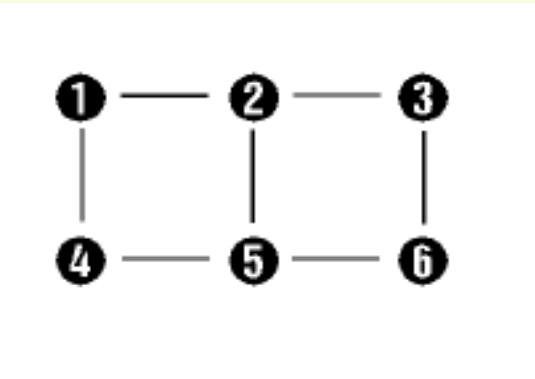
Square

Neighbourhoods of distance 1 & 2
about 'red' node



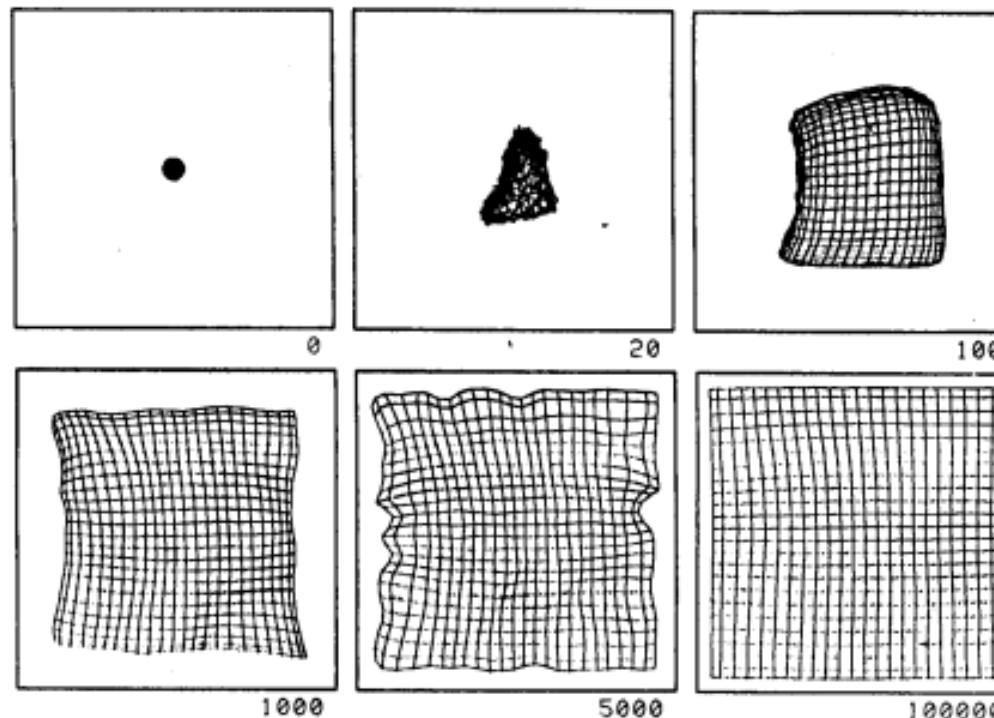
Hex

A Simple Example



Kohonen's Example

5. Self-Organizing Feature Maps



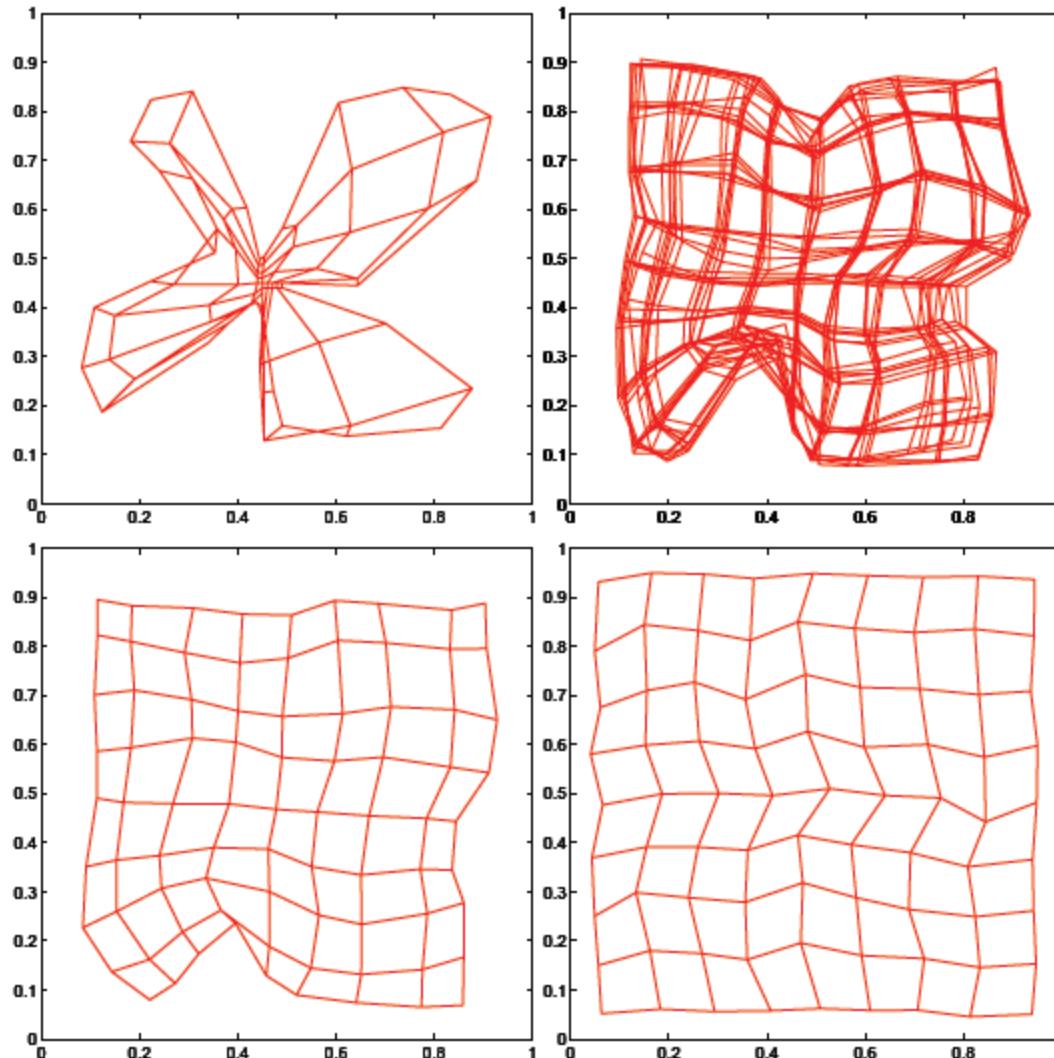
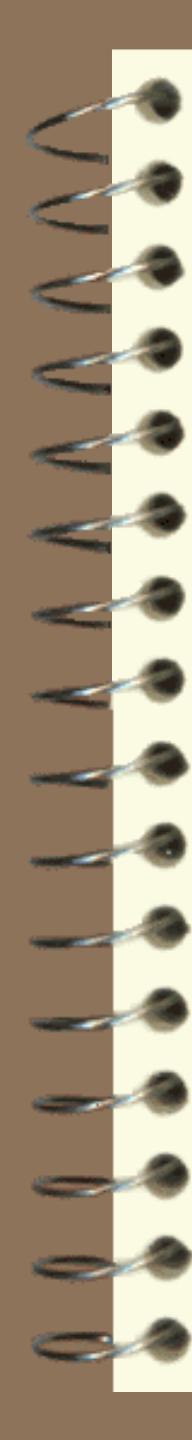


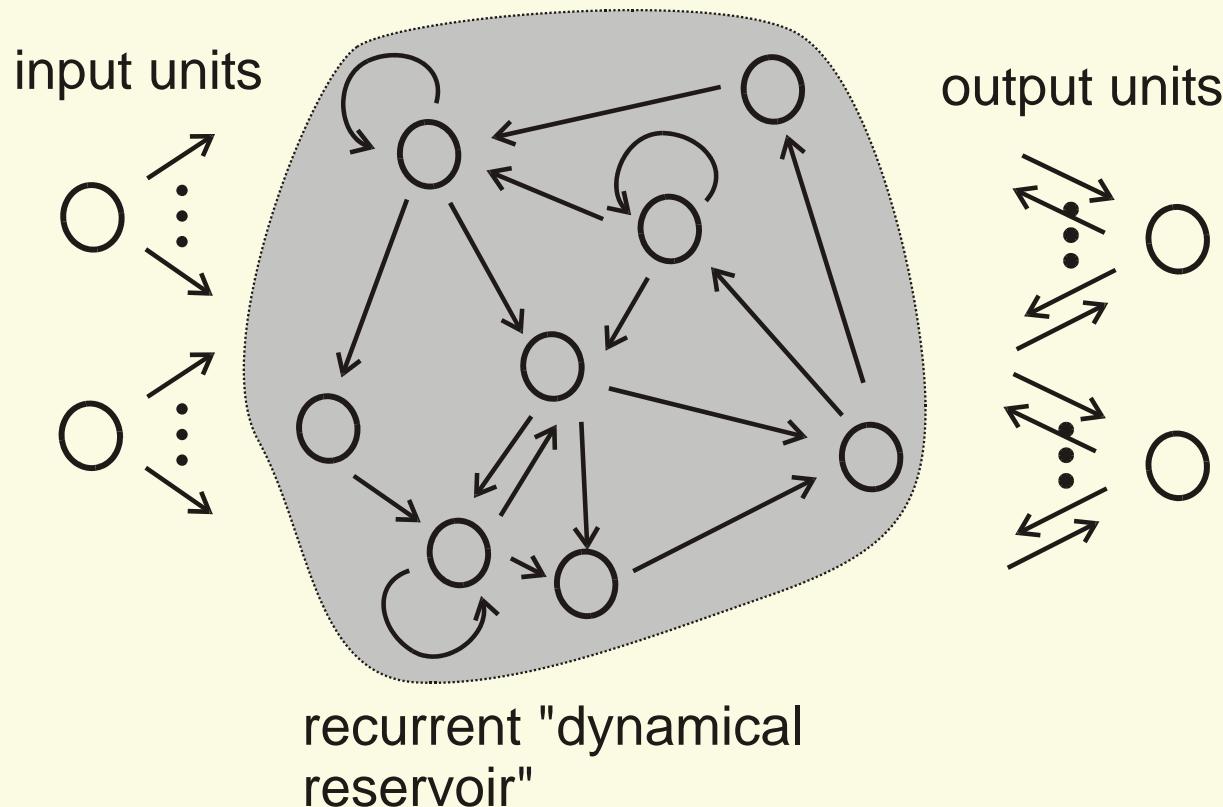
Fig. 15.7. Mapping a square with a two-dimensional lattice. The diagram on the upper right shows some overlapped iterations of the learning process. The diagram below it is the final state after 10000 iterations.



Echo State Network

- Proposed by Herbert Jaeger and Harald Haas at Jacobs University in 2004.
- Also called reservoir computing.
- It is a recurrent neural network with sparse connections and random weights among hidden neurons.

ESN Architecture



State Equations

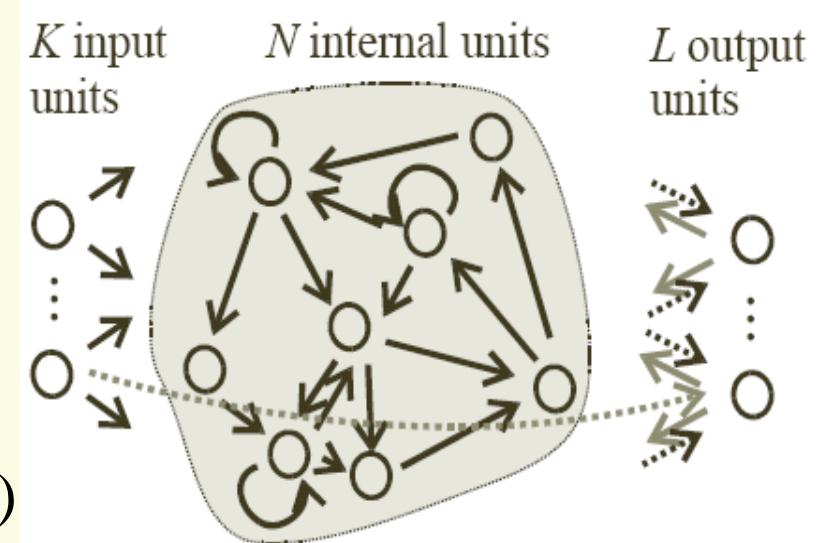
$$u(n) = (u_1(n), \dots, u_K(n))'$$

$$x(n) = (x_1(n), \dots, x_N(n))'$$

$$y(n) = (y_1(n), \dots, y_L(n))'$$

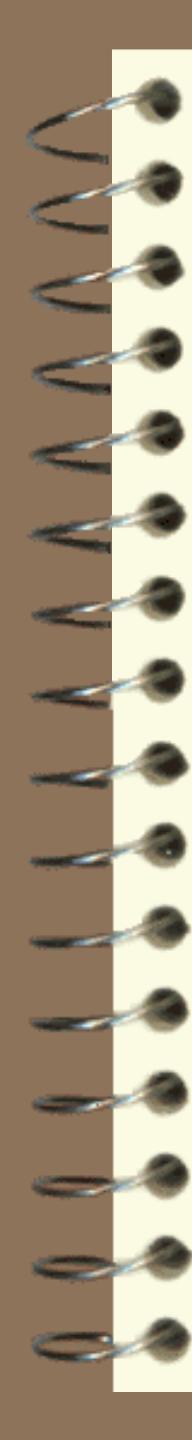
$$W^{in} = (w_{ij}^{in}), W = (w_{ij}),$$

$$W^{out} = (w_{ij}^{out}), W^{back} = (w_{ij}^{back})$$



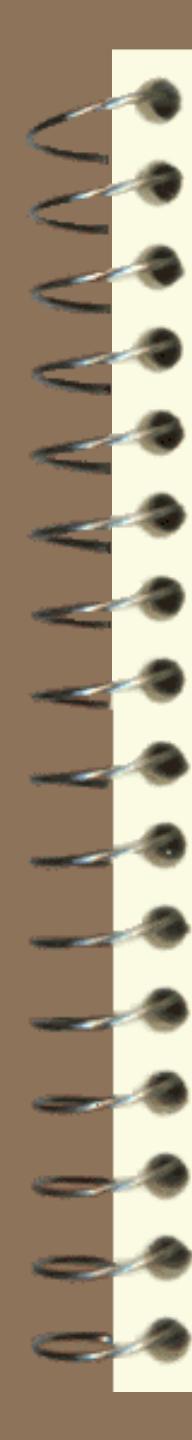
$$x(n+1) = f(W^{in}u(n+1) + Wx(n) + W^{back}y(n))$$

$$y(n+1) = f^{out}(W^{out}(u(n+1), x(n+1), y(n)))$$



Fuzzy Logic

- Developed by Prof. Lofti Zadeh at the University of California - Berkeley in late 1965.
- A generalization of classical logic.
- Fuzzy logic describes one kind of uncertainty: impreciseness or ambiguity.
- Probability, on the other hand, describes the other kind of uncertainty: randomness.



Membership Function

Let X be a classical set. A membership function of fuzzy set A $u_A : X \rightarrow [0, 1]$ defines the fuzzy set A of X .

Crisp sets are special case of fuzzy sets where the value of the membership function are 0 and 1 only.

Membership Functions

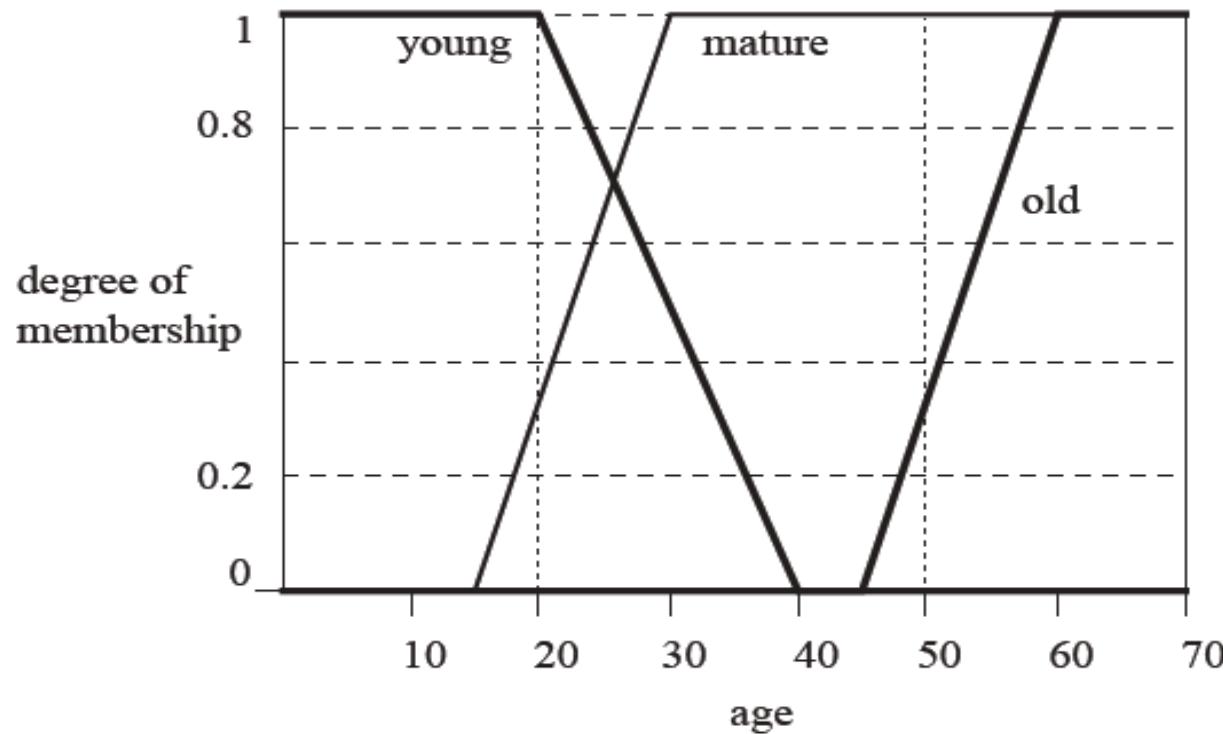
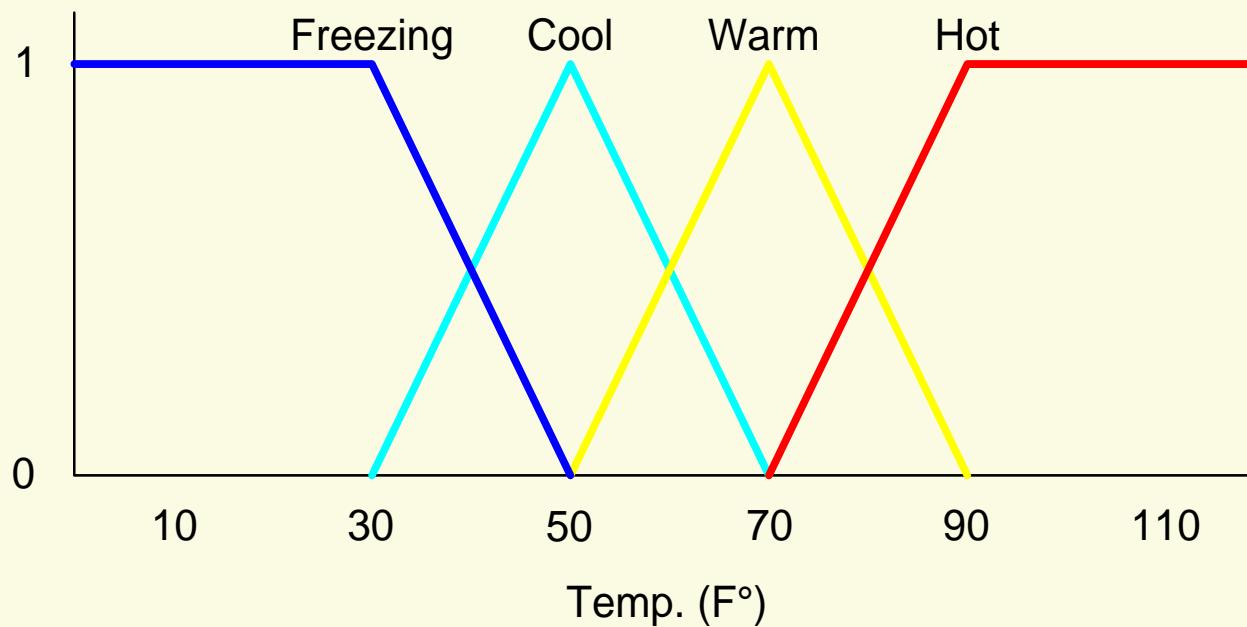


Fig. 11.1. Membership functions for the concepts young, mature and old

Membership Functions

Temp: {Freezing, Cool, Warm, Hot}

Degree of Truth or "Membership"



Membership Functions

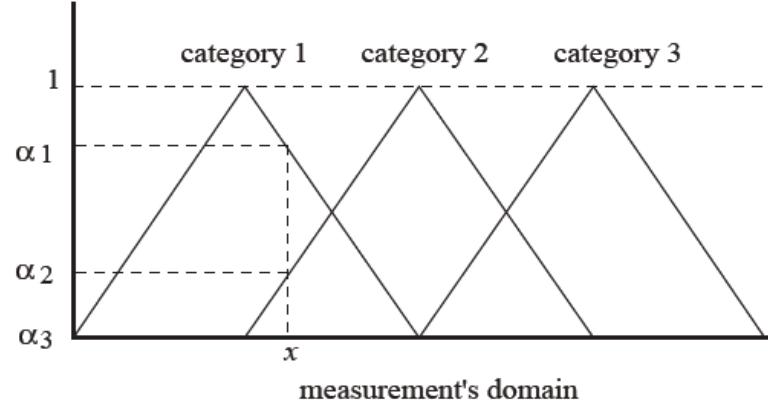


Fig. 11.10. Categories with triangular membership functions

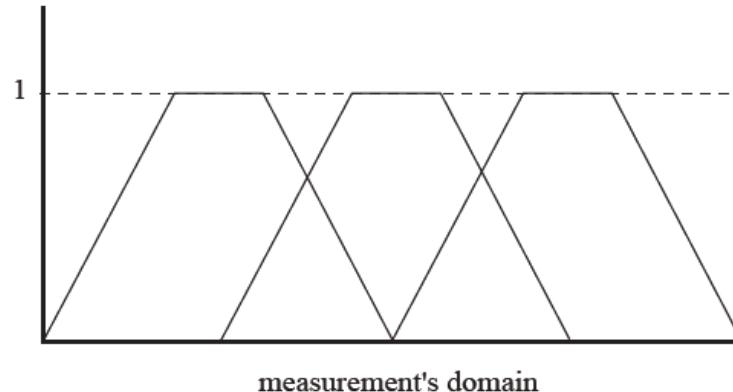


Fig. 11.11. Categories with trapezium-shaped membership functions

Fuzzy Set

Fuzzy set A is the set of all pairs $(x, u_A(x))$ where x belongs to X ; i.e., $A = \{(x, u_A(x)) \mid x \in X\}$

If X is discrete,
$$A = \sum_i u_A(x_i) / x_i$$

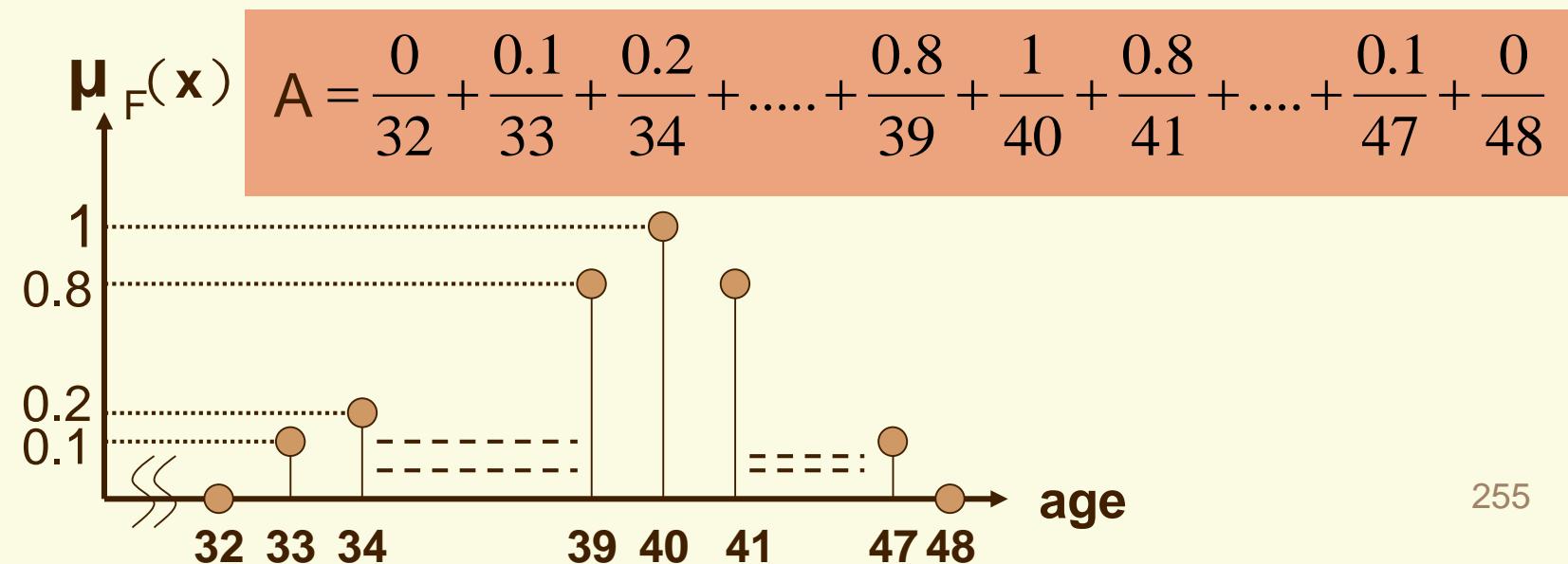
If X is continuous,
$$A = \int_X u_A(x) / x$$

