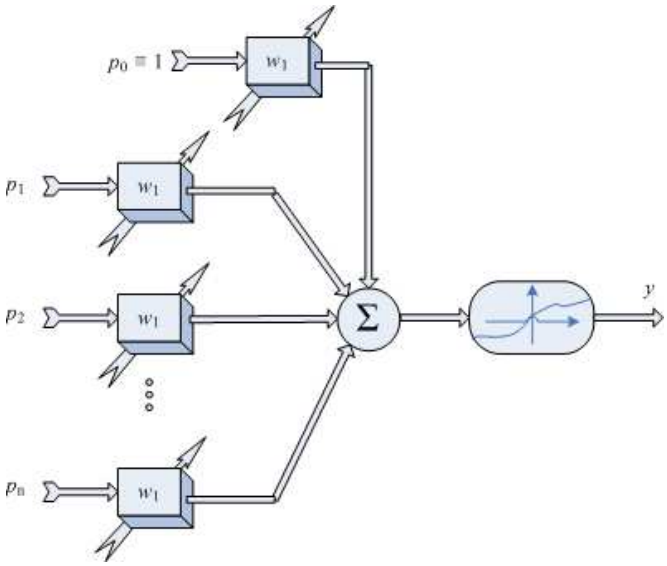
Engenharia
Elétrica
Universidade de Brasília



Introdução aos Sistemas Inteligentes

Aplicações em Engenharia de
Redes Neurais Artificiais, Lógica Fuzzy e
Sistemas Neuro-Fuzzy

Adolfo Bauchspiess

Brasília, novembro de 2008

RESUMO

Objetivo do Curso

O mini-curso **Sistemas Inteligentes – Redes Neurais e Lógica Fuzzy** tem como objetivo apresentar técnicas de projeto para a engenharia, ***Redes Neurais Artificiais*** e ***Lógica Fuzzy***, que se originaram de pesquisas na área de inteligência artificial.

Motivação

Redes Neurais Artificiais:

Programas de computador podem *emular* a maneira como o cérebro resolve problemas: uma grande rede altamente interconectada de neurônios relativamente simples trabalhando em paralelo. Convencionou-se assim, chamar de Redes Neurais Artificiais toda estrutura de processamento paralelo que utiliza o ***conexionismo*** como paradigma. Talvez a característica mais importante das Redes Neurais Artificiais seja a sua capacidade de ***aprendizado***, i.e., problemas complexos em engenharia podem ser resolvidos treinando-se a rede pela apresentação do padrão de comportamento desejado.

Lógica Difusa (Fuzzy):

Seres humanos tomam decisões considerando não valores exatos, mas sim utilizando uma lógica que leva em conta um certo "grau de pertinência" das variáveis envolvidas no processo decisório. Não se liga, por exemplo, o ar condicionado em 27°C, às 9^{57} h, e umidade relativa do ar em 77%, mas sim, quando está "quente", no "começo da manha" e quando o ar está "abafado". Estas variáveis *lingüísticas* podem ser melhor descritas e manipuladas num *conjunto Fuzzy*. A Lógica Fuzzy é assim, uma generalização da lógica clássica que permite incluir a imprecisão ("fuzziness") nos processos decisórios.

Metodologia

Apresentação em *Power Point* acompanhada de apostila do curso. O curso será orientado a aplicações das *Toolboxes* do MatLab: Neural Network e Fuzzy Logic, que serão utilizadas durante o mini-curso.

Conteúdo

Resumo

1. Introdução

Inteligência Artifical

Sistemas Inteligentes

Engenharia do Conhecimento

Algoritmos Genéticos

2. Redes Neurais Artificiais

Estruturas e Dinâmicas de Redes Neurais Artificiais

Sinais e Funções de Ativação

Algoritmos e Estratégias de Aprendizado

Implementações de Redes Neurais

Perceptron Multicamadas

Rede de Hopfield

Rede de Base Radial

3. Lógica Fuzzy

Lógica Fuzzy e Sistemas Fuzzy

Inferência Fuzzy

ANFIS

4. Ferramentas Computacionais

MatLab – Neural Network Toolbox

MatLab – Fuzzy Logic Toolbox

5. Exemplos de Aplicações em Engenharia

Reconhecimento Óptico de Caracteres (OCR)

Identificação de Falhas em Estruturas Mecânicas

Controle Fuzzy de Processo de Nível de Líquidos

Conclusões

Bibliografia

1. Introdução

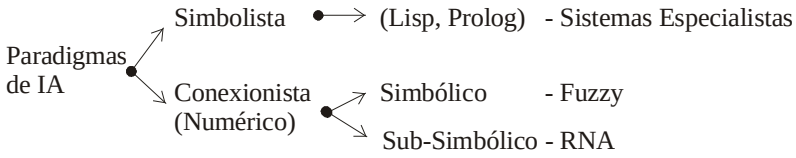
O mini-curso **Sistemas Inteligentes – Redes Neurais e Lógica Fuzzy** tem como objetivo apresentar técnicas de projeto para a engenharia, ***Redes Neurais Artificiais*** e ***Lógica Fuzzy***, que se originaram de pesquisas na área de inteligência artificial. Nesta introdução pretende-se contextualizar o programa proposto em relação ao domínio mais amplo da inteligência artificial e da engenharia do conhecimento. São apresentados argumentos que permitem justificar a utilização destes novos paradigmas em relação aos paradigmas "convencionais".

1.1. Inteligência Artificial

É o ramo da ciência que estuda o conjunto de paradigmas que pretendem justificar como um comportamento inteligente pode emergir de implementações artificiais, em computadores. O que pode ser considerado um sistema inteligente é, no entanto, ainda bastante polêmico. Um subterfúgio permite identificar sistemas inteligentes de forma indireta. Considera-se um programa de computador *inteligente* quando realiza uma tarefa, que se fosse feita por um ser humano, seria considerada inteligente.

Sistemas complexos não devem ser confundidos com sistemas inteligentes. Assim um robô manipulador que aplica pontos de solda na carroçaria de veículos, apesar de realizar uma seqüência complexa de movimentos, ter requisitos de operação em tempo real e segurança aguçados não é considerado inteligente. Este robô apenas repete uma seqüência de movimentos previamente armazenada. Falta a este sistema a capacidade de se adaptar a situações completamente novas. Uma das características de sistemas inteligentes é justamente a capacidade de aprender, de se adaptar a um ambiente desconhecido ou a uma situação nova.

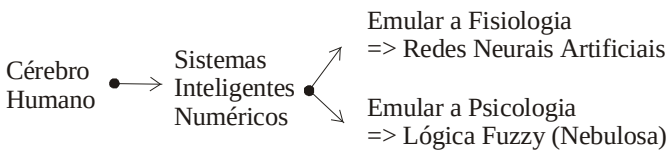
Inteligência: aprendizado, adaptação, compreensão.



A abordagem conexionista considera ser virtualmente impossível ***transformar em algoritmos*** - i.é, reduzir a uma seqüência de passos lógicos e aritméticos – diversas tarefas que a mente humana executa com facilidade e rapidez, como por exemplo:

- Reconhecer rostos,
- Compreender e traduzir línguas,
- Evocação de memória pela associação.

O processo computacional deve reproduzir a capacidade do cérebro de se auto-organizar⇒aprender!



1.2. Métodos em Engenharia do Conhecimento.

De acordo com a disponibilidade de dados e/ou teoria, diferentes métodos são indicados para cada situação particular. A figura 1 apresenta os principais métodos:

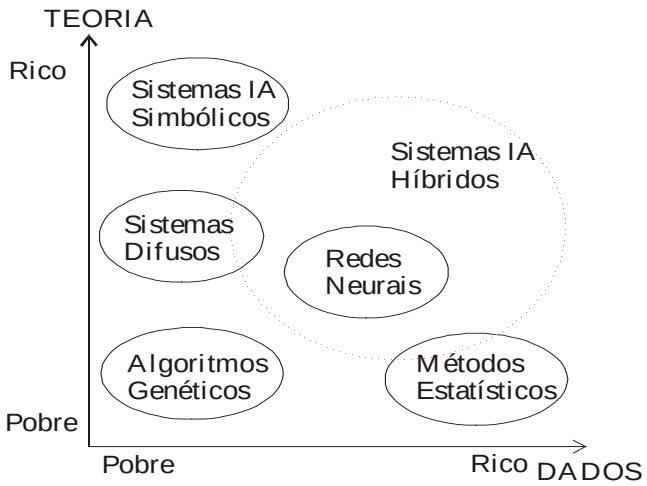


Figura 1 – Métodos de engenharia do conhecimento.

Quando existem apenas exemplos (amostras representativas) de um dado processo, sem regras que expliquem a sua geração, então os métodos estatísticos permitem obter os melhores resultados. No extremo oposto do gráfico estão os métodos de Inteligência Artificial Simbólicos, como por exemplo os sistemas especialistas. Neste caso o importante são as regras e o processo de inferência que permite resolver um certo problema. A teoria - lógica clássica baseada em axiomas - é muito bem estabelecida e prescinde de exemplos para a implementação do sistema especialista.

Os algoritmos genéticos, uma das vertentes da computação evolucionária, por ser justificada exclusivamente pela heurística, está no canto inferior esquerdo do gráfico. Não existe muita base teórica para a sua utilização e tampouco têm-se garantias de que um conjunto de amostras rico em informações sobre o processo, leve à solução do problema. Estes métodos tiveram origem na esperança de que os processos de transmissão de material genético entre gerações de populações, levando eventualmente a indivíduos mais aptos, pudesse ser mimetizado em um programa de computador. Dado um conjunto inicial de possíveis soluções sub-ótimas estas podem ser combinadas (“cruzamento de material genético”) sucessivamente (“gerações”) até obter-se a solução ótima do problema.

Os sistemas difusos, fundamentados pela lógica fuzzy, têm uma boa base teórica. Podem ser construídos a partir de regras formuladas por especialistas da aplicação em particular. Amostras do processo não são necessárias. Estes sistemas utilizam uma lógica multi-valorada que permite “graus de pertinência” e não verdades ou falsidades absolutas. A dificuldade de sua aplicação está muitas vezes justamente na escolha dos novos graus de liberdade obtidos.

Os sistemas baseados em Redes Neurais Artificiais (RNA) ocupam uma região intermediária em relação ao gráfico Teoria x Dados. Estes sistemas exploram razoavelmente bem as amostras do processo. De fato uma das grandes vantagens desta abordagem é a possibilidade de treinamento das RNAs a partir dos dados. Não são necessárias regras ou uma teoria que descreva o processo, as RNAs simplesmente “aprendem” com os exemplos. Estes exemplos são apresentados sucessivamente à RNA, que se adapta um pouco a cada exemplo. O comportamento desejado é reforçado e o comportamento indesejado é reprimido até que o sistema realize a tarefa almejada. Tarefas típicas para RNAs são a classificação de padrões e a aproximação de funções não lineares.

Cada método apresentado possui aplicações para as quais fornece melhores resultados e situações em que não são indicados. Diversas implementações de produtos comerciais, como por exemplo sistemas de OCR (“Optical Character Recognition”), demonstraram que os melhores resultados podem ser obtidos da combinação pela diferentes métodos. Em OCR combinam-se métodos estatísticos com técnicas de extração de características (“feature extraction”) e redes neurais artificiais.

1.3. Paradigma Simbolista versus Paradigma Conexionista

Nesta seção apresentam-se alguns argumentos que destacam vantagens do paradigma conexionista em relação ao paradigma simbolista.

Percepção

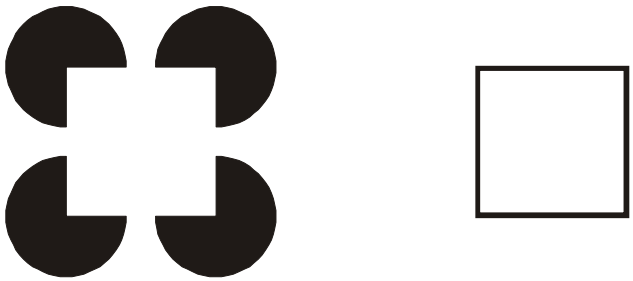


Figura 2 – Quadro de Kanizsa, 1976

O quadro de Kanizsa ilustra a capacidade que temos em lidar com padrões incompletos. Neste caso "interpolamos" a informação que falta e vemos um quadrado branco. Este quadrado não existe formalmente e um programa de computador para interpretação de imagens muito provavelmente não o encontraria, por não existir um *elemento de quatro lados* nesta figura. Padrões incompletos ocorrem muitas vezes em engenharia na forma de sinais ruidosos e faz-se necessário reconstruir a informação original. As redes neurais artificiais podem ser treinadas para lidar com padrões incompletos.

Paradoxos

Epimenides, de Creta, afirmava: *Todos em Creta mentem*. Como interpretar esta afirmação? Se Epimenides diz a verdade, então de fato ele está mentindo; se porém ele está mentindo, então a sua afirmação é verdadeira. É uma situação sem saída. Na lógica clássica não há solução para o paradoxo.

Em lógica Fuzzy os paradoxos podem ser reduzidos a "*meias verdades*" ou "*meias mentiras*", como se queira, através de uma lógica multi-valorada. O verdadeiro (1) e o falso (0) são substituídos por *graus de pertinência* que podem assumir qualquer valor entre 0 e 1. O valor 0,5 descreve um paradoxo.

Consideremos o conceito linguístico *quente*. Os seres humanos tomam decisões baseados neste conceito. Existe portanto uma *lógica*, um formalismo, que permite operar com este conceito. Uma temperatura ambiente de 10°C com certeza não é *quente.* Atribuímos a ela e a todos os valores de temperatura abaixo de 10°C o valor 0. Por outro lado há consenso de que 30°C é *quente.* Atribuímos a esta temperatura o grau de pertinência 1 no conjunto das temperaturas quentes. Entre estes dois extremos podemos arbitrar uma transição suave. Ao ponto central, em 20°C pode ser atribuído o valor 0,5 *quente*. Isto é, em palavras, 20°C não é *quente* e também não deixa de sê-lo.

A lógica *fuzzy* permite operacionalizar estes conceitos lingüísticos construindo com elas um conjunto de regras e permitindo com elas a inferência, i.é, a geração de fatos novos a partir de premissas.

Conjectura de Gödel

Kurt Gödel apresentou em 1931 uma conjectura que abalou a convicção matemática dominante. De maneira simplificada a conjectura de Gödel é:

"Toda formulação axiomática livre de contradições da teoria dos números contém sentenças que não podem ser verificadas e tampouco negadas"

Antes de Gödel imaginava-se ser possível provar ou negar qualquer sentença matemática partindo-se de axiomas. Caso isto fosse possível então programas de computador poderiam ser escritos para resolver qualquer problema que pudesse ser formalizado.

Douglas Hofstadter apresenta em seu hoje clássico *Gödel, Escher, Bach: an Eternal Golden Braid,Basic Books, New York, 1979* uma ilustração para a conjectura de Gödel.

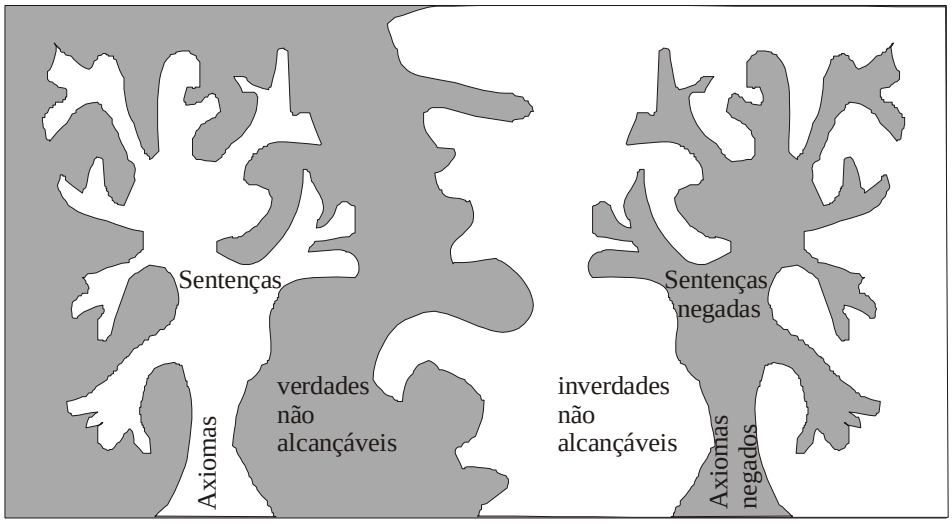


Figura 3 – Visualização do conjunto de sentenças verdadeiras, falsas, que podem ou não ser provadas.

A conjectura de Gödel nos leva a considerar que existam sistemas que funcionam de forma *correta*, que porém não podem ser uma reproduzidos por uma *seqüência de passos lógicos e aritméticos*.