Support set of A is $\text{supp}(A) = \{x \in X \mid u_A(x) > 0\}$

Discrete Fuzzy Set

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \frac{\mu_A(x_3)}{x_3} + \dots + \frac{\mu_A(x_n)}{x_n}$$

Example: middle age



Fuzzy Set

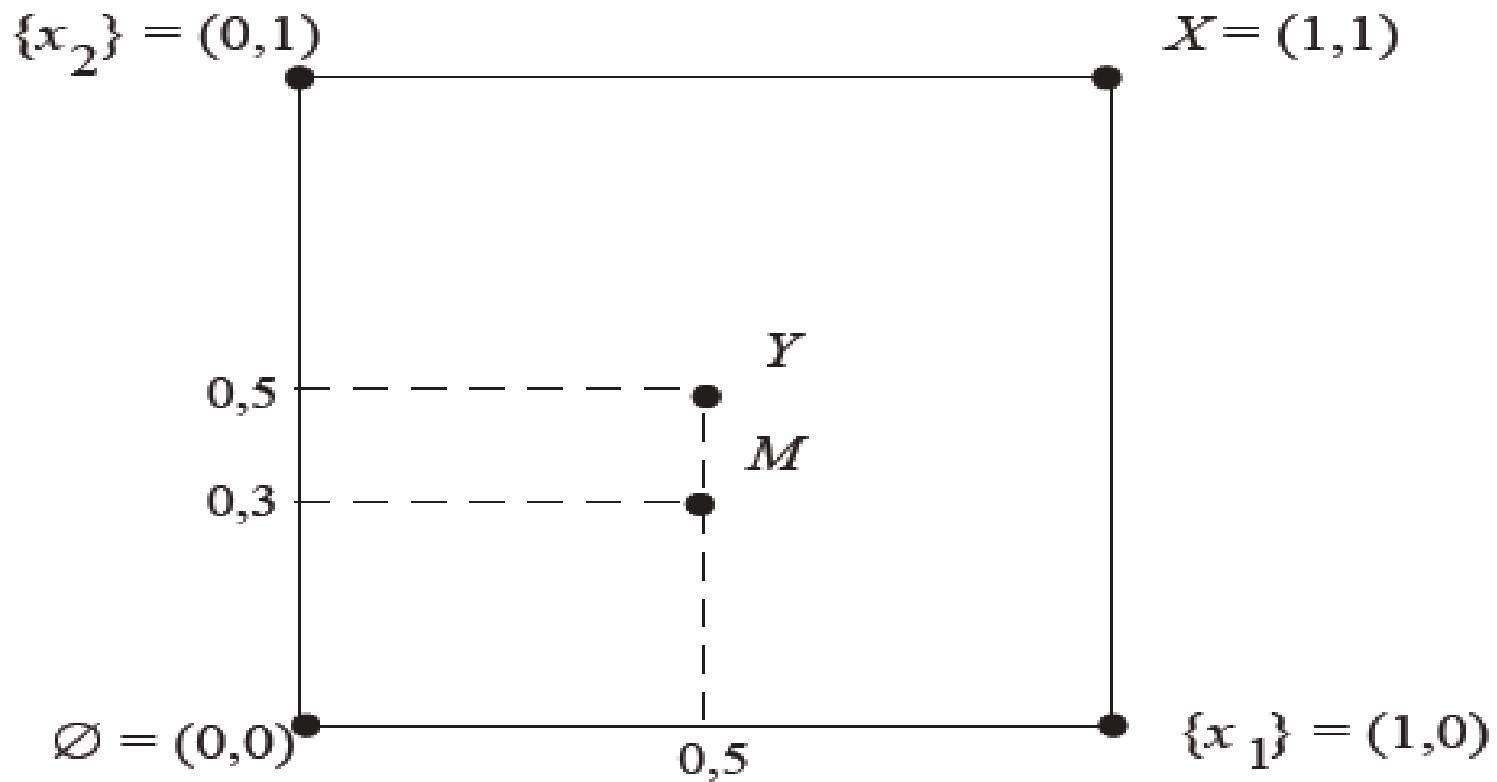
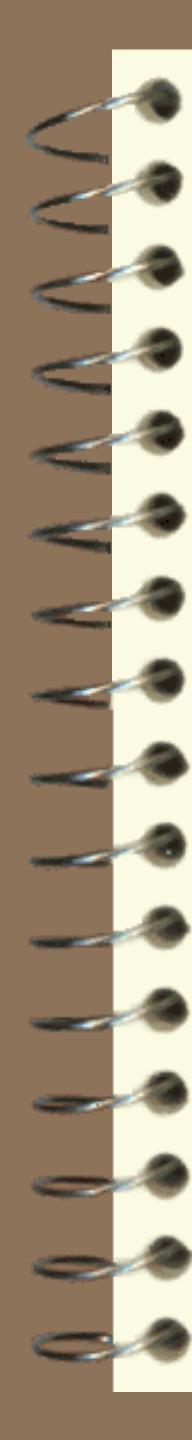


Fig. 11.2. Geometric visualization of fuzzy sets



Fuzzy Set Terminology

- Fuzzy singleton: A fuzzy set where its support set contain a single point only with $u_A(x)=1$.
- Crossover point: $x \in X$ such that $u_A(x) = 0.5$
- Kernel of a fuzzy set A : All x such that $u_A(x)=1$; i.e., $\text{ker}(A) = \{x \in X \mid u_A(x) = 1\}$
- Height of a fuzzy set A : Supremum of $u_A(x)$ over x ; i.e., $\text{ht}(A) = \sup_{x \in X} u_A(x)$

Fuzzy Set Terminology

- Normalized fuzzy set A : Its height is unity; i.e., $\text{ht}(A)=1$. Otherwise, it is subnormal.
- α -cut of a fuzzy set A : A crisp set

$$A_\alpha = \{x \in X \mid u_A(x) \geq \alpha\}$$

- Convex fuzzy set A :

$$\forall \lambda \in [0,1], \forall x, y \in X$$

$$u_A(\lambda x + (1-\lambda)y) \geq \min(u_A(x), u_A(y))$$

i.e., any α -cut is a convex set.

Logic Operations on Fuzzy Sets

- Union of two fuzzy sets:

$$\mu_{A \cup B}(x) = \max\{ \mu_A(x), \mu_B(x) \}$$

- Intersection of two fuzzy sets:

$$\mu_{A \cap B}(x) = \min\{ \mu_A(x), \mu_B(x) \}$$

- Complement of a fuzzy set:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

Logic Operations on Fuzzy Sets

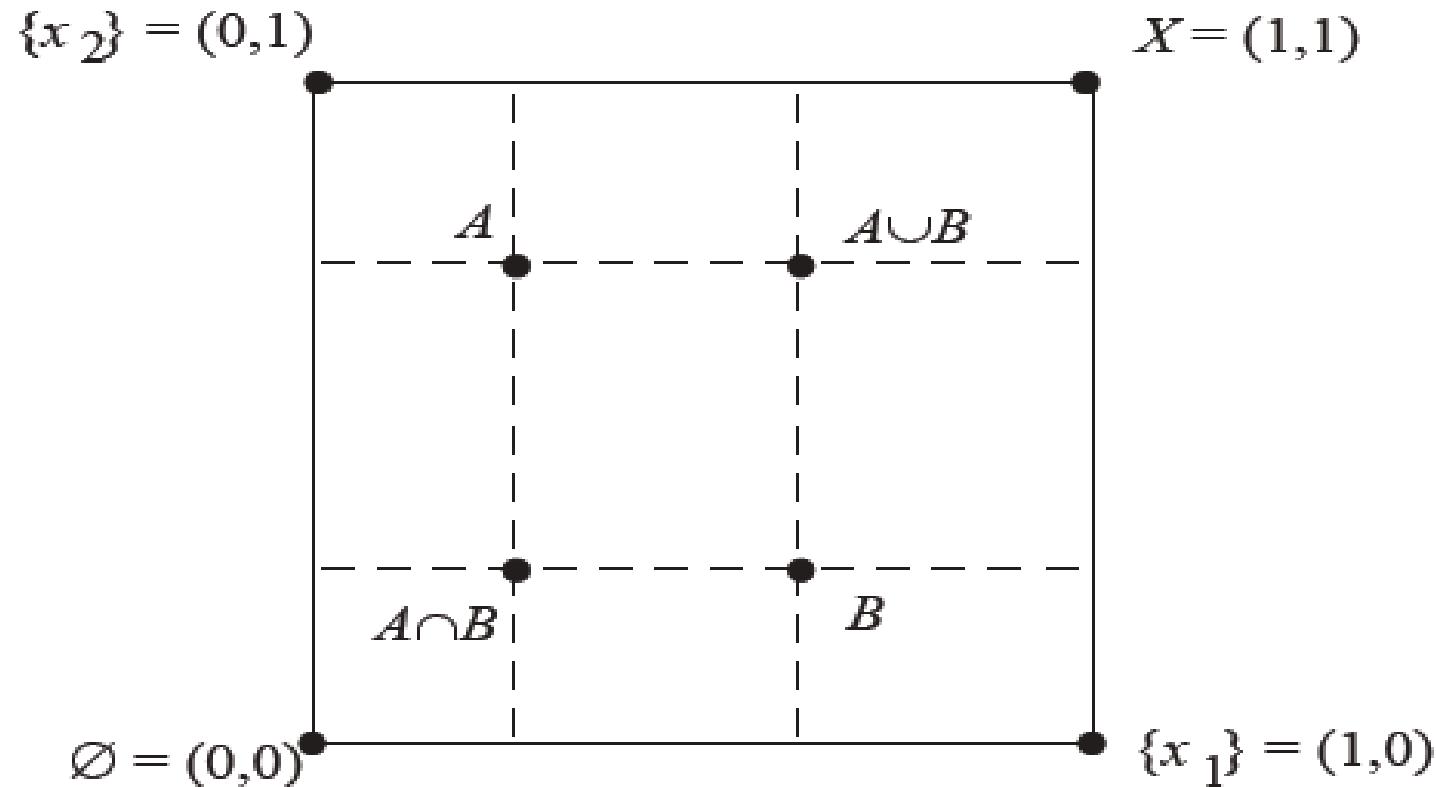


Fig. 11.4. Intersection and union of two fuzzy sets

Cardinality and Entropy of Fuzzy Sets

- Cardinality: $|A|$ is defined as the sum of the membership function values of all elements in X ; *i.e.*, $|A| = \sum_{x \in X} \mu_A(x)$ or $|A| = \int_X \mu_A(x) dx$
- Entropy: $E(A)$ measures fuzziness and is defined as

$$E(A) = \frac{|A \cap \bar{A}|}{|A \cup \bar{A}|}$$

Cardinality of Fuzzy Sets

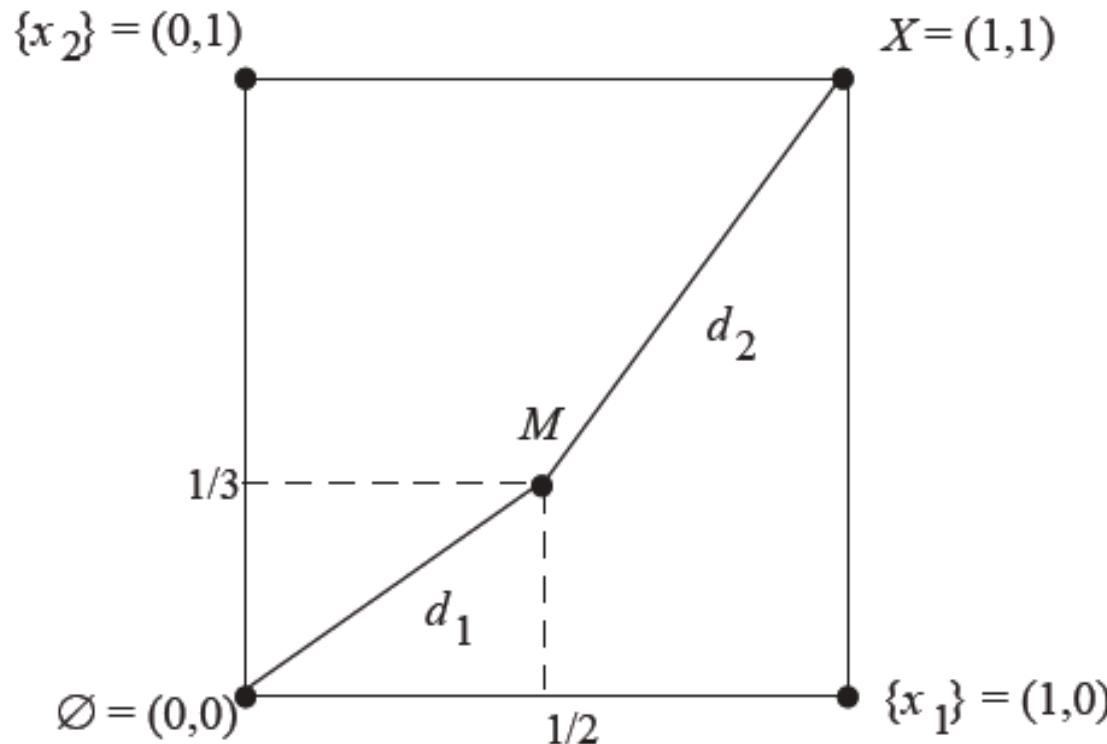


Fig. 11.3. Distance of the set M to the universal and to the void set

Entropy of Fuzzy Sets

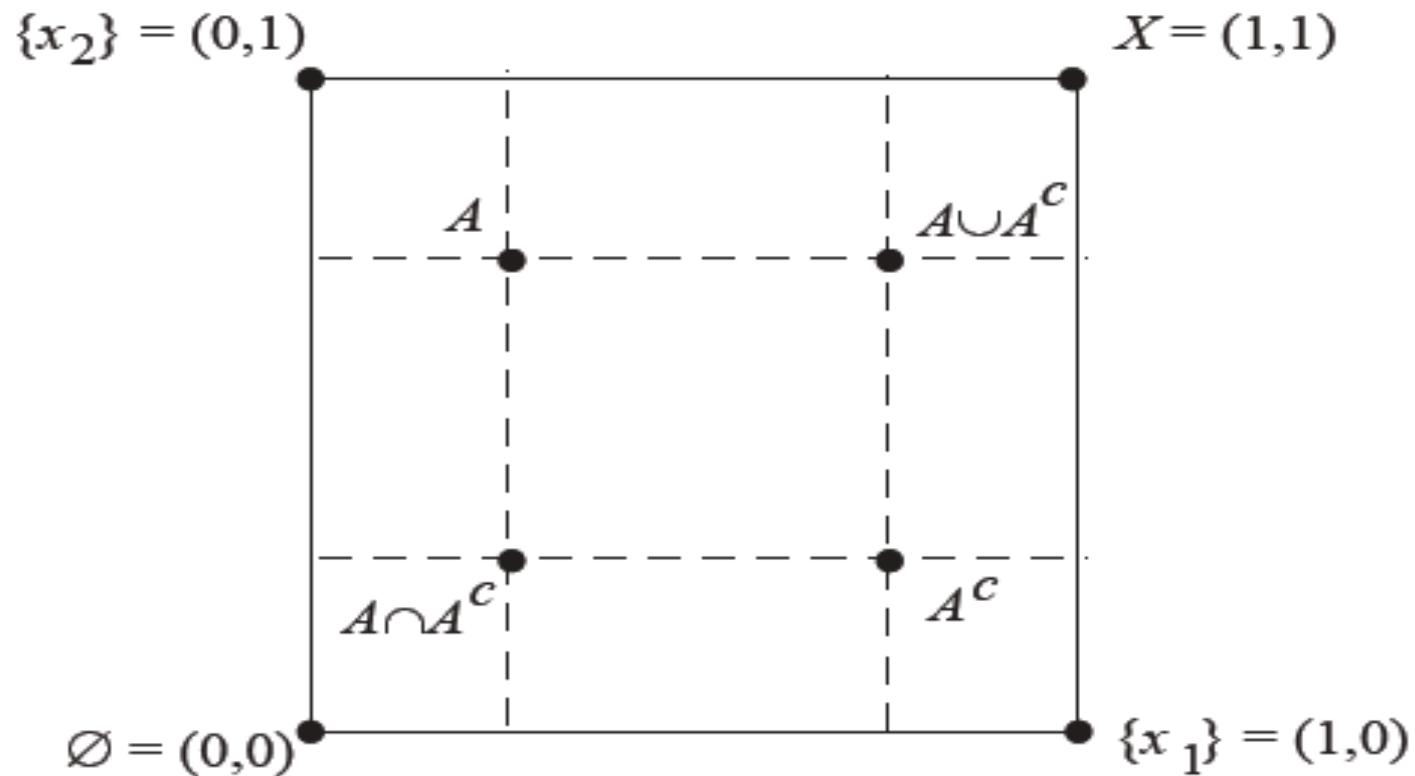


Fig. 11.5. Complement A^c of a fuzzy set

Entropy of Fuzzy Sets

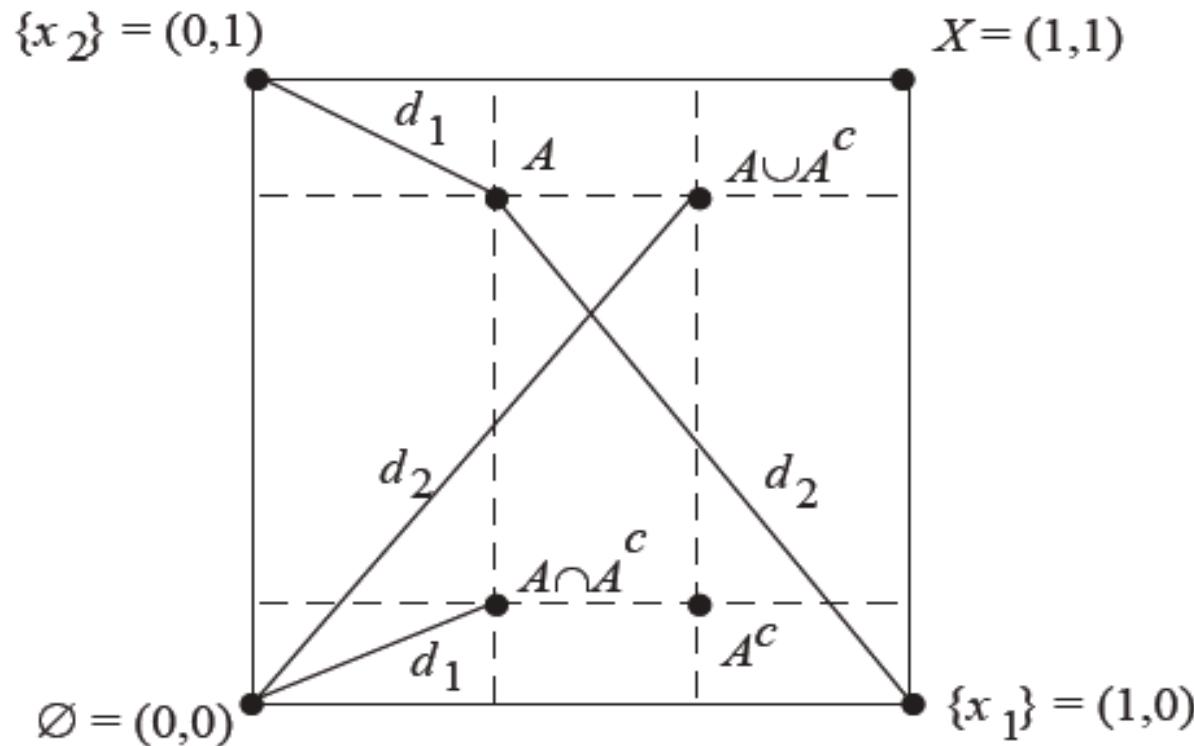


Fig. 11.6. Geometric representation of the entropy of a fuzzy set

Logic Operations on Fuzzy Sets

- Equality: For all x , $u_A(x) = u_B(x)$
- Degree of equality:

$$E(A, B) = \deg(A = B) = \frac{|A \cap B|}{|A \cup B|}$$

- Subset: $A \subseteq B$, if $u_A(x) \leq u_B(x), \forall x \in X$
- Subsethood measure:

$$S(A, B) = \deg(A \subseteq B) = \frac{|A \cap B|}{|A|}$$

Properties of Fuzzy Sets

Union: $A \subseteq A \cup B, B \subseteq A \cup B$

Intersection: $A \cap B \subseteq \underline{\underline{A}}, A \cap B \subseteq B$

Double negation law: $\underline{\underline{A}} = A$

DeMorgan's laws: $\overline{\overline{A} \cup B} = \overline{\overline{A}} \cap \overline{\overline{B}}$

$$\overline{\overline{A} \cap B} = \overline{\overline{A}} \cup \overline{\overline{B}}$$

However, $A \cup \overline{\overline{A}} \neq X, A \cap \overline{\overline{A}} \neq \emptyset$

Fuzzy Relations

$$R(x_1, x_2, \dots, x_n) = \int_{X_1 \times X_2 \dots X_n} u_R(x_1, x_2, \dots, x_n) / (x_1, x_2, \dots, x_n)$$

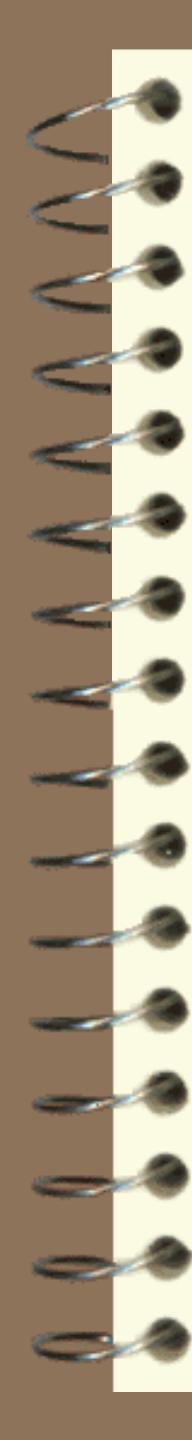
$$R(x_1, x_2, \dots, x_n) = \{((x_1, x_2, \dots, x_n), u_R(x_1, x_2, \dots, x_n)) \mid (x_1, x_2, \dots, x_n) \in X_1 \times X_2 \dots X_n\}$$

Binary fuzzy relations are most common.

Reflexive: $u_R(x, x) = 1$

Symmetric: $u_R(x, y) = u_R(y, x)$

Transitive: $u_R(x, z) = \max_y \min_{(x, z)} [u_R(x, y), u_R(y, z)]$



Fuzzifiers and Defuzzifiers

- Fuzzifier: A mapping from a real-valued set to a fuzzy by means of a membership function.
- Defuzzifier: A mapping from a fuzzy set to a real-valued set.