1.4. Solução Heurística de Problemas

A palavra heurística vem do grego e significa "descobrir". Tem a ver com o uso do "senso comum" na solução de problemas. Exemplos:

- Como fritar um ovo
- Como estacionar um carro

Um problema interessante que admite solução heurística é o do caixeiro viajante (TSP- Travelling Salesman Problem), que consiste em encontrar o menor caminho que liga N cidades e cada cidade deve ser visitada uma única vez. Uma heurística que funciona muito bem é a do "vizinho mais próximo". Partindo-se de uma cidade qualquer vai-se sempre para a cidade mais próxima que ainda não foi visitada. A trajetória resultante não é em geral ótima, mas é bastante satisfatória.

A seqüência de perfuração de uma placa de circuito impresso ou a sua montagem com componentes eletrônicos são exemplos de aplicação do TSP.

Uma heurística não garante a solução ótima. Permite, no entanto, em geral uma grande redução de custo e tempo.

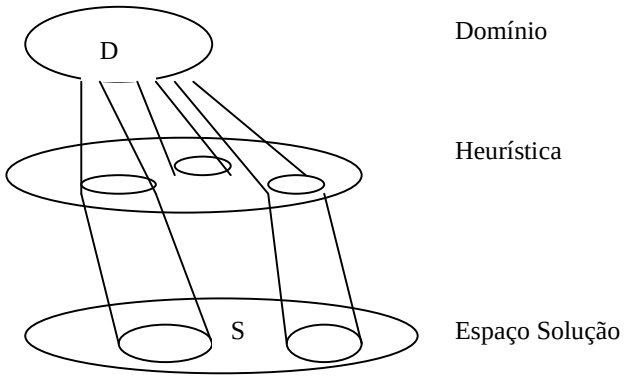


Figura 4 – Uma regra heurística leva do espaço domínio ao espaço solução.

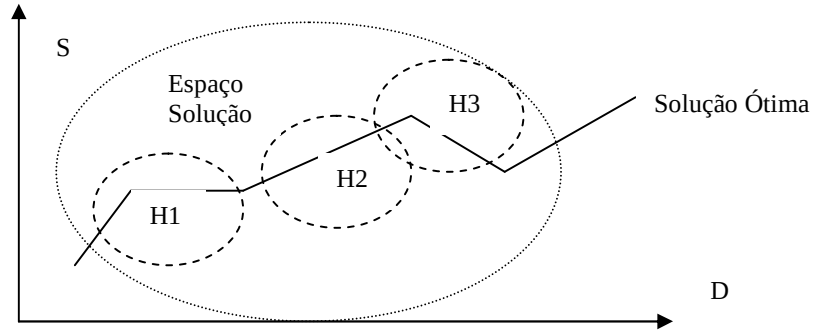


Figura 5 – Heurísticas fornecem soluções sub-ótimas.

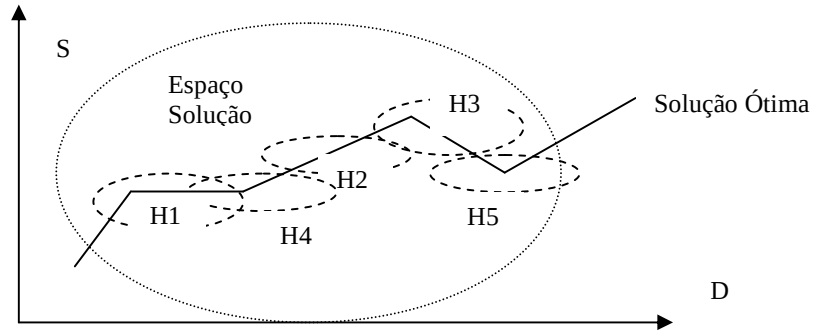


Figura 6 – Heurísticas "bem-formadas" estão próximas da solução ótima.

Heurísticas "bem-formadas" fornecem soluções que estão muito próximas da solução ótima.

Uma regra heurística é da forma:

SE <condição> ENTÃO <conclusão>

- Sistemas difusos – representação de conhecimento heurístico por meio regras fuzzy.
- Redes Neurais Artificias – aprendem heurística a partir dos dados.

Ambos são ***aproximadores universais***, isto é, permitem aproximar uma função qualquer com precisão arbitrária.

1.5. Algoritmos Genéticos

Os algoritmos genéticos, bem como a assim denominada computação evolucionária, são métodos heurísticos para a solução de problemas. John Holland propôs os algoritmos genéticos em 1975 inspirado pela teoria da evolução de Charles Darwin. Os conceitos básicos dos AG são:

- Gen
- Cromossomo – representa um indivíduo, uma solução possível
- População
- Cruzamento
- Mutação
- Critério de Avaliação
- Seleção

A idéia básica é de buscar a solução ótima para um problema partindo-se que uma População Inicial, que representam um conjunto inicial de candidatos à solução. A partir do Cruzamento destes indivíduos, chega-se a uma nova geração. A Seleção dos que estiverem mais próximos da solução ótima, i.é., os mais aptos, permite levar a uma nova geração. E assim por diante. Eventualmente pode-se alterar alguns gens, representando uma mutação. Ver figura 7.

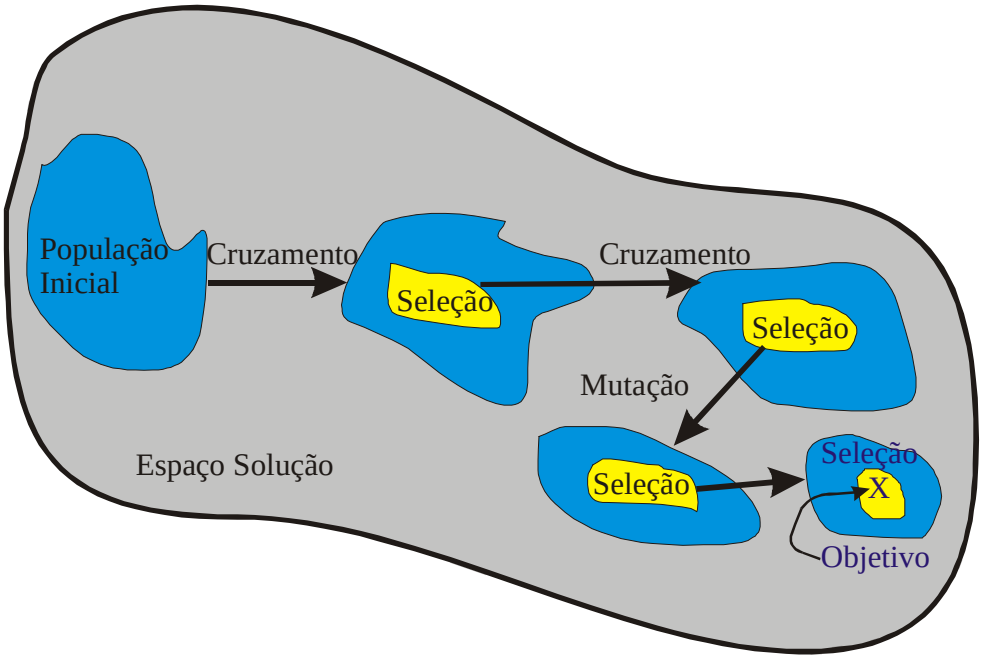


Figura 7 – Representação gráfica do algoritmo genético.

Exemplo: Considere o "jogo" de adivinhar o número 001010 ("Mastermind")

O critério de seleção é a proximidade do número 001010.

População Inicial	Avaliação	Seleção
A 010101	1	
B 111101	1	
C 011011	4	*
D 101100	3	*

Nova população em que os indivíduos C e D são os pais.

	Nova População	Avaliação	Seleção
C 01:1011	E 01 11 00	3	
D 10:1100	F 10 10 11	4	*
C 0110:11	G 01 10 00	4	*
D 1011:00	H 10 11 11	3	

Nova população em que os indivíduos F e G são os pais.

	Nova População	Avaliação	Seleção
F 1:0 10 11	I 11 10 00	3	
G 0:1 10 00	J 00 10 11	5	*
F 10 1:0 11	K 10 10 00	4	*
G 01 1:0 00	L 01 10 11	4	

Nova população em que os indivíduos J e K são os pais.

	Nova População	Avaliação	Seleção
J 00 10: 11	M 00 10 00	5	
K 10 10: 00	N 10 10 11	5	*
J 00 10 1:1	O 00 10 10	6	Sucesso – FIM
K 10 10 0:0	P 10 10 01	3	

Sucesso após 16 questões. Por busca exaustiva teríamos $2^6 = 64$ combinações possíveis.

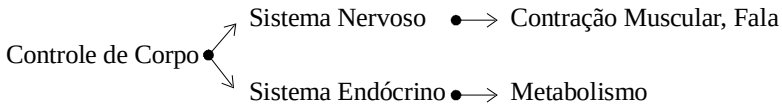
2. Redes Neurais Artificiais

As redes neurais artificiais que são utilizadas em engenharia foram inspiradas nas redes neuronais biológicas. No entanto convencionou-se chamar redes neurais artificiais a toda topologia de processamento de sinais constituída de vários elementos processadores simples altamente interconectados, i.é, estruturas baseadas no *conexionismo.* Inicialmente este capítulo trata dos fundamentos biológicos, para então apresentar as redes neurais artificiais mais conhecidas.

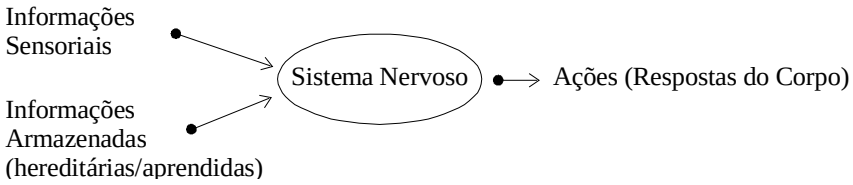
2.1. Fundamentos Biológicos

Esta introdução aos fundamentos biológicos apresenta apenas os elementos necessários à compreensão da anatomia (estrutura) e fisiologia (funcionamento) dos neurônios biológicos que serviram de inspiração para as primeiras redes neurais artificiais.

O sistema nervoso atua em conjunto com o sistema endócrino no controle do corpo.



O sistema nervoso obtém informações do meio ambiente através de sensores que são combinadas com informações armazenadas para produzir as ações do corpo.



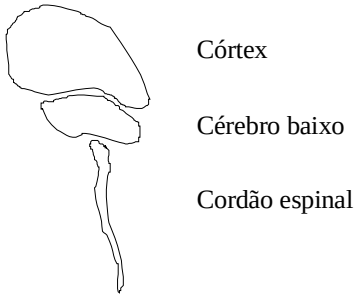
Apenas uma pequena parte das informações obtidas é relevante para o funcionamento do corpo.

O sistema nervoso pode ser considerado em três níveis:

- Cordão espinal
- Cérebro baixo
- Córtex cerebral

Cada um constituído por neurônios de diferentes anatomias.

Estima que o cérebro humano tenha por volta de 10^{11} neurônios, cujo comprimento total somado chega a 10^{14} metros.



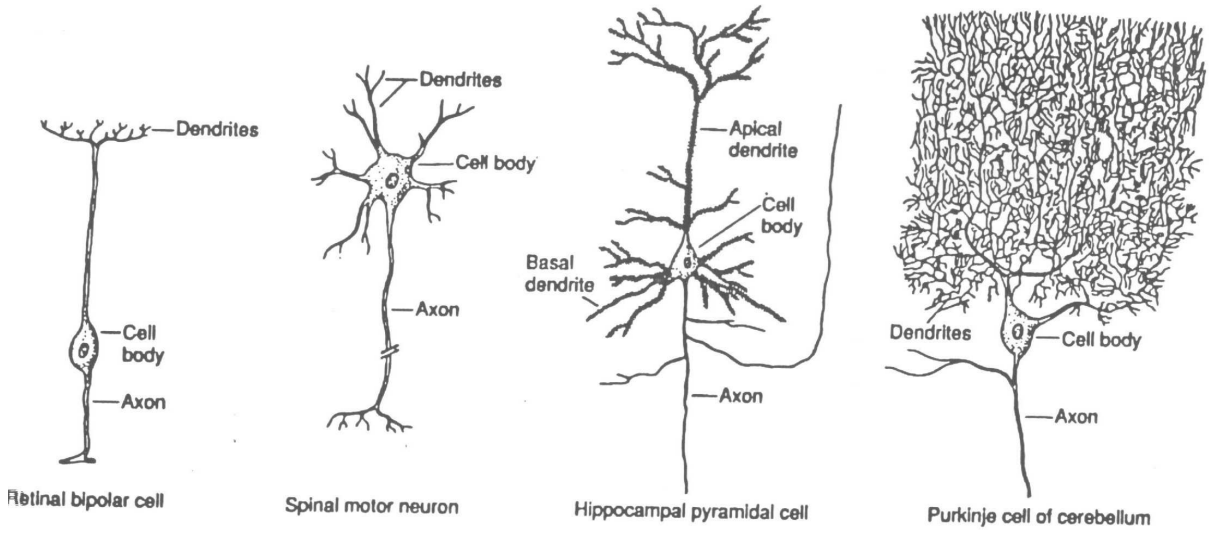


Figura 8: Tipos de neurônios

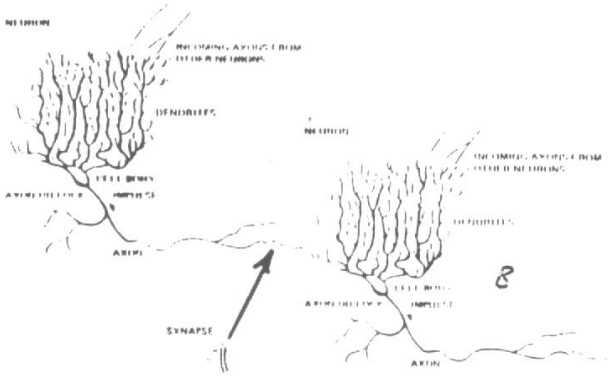


Figura 9: Conexão Sináptica

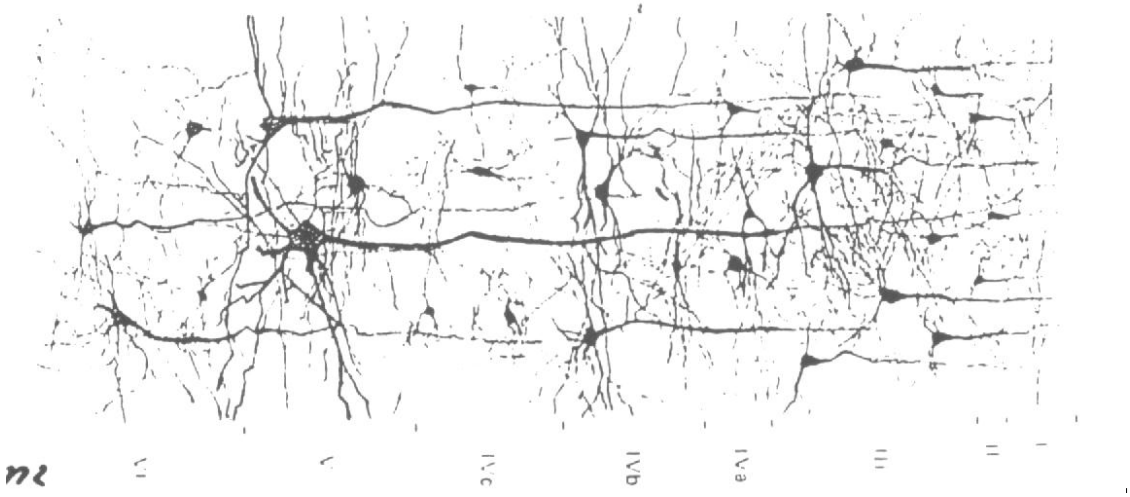


Figura 10: Lâmina que mostra o padrão de conexão dos neurônios em camadas.

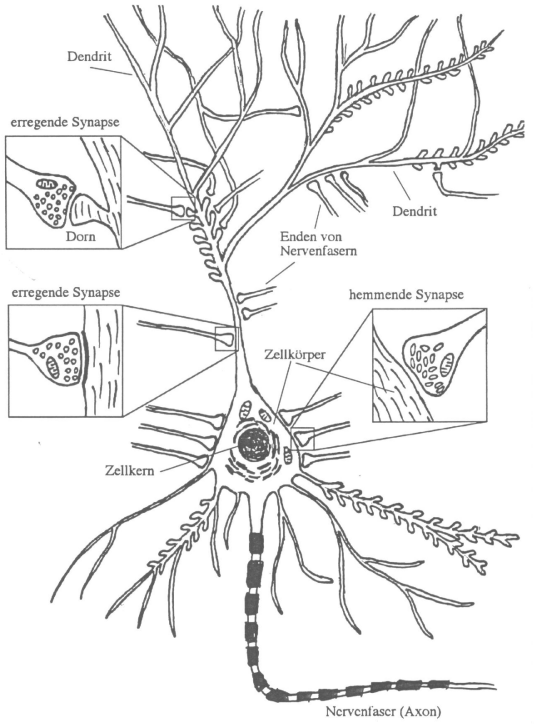


Figura 11: Sinapses Excitatórias e Inibitórias

Níveis de processamento da informação pelo cérebro
- estrutural
- fisiológico
- cognitivo

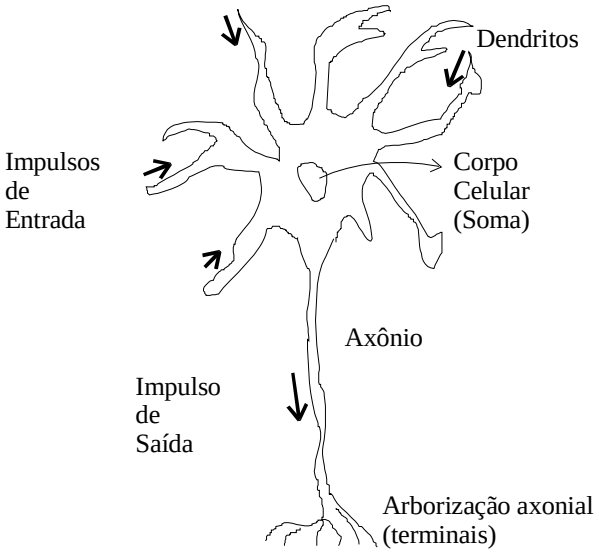


Figura 12– Neurônio Biológico

Sinapse – conexão entre o terminal de um neurônio aos dendritos de outros neurônios.