Typical Defuzzifiers

- Centroid (also known as center of gravity and center of area) defuzzifier:

$$x^* = \frac{\int_X x \mu(x) dx}{\int_X \mu(x) dx} \text{ or } x^* = \frac{\sum_i x_i \mu_A(x_i)}{\sum_i \mu_A(x_i)}$$

- Center average (mean of maximum) defuzzifier:

$$x^* = \frac{\sum_i x_i ht(x_i)}{\sum_i ht(x_i)}$$

Centroid Defuzzifier

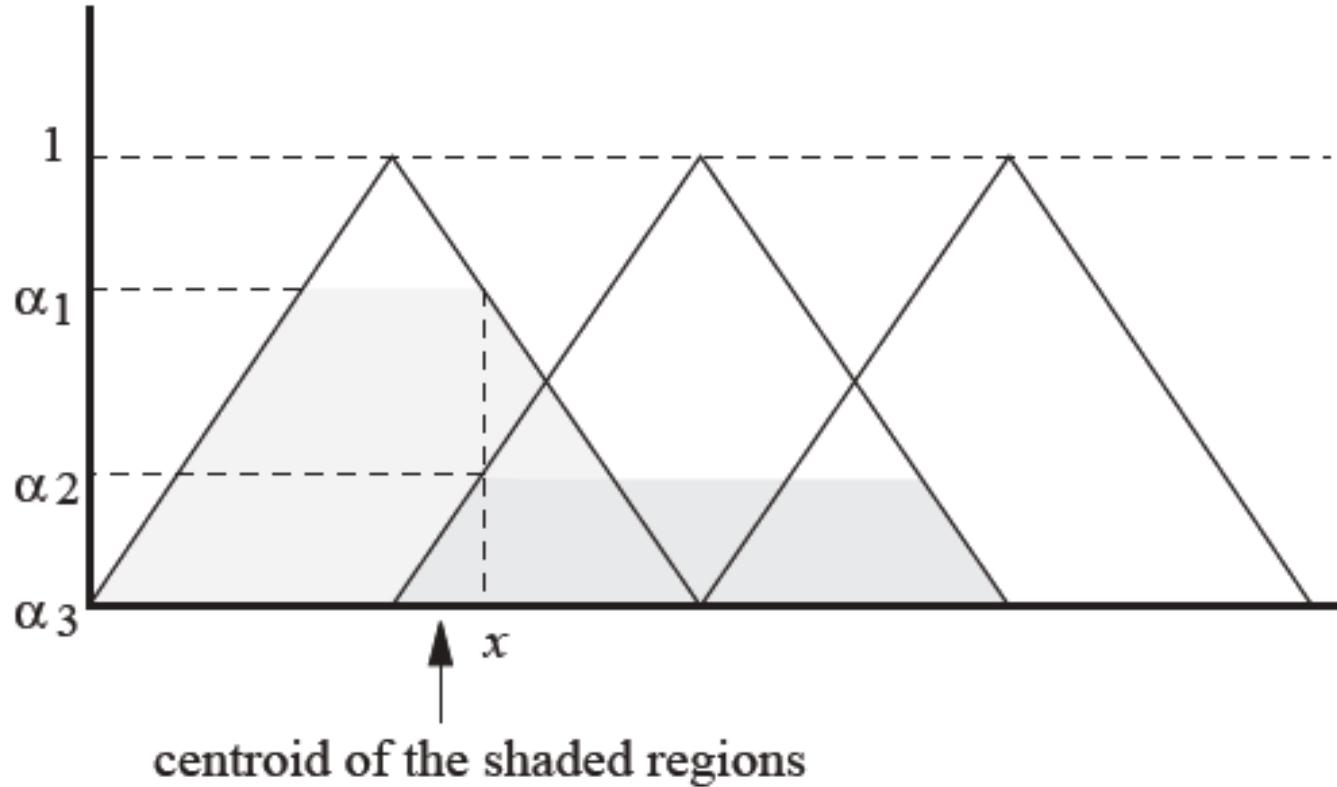
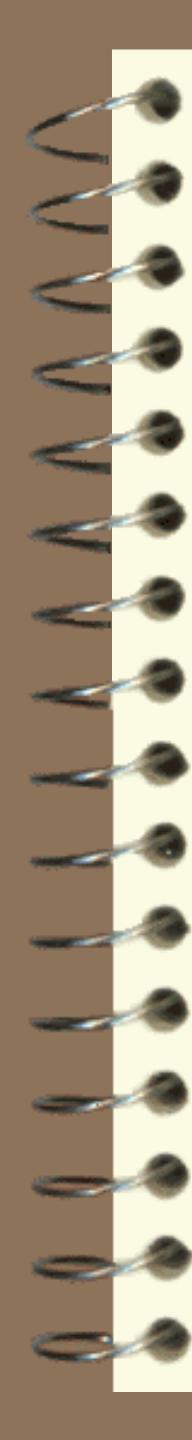
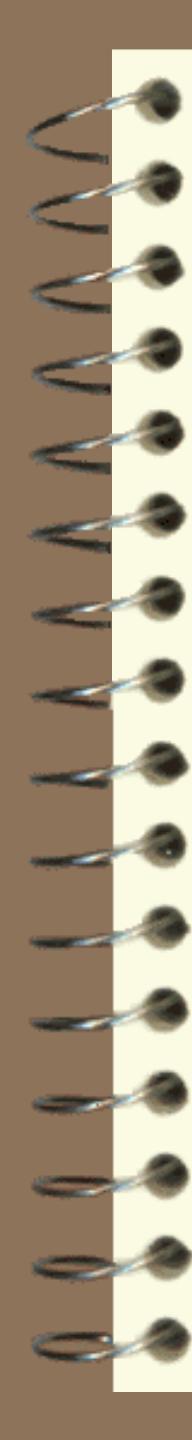


Fig. 11.12. The centroid method



Linguistic Variables

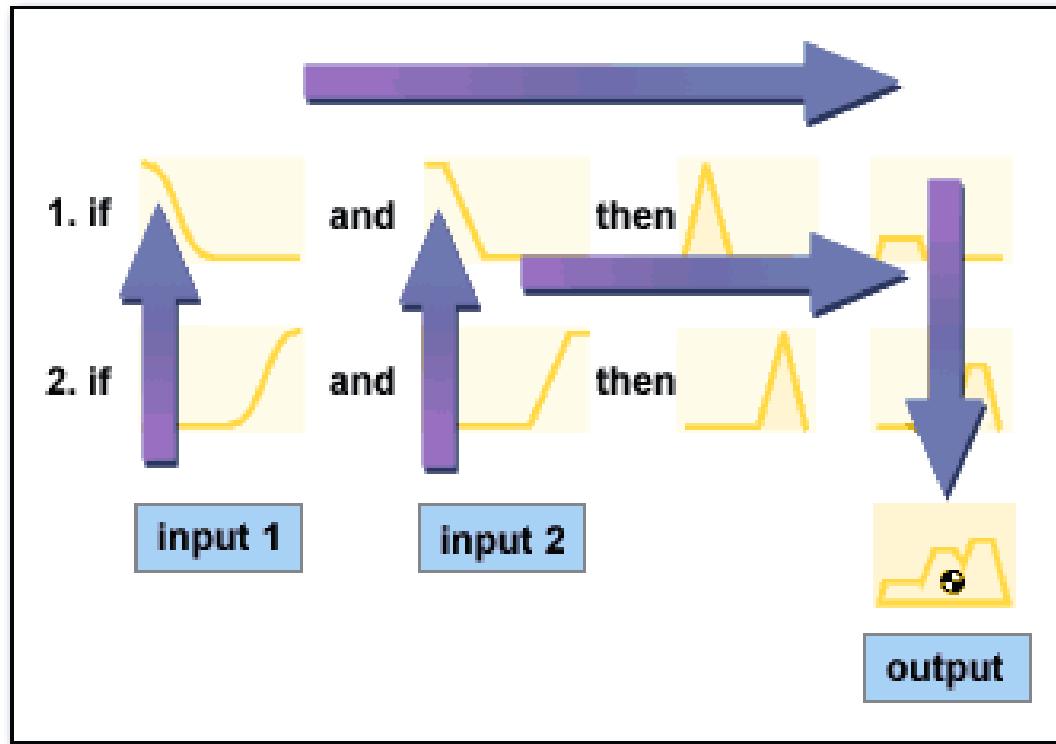
- Linguistic variables are important in fuzzy logic and approximate reasoning.
- Linguistic variables are variables whose values are words or sentences in natural or artificial languages.
- For example, *speed* can be defined as a linguistic variable and takes values of *slow*, *fast*, and *very fast*.



Fuzzy Inference Process

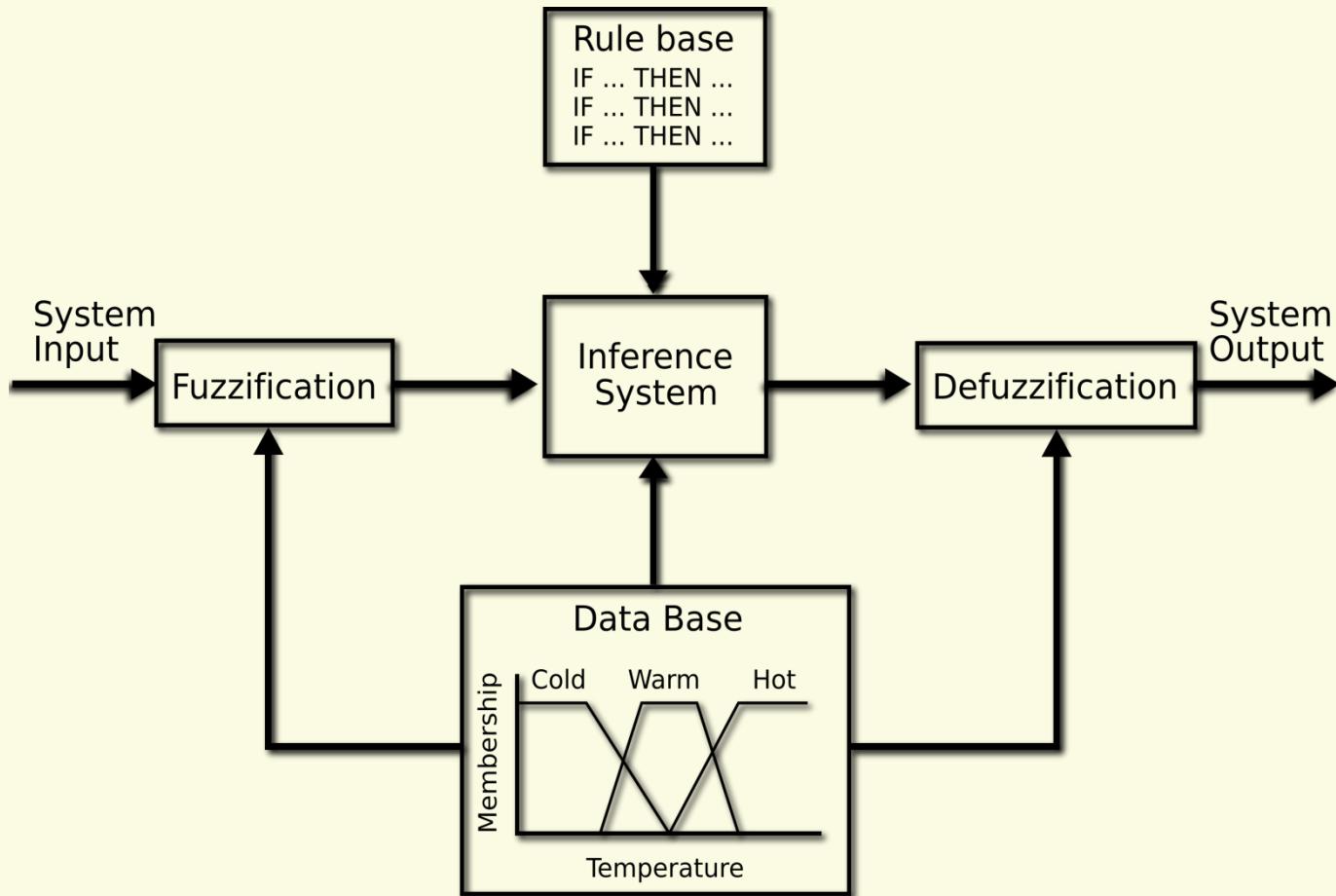
- When imprecise information is input to a fuzzy inference system, it is first fuzzified by constructing a membership function.
- Based on a fuzzy rule base, the fuzzy inference engine makes a fuzzy decision.
- The fuzzy decision is then defuzzified to output for an action.
- The defuzzification is usually done by using the centroid method.

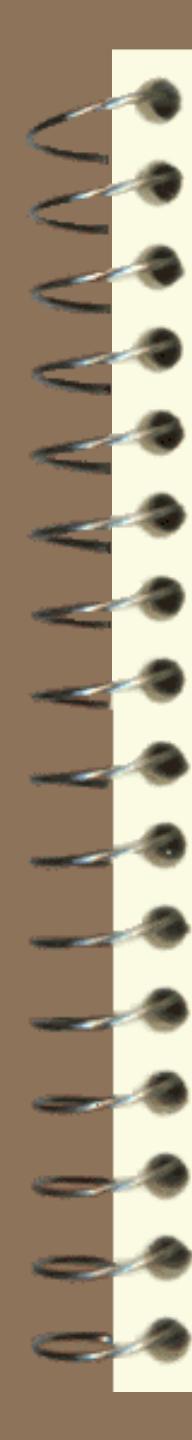
Fuzzy Inference Process



Interpreting the Fuzzy Inference Diagram

Fuzzy Inference System





An Electrical Heater Example

Rule Base:

- R1: If temperature is cold, then increase power.
- R2: If temperature is normal, then maintain.
- R3: If temperature is warm, then reduce power.
- At 12° , $T = \text{cold}/0.5 + \text{normal}/0.3 + \text{warm}/0.0$,
- $A = \text{increase}/0.5 + \text{maintain}/0.3 + \text{reduce}/0.0$.

An Electrical Heater Example

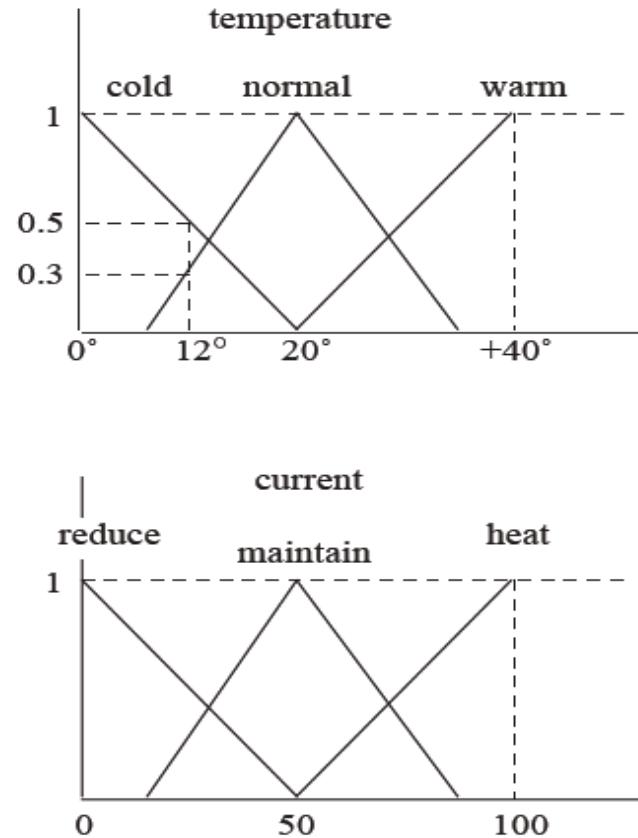


Fig. 11.14. Membership functions for temperature and electric current categories

An Electrical Heater Example

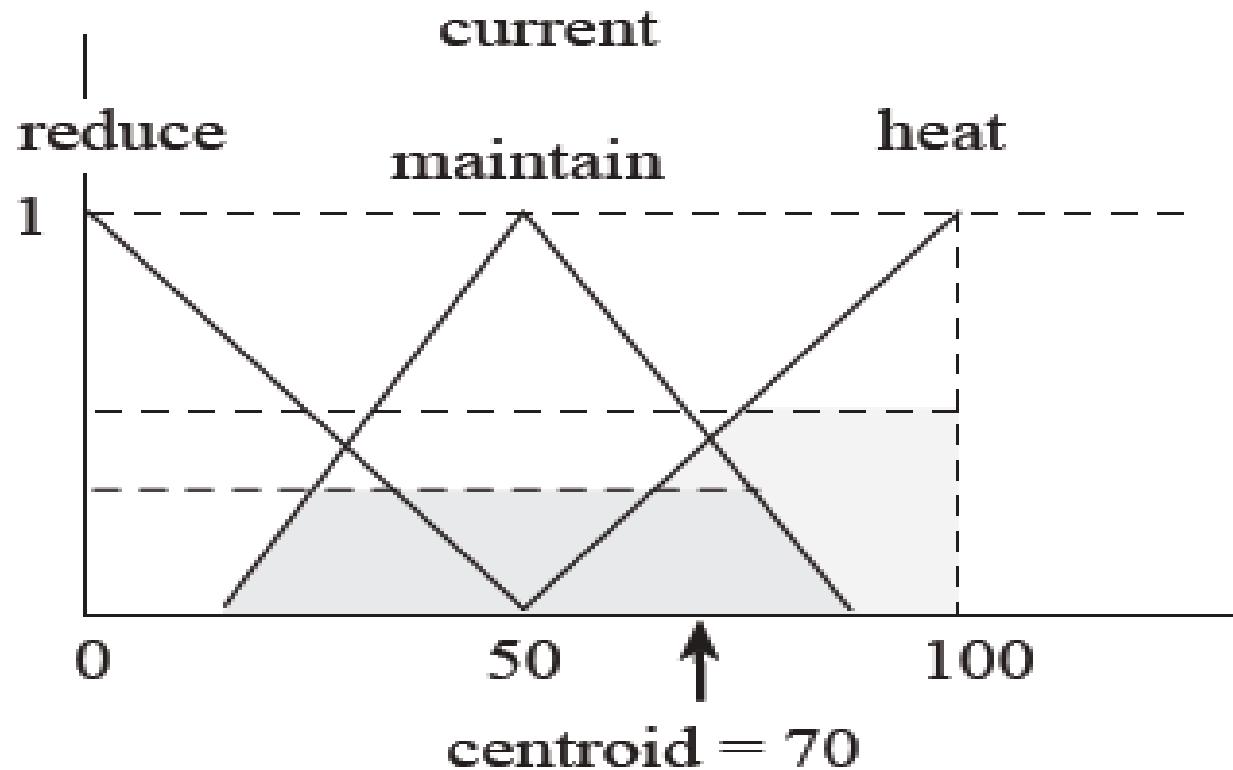
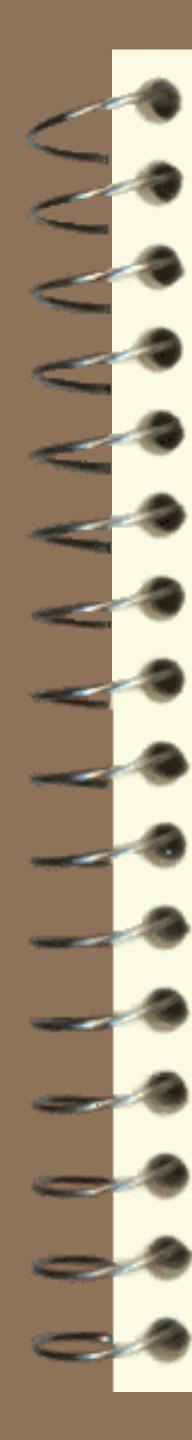
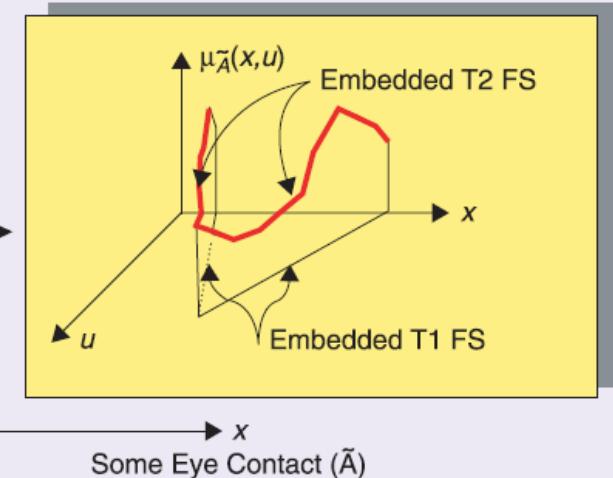
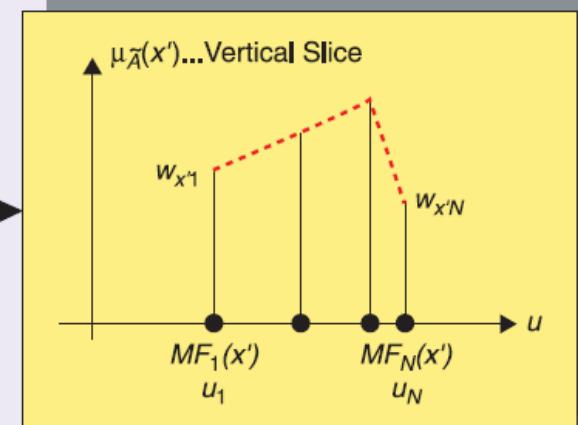
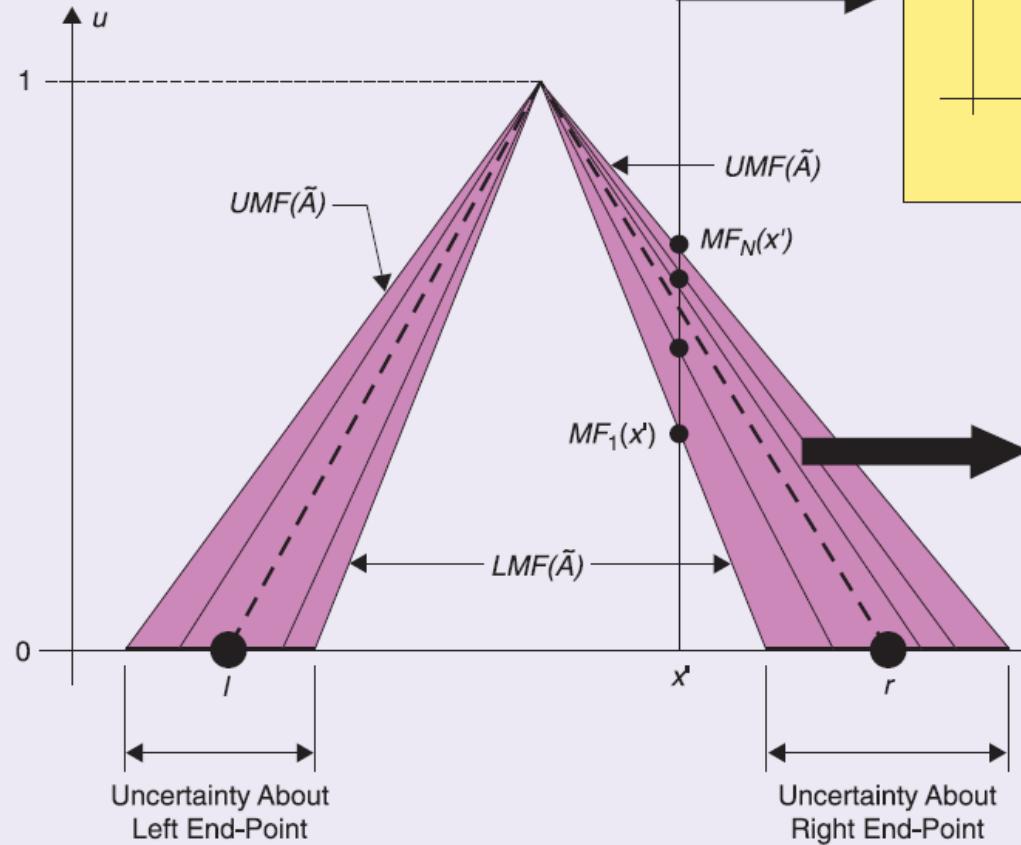