Muitas formas de conexão entre neurônios, contudo observa-se um padrão de conexão bastante seletivo e específico.

O Fluxo da informação (corrente elétrica) é sempre dos dendritos para o axônio.

Ocorre um processo de integração (soma) dos estímulos de entrada, produzindo um impulso elétrico que se propaga através do axônio, caso a soma das entradas seja maior que um certo limiar.

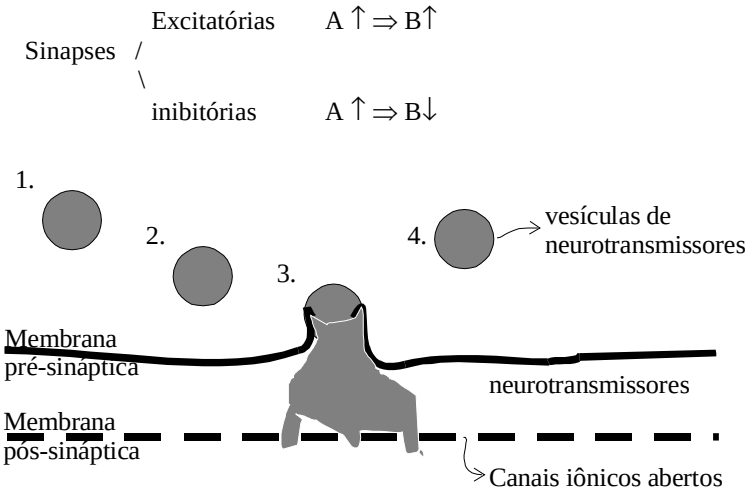


Figura 13 - Neurotransmissores no gap sináptico.

Apenas um tipo de neurotransmissor é liberado em uma dada ativação e o efeito da sinapse é sempre toda excitatória ou toda inibitória.

2.2. O Potencial de Ação

O impulso nervoso ou potencial de ação é uma onda de despolarização que se propaga ao longo da membrana. Ocorre quando a membrana é despolarizada suficientemente para cruzar o limiar de disparo. A velocidade de propagação é de até 150m/s em células mielinizadas.

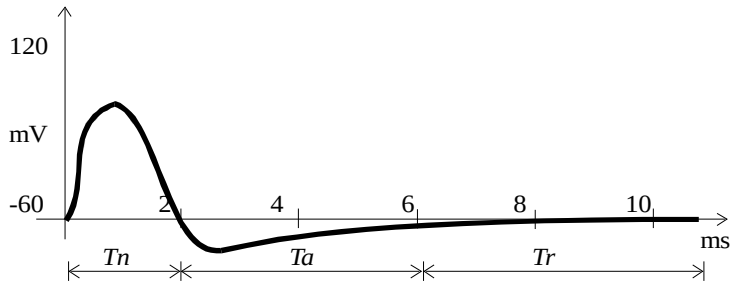


Figura 14 – O potencial de ação.

Tn – duração do impulso nervoso.
Ta – período de refração absoluta.
Tr – período de refração relativa.

A figura 14 ilustra a tensão elétrica que pode ser observada em um ponto fixo do axônio ao longo do tempo quando da passagem de um impulso nervoso. Depois de *Tn*, a duração do impulso, há um intervalo de tempo *Ta* em que o neurônio não pode disparar, independente do potencial acumulado pelo soma. No período de refração relativa é possível que o neurônio dispare, desde que o potencial acumulado pelo soma, mais elevado que de costume, seja suficiente. Depois de aprox. 10 ms, o neurônio está novamente em seu estado de repouso e pode disparar novamente em seu potencial de disparo usual.

2.3. Integração Espaço/Temporal dos Estímulos

A membrana do axônio mantém por algum tempo a memória da atividade sináptica → integração temporal. Efeito combinado de todos os estímulos excitatórios e inibitórios → integração espacial.

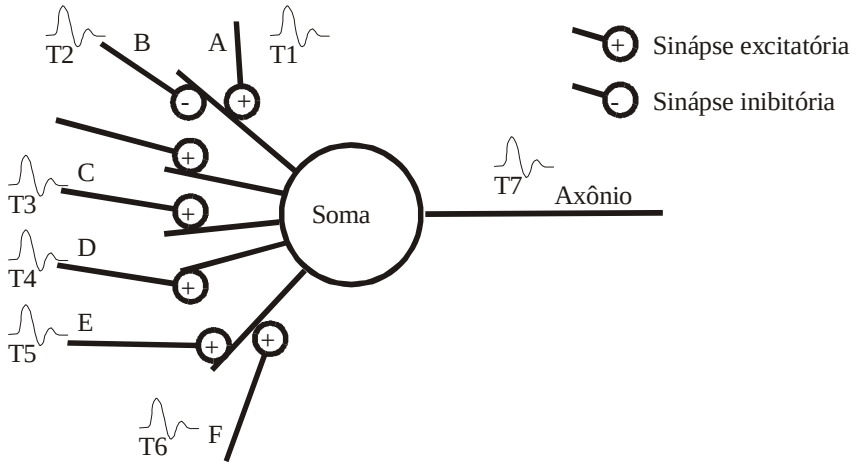


Figura 15 – Sinápses excitatórias e inibitórias

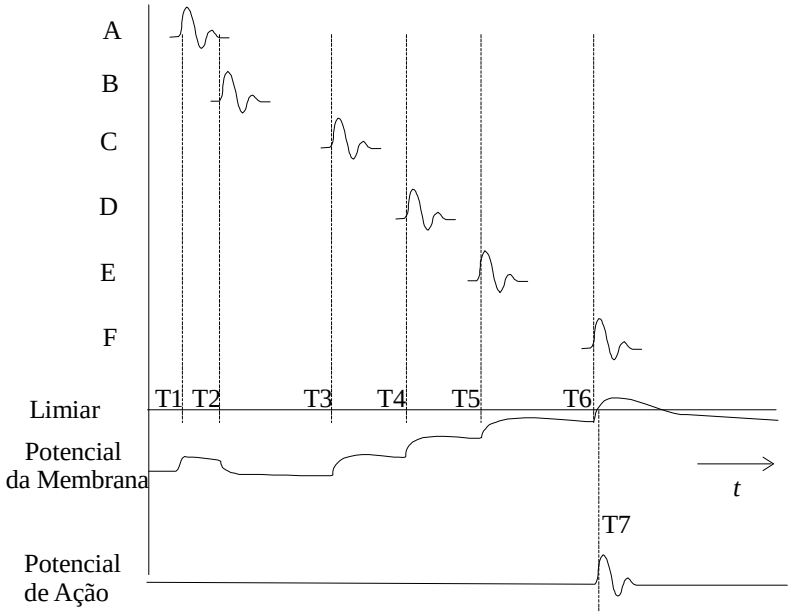


Figura 16 – Integração espaço/temporal dos estímulos

Freqüência máxima de pulsos no axônio: $f_{\max} = \frac{1}{T_a + T_n}$

Durante o período refratário relativo, a ocorrência de novos impulsos nervosos com freqüência crescente só é possível com o aumento da despolarização.

O neurônio codifica em freqüência de pulsos o resultado da integração espaço/temporal.

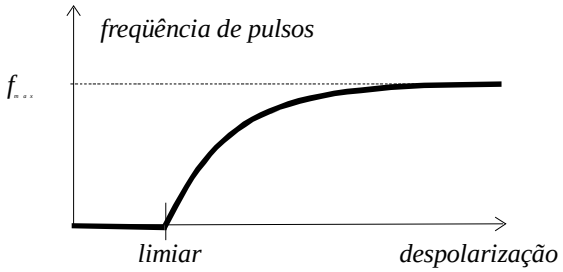


Figura 17 – Freqüência de pulsos em função da desporalização do neurônio.

$$f_T = g\left(\int_0^T \sum_i \alpha_i(t) x_i(t) dt\right)$$

f_T – freqüência média de impulsos nervosos no intervalo de tempo T,
$\alpha_i(t)$ – ganhos sinápticos,
$x_i(t)$ – entradas dos neurônios.

A característica básica do funcionamento do neurônio biológico que inspira as redes neurais artificiais é o comportamento monotônico saturado de sua resposta em termos de freqüência de pulsos. As RNAs utilizam a função pulso, sigmóide e tangente hiperbólica para simular o neurônio natural.

Neurônios com conexão lateral

Embora a maioria das estruturas de processamento da informação em neurônios seja unidirecional e organizada em camadas têm-se também exemplos de conexão lateral que inspiram redes com realimentação.

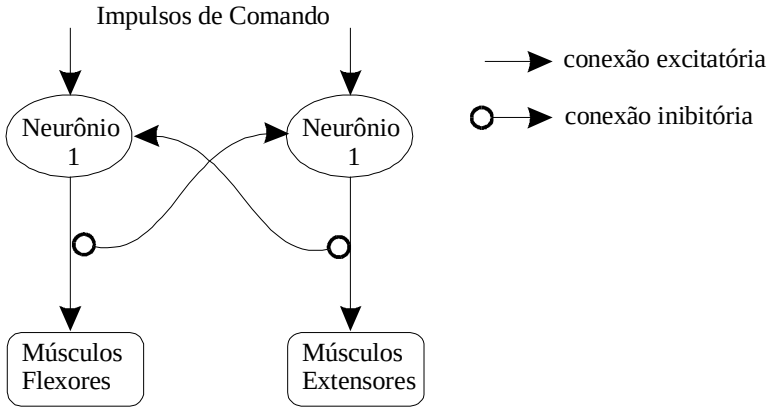


Figura 18 – Circuito neural inibidor dos antagonismos.

2.4. Modelo básico do neurônio artificial

Um modelo computacional básico para o neurônio considera a soma ponderada de entradas que são submetidas a uma função de ativação contínua com saturação. Em vez de pulsos é muito mais simples processar informação analógica (contínua).

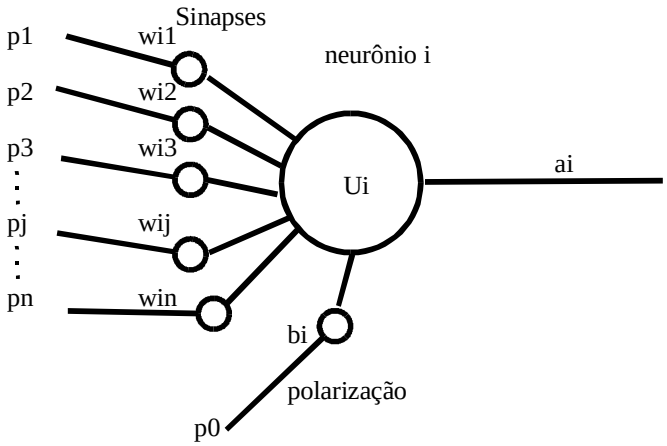


Figura 19 – Neurônio artificial básico

A soma ponderada das entradas é representada por

$$u_i = \sum_{j=1}^{n} w_{ij} p_j + b_i = \mathbf{w}_i^t \mathbf{p} + b_i$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}, \qquad \mathbf{w} = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{bmatrix}.$$

Sinapse excitatória $w_{ij} > 0$,
Sinapse inibitória $w_{ij} < 0$.

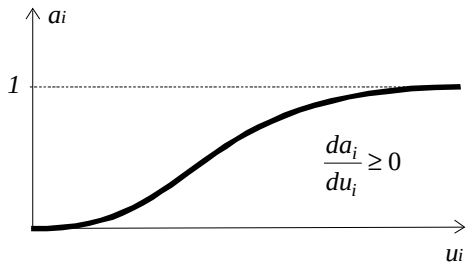


Figura 20 – Função de ativação sigmóide (em forma de "S", sigma em grego).

Da mesma forma como os neurônios naturais estão organizados em camadas, as redes neurais artificiais geralmente tem esta mesma topologia. Como o tecido neuronal apresenta especialização em sistemas específicos e não específicos, os sistemas artificias processam a informação de forma semelhante, sendo que conceitos mais abstratos são representados por camadas mais profundas.

2.5. Comparação Cérebro x Computador

A seguinte comparação entre o cérebro e o computador é apenas dar mostrar a ordem de grandeza dos elementos processadores envolvidos.

	Cérebro	*Computador*
# elementos processadores	~ 10^{11} neurônios	~ 10^{9} transistores
Forma de processamento	Massivamente paralelo	Em geral serial
Memória	Associativa	Endereçada
Tempo de chaveamento	~ 1 ms	~ 1 ns
Chaveamentos /s	~ 10^{3} /s	~ 10^{9} /s
Chaveamentos totais (teórico)	~ 10^{14} /s	~ 10^{18} /s
Chaveamentos totais (real)	~ 10^{12} /s	~ 10^{10} /s

Regra dos 100 passos:

Pessoas reconhecem um rosto conhecido em ~ 0.1 *s*. Considerando 1*ms* por neurônio, implica que o cérebro executa no máximo 100 passos seqüênciais até reconhecer o padrão.

Em 100 passos seqüênciais um computador convencional (arquitetura von Neumann) não faz praticamente nada.

⇒ Novas arquiteturas de processamento paralelo massivo são necessárias.

2.6. Perspectiva Histórica de RNAs

Apresentam-se algumas datas, pesquisadores e inovações que tiveram grande repercussão na área de Redes Neurais Artificiais:

1943	McCulloch	Neurônio Booleano	Entusiasmo Inicial
1949	Hebb	Regra de aprendizado	
1957	Rosenblatt	Perceptron	
1960	Widrow-Hoff	ADALINE/MADALINE LMS	
	Rosenblatt	Perceptron Multicamadas, sem treinamento	
1969	Minsky-Papert	*Perceptrons*	Desencantamento
1974	Werbos	Algoritmo *Error Backpropagation* – sem repercussão	
1982	Hopfield	Rede realimentada	
1986	Rumelhart, Hinton & Williams		Ressurgimento
		PDP – MIT *Backpropagation* p/ Perceptron Multicamadas	
		Função de ativação contínua sigmóide	
1987	Kosko	BAM	

O neurônio booleano e o perceptron tiveram grande repercussão quando foram propostos, pois imaginava-se que poderiam servir de modelo para os processos decisórios mentais. A regra de aprendizado por reforço positivo de Hebb era bastante intuitiva e podia ser verificada na prática educacional.

A dificuldade de representar funções não-linearmente separáveis, como o x-or, apontada por Minsky & Papert em *Perceptrons* levou ao desencantamento da comunidade científica em relação às RNAs.

Apenas a partir do algoritmo *backpropagation* apresentado pelo grupo de processamento paralelo distribuído do MIT em 1986 é que houve um ressurgimento do interesse pelas redes neurais artificiais.

Vários congressos e revistas científicas que tratam de RNAs atestam o alcance desta técnica na comunidade científica. No Brasil o CBRN – Congresso Brasileiro de Redes Neurais teve a sua 5ª edição realizada no Rio de Janeiro em Março de 2001. O SBAI – Simpósio Brasileiro de Automação Inteligente, também em sua 5ª edição em 2001, também reserva grande espaço para aplicações de RNAs.

2.7. O Neurônio de McCulloch (1943)

Caso particular de descriminador de n entradas $\{p_1, p_2, \ldots p_n\}$

$$a = F\left(\sum_{i=1}^{n} w_i p_i - b\right) = F(\mathbf{w}^t\mathbf{p} - b) \rightarrow a \in [0;1] \text{ degrau}$$

F = função degrau

Divisão do espaço euclidiano $\Re^n$ em duas regiões A e B

$$\mathbf{w}^t\mathbf{p} - b > 0 \Rightarrow x \in A \ (a = 1)$$

$$\mathbf{w}^t\mathbf{p} - b < 0 \Rightarrow x \in B \ (a = 0)$$

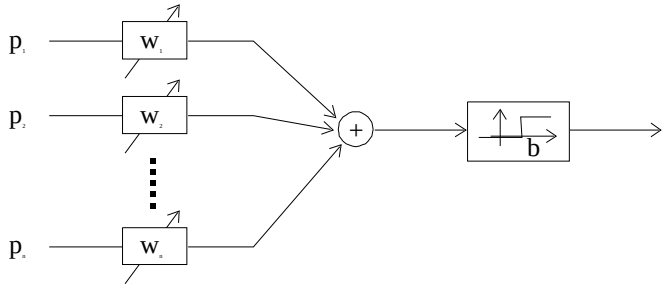


Figura 21 - O neurônio de MacCulloch

Para n=2

$$\sum_{i=1}^{2} w_i p_i = b$$

$$w_1 p_1 + w_2 p_2 = b$$

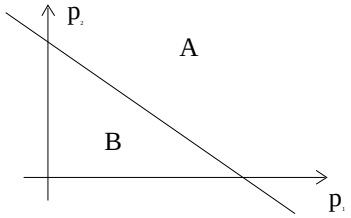


Figura 22 – Separação do plano em duas regiões pelo neurônio de McCulloch

O neurônio de McCulloch como classificador de padrões

Sejam $\left.\begin{array}{l}\Phi_1 = \{\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_k\} \\ \Phi_2 = \{\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_m\}\end{array}\right\}$ aglomerações de pontos

duas coleções de vetores n-dimensionais.

O descriminador dever fornecer a=1 se $p \in \Phi_1$ e a=0 se $p \in \Phi_1$. Isto só é possível se houver um hiperplano que separe os dois aglomerados:

$$\sum_{i=1}^{'n} w_i p_i = \mathbf{w}^t\mathbf{p} = b \qquad (3.1)$$

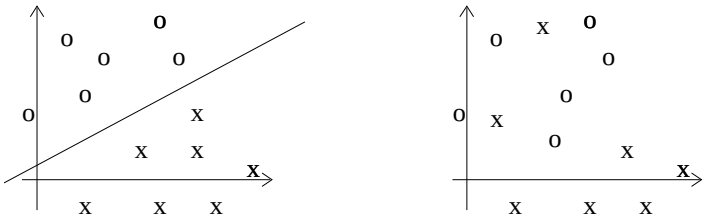


Figura 23 – Coleções linearmente separáveis e linearmente dependentes (não-separáveis).

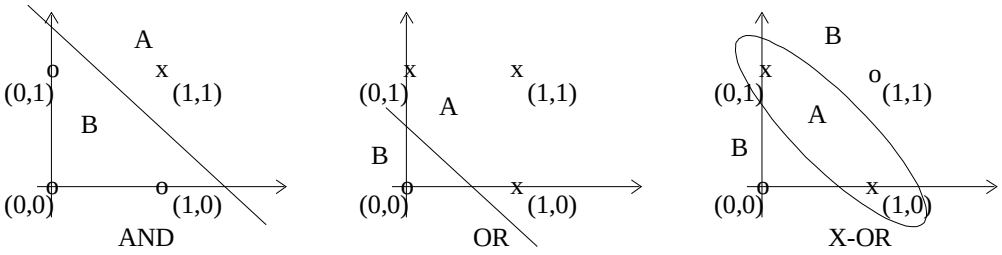


Figura 24 – Algumas funções booleanas de duas variáveis representadas no plano binário.

2.8. Classificadores Lineares e Não-Lineares

Objetivo: Treinar uma rede para determinar a equação do hiperplano separador.
Para *n* entradas, $m = 2^n$ padrões de entrada.

Existem $2^m = 2^{2^n}$ possíveis funções lógicas conectando *n* entradas para uma saída binária.

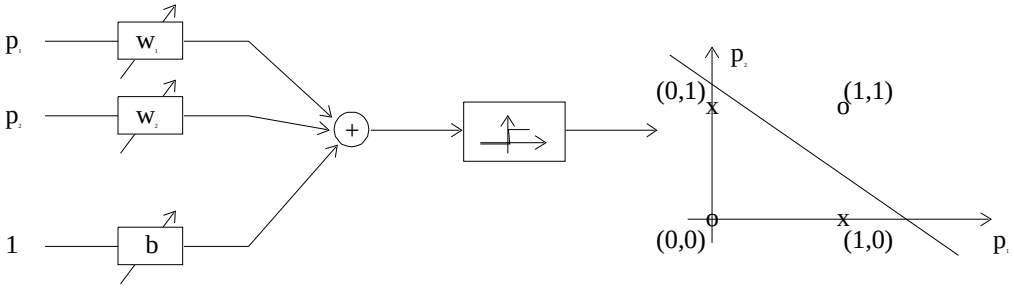


Figura 25 – O neurônio de McCulloch e o X-OR

A função X-OR não é linearmente separável e assim não pode ser aprendida pelo neurônio booleano de McCulloch. Widner apresentou em 1960 um estudo sobre as funções linearmente separáveis:

Tabela 1 – Funções linearmente separáveis

n	nº de padrões binários	nº de funções lógicas	linearmente separáveis	% linearmente separável
1	2	4	4	100
2	4	16	14	87,5
3	8	256	104	40,6
4	16	65536	1.772	2,9
5	32	$4{,}3 \times 10^{9}$	94.572	$2{,}2 \times 10^{-3}$
6	64	$1{,}8 \times 10^{19}$	5.028.134	$3{,}1 \times 10^{-13}$

As funções lógicas de uma variável:

$A,\ \overline{A}$, 0, 1

As funções lógicas de duas variáveis:

$A, B, \overline{A}, \overline{B}$, 0, 1, $A \vee B, A \wedge B, \overline{A} \vee B, \overline{A} \wedge B,\ \ A \vee \overline{B}, A \wedge \overline{B}, \overline{A} \vee \overline{B}, \overline{A} \wedge \overline{B},\ \ A \oplus B, \overline{A \oplus B}$

Verifica-se por esta tabela que neurônios que podem apenas representar funções linearmente separáveis são de pouca utilidade, uma vez que uma percentagem ínfima ($3{,}1 \times 10^{-13}$ % para sistemas com seis entradas) de funções lógicas está nesta categoria.

Perceptron binário de duas etapas

A utilização de mais um neurônio, em uma camada adicional, pode, em princípio, realizar um sistema para classificar polígonos convexos. A dificuldade é justamente encontrar um procedimento de treinamento para este tipo de rede.

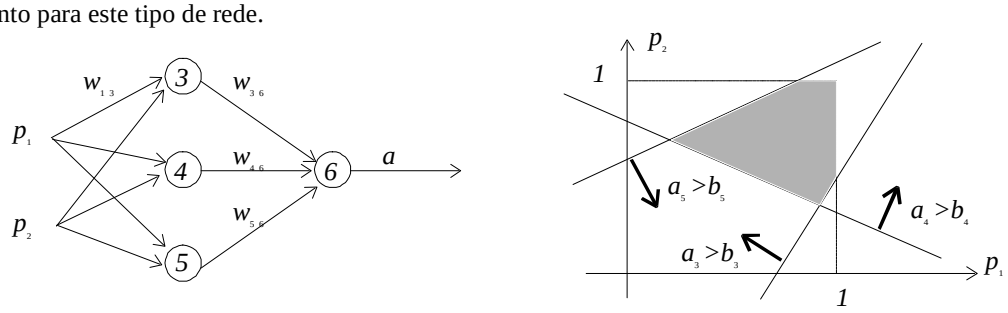


Figura 26 – Classificação de polígonos convexos com o neurônio booleano.

O neurônio 6 na figura 26 implementa um E-lógico fazendo-se $b_6 = \sum_{i=3}^{5} w_{i6}$. Outra função binária, p.ex: OR, NAND, poderia também ser facilmente implementada.

Por exemplo: $w_{36} = w_{46} = w_{56} = \frac{1}{3}$; $\quad b_6 = 1 \quad \Rightarrow a_6 = 1$ *se e somente se* $a_3 = a_4 = a_5 = 1$.

Perceptron binário de três etapas

Pode ser utilizado para representar sobreposições e cortes de polígonos convexos.

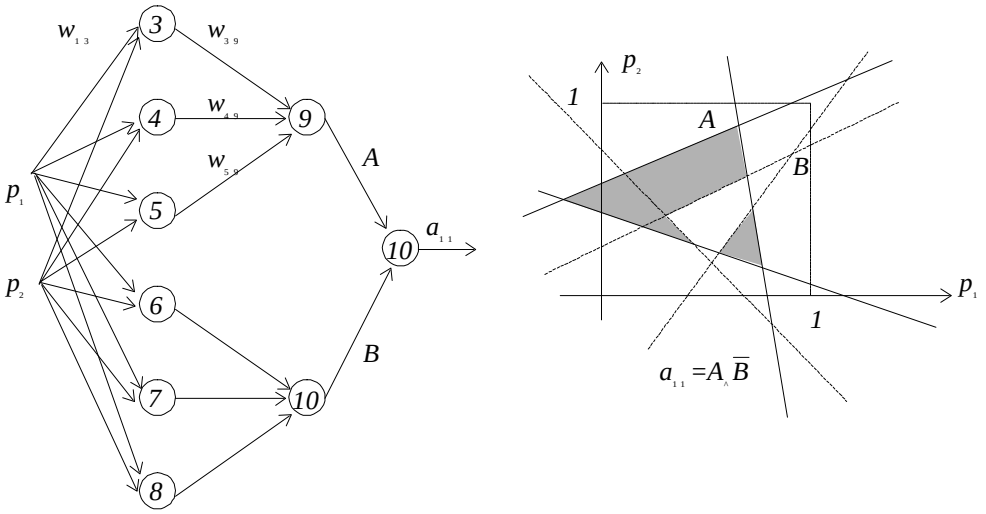


Figura 27 – Classificação de região côncava e não contígua com o neurônio booleano.

Novamente a configuração de três camadas de perceptrons pode apenas em princípio representar mapeamentos complexos da entrada para a saída. Um gerador de números aleatórios poderia ser utilizado para os pesos, na esperança de que eventualmente a combinação necessária de pesos seja encontrada.

Sem um procedimento factível de treinamento estas redes de neurônios booleanos são apenas curiosidades acadêmicas.

2.9. Neurônios e Redes Neurais Artificiais

As redes neurais artificiais podem ser classificadas quanto à:

- Micro-Estrutura – características de cada neurônio na rede
- Meso-Estrutura – organização da rede
- Macro-Estrutura – associação de redes eventualmente com processamento analítico para abordar problemas complexos.

Micro-Estrutura:

A micro-estrutura é definida pelas características de cada neurônio na rede, em particular pela sua função de ativação.

Na Toolbox de Redes Neurais do MATLAB® todos os neurônios de uma camada devem ter a mesma função de ativação. Em outros ambientes de simulação de redes neurais não há esta restrição, como p.ex. no NeuralWorks®.

Cada neurônio artificial possui um número *n* de entradas cuja soma ponderada pelos pesos w_i passa pela função de ativação para gerar a saída do neurônio. Os pesos w_i são variáveis apenas na fase de treinamento. Depois desta fase o neurônio passa a ser um função não-linear de $\Re^n \rightarrow \Re$.

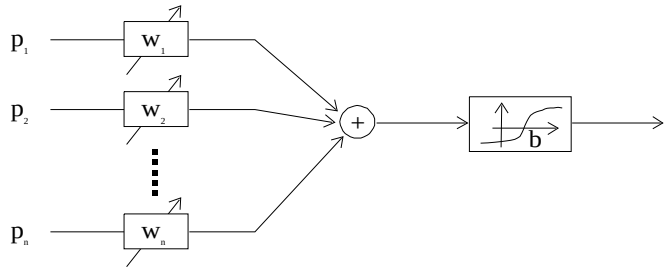


Figura 28 – O neurônio artificial.

Modelo Matemático, para a função de ativação *sinal*: $u = \sum_{i=1}^{'n} w_i p_i, \qquad a = f(u) = \begin{cases} +1 \ se \ u \geq b \\ -1 \ se \ u < b \end{cases}$

Uma entrada de polarização é freqüentemente utilizada, permitindo uma soma ponderada das entradas não nula quando a soma das entradas é zero. O parâmetro *b*, de deslocamento da função de ativação, torna-se assim desnecessário.

$$u = \sum_{i=1}^{'n} w_i p_i - b, \quad a = f(u) = \begin{cases} +1 \ se \ u \geq 0 \\ -1 \ se \ u < 0 \end{cases}$$

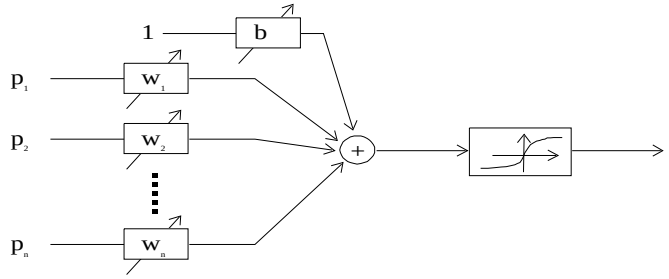
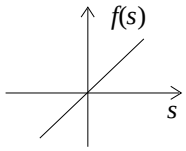
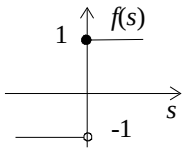
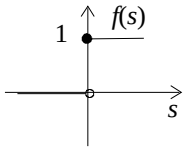
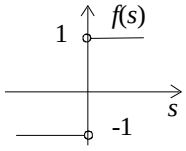
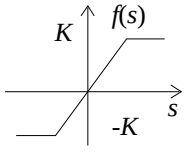
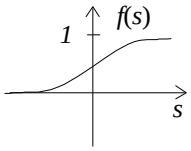
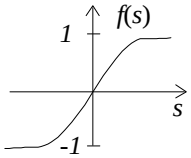


Figura 29 – O neurônio artificial com entrada de polarização.

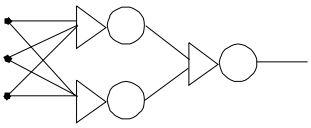
2.10. Funções de ativação típicas

Linear	$f(s) = s$	Hopfield BSB	purelin	
Sinal	$f(s) = \begin{cases} +1 \; se \; s \geq 0 \\ -1 \; se \; s < 0 \end{cases}$	Perceptron	hardlims	
Degrau	$f(s) = \begin{cases} +1 \; se \; s \geq 0 \\ 0 \; se \; s < 0 \end{cases}$	Perceptron BAM	hardlim	
Hopfield/ BAM	$f(s) = \begin{cases} +1 \; se \; s > 0 \\ -1 \; se \; s < 0 \\ inalterado \; se \; s = 0 \end{cases}$	Hopfield BAM		
BSB ou Limiar Lógico	$f(s) = \begin{cases} -K \; se \; s \leq -K \\ s \; se \; -K < s < +K \\ +K \; se \; s \geq +K \end{cases}$	BSB	satlin satlins	
Logística	$f(s) = \frac{1}{1+e^{-s}}$	Perceptron Hopfield BAM, BSB	logsig	
Tangente Hiperbólica	$f(s) = \tanh(s) = \frac{1-e^{-2s}}{1+e^{-2s}}$	Perceptron Hopfield BAM, BSB	tansig	

2.11. Meso-Estrutura

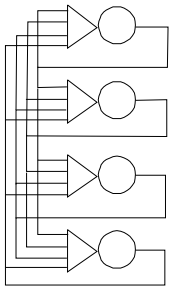
Meso-Estrutura – organização da rede
neurônios por camada
camadas da rede
tipo de conexão (*forward*, *backward*, lateral).