Fig. 11.15. Centroid computation

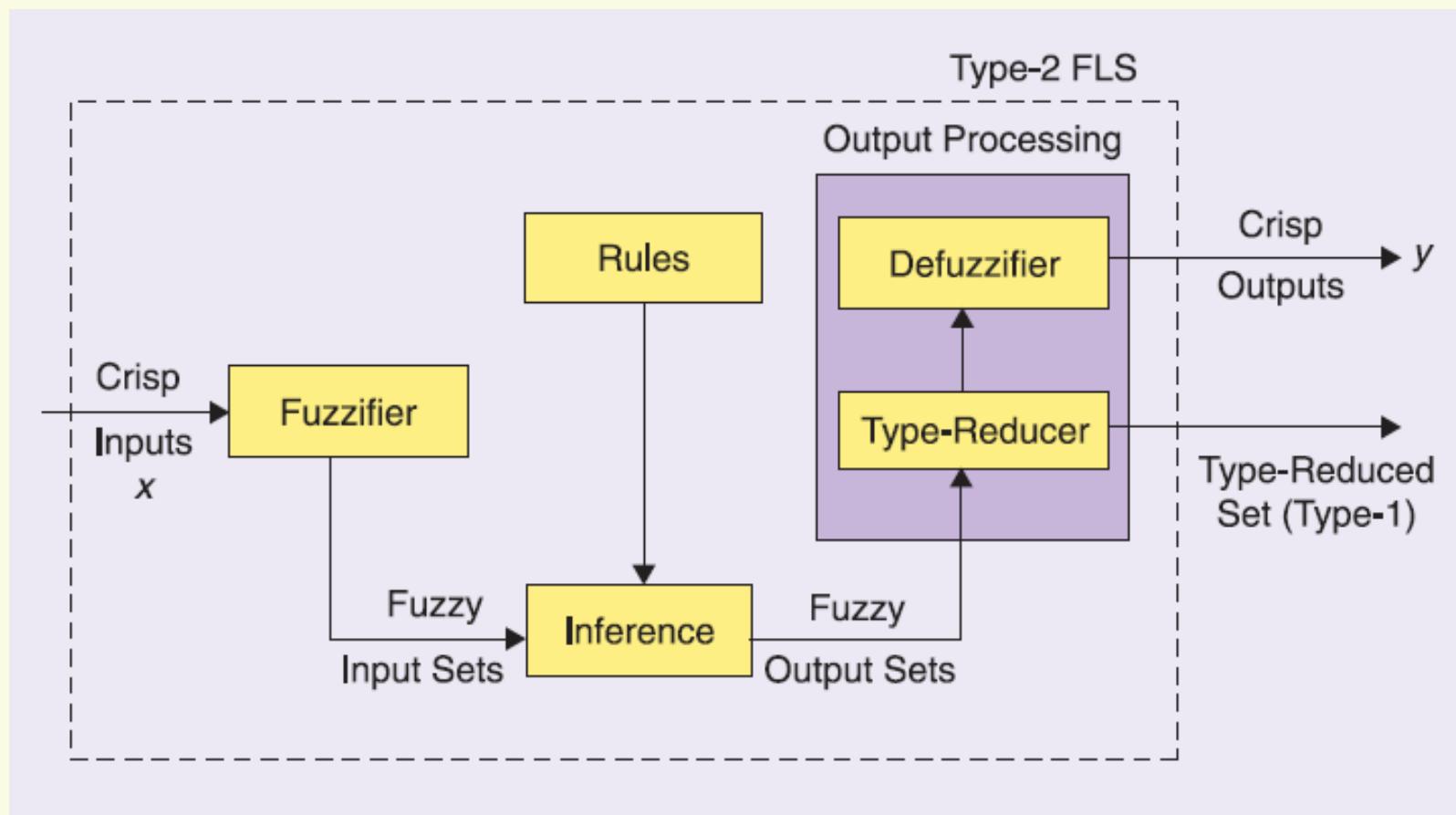


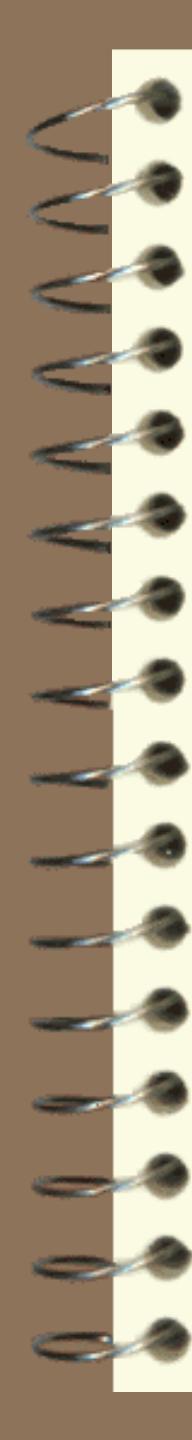
Type-2 Fuzzy Logic

- A generalization of type-1 fuzzy logic to handle the uncertainty of membership function by using fuzzy membership function
- Proposed by Prof. Zadeh in 1975 (ten years after type-1), but became popular in recent few years



Type-2 Fuzzy System





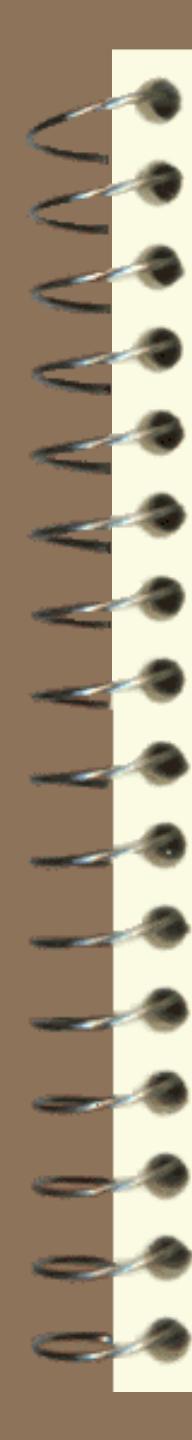
Evolutionary Computation

- Population-based stochastic and meta-heuristic search algorithms for global or multi-objective optimization
- Motivated by the natural evolution based on Darwinist or other principles
- Use collective wisdom to accomplish given tasks via efforts of many generations.

Swarm-Based Search

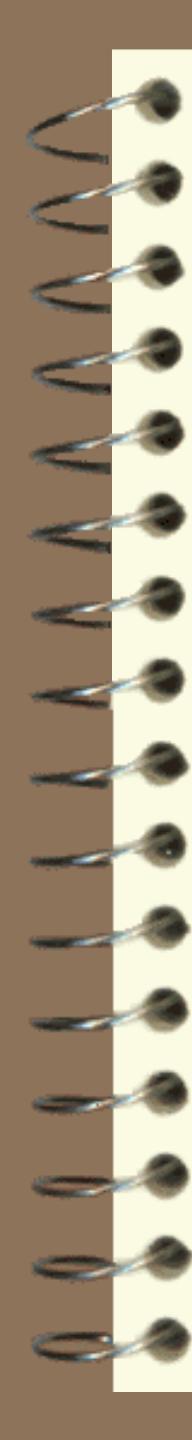
Swarm is better than individual





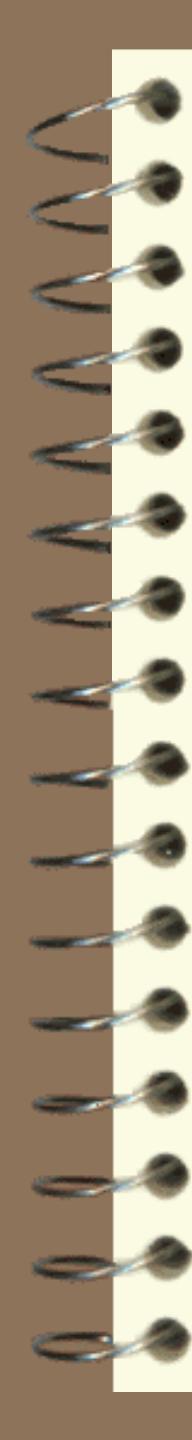
Evolutionary Algorithms

- Evolutionary programming (Lawrence Fogel, 1960's)
- Evolutionary strategies (Ingo Rechenberg and Hans-Paul Schiwefel, 1960's)
- Genetic algorithms (John Holland, 1970's)
- Genetic programming (John Koza,
- 1990's)



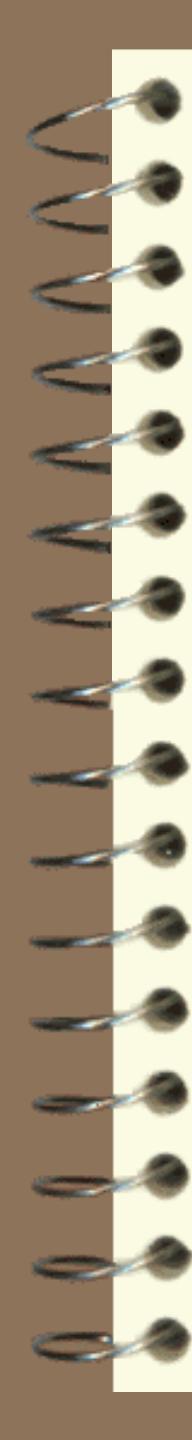
Genetic Algorithms

- A stochastic search method simulating the evolution of population of living species.
- Optimize a fitness function which is not necessarily continuous or differentiable.
- A genetic algorithm generates a population of seeds instead of one in traditional algorithms.
- The computation of the population can be carried out in parallel.



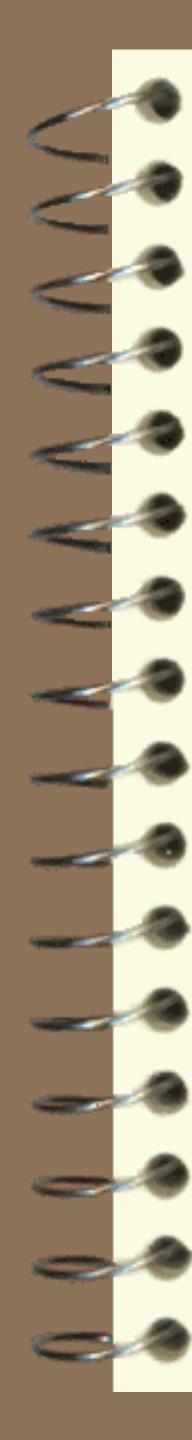
Elements in Genetic Algorithms

- ★ A coding of the optimization problem to produce the required discretization of decision variables in terms of strings.
- ★ A reproduction operator to copy individual strings according to their fitness.
- ★ A set of information-exchange operators; e.g., crossover, for recombination of search points to generate new and better population of points.
- ★ A mutation operator for modifying data.



Reproduction Operator

1. Sum the fitness of all the production members and call the result total fitness.
2. Generate a random number n between 0 and total fitness under uniform distribution.
3. Return the first population member whose fitness, added to the fitnesses of the preceding population members (running total), is greater than or equal to n .



Crossover Operator

- Select offspring from the population after reproduction.
- Two strings (parents) from the reproduced population are paired with probability P_c .
- Two new strings (offspring) are created by exchanging bits at a crossover site.

Crossover Operation

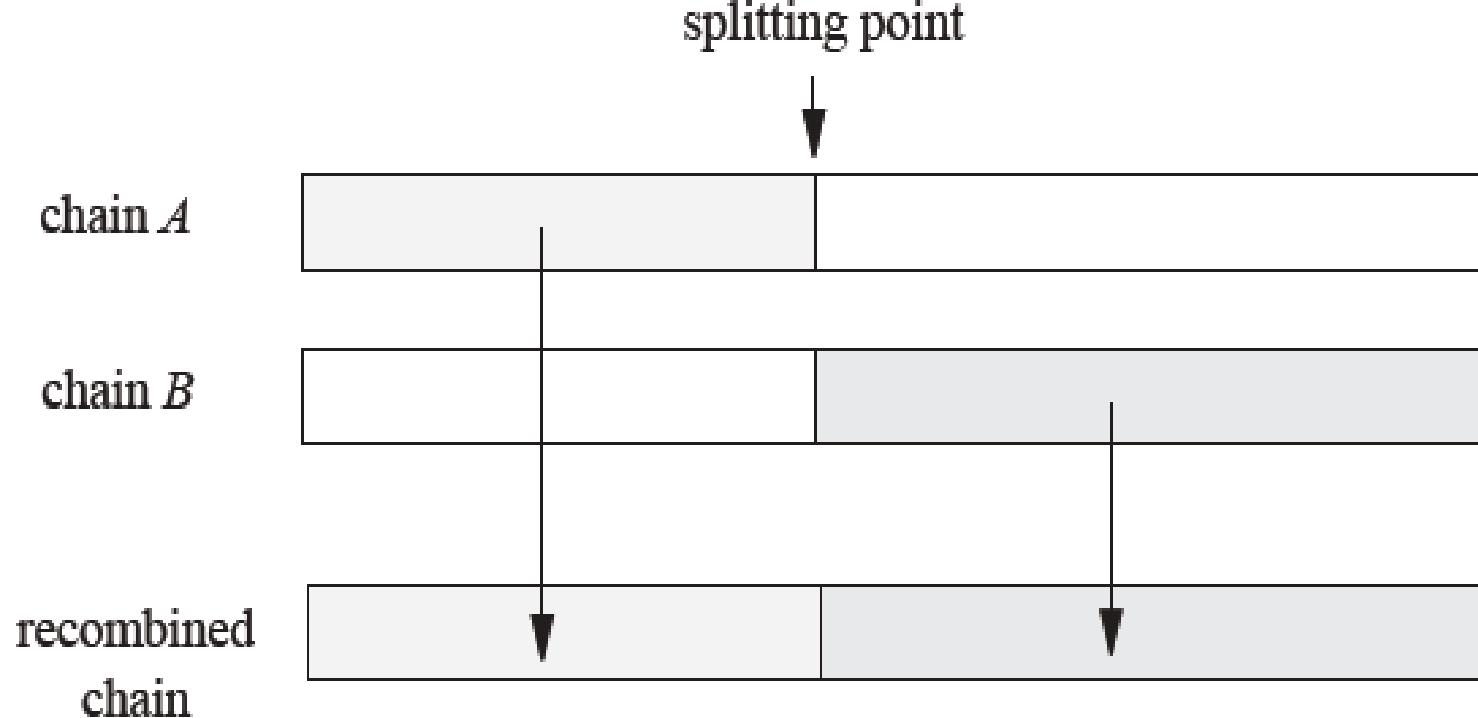
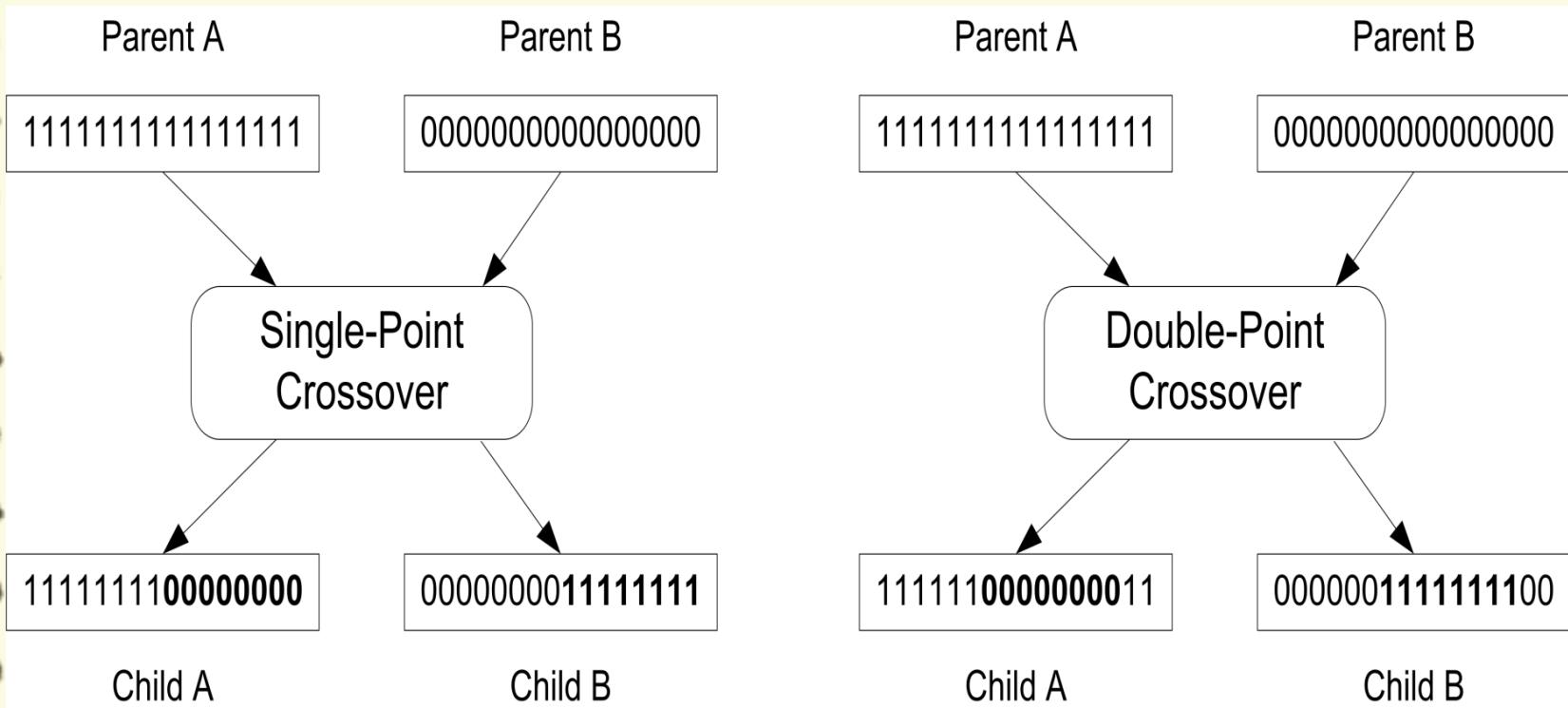
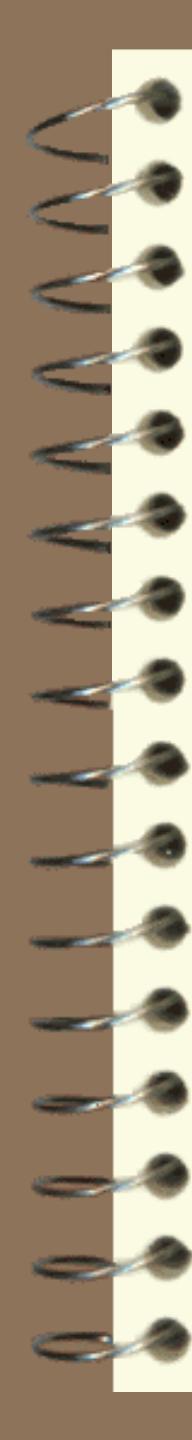


Fig. 17.3. An example of crossover

Crossover Operation

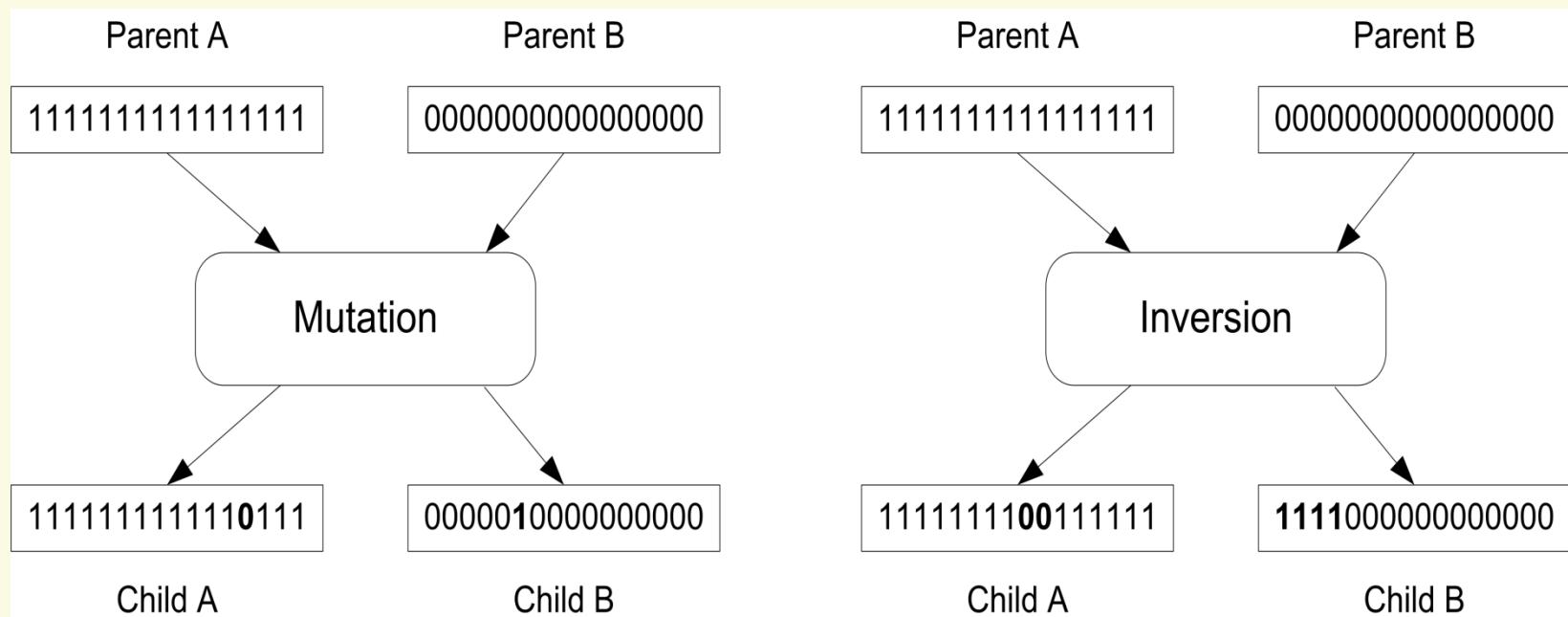


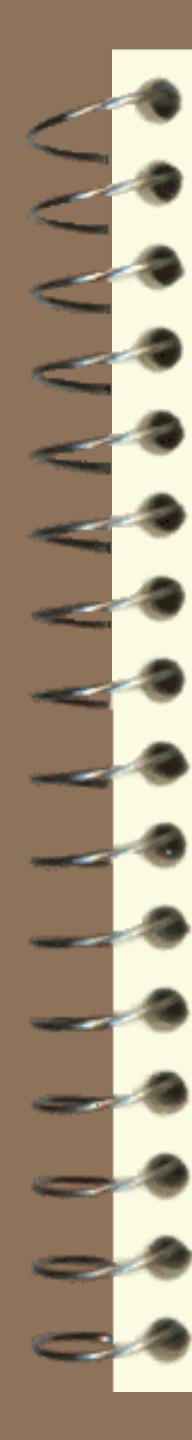


Mutation Operator

- Reproduction and crossover produce new string without introducing new information into the population at bit level.
- To inject new information into offspring.
- Invert chosen bits randomly with a lower probability P_m

Mutation Operation



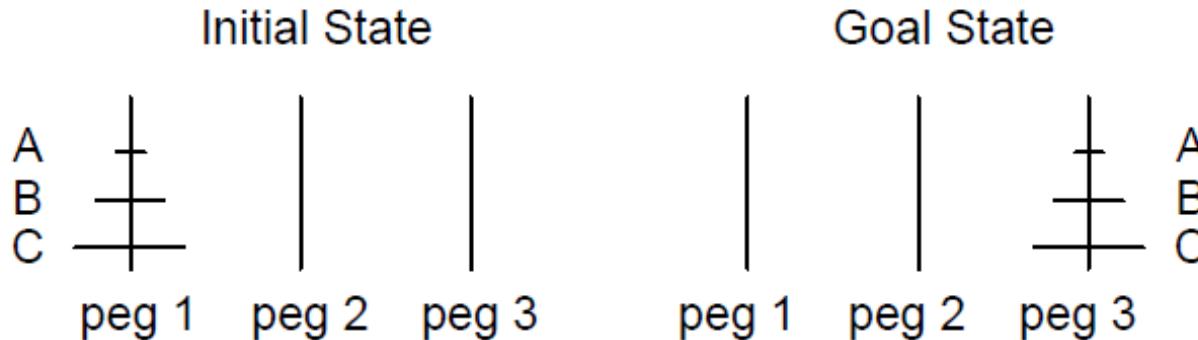


Towers of Hanoi

- Towers of Hanoi puzzle: move a number of disks from one peg to another among three pegs, to restore the original piling order.
- Any move must satisfy constraints: e.g., we can only move a disk that has no disk above it.

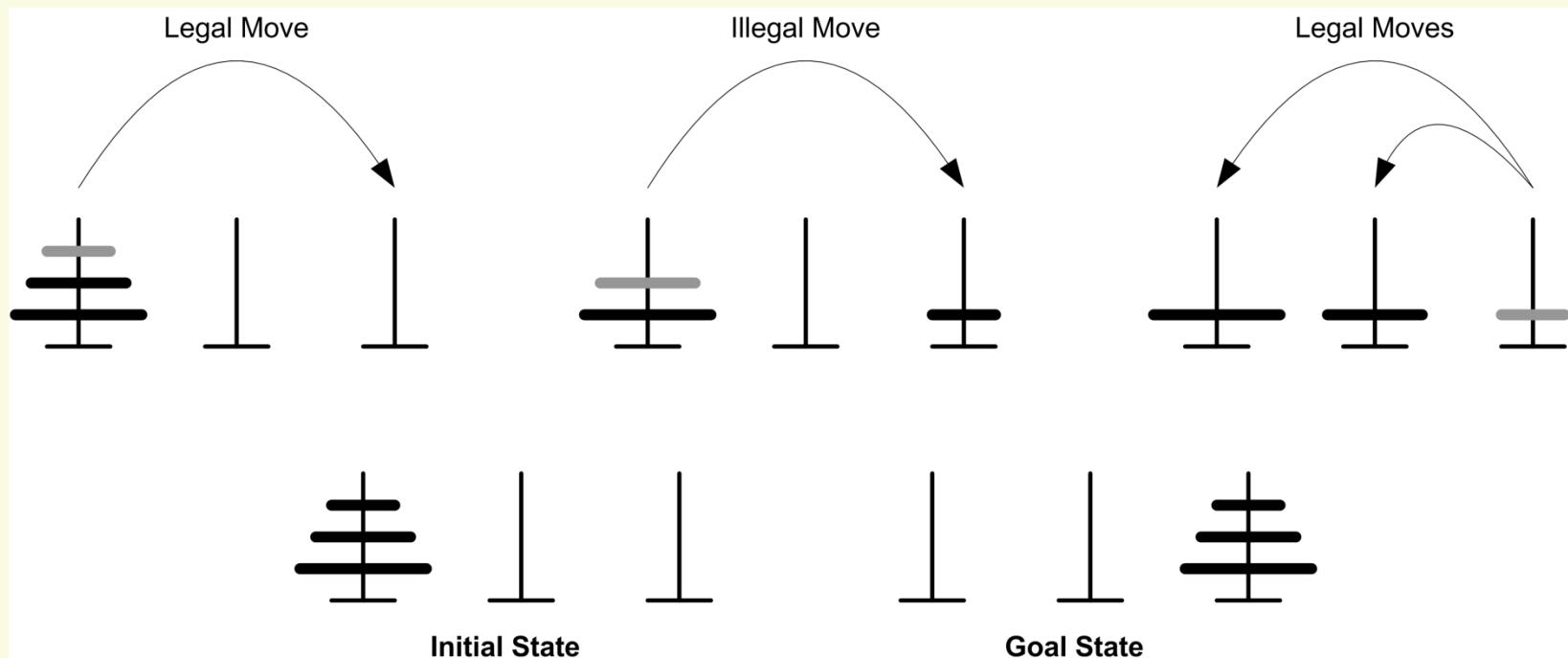
Towers of Hanoi

There are n (say 60) disks, A, B, C, ... of graduated sizes. There are also three pegs, 1, 2, and 3. Initially the disks are stacked on peg 1, with the smallest on top and then the 2nd smallest at the 2nd position from the top, and so on, with the largest at the bottom. The problem is to transfer the stack to peg 3, as in the figure, given that (a) only one disk can be moved at a time and (b) no disk may be placed on top of a smaller disk at all time.