1-*Feedforward* Multicamadas



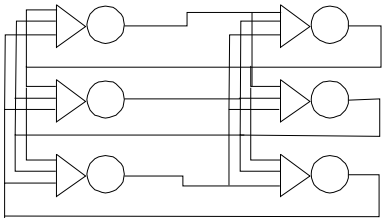
Perceptron Multicamadas (MLP)

2- Camada simples conectada lateralmente

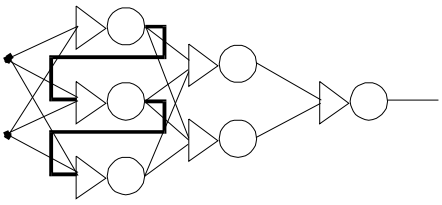


BSB (auto-realimentação), Hopfield

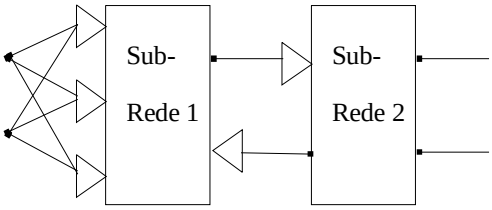
3 – Bicamadas *Feedforward/Feedbackward*



4 – Rede Multicamadas Cooperativa/Comparativa

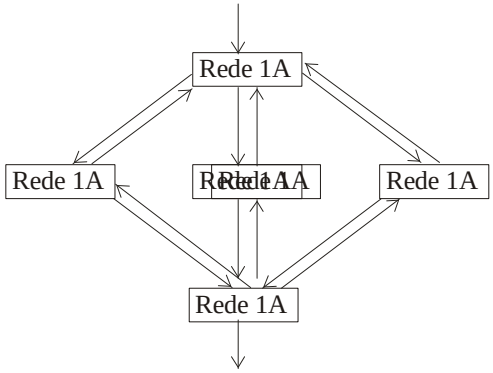


5 – Rede Híbrida



2.12. Macro-Estrutura Neural

de redes
tipo de conexão
tamanho das redes
grau de conectividade



2.13. Aprendizado Supervisionado: A regra delta

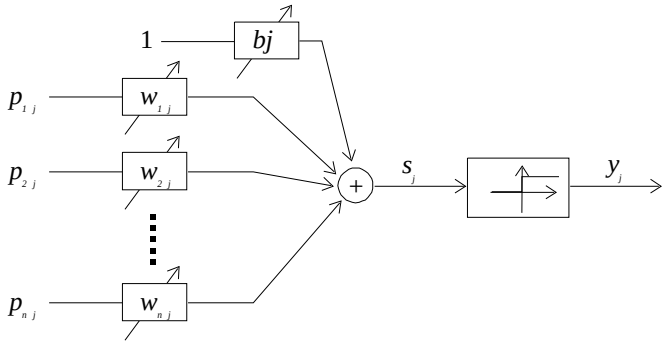
- Regra delta → perceptron
- Regra delta de Widrow-Hoff (LMS) → ADALINE, MADALINE
- Regra delta generalizada

Perceptron – Rosenblatt, 1957

Dinâmica:

$$s_j = \sum_i w_{ij} p_{ij} + b_j$$

$$y_j = f(s_j) = \begin{cases} +1 \; se \; s_j \geq 0 \\ 0 \; se \; s_j < 0 \end{cases}$$



$$\delta_j = d_j - y_j$$

$$w_{ij} \leftarrow w_{ij} + \mu \, \delta_j x_{ij}$$ Regra delta

μ - taxa (fator) de aprendizagem
$\delta_j = 0 \rightarrow$ o peso não é alterado.

2.14. ADALINE e MADALINE – Widrow & Hoff, 1960

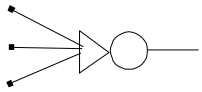


Figura 30 – ADALINE – ADAptive LINear Element.

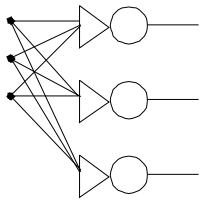


Figura 31 – MADALINE – Multiple ADAptive LINear Element.

Treinamento

$$\varepsilon_j = d_j - s_j = d_j - \left(\sum w_{ij} p_{ij} + b_j \right)$$

$$\boxed{w_{ij} \leftarrow w_{ij} + \frac{\mu \varepsilon_j x_{ij}}{\sum x_k^2}}$$ Regra delta de Widrow-Hoff

LMS – Least Mean Squared (Menor erro quadrático médio).

0.1 $< \mu <$*1* – estabilidade e velocidade de convergência.
Entradas bipolares $\pm 1 \Rightarrow \Sigma x^2_k$ = n, independente do padrão.

Funções no MatLab: NEWLIN, NEWLIND, SIM, ADAPT, LEARNWH

2.15. O Algoritmo LMS

Desenvolvido por Widrow & Hoff em 1960. Utilizado nas redes MADALINE.

Objetivo: Aprender a função $f : \Re^n \rightarrow \Re$ das amostras ($\boldsymbol{x}_k$, d_k)

$\{ \boldsymbol{x}_k \}$, $\{d_k\}$ e $\{e_k\} \rightarrow$ processos estocásticos estacionários
$e = d - y \rightarrow$ erro estocástico atual

$$y = \sum_{i=1}^{n} x^i w^i = \mathbf{x}\mathbf{w}^t \rightarrow \text{neurônio linear}$$

Valor esperado do erro quadrático médio

$$\begin{aligned} E[e^2] &= E[(d-y)^2] \\ &= E[(d-\boldsymbol{x}\boldsymbol{w}^t)^2] \\ &= E[d^2] - 2E[d\boldsymbol{x}]\boldsymbol{w}^t + \boldsymbol{w}E[\boldsymbol{x}^t\boldsymbol{x}]\boldsymbol{w}^t \end{aligned}$$

assumindo-se para tanto $\boldsymbol{w}$ determinístico.
Define-se

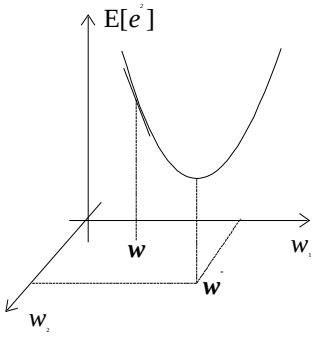
$E[\boldsymbol{x}^t\boldsymbol{x}] \equiv \boldsymbol{R} \rightarrow$ matriz de autocorrelação de entrada
$E[d\boldsymbol{x}] \equiv \boldsymbol{P} \rightarrow$ vetor de correlação cruzada

Com esta notação têm-se:

$$E[e^2] = E[d^2] - 2\boldsymbol{P}\boldsymbol{w}^t + \boldsymbol{w}\boldsymbol{R}\boldsymbol{w}^t$$

O vetor de pesos sinápticos ótimo $\boldsymbol{w}^*$ pode ser obtido fazendo-se nulo o gradiente das derivadas parciais em relação à $\boldsymbol{w}$:

$$\mathbf{0} = 2\boldsymbol{w}^*\boldsymbol{R} - 2\boldsymbol{P}$$



$\boxed{\boldsymbol{w}^* = \boldsymbol{P}\boldsymbol{R}^{-1}}$ Solução analítica da otimização (solvelin.m)

Desenvolvimento de um algoritmo iterativo:

Conhecendo-se $\boldsymbol{P}$ e $\boldsymbol{R}$, $\boldsymbol{R}^{-1}$ existe, então, para uma valor particular de $\boldsymbol{w}$:

$$\nabla_w E[e^2] = 2\boldsymbol{w}\boldsymbol{R} - 2\boldsymbol{P}$$

Pós-multiplicando por ½ $\boldsymbol{R}^{-1}$

$$\tfrac{1}{2} \nabla_w E[e^2] \boldsymbol{R}^{-1} = \boldsymbol{w} - 2\boldsymbol{P}\boldsymbol{R}^{-1} = \boldsymbol{w} - \boldsymbol{w}^*$$

Assim, o vetor sináptico ótimo

$\boldsymbol{w}^* = \boldsymbol{w} - ½ \nabla_w \mathrm{E}[e^2] \boldsymbol{R}^{-1}$

pode ser calculado em uma iteração.
Reescrevendo de forma iterativa e escalonando por um coeficiente de aprendizado c_k:

$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - c_k \nabla_w \mathrm{E}[e^2] \boldsymbol{R}^{-1}$ ($c_k = ½ \rightarrow$ método de Newton)

Hipótese LMS:

$$\mathrm{E}[e^2_{k+1} | e^2_0, e^2_1, \ldots e^2_k] = e^2_k$$

e assumindo $\boldsymbol{R} = \boldsymbol{I} \rightarrow$ algoritmo do gradiente descendente estimado:

$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - c_k \nabla_w e^2_k$

Gradiente de e^2_k em relação a $\boldsymbol{w}$

$$\begin{aligned}
\nabla_w e^2_k &= \left[\frac{\partial e_k^2}{\partial w_1}, \quad \cdots \quad \frac{\partial e_k^2}{\partial w_n} \right] \\
&= \left[\frac{\partial (d_k - y_k)^2}{\partial w_1}, \quad \cdots \quad \frac{\partial (d_k - y_k)^2}{\partial w_n} \right] \\
&= \left[-2(d_k - y_k) \frac{\partial y_k}{\partial w_1}, \quad \cdots \quad -2(d_k - y_k) \frac{\partial y_k}{\partial w_n} \right] \\
&= -2 e_k \left[\frac{\partial y_k}{\partial w_1}, \quad \cdots \quad \frac{\partial y_k}{\partial w_n} \right] \\
&= -2 e_k \left[x_k^1, \quad \cdots \quad x_k^n \right] \qquad (y_k = \mathbf{x}_k \mathbf{w}_k^t) \\
&= -2\, e_k \boldsymbol{x}_k
\end{aligned}$$

Assim o algoritmo LMS reduz-se a:

$\boxed{\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + 2c_k\, e_k \boldsymbol{x}_k}$ assume-se $c_k > 0$

Regra delta: $w_{ij} \leftarrow w_{ij} + \mu \delta_{ij} x_{ij}$

Regra delta WH: $w_{ij} \leftarrow w_{ij} + \dfrac{\mu \varepsilon x_{ij}}{\sum x_k^2}$

Esta fórmula computacionalmente simples utiliza apenas valores conhecidos em cada iteração. O operador linear $\boldsymbol{R} : \Re^n \rightarrow \Re^n$, com autovalores $\boldsymbol{e}_i \boldsymbol{R} = \lambda_i \boldsymbol{e}_i$ representa a "incerteza" das amostras estocásticas de $f : \Re^n \rightarrow \Re$. Influi fortemente na convergência do algoritmo. Na prática

$$0 < \mu < \frac{1}{traço(\boldsymbol{R})}$$

2.16. O Perceptron Multicamadas e o algoritmo “backpropagation”

Rumelhart, Hinton e Williams, 1986

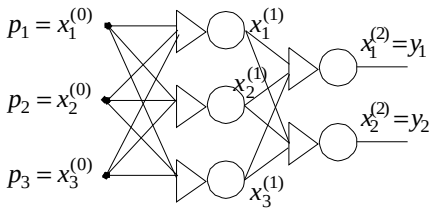


Figura 32 – Perceptron Multicamadas (MLP)

Dinâmica: EP – Elemento Processador (neurônio).

$$s_j^{(k)} = w_{0j}^{(k)} + \sum_i w_{ij}^{(k)} x_i^{(k-1)}$$

$$x_j^{(k)} = f(s_j^{(k)}),$$

com *f* contínua diferenciável.

Treinamento

$\varepsilon^2 = \sum_{j=1}^{m} (d_j - y_j)^2$ - erro quadrático

$\mathbf{w}_j^{(k)} = (w_{oj}^{(k)}, w_{1j}^{(k)}, ..., w_{mj}^{(k)})$ - vetor de pesos do EP *j*.

$\mathbf{x}_j^{(k-1)} = (1, x_{1j}^{(k-1)}, ..., x_{nj}^{(k-1)})$ - vetor de entradas do EP *j*.

Gradiente instantâneo:

$$\nabla_j^{(k)} = \frac{\partial \varepsilon^2}{\partial \mathbf{w}_j^{(k)}} = \left[\frac{\partial \varepsilon^2}{\partial w_{0j}^{(k)}}, \quad \frac{\partial \varepsilon^2}{\partial w_{1j}^{(k)}}, \quad \cdots \quad \frac{\partial \varepsilon^2}{\partial w_{mj}^{(k)}} \right]$$

$$\nabla_j^{(k)} = \frac{\partial \varepsilon^2}{\partial \mathbf{w}_j^{(k)}} = \frac{\partial \varepsilon^2}{\partial s_j^{(k)}} \frac{\partial s_j^{(k)}}{\partial \mathbf{w}_j^{(k)}}$$

como $s_j^{(k)} = \mathbf{w}_j^{(k)} \mathbf{x}_j^{(k-1)}$

$$\frac{\partial s_j^{(k)}}{\partial \mathbf{w}_j^{(k)}} = \mathbf{x}_j^{(k-1)}$$

e portanto

$$\nabla_j^{(k)} = \frac{\partial \varepsilon^2}{\partial \mathbf{w}_j^{(k)}} = \frac{\partial \varepsilon^2}{\partial s_j^{(k)}} \mathbf{x}_j^{(k-1)}$$

O ***erro derivativo quadrático*** é definido por $\boxed{\delta_j^{(k)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_j^{(k)}}}$

$\nabla_j^{(k)} = -2\delta_j^{(k)}\mathbf{x}_j^{(k-1)}$

Para a camada de saída, o erro derivativo quadrático é:

$$\delta_j^{(k)} = -\frac{1}{2}\frac{\partial \sum_{i=1}^{N_k}(d_i - y_i)^2}{\partial s_j^{(k)}} = -\frac{1}{2}\frac{\partial \sum_{i=1}^{N_k}(d_i - f(s_i^{(k)}))^2}{\partial s_j^{(k)}}$$

e como as derivadas parciais para $i \neq j$ se anulam

$$\delta_j^{(k)} = -\frac{1}{2}\frac{\partial (d_j - f(s_j^{(k)}))^2}{\partial s_j^{(k)}} = -(d_j - f(s_j^{(k)}))\frac{\partial (d_j - f(s_j^{(k)}))}{\partial s_j^{(k)}} = (d_j - x_j^{(k)})\, f'(s_j^{(k)})$$

Erro na saída associado ao EP_j da última camada:

$\varepsilon_j^{(k)} = d_j - x_j^{(k)} = d_j - y_j$

implicando em $\boxed{\delta_j^{(k)} = \varepsilon_j^{(k)}.f'(s_j^{(k)})}$

Desenvolvimento para uma camada oculta (*k*):

- O erro quadrático da camada *k* é determinado pelas saídas lineares da camada *k+1*.

$$\delta_j^{(k)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_j^{(k)}}$$

Desenvolvimento para uma camada oculta (*k*):

- O erro quadrático da camada *k* é determinado pelas saídas lineares da camada *k+1*.

$$\delta_j^{(k)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_j^{(k)}} = -\frac{1}{2}\sum_{i=1}^{N_{k+1}}\left(\frac{\partial \varepsilon^2}{\partial s_i^{(k+1)}}\frac{\partial s_i^{(k+1)}}{\partial s_i^{(k)}}\right) \text{ (regra da cadeia)}$$

$$= \sum_{i=1}^{N_{k+1}}\left(\left(-\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_i^{(k+1)}}\right)\frac{\partial s_i^{(k+1)}}{\partial s_i^{(k)}}\right) = \sum_{i=1}^{N_{k+1}}\left(\delta_i^{(k+1)}\frac{\partial s_i^{(k+1)}}{\partial s_i^{(k)}}\right)$$

lembrando que $s_j^{(k)} = w_{0j}^{(k)} + \sum_{i=1}^{N_k} w_{ij}^{(k)} x_i^{(k-1)}$, tem-se

$$\delta_j^{(k)} = \sum_{i=1}^{N_{k+1}}\left(\delta_i^{(k+1)}\frac{\partial}{\partial s_i^{(k)}}\left(w_{0i}^{(k+1)} + \sum_{l=1}^{N_k} w_{li}^{(k+1)} f\left(s_l^{(k)}\right)\right)\right)$$

$$\delta_j^{(k)} = \sum_{i=1}^{N_{k+1}}\left(\delta_i^{(k+1)}\sum_{l=1}^{N_k} w_{li}^{(k+1)}\frac{\partial}{\partial s_i^{(k)}} f\left(s_l^{(k)}\right)\right)$$

observando que

$\frac{\partial}{\partial s_j^{(k)}} f\left(s_l^{(k)}\right) = 0$ se $l \neq j$ e que $\frac{\partial}{\partial s_j^{(k)}} f\left(s_j^{(k)}\right) = f'\left(s_j^{(k)}\right)$

temos

$$\delta_j^{(k)} = \sum_{i=1}^{N_{k+1}}\left(\delta_i^{(k+1)} w_{ji}^{(k+1)} f'\left(s_j^{(k)}\right)\right)$$

$$\delta_j^{(k)} = \underbrace{\left(\sum_{i=1}^{N_{k+1}} \left(\delta_i^{(k+1)} w_{ji}^{(k+1)} \right) \right)}_{\equiv \varepsilon_j^{(k)}} f'\left(s_j^{(k)} \right)$$

Definindo-se $\varepsilon_j^{(k)} = \sum_{i=1}^{N_{k+1}} \delta_i^{(k+1)} w_{ji}^{(k+1)}$

Tem-se o erro derivativo quadrático como: $\delta_j^{(k)} = \varepsilon_j^{(k)} . f'(s_j^{(k)})$

A atualização dos pesos é feita por

$$\mathbf{W}_j^{(k)}(n+1) = \mathbf{W}_j^{(k)}(n) + \mu\left(-\vec{\nabla}\mathbf{W}_j^{(k)}(n) \right)$$

$\mu > 0 \rightarrow$ taxa de aprendizagem
n $\rightarrow$ iteração corrente

$$\boxed{\mathbf{W}_j^{(k)}(n+1) = \mathbf{W}_j^{(k)}(n) + 2\mu\delta_j^{(k)}(n)\mathbf{X}_j^{(k-1)}(n)}$$

O Algoritmo Backpropagation:

1 $w_{ij}^{(k)} \leftarrow random$, inicializar a rede

2 p/ (**x**,**d**), par de treinamento, obter y. Propagação feedforward. $\varepsilon^2 = \sum_{j=1}^{m} (d_j - y_j)^2$

3 k $\leftarrow$ *última camada*

4 para todo elemento *j* da camada *k* faça:

Calcule $\varepsilon_j^{(k)}$ empregando $\varepsilon_j^{(k)} = d_j - x_j^{(k)} = d_j - y_j$ se *k* for a última camada,

$\varepsilon_j^{(k)} = \sum_{i=1}^{N_{k+1}} \delta_i^{(k+1)} w_{ji}^{(k+1)}$ se for uma camada oculta;

Calcule $\delta_j^{(k)} = \varepsilon_j^{(k)} . f'(s_j^{(k)})$

5 $k \leftarrow k-1$ se k>0 vá para o passo 4, senão prossiga.

6 $\mathbf{W}_j^{(k)}(n+1) = \mathbf{W}_j^{(k)}(n) + 2\mu\delta_j^{(k)}(n)\mathbf{X}_j^{(k-1)}(n)$

7 para o próximo par de treinamento vá para o passo 2.

O algoritmo *Error Backpropagation* levou a uma grande aceitação das RNAs por parte da comunidade científica, uma vez que redes multicamadas podem ser treinadas a partir dos dados que representam amostras significativas do processo em questão.

2.17. Considerações práticas sobre o algoritmo *Backpropagation*

Nesta forma mostrada anteriormente algoritmo *Error Backpropagation* é em geral muito lento e suscetível a "patologias" de treinamento. A *paralisia da rede* ocorre em regiões de gradiente próximos de zero (platôs). Dependendo das condições iniciais o treinamento fica preso em *mínimos locais* da superfície de erro.

O critério de parada do treinamento considera em geral um limite máximo do número de épocas de treinamento. Além disso o treinamento pode terminar quando a soma do erro quadrático ou a média deste atinge o seu objetivo.

Para acelerar a convergência utilizam-se variantes do *Backpropagation.* A função *trainbpm* (com momento, m_C) considera não apenas o gradiente local, mas também tendências recentes da superfície de erro. Atua como um filtro passa-baixas, ignorando características menores da superfície de erro. O treinamento "desliza" sobre mínimos locais não muito pronunciados.

$$\Delta \mathbf{W}_j^{(k)}(n+1) = m_C \Delta \mathbf{W}_j^{(k)}(n) + (1-m_C) 2\mu \delta_j^{(k)}(n) \mathbf{X}_j^{(k-1)}(n) \qquad m_C \approx 0{,}95 \text{ (típico)}$$

A função *trainbpx* implementa uma taxa de aprendizagem adaptativa (e com momento, m_C)

Taxa de aprendizagem: 1,04 $\frac{\textit{erro novo}}{\textit{erro anterior}}$ redução de μ: 0,7 aumento de μ: 1,05

O algoritmo de *Levenberg-Marquardt* é um dos mais rápidos; utiliza a Matriz *J* Jacobiana das derivadas do erro (*e*) em relação aos pesos.

$$\Delta \mathbf{W}_j^{(k)} = (J^T J + \mu J)^{-1} J^T e$$

Se o fator de escala μ é muito grande temos o algoritmo do gradiente descendente. Se μ é muito pequeno torna-se o algorimto de Gauss-Newton, que é mais preciso e assim adequado para uma rápida convergência próxima a um mínimo.

2.18. Redes Neurais Recorrentes - A Rede de Hopfield

A rede neural de Hopfield possui uma única camada de neurônios realimentados e implementa assim uma memória auto-associativa, isto é, ao ser apresentado um padrão de *n* bits a rede retorna um padrão armazenado de *n* bits que lhe é mais próximo (associado). Pela facilidade de treinamento e velocidade de operação a rede deHopfield tem sido escolhida para implementações em VLSI.

Sistemas realimentados precisam ser projetados com cuidado pois uma escolha inadequada dos pesos pode levar o sistema a apresentar comportamento instável. A escolha dos pesos da rede de Hopfield garante a sua estabilidade. A estrutura desta rede é mostrada na figura a seguir.

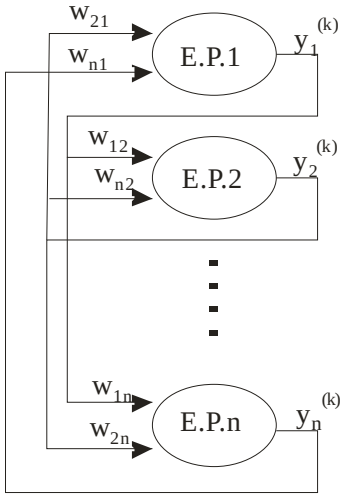


Figura 33 – Rede de Hopfield com n Elementos Processadores.

Dinâmica:

$$s_j^{(k)} = \sum_{i=1}^{n} w_{ij}\, y_i^{(k)}$$

$$y_j^{(k+1)} = f\left(s_j^{(k)}\right)$$

Inicialização da rede: $\mathbf{y}^{(0)} = \mathbf{x}$

Vetor de saída: $\mathbf{y}^{(k)} = \left[y_i^{(k)}\right]$

Para Sistemas Binários utiliza-se:

$$f(s_j) = \begin{cases} 1 \; se \; s_j > L_j \\ 0 \; se \; s_j < L_j \\ mantem \, valor \, anterior, \; se \; s_j = L_j \end{cases}$$

Desta forma a rede de Hopfield pode ser vista como um sistema IIR (Infinite Impulse Response) com entrada nula, uma vez que pode ser descrita por um conjunto de equações a diferenças.

Formas de operação:

- Assíncrona
- Síncrona
- Seqüêncial

Define se o estado da rede como o conjunto de todas as saídas correntes.

Exemplo:

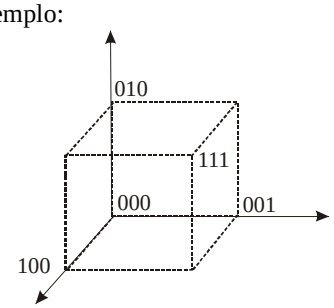


Figura 34 – Rede de Hopfield com 3 E.P. ⇒ 8 estados possíveis.

Aprendizagem:
Os padrões a serem armazenados na memória associativa são escolhidos à priori.
Cada padrão p: $A_p = \left[a_1^p \quad a_2^p \quad \ldots \quad a_n^p \right]$, com $a_i^p = 0$ ou 1
m padrões distintos, $L_i = 0$.

$$w_{ij} = \sum_{p=1}^{m} (2a_i^p - 1)(2a_j^p - 1)$$

A expressão $(2a_i^p - 1)$ converte 0 e 1 para –1 e +1.

w_{ij} é incrementado de 1 se $a_i^p = a_j^p$, e decrementado de 1 caso contrário.

Este procedimento é repetido para todos i, j e para todos os padrões A_p.
⇒ Adicionar padrões à memória é um processo análogo ao reforço no ensino.

Exemplo:

Símbolo	Vetor de Treinamento
L	A_1 = [1 0 0 1 0 0 1 1 1]
T	A_2 = [1 1 1 0 1 0 0 1 0]
+	A_3 = [0 1 0 1 1 1 0 1 0]

a_1	a_2	a_3
a_4	a_5	a_6
a_7	a_8	a_9

1		
1		
1	1	1

1	1	1
	1	
	1	

	1	
1	1	1
	1	

Matriz de pesos após treinamento:

$$W = \begin{bmatrix} 0 & -1 & 1 & -1 & -1 & -3 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 3 & 1 & -3 & 1 & -3 \\ 1 & 1 & 0 & -3 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -3 & 0 & -1 & 1 & 1 & 1 & 1 \\ -1 & 3 & 1 & -1 & 0 & 1 & -3 & 1 & -3 \\ -3 & 1 & -1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & -3 & -1 & 1 & -3 & -1 & 0 & -1 & 3 \\ 1 & 1 & -1 & 1 & 1 & -1 & -1 & 0 & -1 \\ 1 & -3 & -1 & 1 & -3 & -1 & 3 & -1 & 0 \end{bmatrix}$$

Consideremos agora o seguinte padrão apresentado à rede:

1		1
1		
	1	1

Teremos a seguinte evolução da rede, considerando a operação seqüencial.

EP disparado	Soma do EP	Saída do EP	Novo vetor de saída
1	2	1	1 0 1 1 0 0 0 1 1
2	-3	0	1 0 1 1 0 0 0 1 1
3	-4	0	1 0 **0** 1 0 0 0 1 1
4	1	1	1 0 0 1 0 0 0 1 1
5	-4	0	1 0 0 1 0 0 0 1 1
6	-4	0	1 0 0 1 0 0 0 1 1
7	4	1	1 0 0 1 0 0 **1** 1 1
8	0	1	1 0 0 1 0 0 1 1 1
9	4	1	1 0 0 1 0 0 1 1 1
1	2	1	1 0 0 1 0 0 1 1 1
2	-8	0	1 0 0 1 0 0 1 1 1

Assim, após a convergência, a rede retorna o símbolo “L”, como aquele que está mais próximo ao padrão de entrada.

A figura a seguir ilustra o funcionamento da rede de Hopfield em termos de energia da rede.

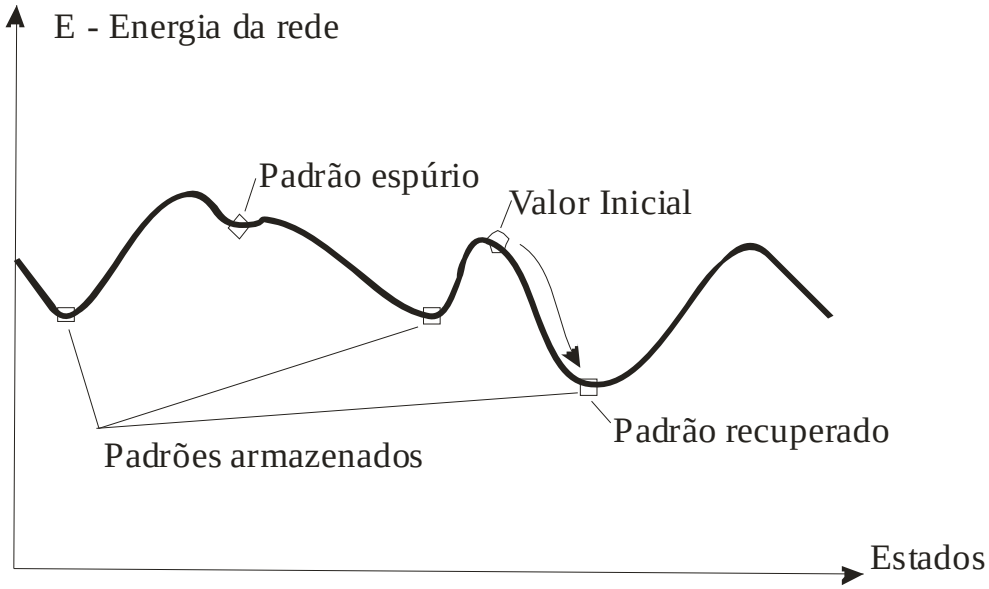


Figura 35 – Padrões e a função de energia típica de uma rede de Hopfield.

Os padrões armazenados na Rede de Hopfield são mínimos locais da função de energia (estados de equilíbrio). A partir de um padrão apresentado (valor inicial) a rede estabiliza no mínimo de energia de sua respectiva “bacia de atração”. Isto é, não se retorna necessariamente o padrão geometricamente mais próximo. Além disso pode acontecer de que haja mínimos locais não desejados, levando a rede a retornar padrões espúrios.

Estabilidade da rede de Hopfield:

A estabilidade da rede de Hopfield é determinada pela matriz **W** = [w_{ij}].

Cohen e Grossberg mostraram em 1983 que "se **W** é simétrica e sua diagonal principal é nula, então a rede recorrente é estável. Esta é uma condição suficiente, mas não necessária para a estabilidade.

Prova:
Considere a seguinte função de Liapunov associada à energia de todos os estados:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} y_i y_j - \sum_j x_j y_j + \sum_j y_j L_j$$

Onde *E* é a energia (artificial) associada ao estado da rede.
A variação da energia devido à variação do estado de um neurônio *k* é dada por:

$$\Delta E_k = -\frac{1}{2}\sum_{i\neq k} w_{ik} y_i \Delta y_k - \frac{1}{2}\sum_{j\neq k} w_{kj} \Delta y_k y_j - x_k \Delta y_k + \Delta y_k L_k$$

pela condição de simetria

$$\Delta E_k = -\sum_{i\neq k} w_{ij} y_i \Delta y_k - x_k \Delta y_k + \Delta y_k L_k$$

e agrupando

$$\Delta E_k = -\left[\sum_{i\neq k} w_{ij} y_i + x_k - L_k\right]\Delta y_k$$

De acordo com a dinâmica de Hopfield, têm-se

$$\Delta E_k = -[s_k - L_k]\Delta y_k$$

As seguintes situações são possíveis:

1. se $s_k > L_k$ então $y_k = 1 \therefore \Delta y_k = 1 \Rightarrow \Delta E_k < 0$
2. se $s_k < L_k$ então $y_k = 0 \therefore \Delta y_k = -1 \Rightarrow \Delta E_k < 0$
3. se $s_k = L_k$ então y_k inalterado $\therefore \Delta y_k = 0 \Rightarrow \Delta E_k = 0$

Portanto a Energia só decresce até que o sistema se estabiliza em um ponto de equilíbrio.

Limitações:
1- Não é retornado necessariamente o padrão mais próximo.
2- Diferenças entre padrões. Nem todos os padrões têm igual ênfase.
3- Padrões espúrios, i.é, padrões evocados que não constam do rol de padrões originais.
4- Número máximo de padrões é limitado. $m \leq 0{,}5n / \log n$ senão esquecimento.

Conclusão:

A rede de Hopfield é mais de interesse acadêmico, pois mostra uma forma de tratar redes neurais recorrentes.

2.19. Redes Neurais de Base Radial

As duas grandes aplicações de RNAs são como classificadores de padrões e como aproximadores de funções. As redes neurais de base radial (RBFs) são uma alternativa interessante para o segundo caso, e se destacam pela rapidez de aprendizado.

Os trabalhos originais nesta área foram feitos por Moody & Darken, 1989, Che et al, 1991 e Girosi, 1992.

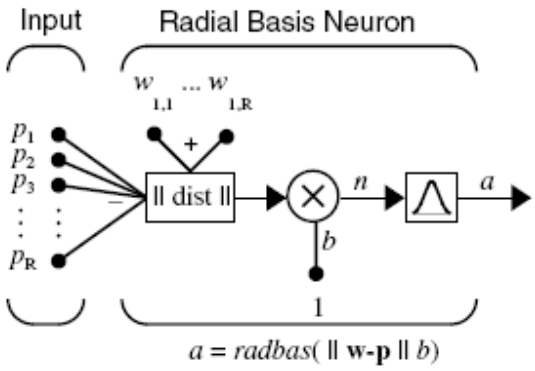
As redes RBF tiveram sua origem na observação de sistemas biológicos:

- O processamento de informações sensoriais é feito por campos sobrepostos de recepção presentes em regiões do córtex cerebral.
- Observa-se uma atividade local de seus elementos processadores.

Neurônio (microestrutura)

Em geral, em relação às RBFs:

- Treinamento rápido, porém necessita mais neurônios que o MLP.
- Permite treinamento incremental. Novos pontos podem ser aprendidos sem perder aprendizado anterior.
- Permite utilizar conhecimento à priori, para localizar neurônios (o que não é possível no MLP).

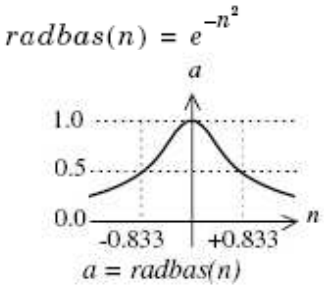


$$a_i = e^{-\frac{\|x_i - \mu_i\|}{\sigma_i^2}} \rightarrow Gaussiana$$

$$Média,\ Variância$$

$$a_i = e^{-\|p_i - w_i\| b}$$

$$a = radbas(dist(w, p) * b)$$



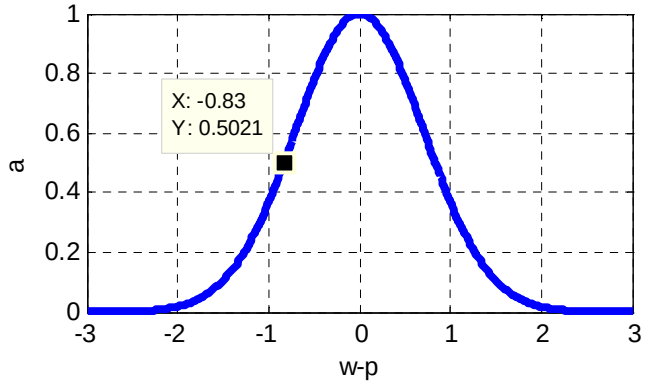
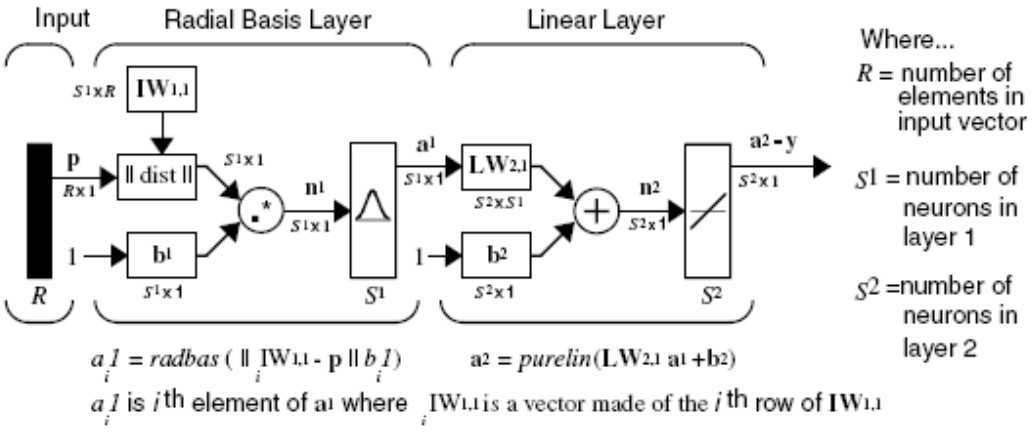


Figura 36 – Neurônio RBF – Função de Base Radial.