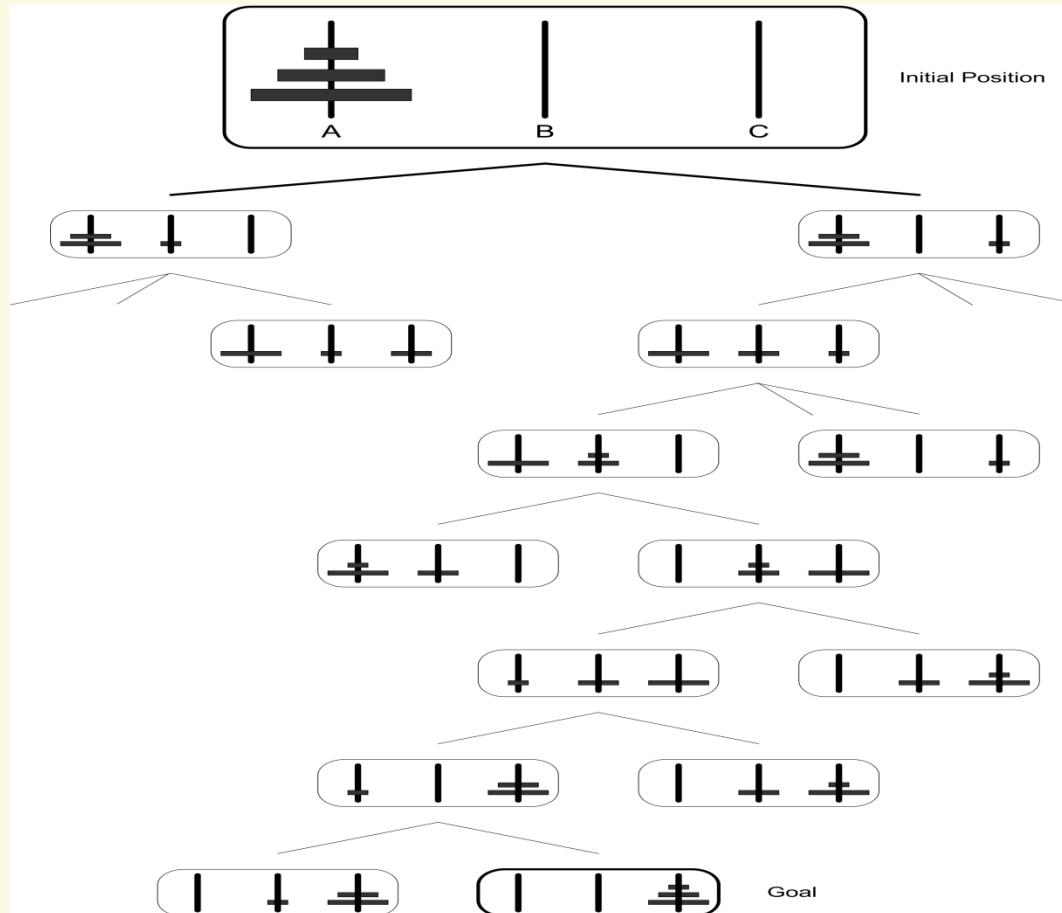


The Tower of Hanoi puzzle

Legal Moves



Towers of Hanoi

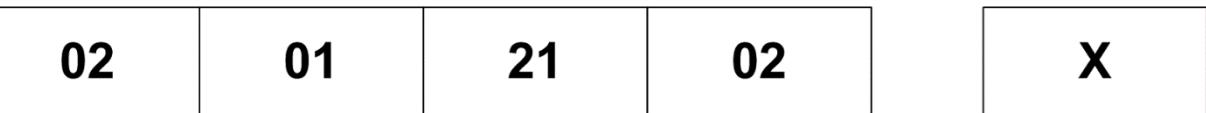


Move Representation

Solution Sequence



Move Disc from
Peg 0 to Peg 2



Move

0

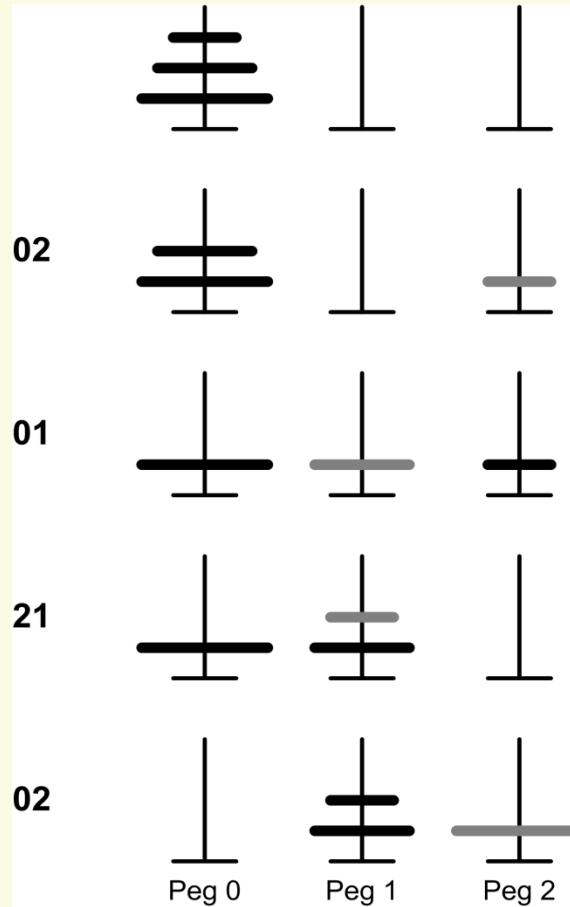
1

2

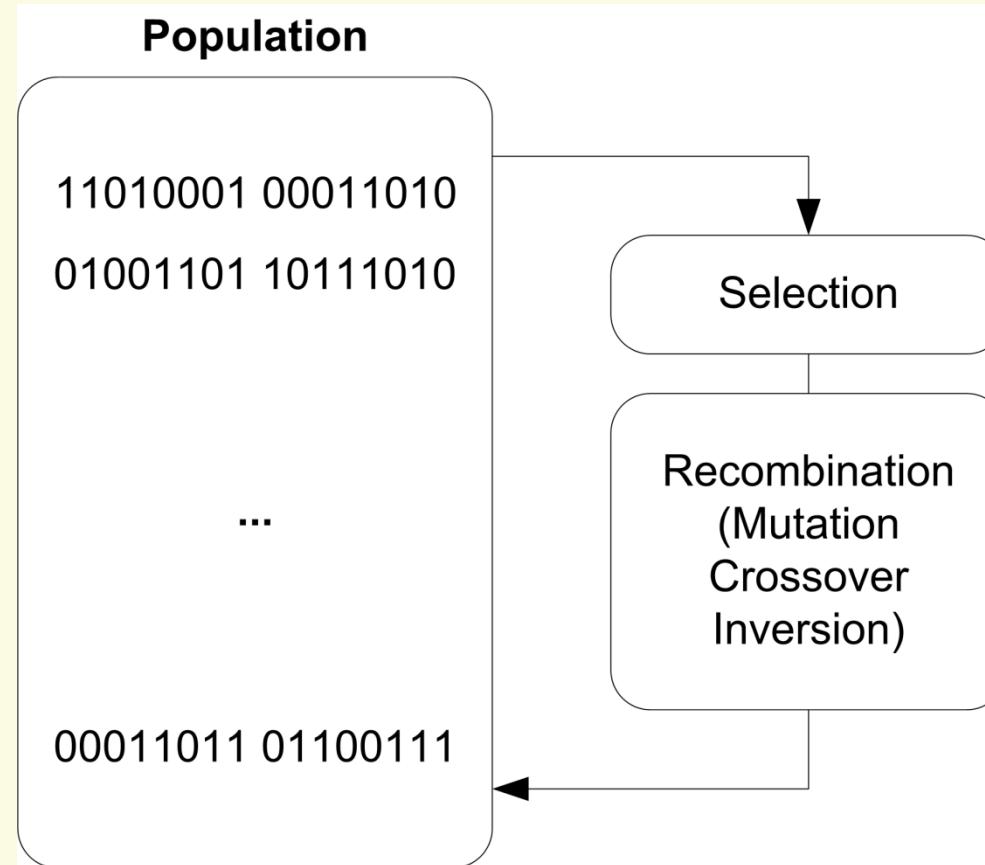
3

N-1

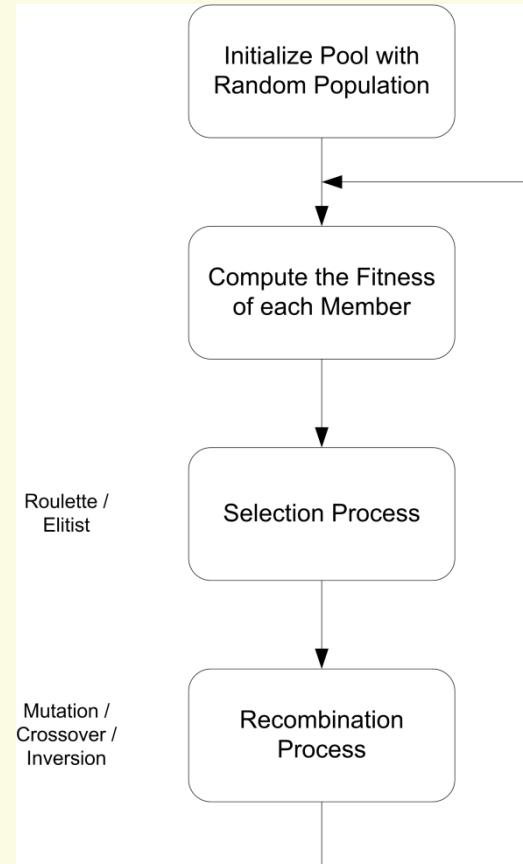
Corresponding Moves



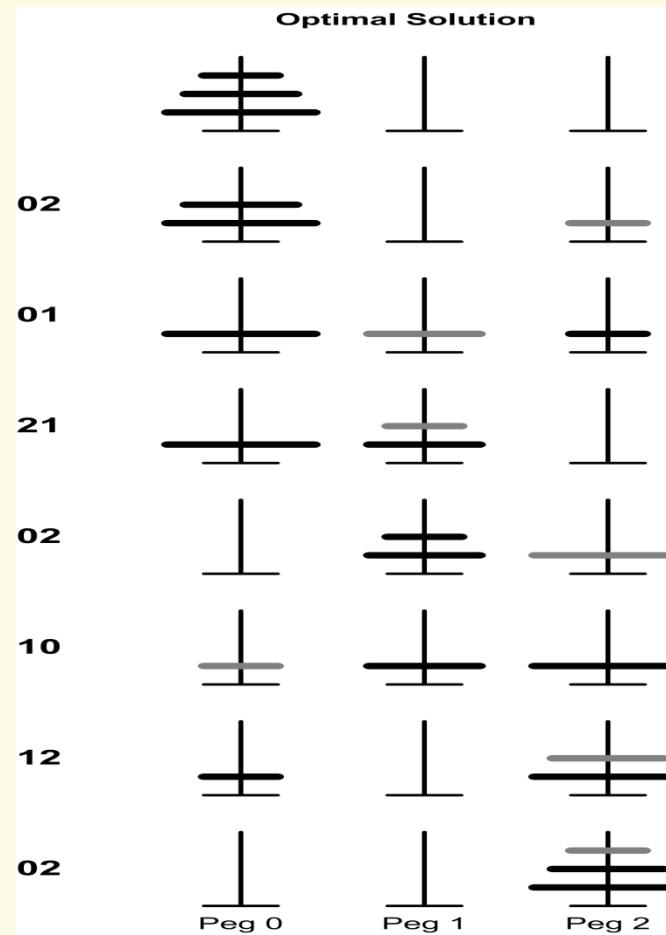
GA Operations



GA Flow Chart



Optimization Process



Encoder-Decoder Design

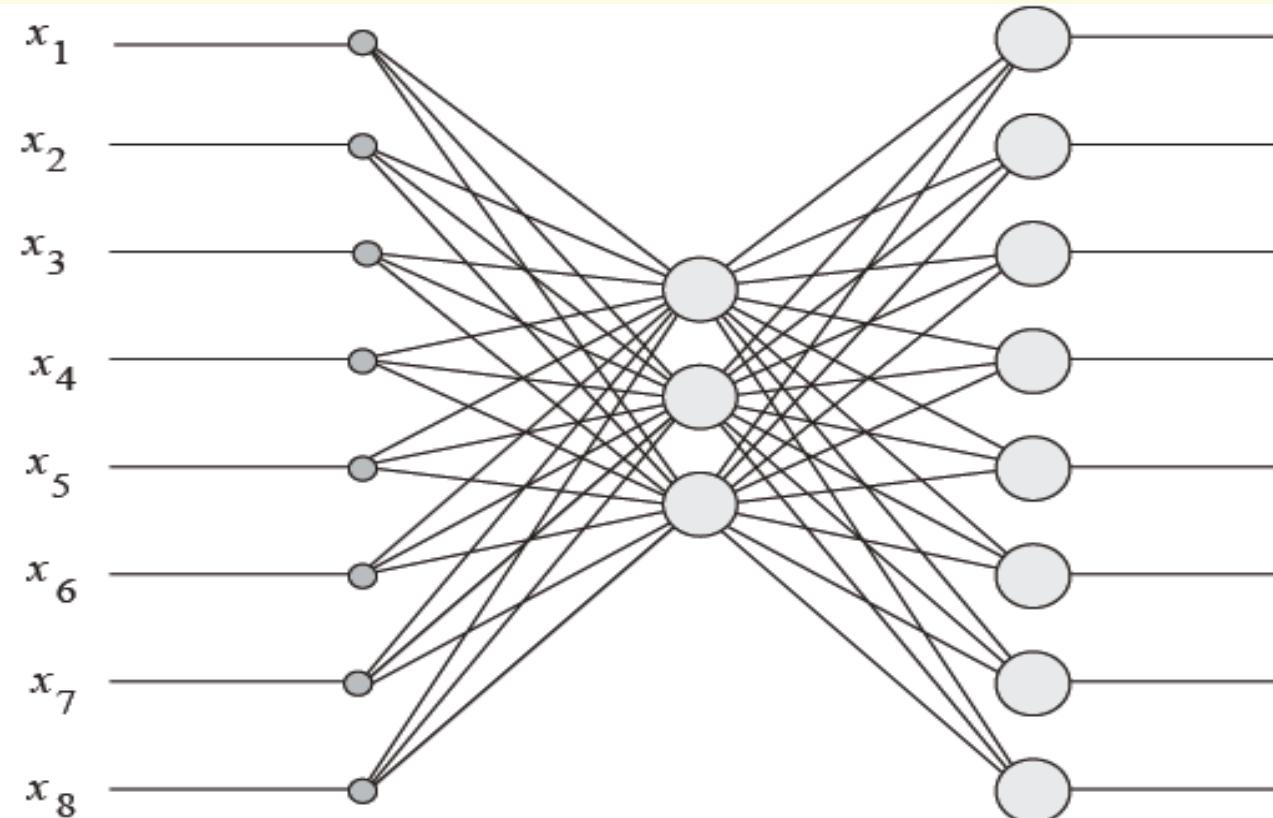


Fig. 17.8. An 8-bit encoder-decoder

Encoder-Decoder Design

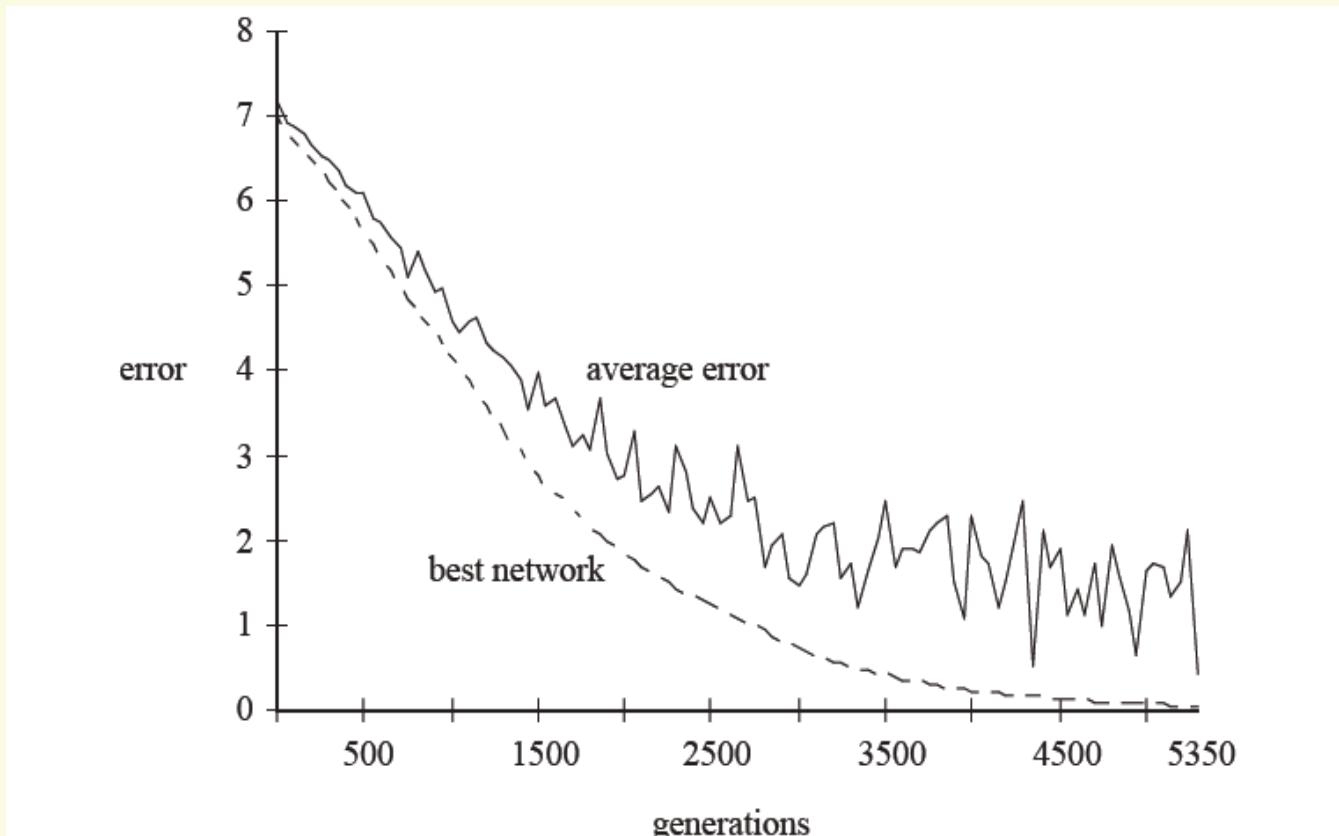
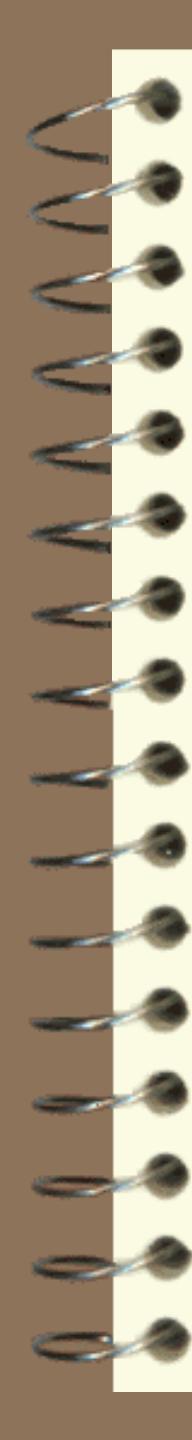
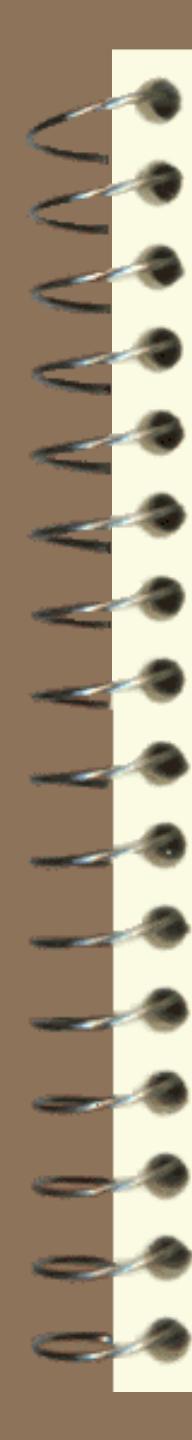


Fig. 17.9. Error function at each generation: population average and best network



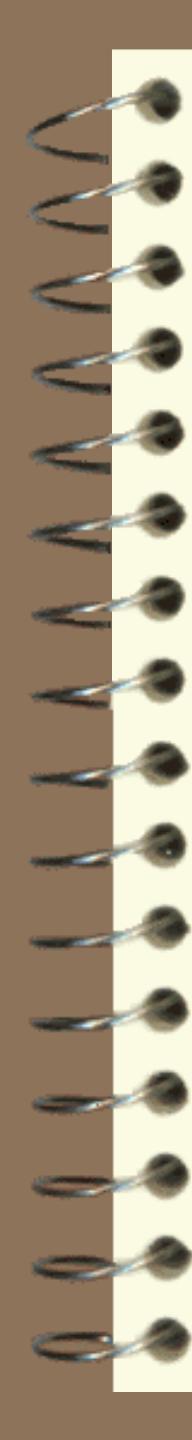
Swarm Intelligence

- Initialized by Gerardo Beni and Jing Wang in 1989 in the context of cellular robotic systems
- Typically made up of a population of simple agents interacting locally with one another and with their environment
- Typical representatives include particle swarm optimization, ant colony optimization, etc.



Particle Swarm Optimization

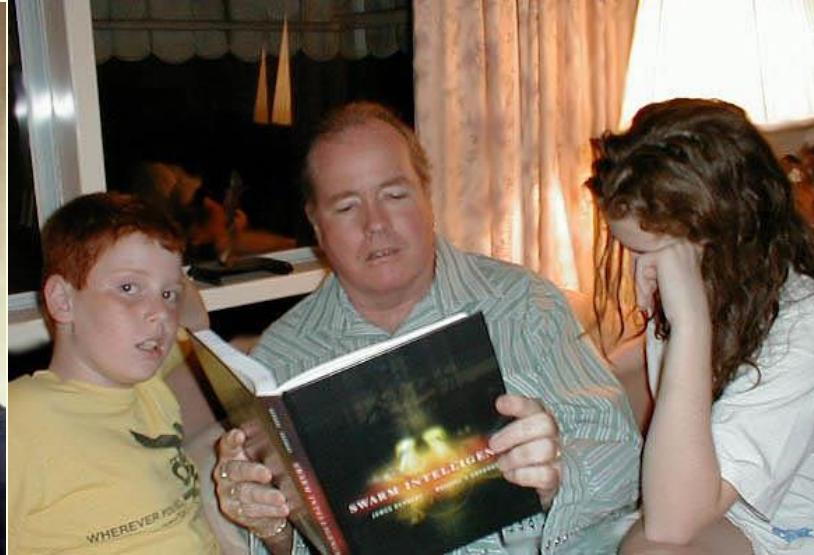
- A robust stochastic optimization technique based on the movement and intelligence of swarms
- Applies the concept of social interaction to problem solving
- It uses a number of agents (particles) that constitute a swarm moving around in the search space looking for the best solution



Particle Swarm Optimization

- Each particle is treated as a point in a multi-dimensional space which adjusts its “flying” according to its own flying experience as well as the flying experience of other particles
- Developed in 1995 by James Kennedy (social-psychologist) and Russell Eberhart (electrical engineering professor).

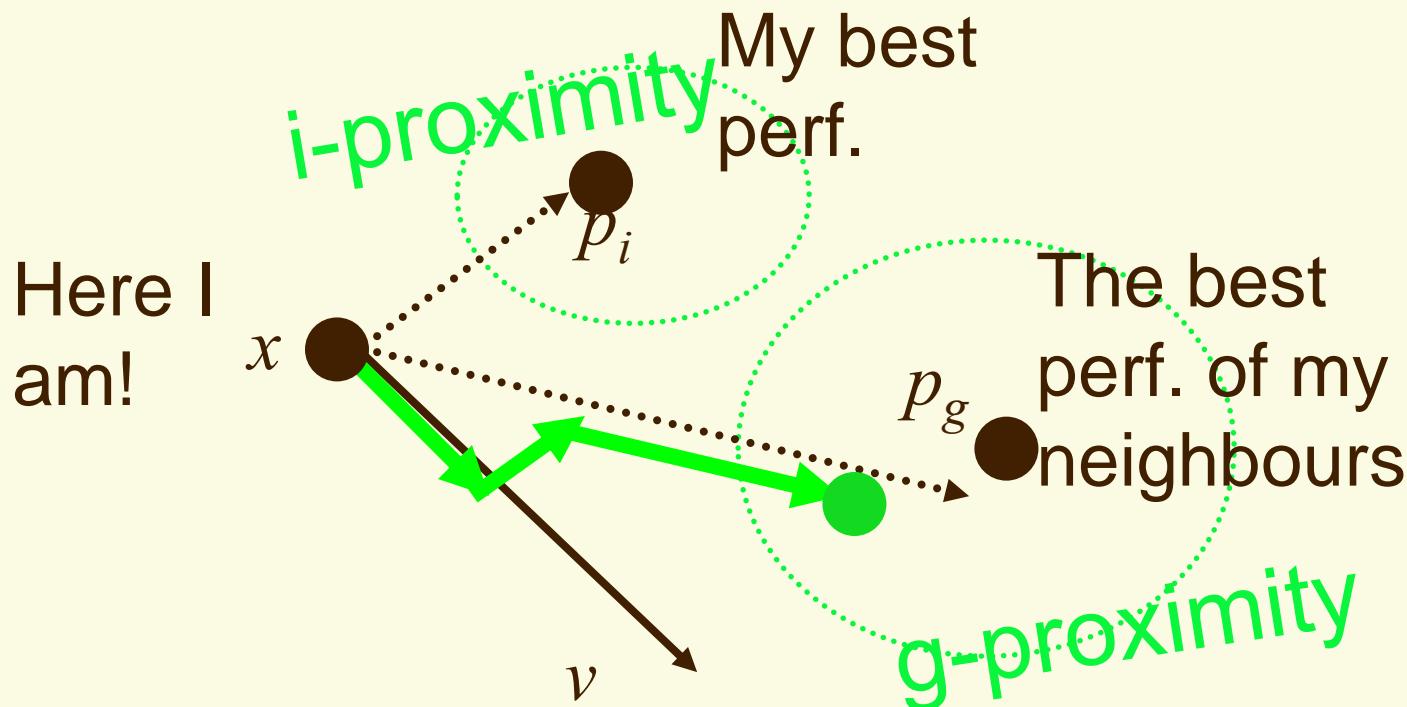
Fathers of PSO



Russ Eberhart

James Kennedy

Psychosocial Compromise



Search Direction

At each time step t
for each particle

for each component i

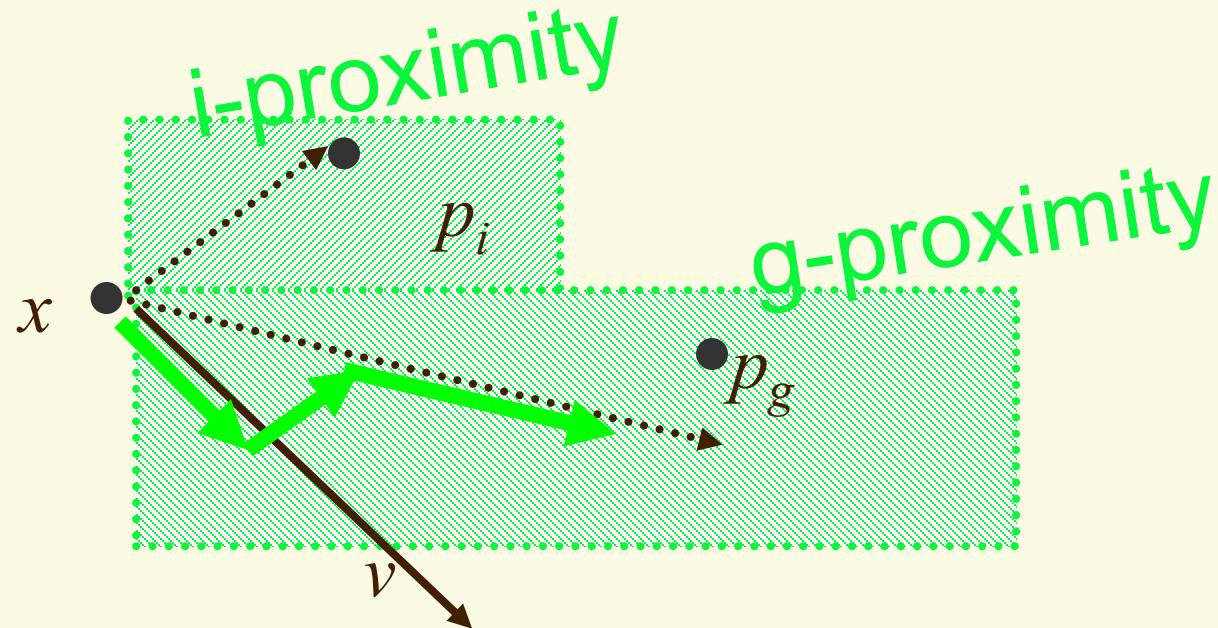
update the velocity

$$v_k(t+1) = \begin{cases} \alpha v_k(t) \\ + \beta rand(0, \varphi_1)(p_{ik} - x_k(t)) \\ + \beta rand(0, \varphi_2)(p_{gk} - x_k(t)) \end{cases}$$