Meso-Estrutura



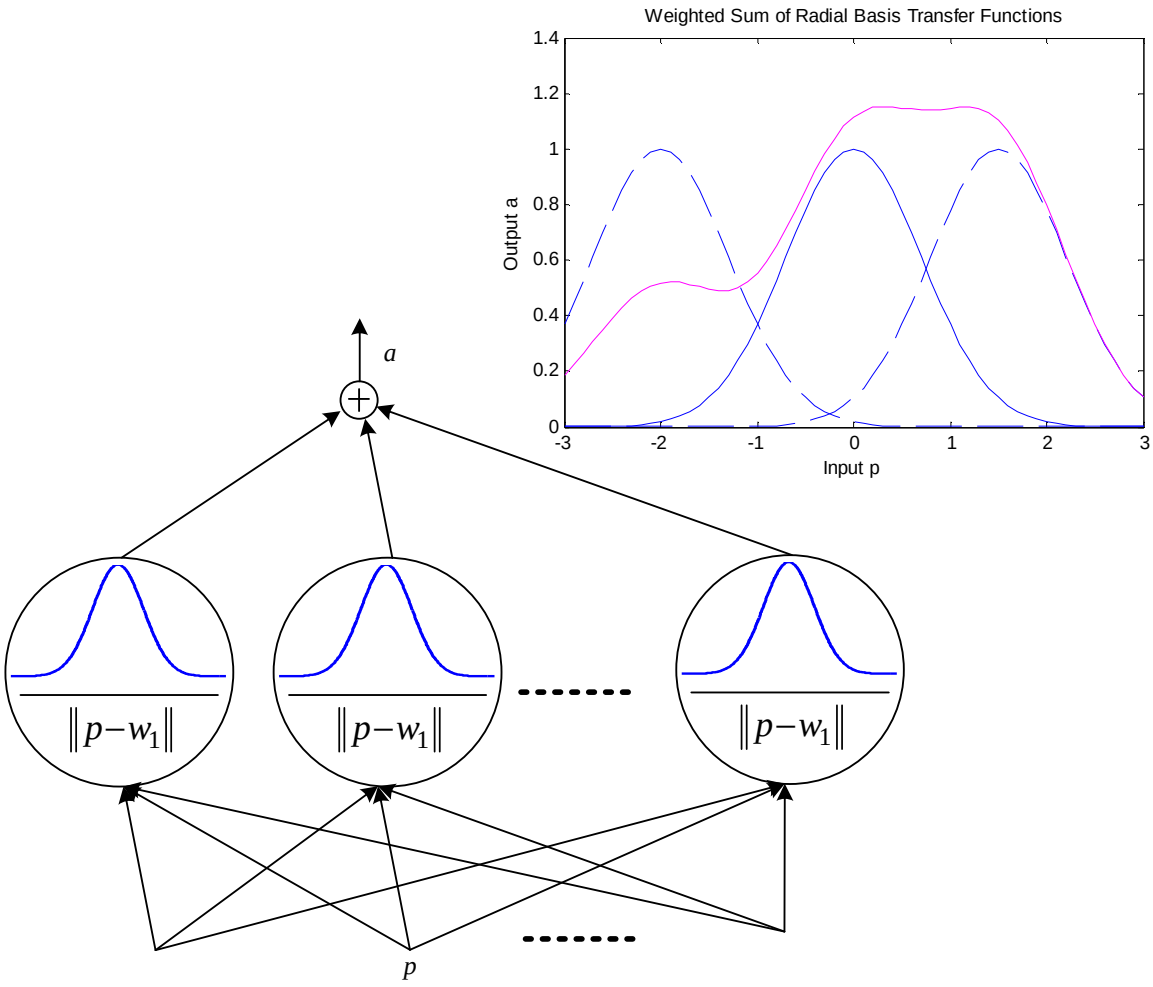


Figura 37 – Rede Neural com Função de Base Radial – Camada gaussiana e camada linear.

Treinamento de Redes de Funções de Base Radial

Projeto Exato – 1 neurônio para cada par entrada saída
→ erro nulo para os dados de treinamento!

net = newrbe(P,T,SPREAD)

P - RxQ matriz dos Q vetores de entrada (“pattern”),
T - SxQ matriz dos Q vetores objetivo (“target”).
SPREAD – espalhamento da função de base radial, default = 1,0 (vale para todos os neurônios).
Em ± SPREAD a gaussiana responde 0,5

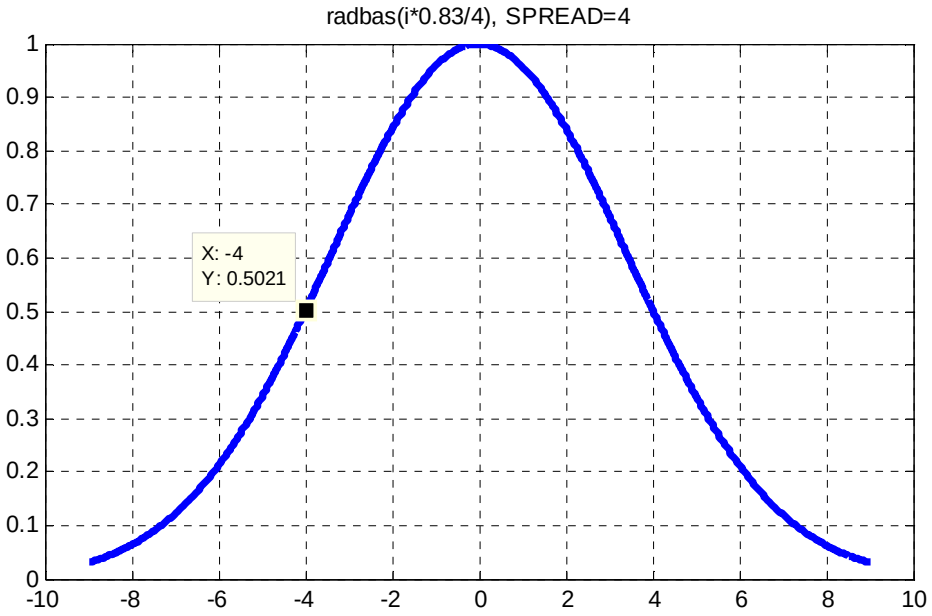


Figura 38 – Função de Base Radial com fator de espalhamento 4.

“Treinamento”:
1- $W_{1i} = p_i'$
2- $b_i = 0{,}83/\text{SPREAD}$
3- Para a camada Purelin S_2: Solvelin $\rightarrow \min \sum erro^2 = 0$

Q restrições (pares entrada/saída)
Cada neurônio possui Q+1 variáveis (Q pesos e um “bias”) → infinitas soluções

→ Solvelin – solução de norma mínima $\sum_j \left(\sum_i w_i^2 + b^2 \right)$

Projeto Incremental – Acrescentam-se neurônios individualmente até que o erro especificado seja satisfeito. A posição onde é inserido o neurônio é escolhida de forma a minimizar o erro $\left(\sum erro^2 \rightarrow \min\right)$ naquele passo.

[net,tr] = newrb(P,T,GOAL,SPREAD,MN,DF)

P - RxQ matriz dos Q vetores de entrada (“pattern”),
T - SxQ matriz dos Q vetores objetivo (“target”),
GOAL - erro médio quadrático pretendido, default = 0,0,
SPREAD – espalhamento da função de base radial, default = 1,0,
MN - Número máximo de neurônios, default é Q,
DF - Número de neurônios adicionados entre apresentações na tela, default = 25.

Casos Patológicos:

Fator de espalhamento muito pequeno leva à perda da capacidade de generalização.

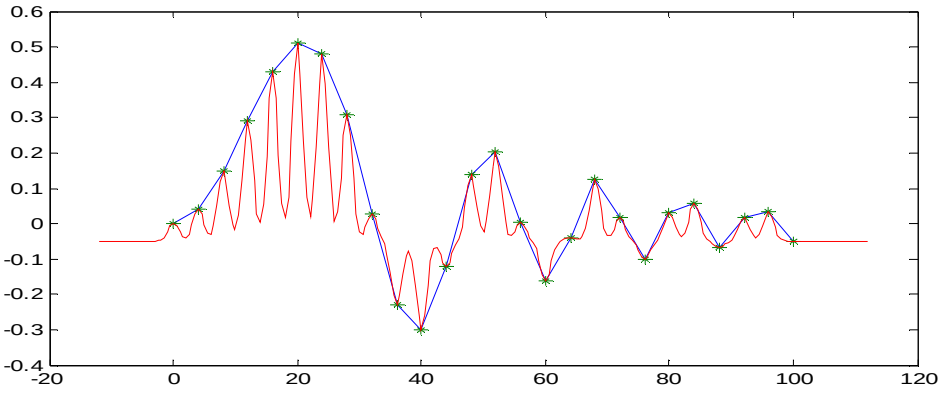


Figura 39 – Aproximação de funções RBF com fator de espalhamento muito pequeno.

Fator de espalhamento muito grande não permite aproximar os detalhes da função.

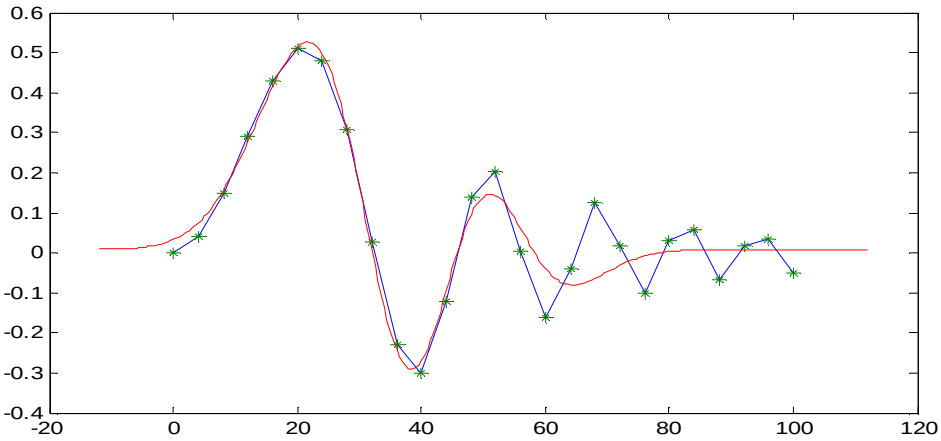


Figura 40 – Aproximação de funções RBF com fator de espalhamento muito grande.

Heurística para a escolha do fator de espalhamento: $|x_{i+1} - x_i| < SPREAD < |x_{\max} - x_{\min}|$

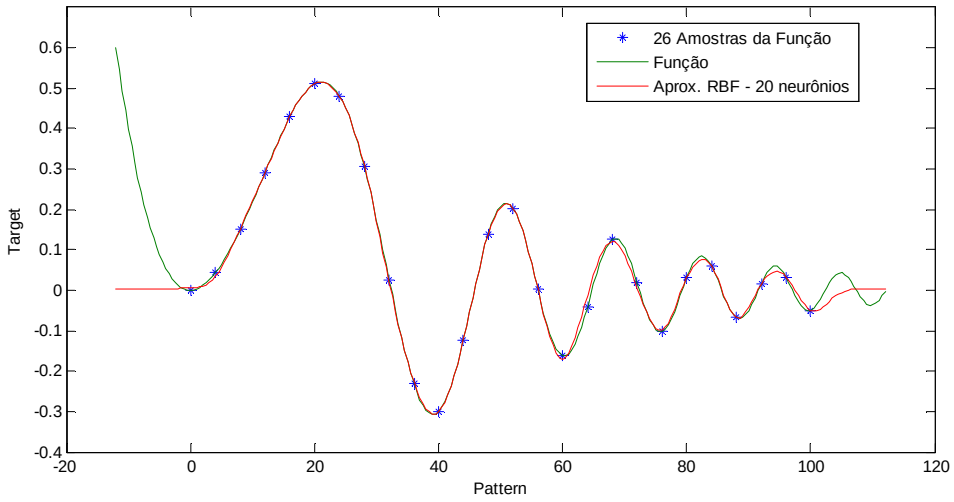


Figura 41 – Aproximação de funções RBF com fator de espalhamento adequado.

O treinamento de uma rede RBF não é ótimo pois os neurônios são acrescidos sem que seja possível alterar a posição dos neurônios anteriores. O treinamento incremental é mais simples e rápido, embora possa haver, teoricamente, uma rede melhor para uma otimização global.

Conclusões

- Treinamento rápido, porém utiliza mais neurônios que o MLP.
- Treinamento incremental, novos pontos podem ser aprendidos sem perder o aprendizado anterior.
- Pode-se utilizar conhecimento a priori, para localizar os neurônios (o que não é possível em uma MLP).

2.20 Aprendizagem Competitiva

Redes Auto-Organizadas – Classificação de Padrões Não-Supervisionada

a = compet(-dist(W,p)+b) { 1 p/ o neurônio vencedor
0 p/ os demais

net = newc(PR,S,KLR,CLR)

PR - Rx2 matrix of min and max values for R input elements.
S - Number of neurons.
KLR - Kohonen learning rate, default = 0.01.
CLR - Conscience learning rate, default = 0.001.

net = train(net,P);
Y = sim(net,P)
Yc = vec2ind(Y)

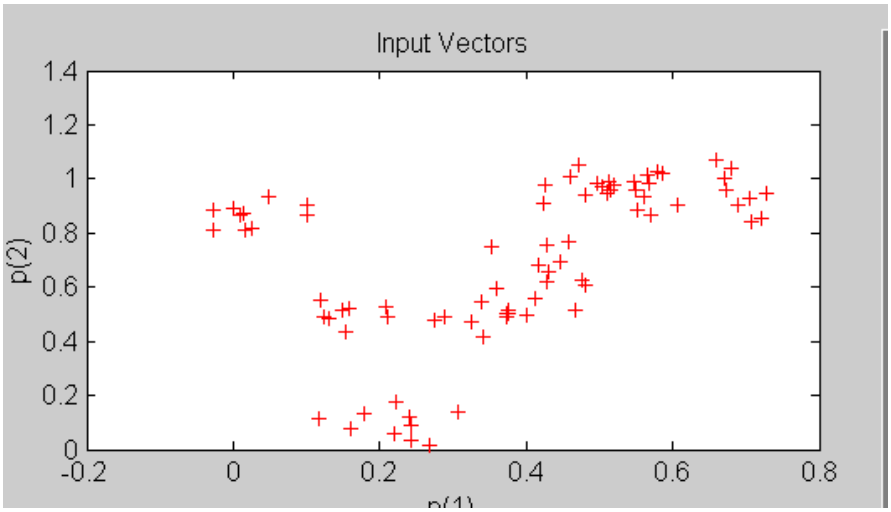


Figura 42 - Padrão a ser classificado.

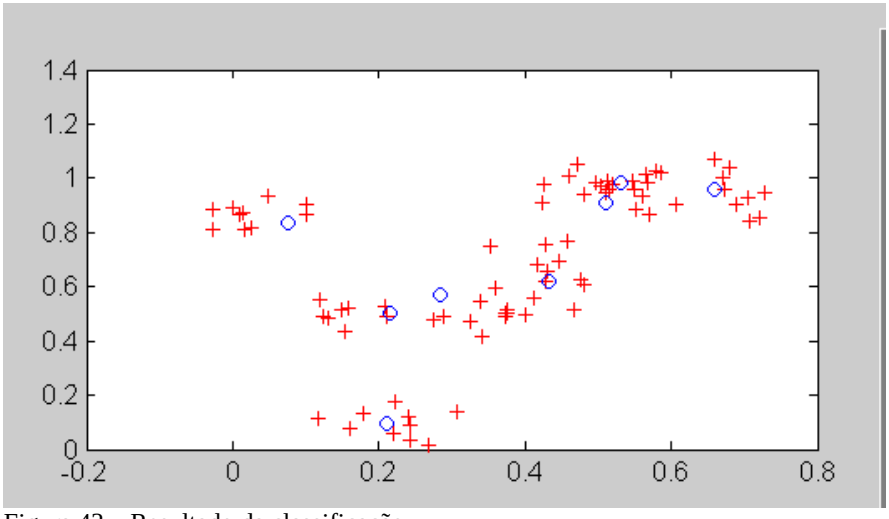


Figura 43 – Resultado da classificação.