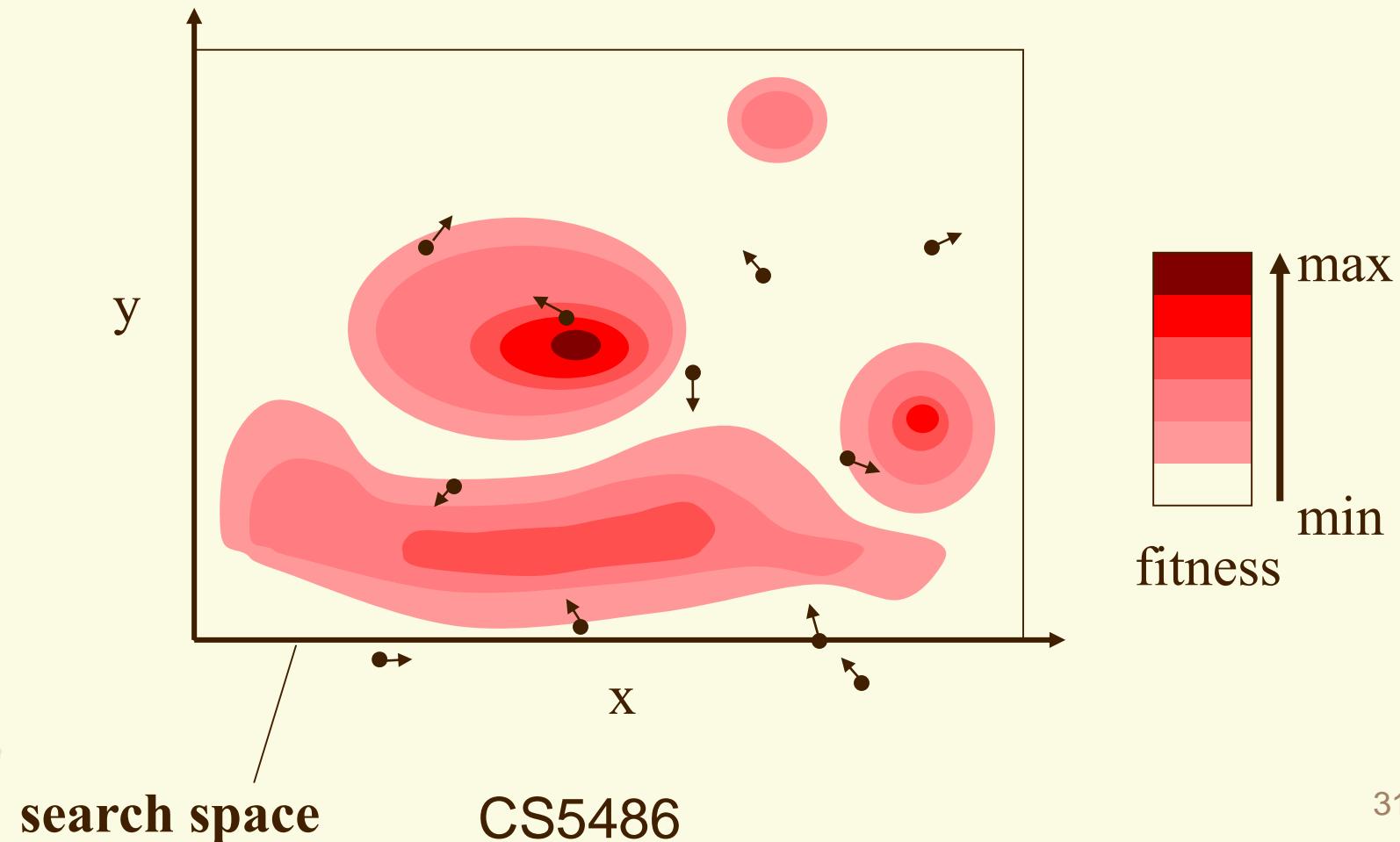
Randomness
inside the
loop

then move $x(t+1) = x(t) + v(t+1)$

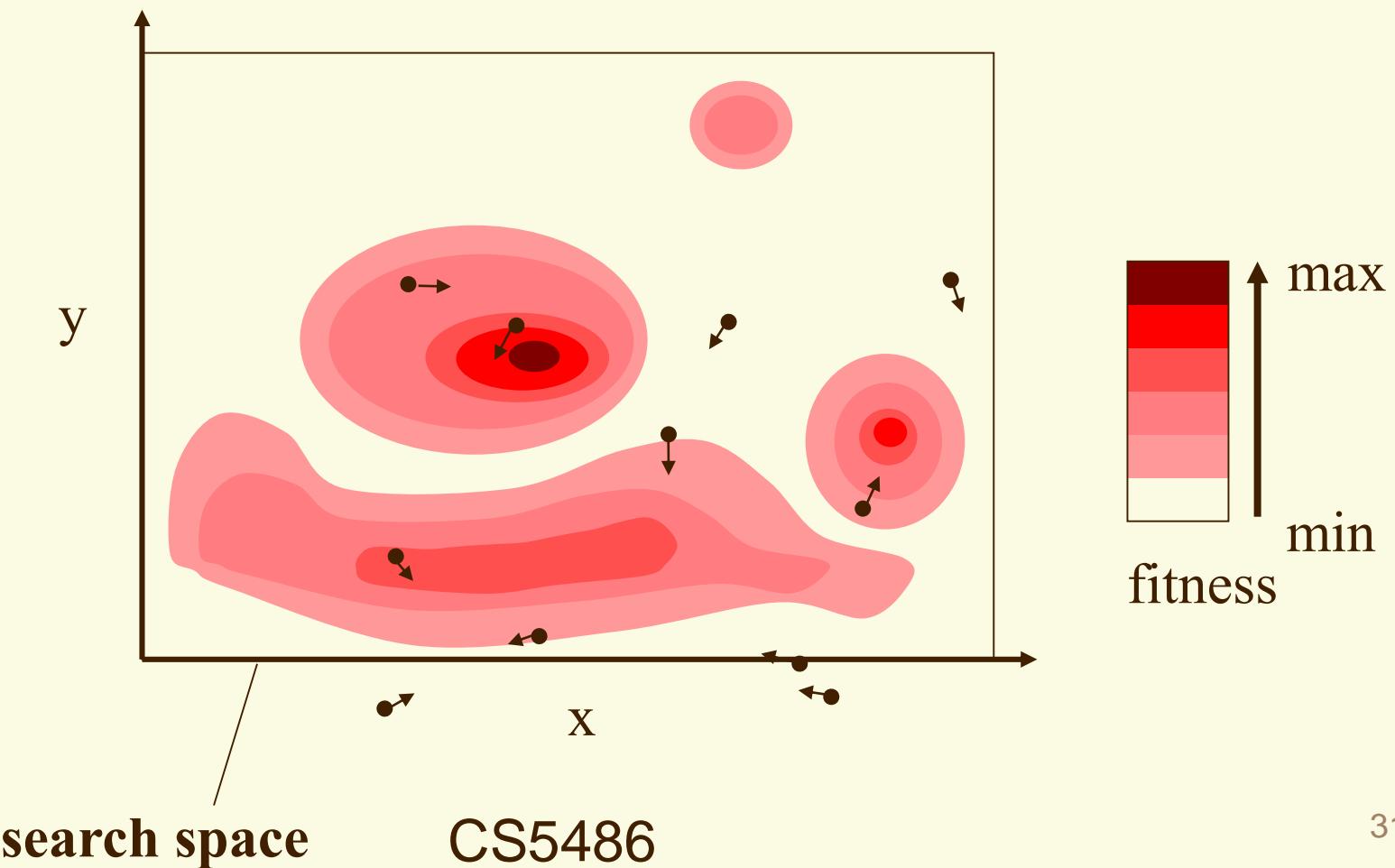
Random Proximity



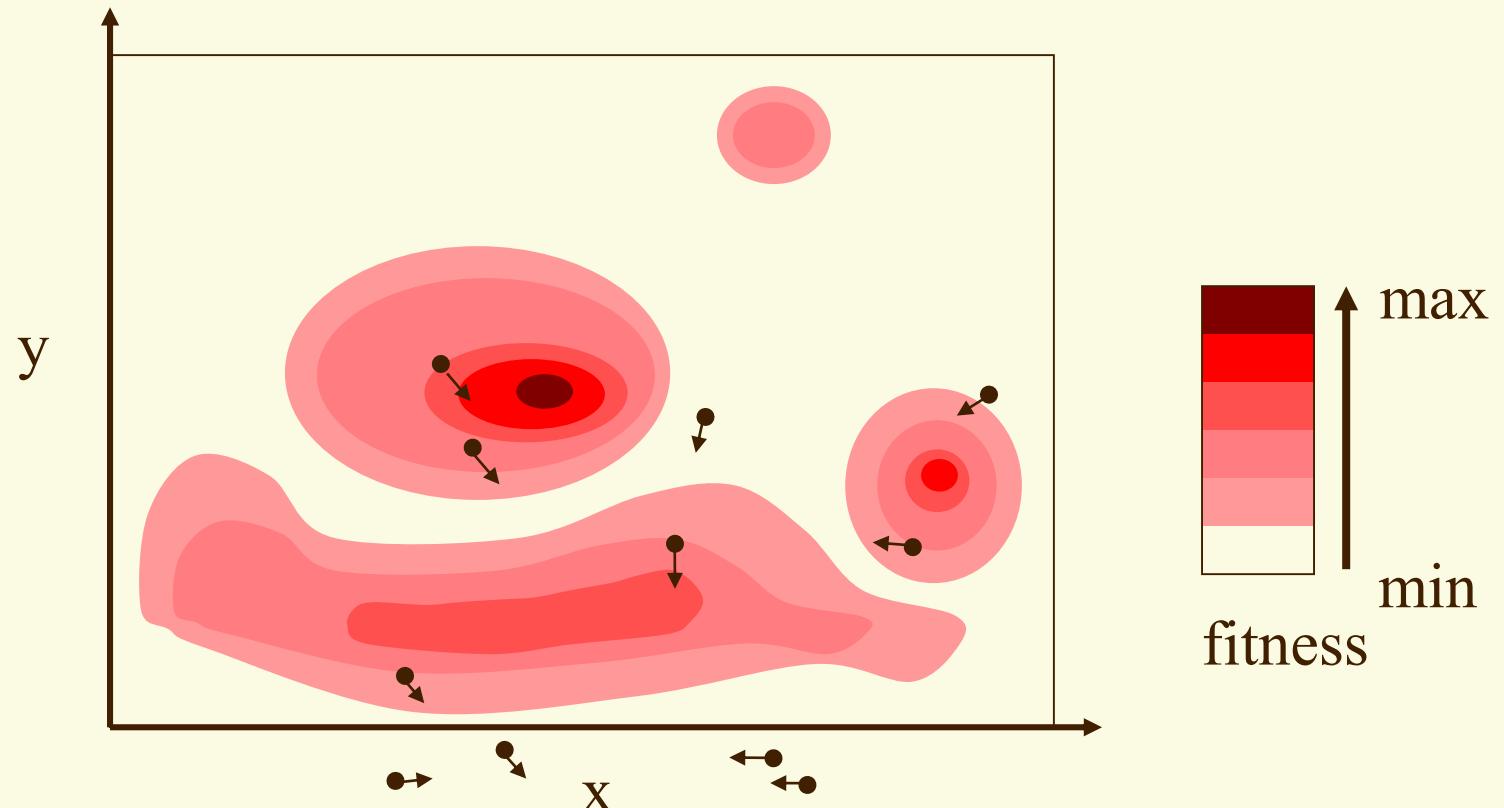
Global Optimization



Global Optimization



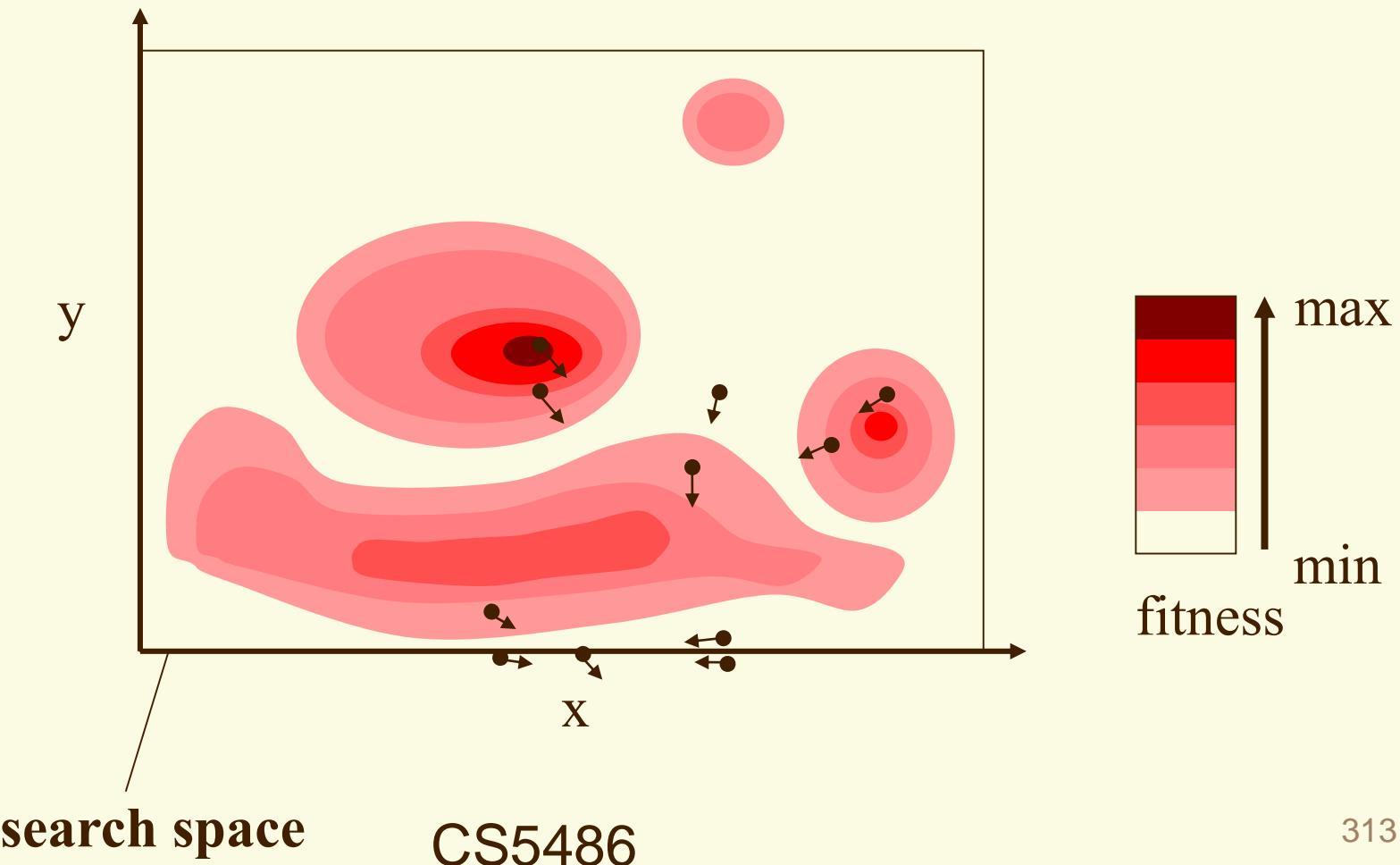
Global Optimization



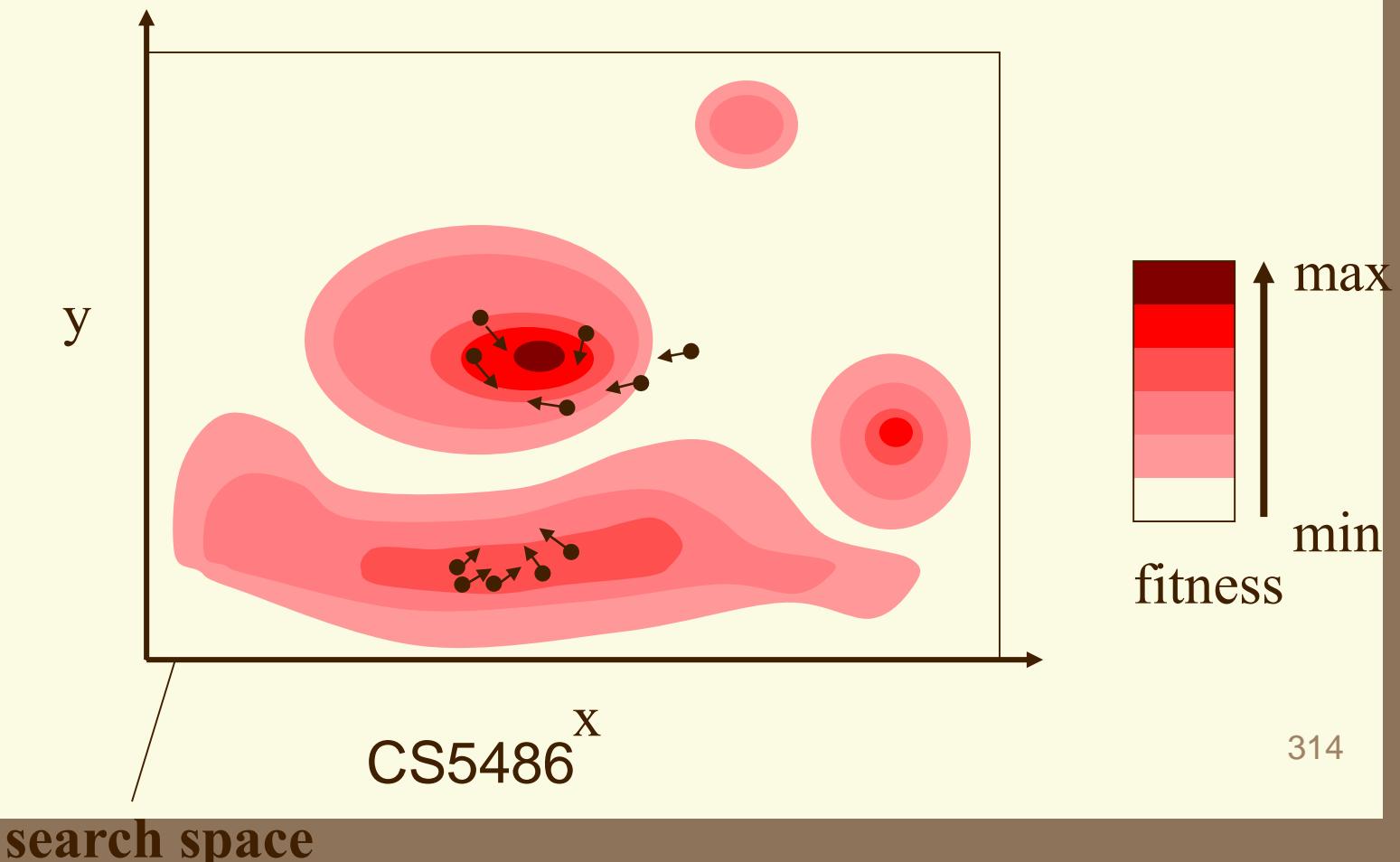
search
space

CS5486

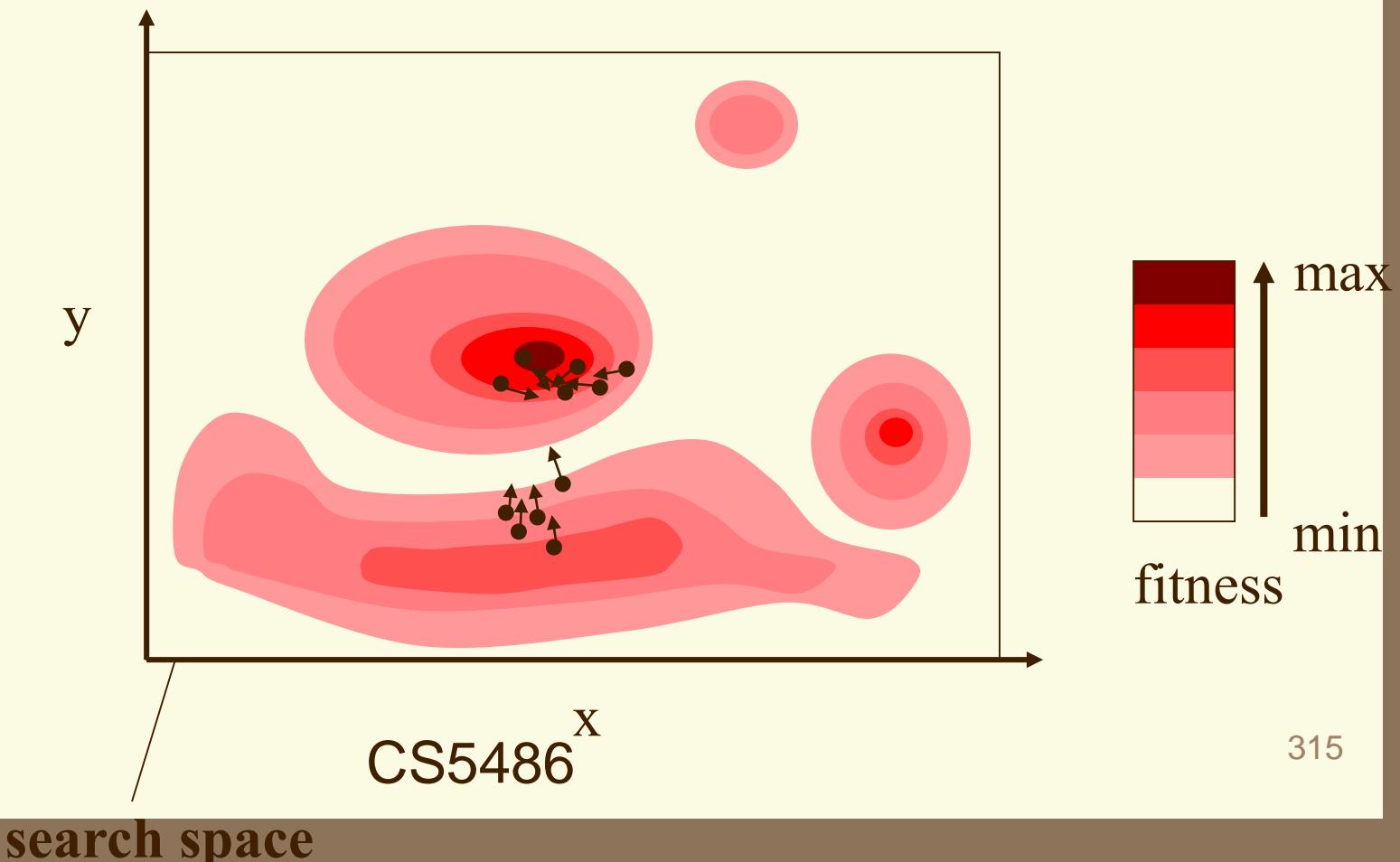
Global Optimization



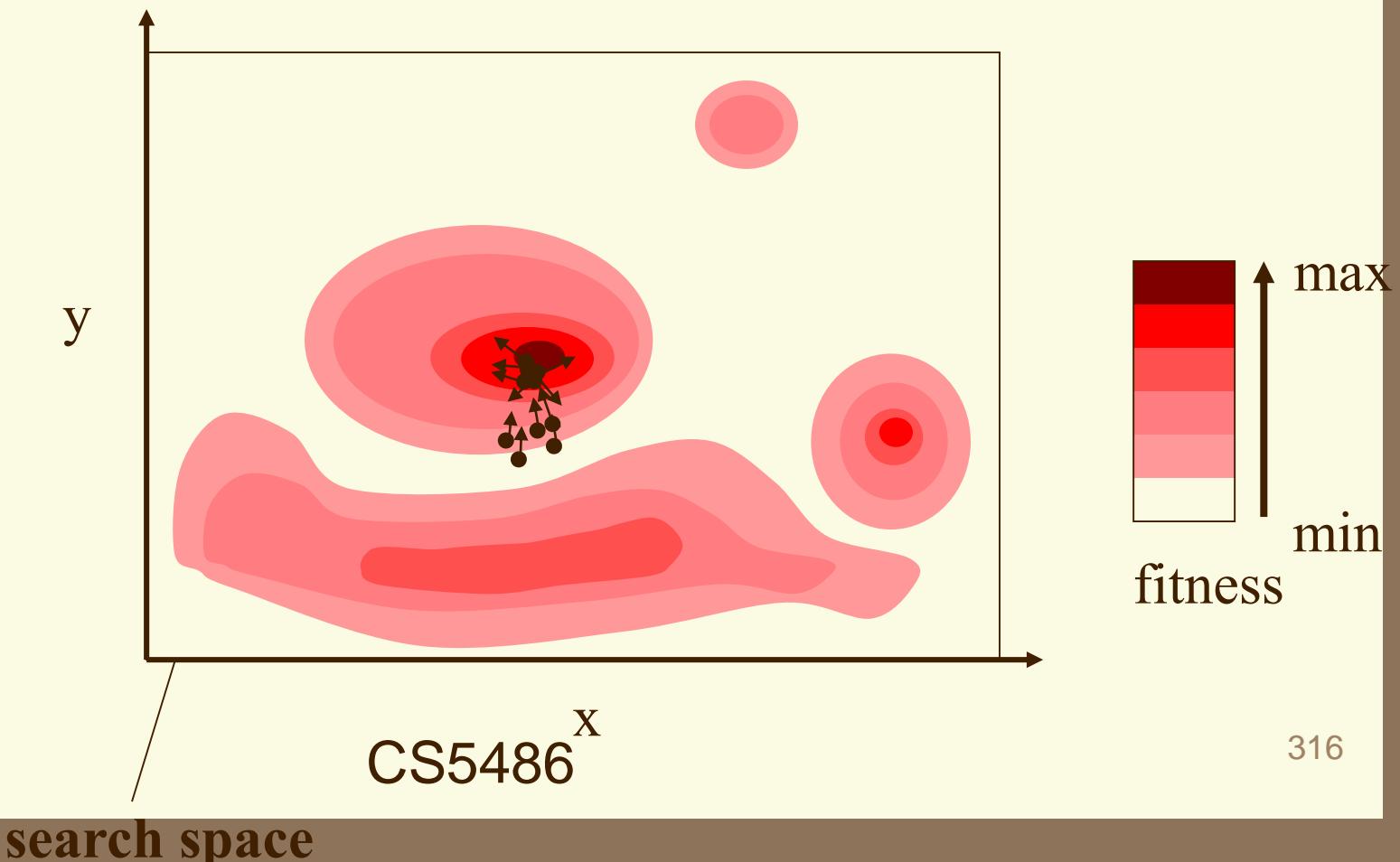
Global Optimization



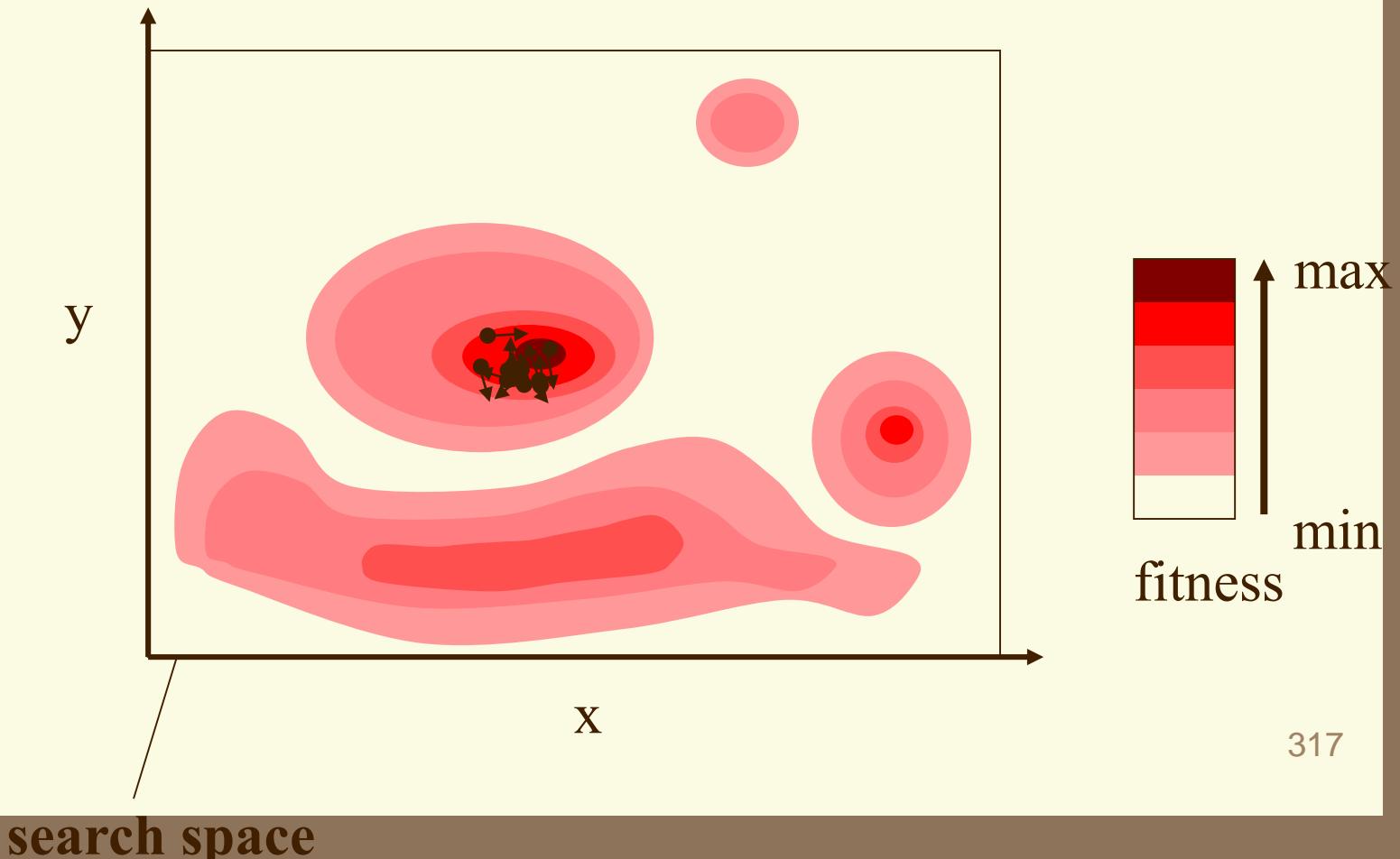
Global Optimization



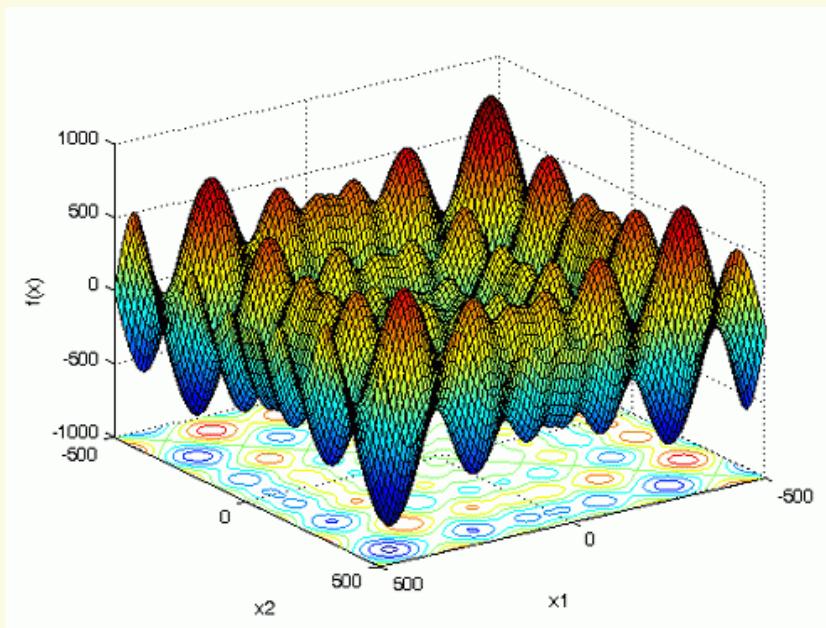
Global Optimization



Global Optimization



Schwefel's function



$$f(x) = \sum_{i=1}^n x_i \sin(\sqrt{|x_i|})$$

where

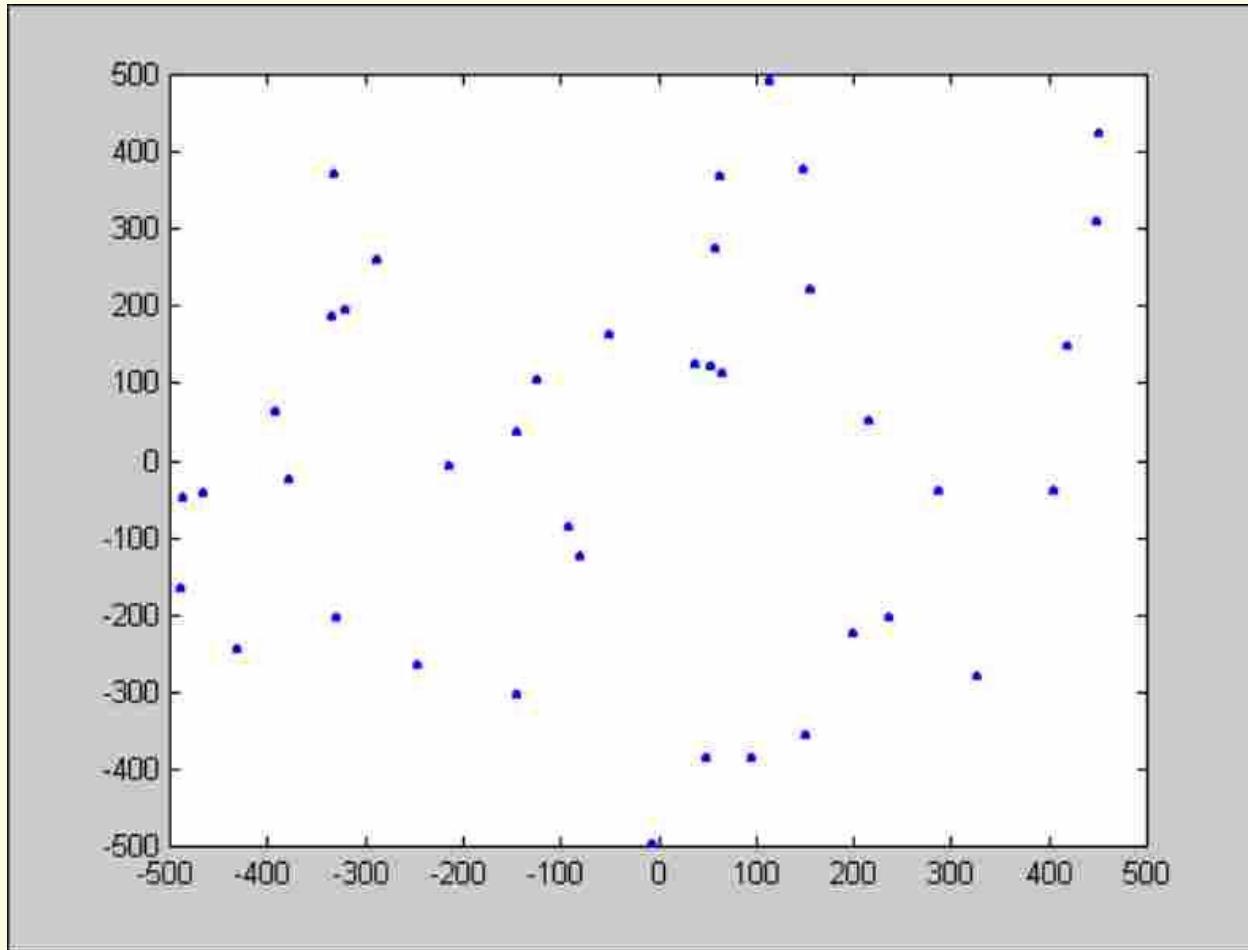
$$-500 \leq x_i \leq 500$$

Global minimum

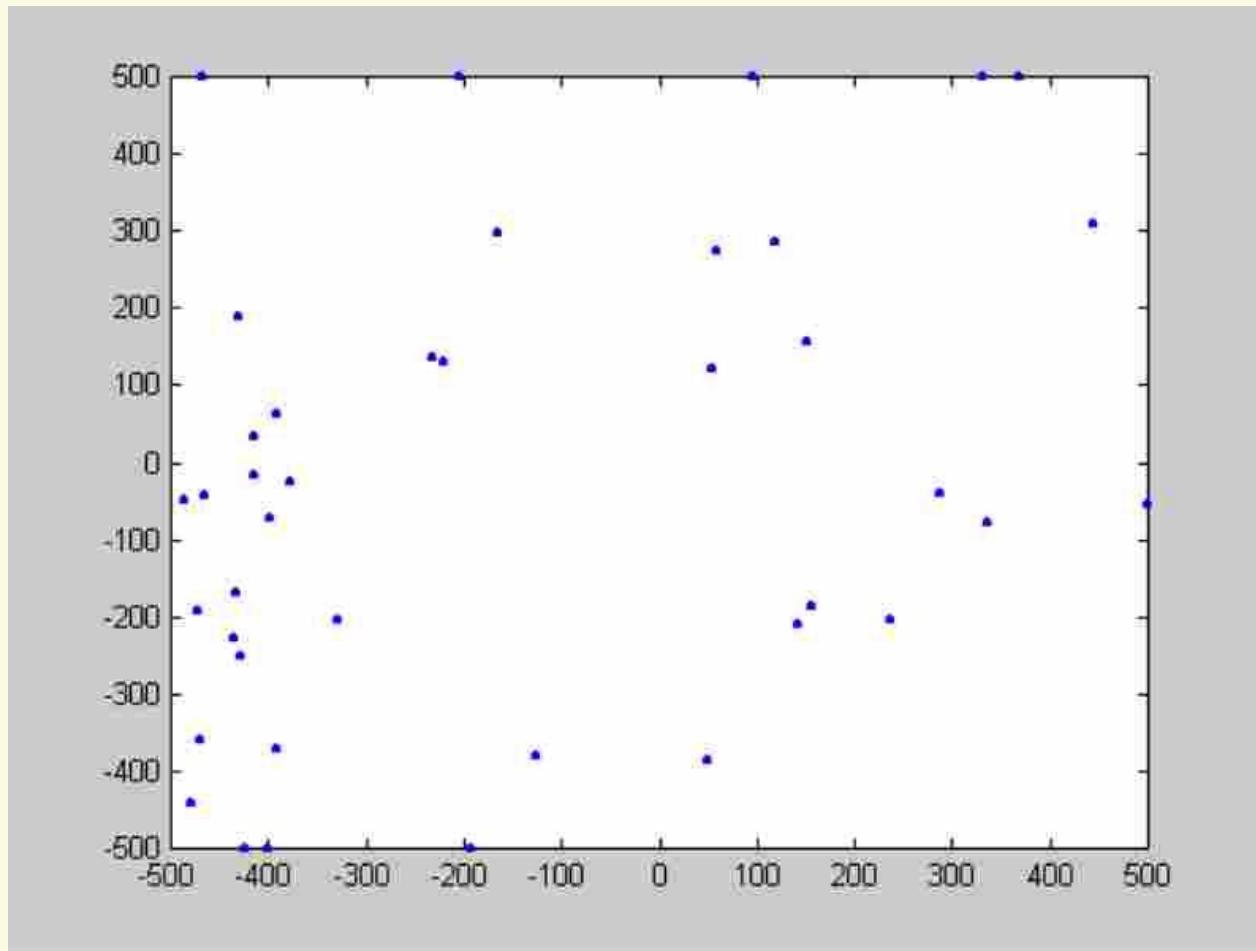
$$f(x^*) = -418.9829n;$$

$$x_i = -420.9687, i=1,2,\dots,n$$

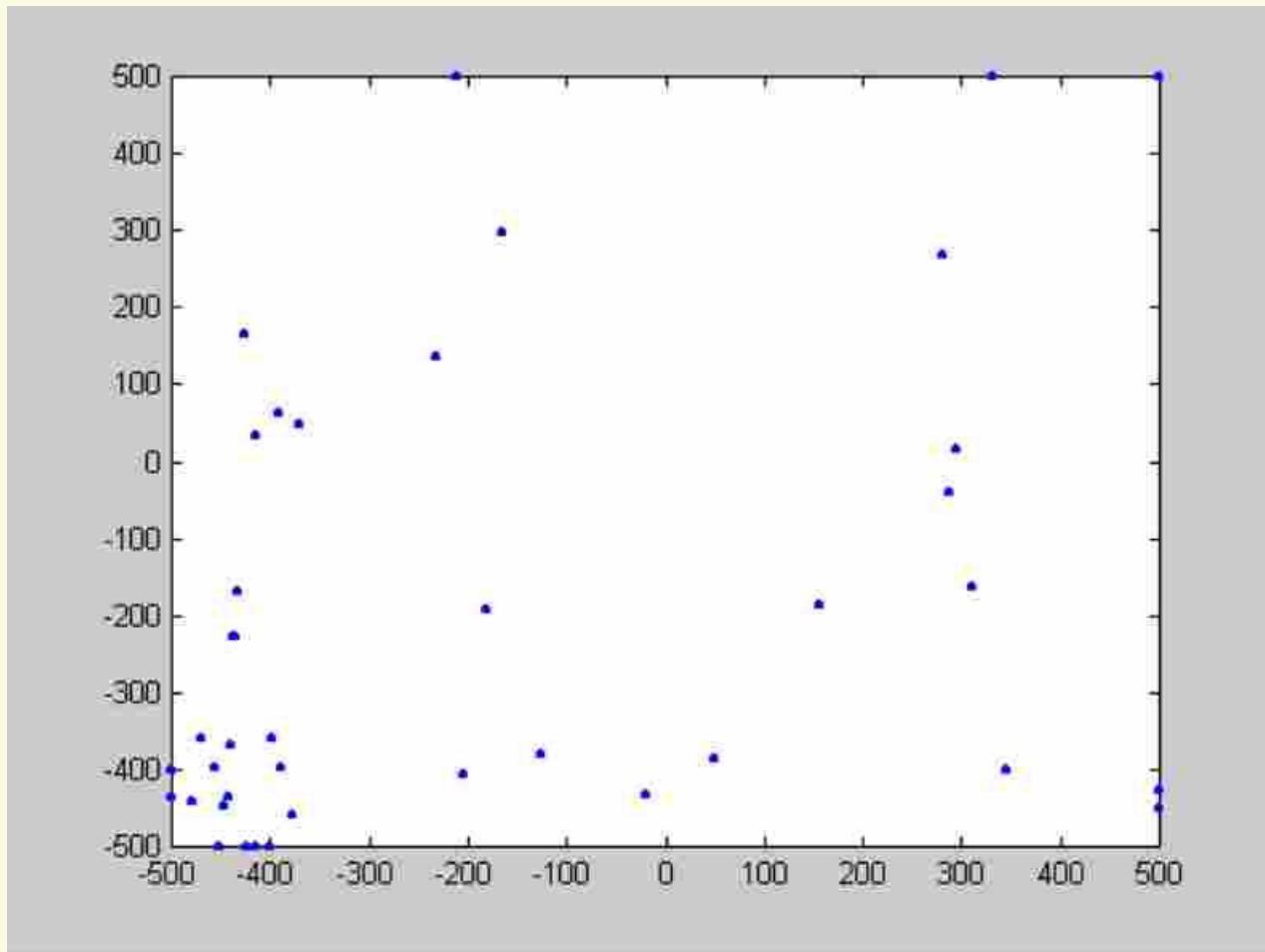
Initialization Swarm



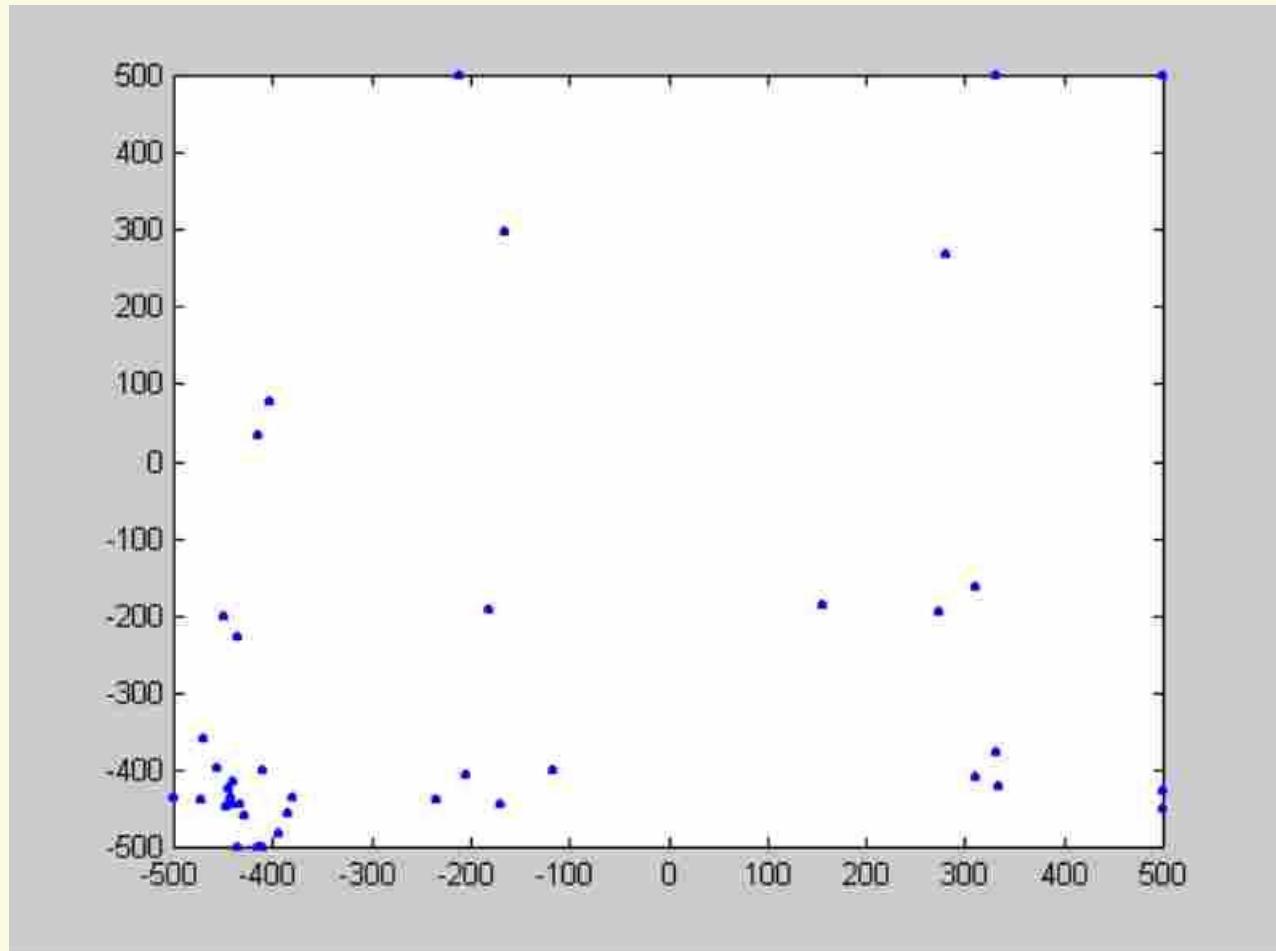
Evolution after 5 Iterations



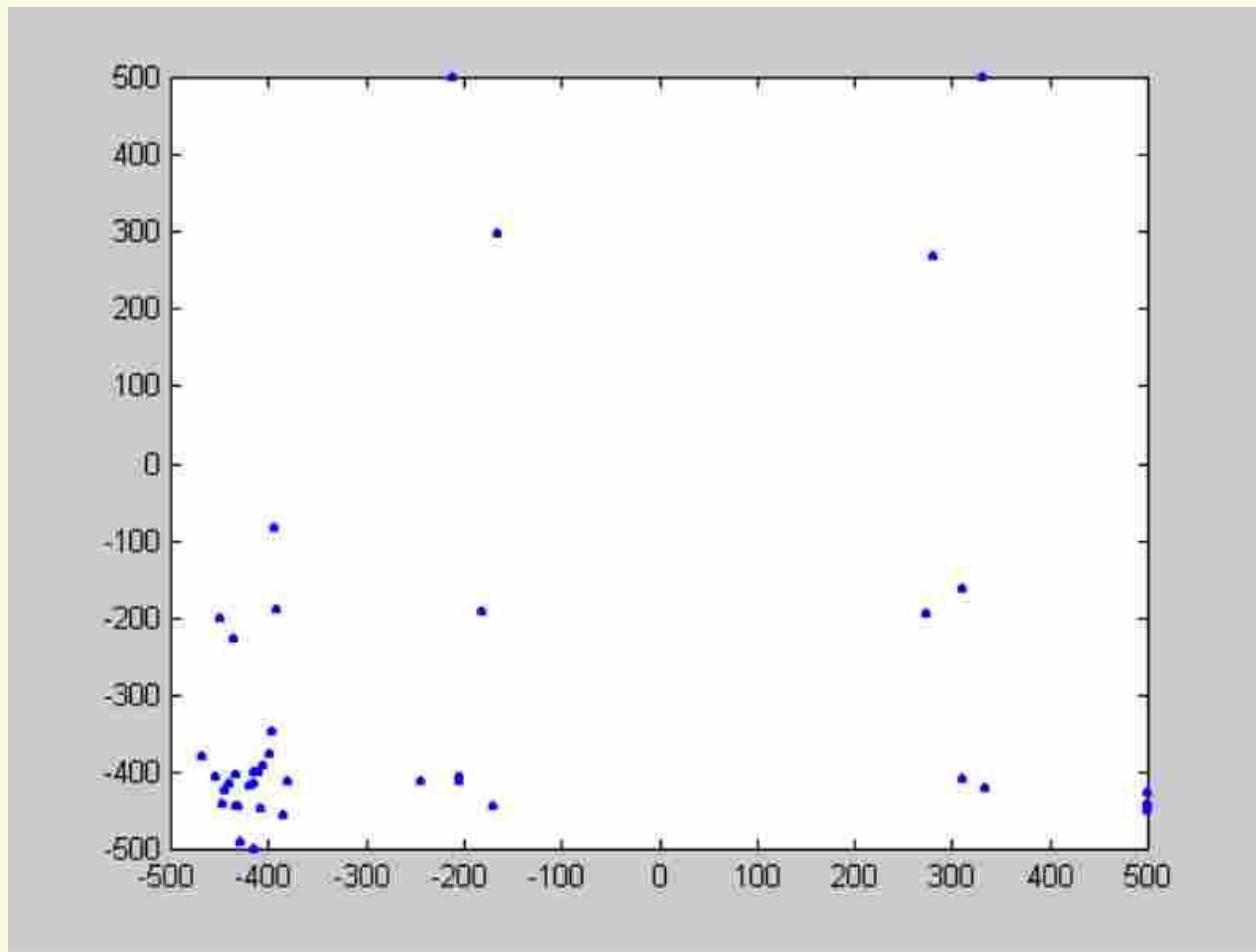
Evolution after 10 Iterations



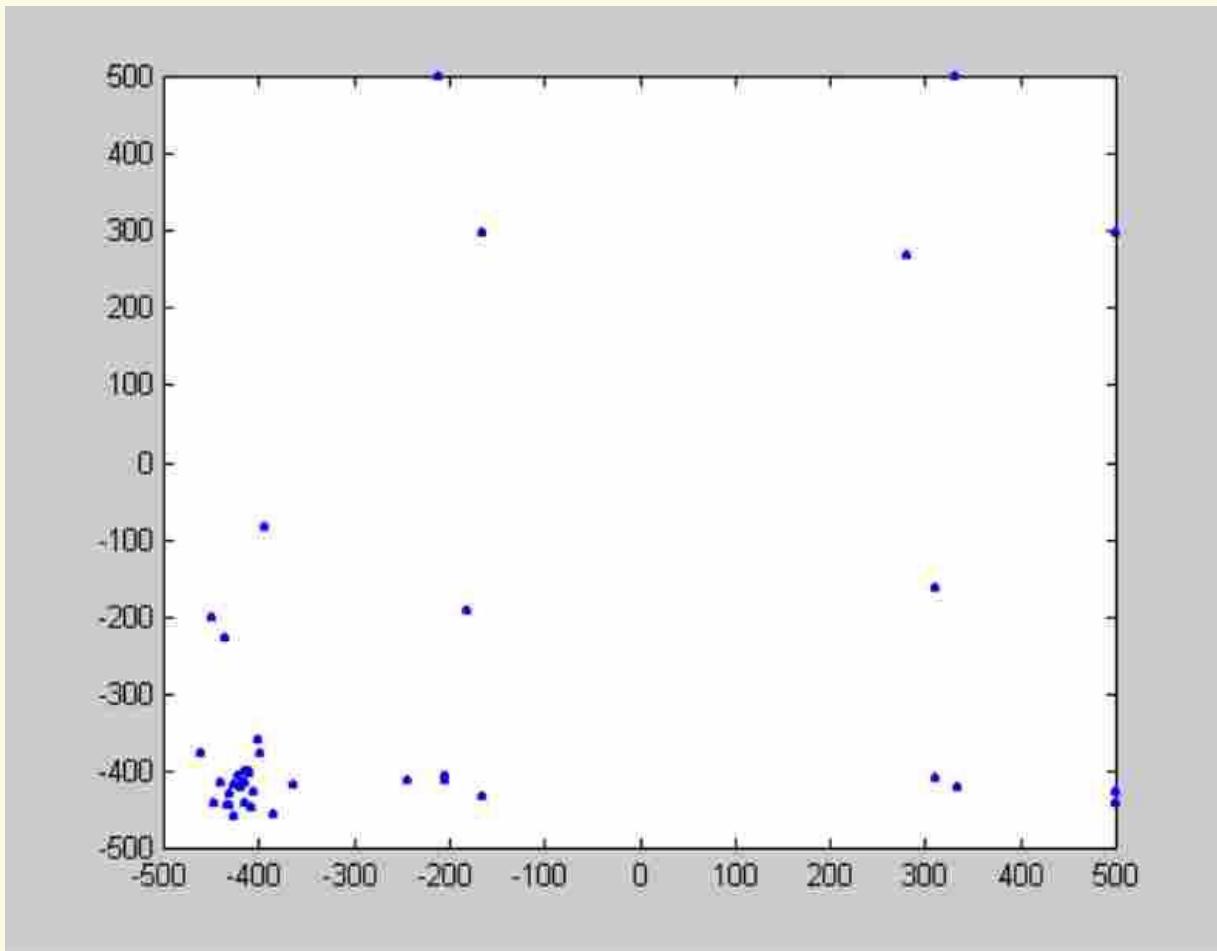
Evolution after 15 Iterations



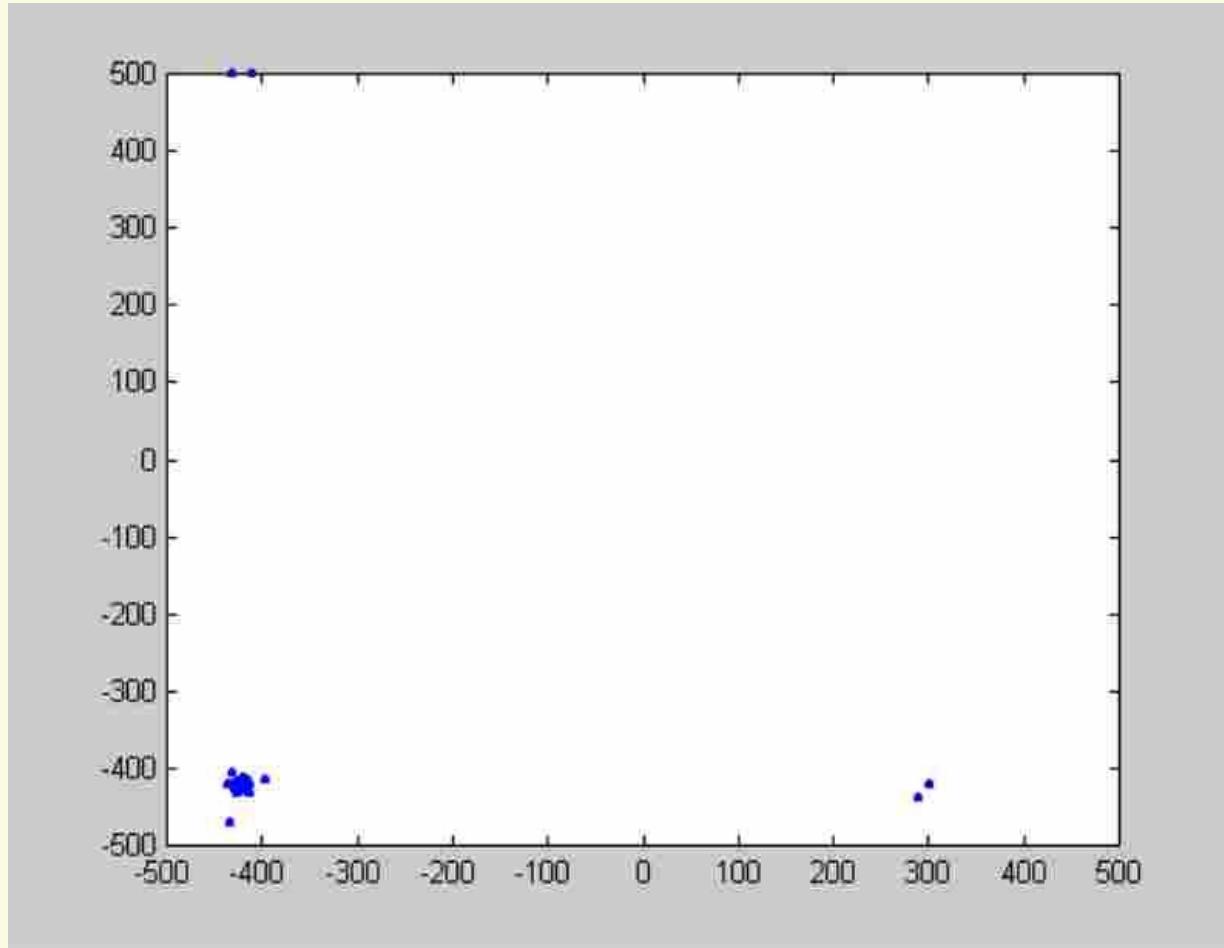
Evolution after 20 Iterations



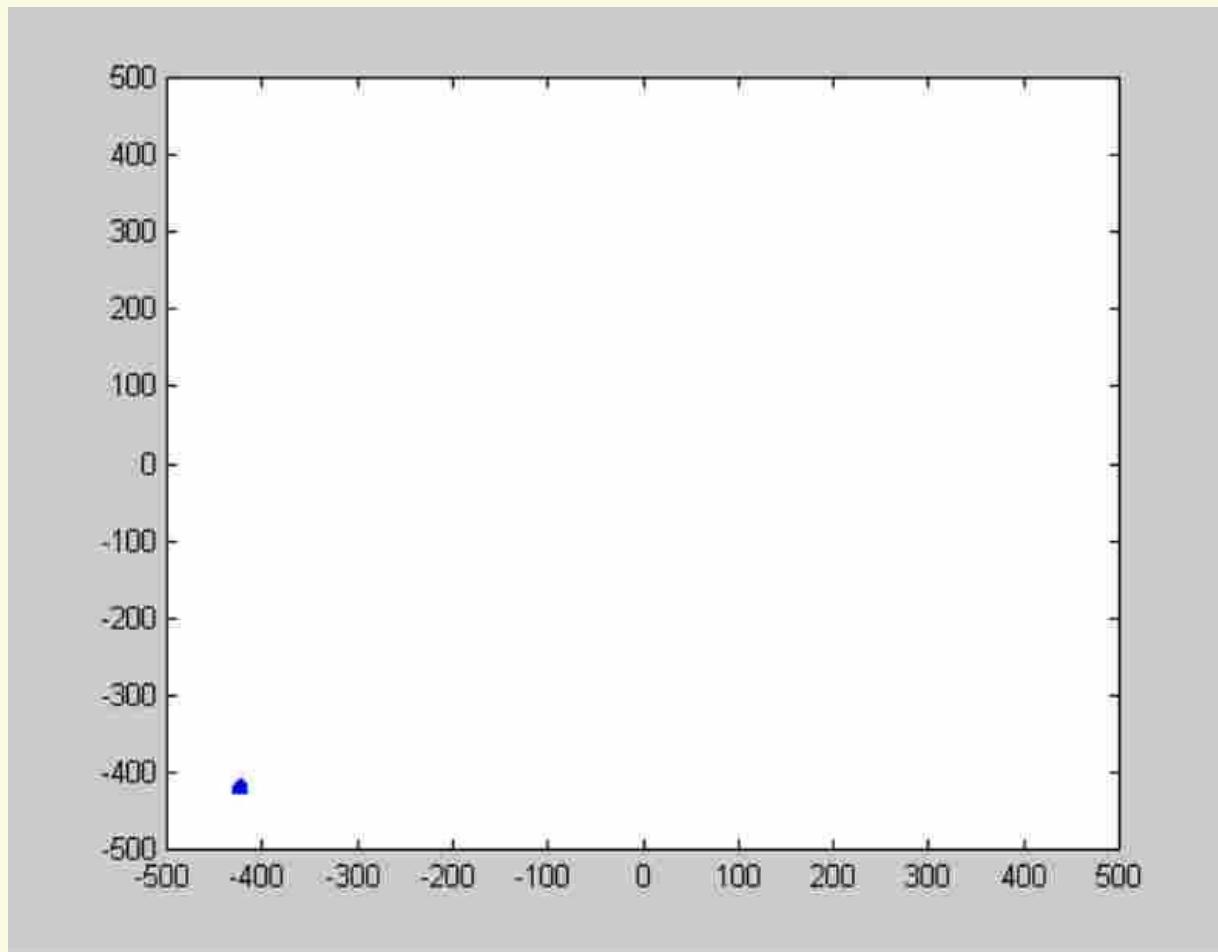
Evolution after 25 Iterations



Evolution after 100 Iterations

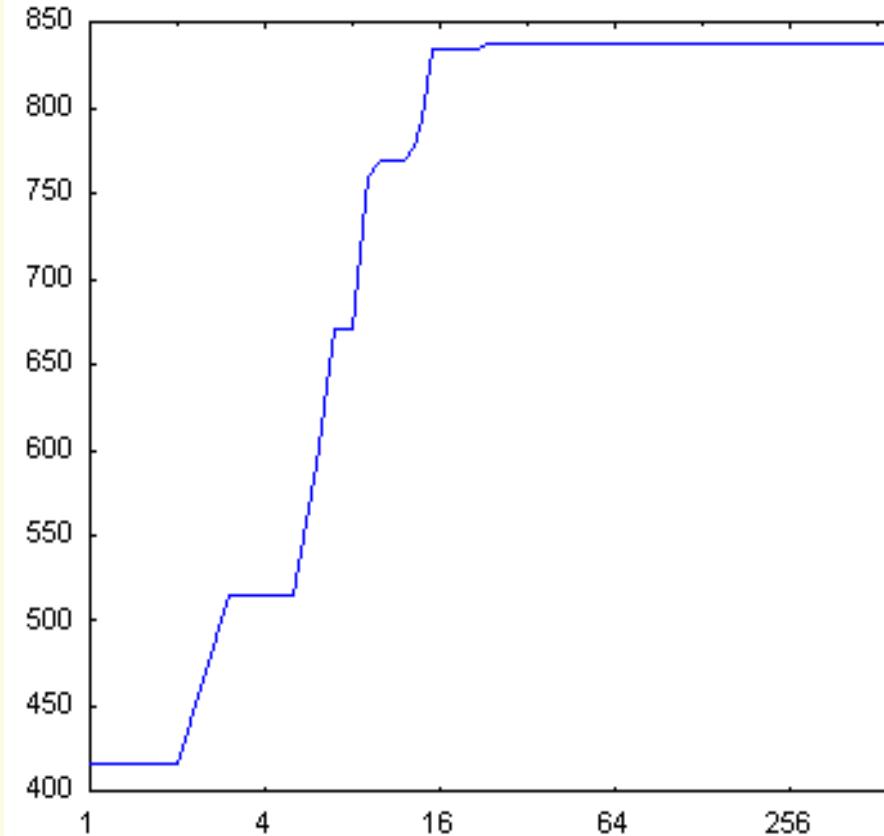


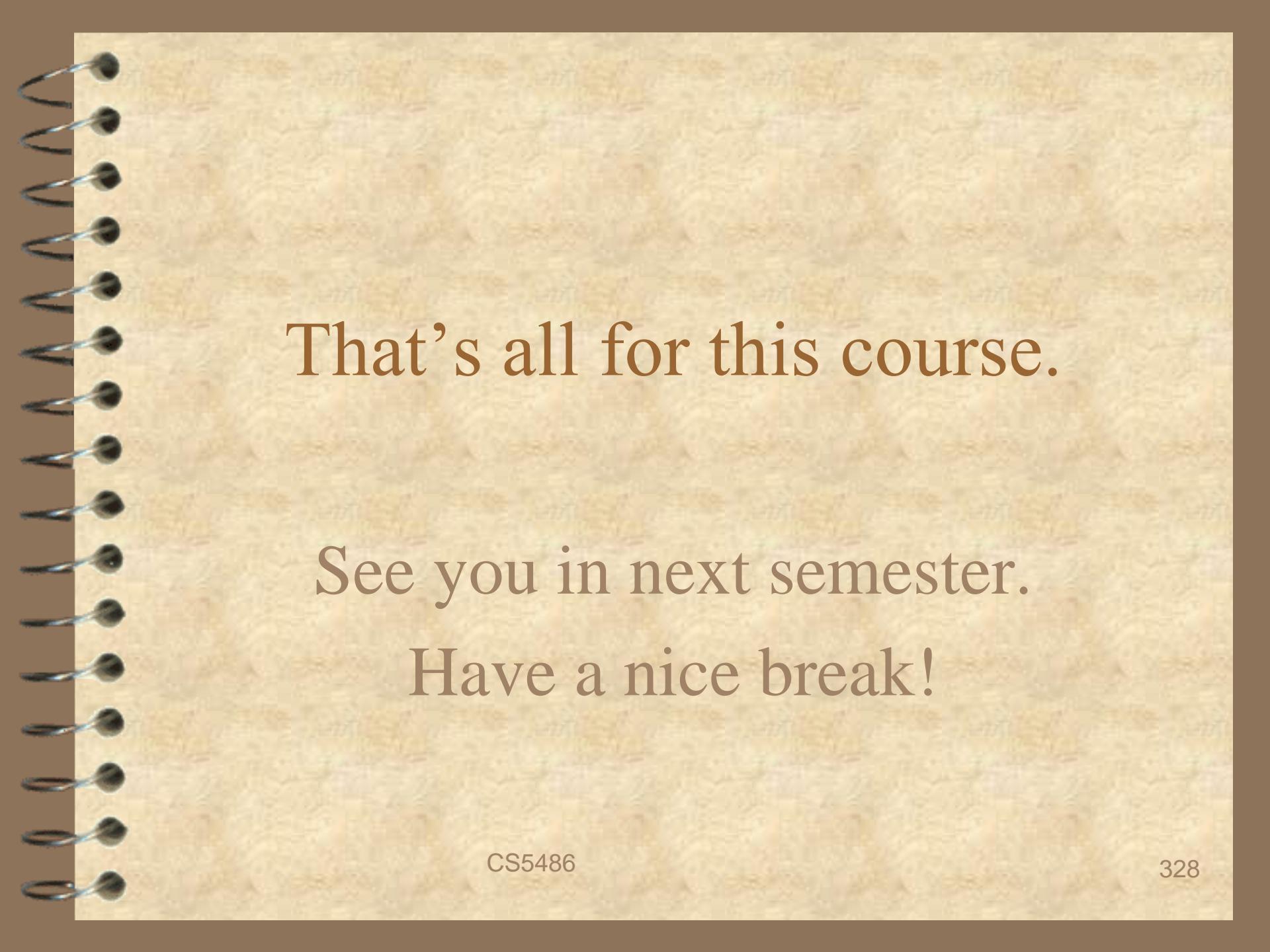
Evolution after 500 Iterations



Search Results

Iteration	Swarm best
0	416.245599
5	515.748796
10	759.404006
15	793.732019
20	834.813763
100	837.911535
5000	837.965771
Global	837.9658





That's all for this course.

See you in next semester.

Have a nice break!