$Wc(t + 1) = Wc(t) + \alpha(t) [X(t) – Wc(t)]$
c – neurônio vencedor

2.21 Quantização Vetorial por Aprendizagem - LVQ - Learning Vector Quantization

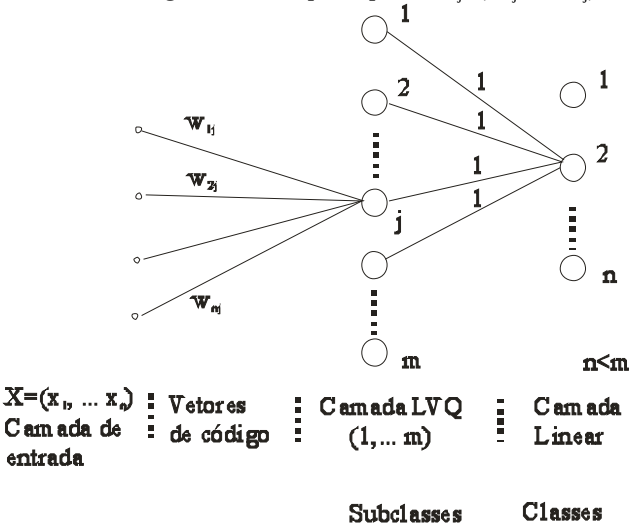
Kohonen 1989

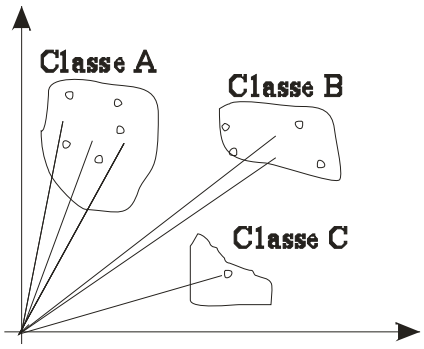
Rede de uma camada ativa, treinamento supervisionado.

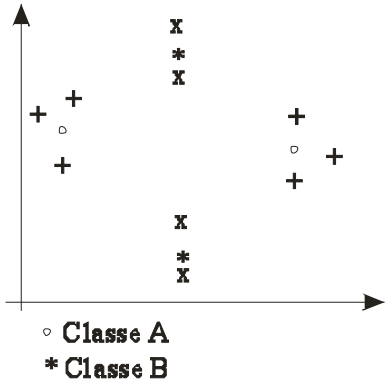
Objetivo: distribuir vetores de código (codebook-vectors) sobre o espaço de entrada de maneira a abrangê-lo da melhor forma possível.

Operação: Um novo vetor de entrada é comparado com os vetores de código e aquele que lhe é mais semelhante indica a classe deste vetor de entrada.

O vetor de código é definido pelos pesos $W_j=(w_{1j},..., w_{nj})$







Existem 3 variantes principais da rede neural LVQ.

LVQ1

Compara-se o vetor de entrada X com os vetores de código Wj. O neurônio *c*, cujo vetor de código Wc é mais semelhante a X é dito o neurônio vencedor. A semelhança é medida por uma norma; em geral utiliza-se a minimização da norma L_2 euclidiana da diferença:

$||X – Wc|| = \min_j (||X - W_j||)$

Obs: X-Wj ->min ; <X|Wj> -> Max

Este algoritmo é uma variante do Classificador do Vizinho mais Próximo, porém não se armazenam todos os padrões; há um procedimento de treinamento de Wj.

Algoritmo de Treinamento

Ajuste de Wj para minimizar o erro de classificação.
O vetor de código Wc do neurônio vencedor, que é mais semelhante ao vetor de entrada X é feito mais semelhante a X caso *c* pertença à mesma classe de X; caso contrário menos semelhante. Os demais neurônios permanecem inalterados.

$Wc(t+1) = Wc(t) + \alpha(t)[X(t)-Wc(t)]$ se classe (Wc) = classe (X)
$= Wc(t) - \alpha(t)[X(t)-Wc(t)]$ se classe (Wc) $\neq$ classe (X)

$Wj(t+1) = Wj(t)$ para todo $j \neq c$

$0 < \alpha(t) < 1$, taxa de aprendizagem, usualmente $\alpha(0) = 0,1$ e decai linearmente.

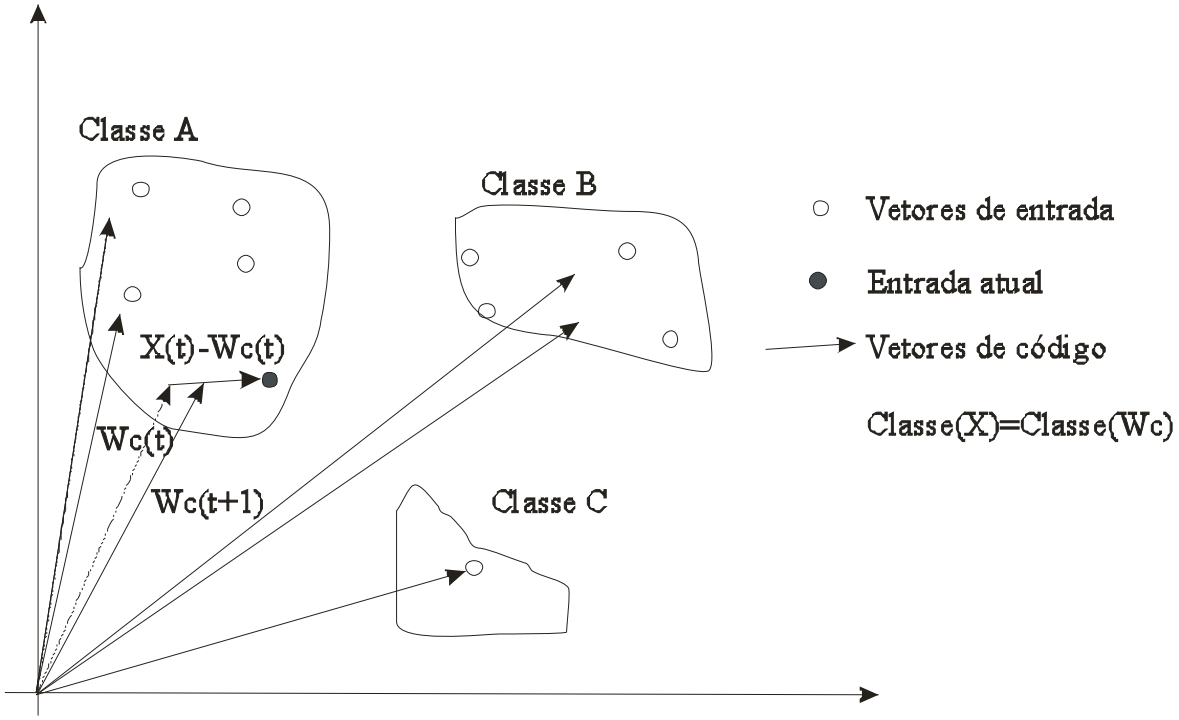


Figura 44 – Visualização to treinamento LVQ1

LVQ2.1

Classificação das entradas como LVQ1
Algoritmo de Treinamento

Busca-se o vetor mais próximo Wi e o segundo mais próximo Wj. Uma adaptação ocorre sse:

1. Classe(Wi) ≠ Classe(Wj)
2. X pertence à classe (Wi) ou à classe (Wj)
3. X está contido na "janela" entre Wi e Wj, $\min\left(\frac{di}{dj}, \frac{dj}{di}\right) > s$

di = dist(X,Wi); dj = dist(X,Wj);

$s = \frac{1-v}{1+v}$ *v*-largura relativa da janela. Kohonen recomenda $0.2 \le v \le 0.3$. Para v=0,2 → s=2/3

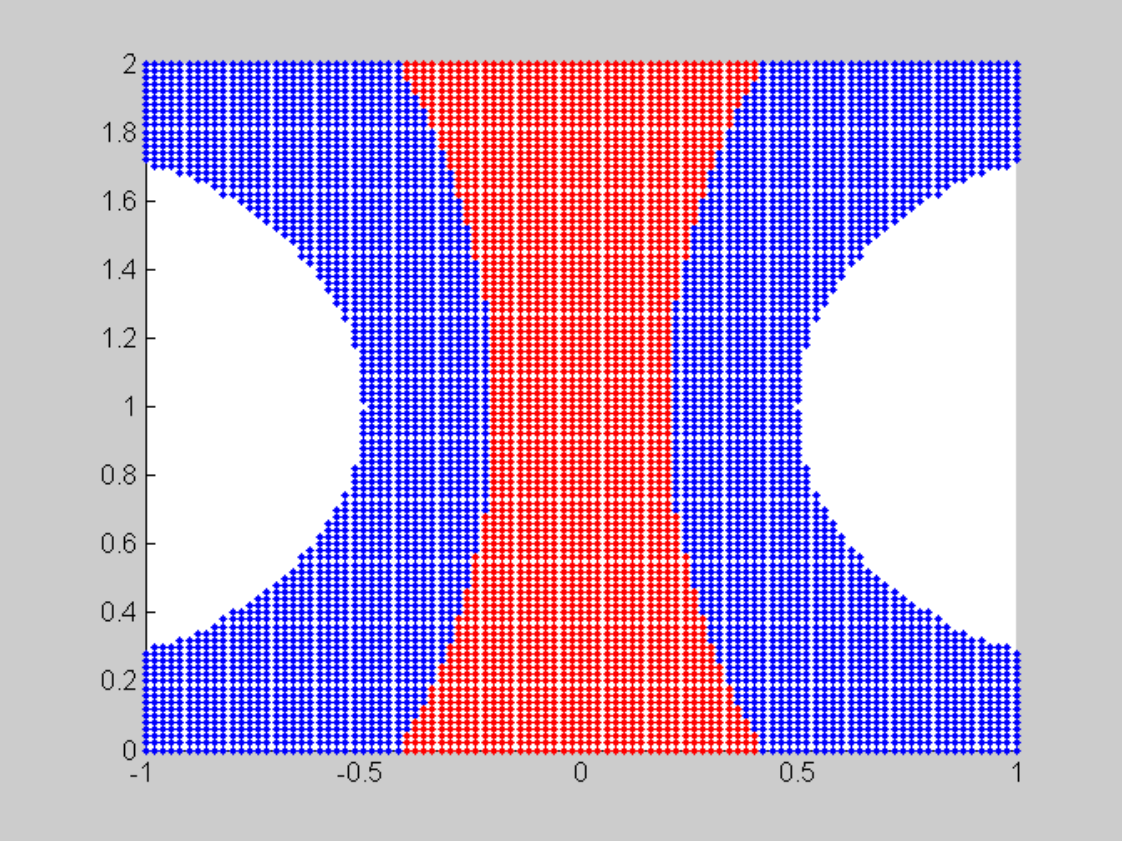


Figura 45 – Janela de treinamento para s=1&Visualização to treinamento LVQ1

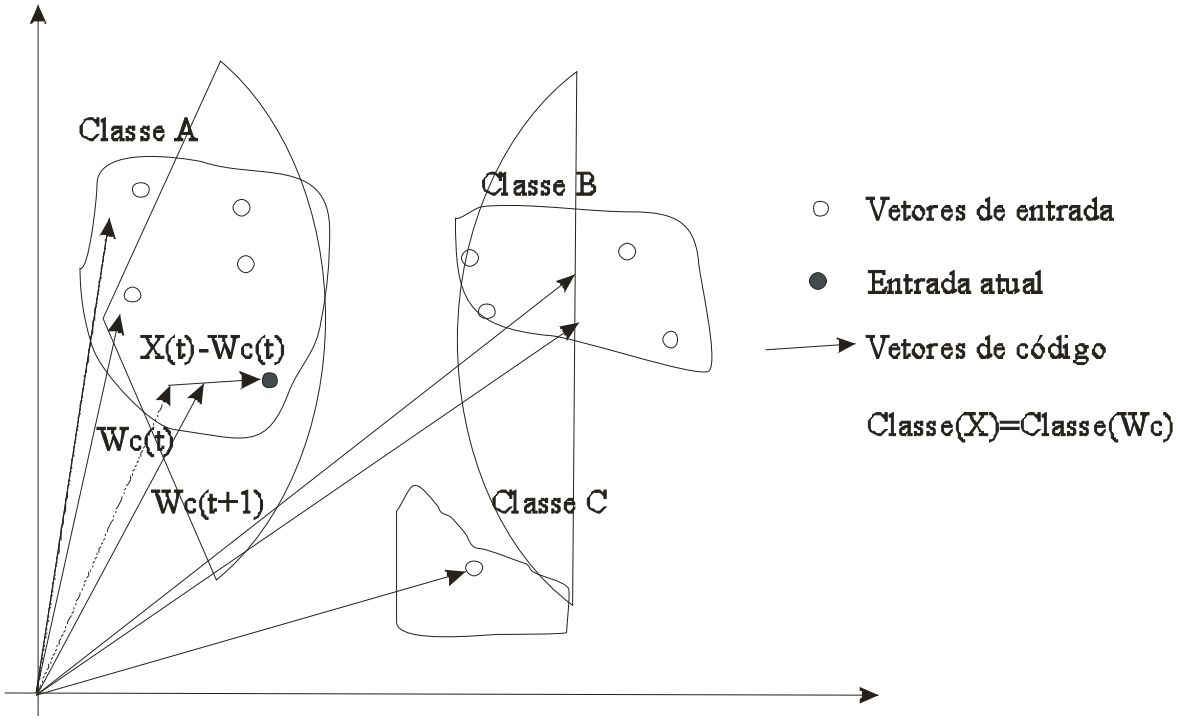


Figura 46 – Visualização to treinamento LVQ2.1

$Wi(t+1) = Wi(t) + \alpha(t)[X(t)-Wi(t)]$
$Wj(t+1) = Wj(t) - \alpha(t)[X(t)-Wj(t)]$

Este algoritmo modifica a fronteira entre as classes, pelo deslocamento dos vetores de código, mas não garante que a distribuição de vetores corresponda à distribuição de probabilidade de entrada.

LVQ3

Modificação do LVQ2.1 em que também ocorre adaptação quando Wi e Wj pertencem ambos à classe(X).

$Wi(t+1) = Wi(t) + \alpha(t)[X(t)-Wi(t)]$
$Wj(t+1) = Wj(t) - \alpha(t)[X(t)-Wj(t)]$

Classe (Wi) = classe (X); classe(Wj) $\neq$ classe(X); X dentro da janela.

$Wi(t+1) = Wi(t) + e\alpha(t)[X(t)-Wi(t)]$
$Wj(t+1) = Wj(t) + e\alpha(t)[X(t)-Wj(t)]$

Classe (X)= classe (Wi) = classe(Wj) Kohonen recomenda $0,1 \leq e \leq 0,5$.

A estabilidade de LVQ3 é melhor que LVQ 2.1.
LVQ1 e LVQ3 ajustam a posição dos vetores de código à distribuição dos vetores de entrada.

LVQ2.1 define a fronteira entre as classes. É recomendado para acentuar a separação de classes treinadas por LVQ1 ou LVQ3.

OLVQ Optimized Learning Vector Quantization

$Wc(t+1) = Wc(t) + \alpha_c(t)[X(t)-Wc(t)]$ Classe (Wc) = classe (X);
$Wc(t+1) = Wc(t) - \alpha_c(t)[X(t)-Wj(t)]$ Classe (Wc) $\neq$ classe (X);

$Wj(t+1) = Wj(t)$ para todo $j \neq c$
Taxa de aprendizagem $\alpha_j(t)$ individualizada.

Conclusões sobre LVQ

Precisão do classificador depende de:
de vetores de código por classe
- inicialização
- treinamento, algoritmo
- taxa de aprendizagem
- critério de parada

A inicializaçãp pode ser feita com os vetores de entrada, isto evita “dead neurons”, neurônios que nunca são vencedores.

Kohonen recomenda começar o treinamento com OLVQ, e 30 a 50 vezes a quantidade de vetores de código.

2.22 Mapas Auto-Organizados (SOM)

As redes SOM (Self-Organizing Maps), também conhecidas como mapas de Kohonen, foram desenvolvidas em diversos trabalhos por Teuvo Kohonen, em 1982, 84, 89, 90 e 92. São uma evolução das LVQs, incluindo relações de vizinhança entre os nerônios.

Princípios:
- Rede Neural de uma camada ativa.
- Treinamento não supervisionado.

- A adaptação dos vetores de código Wj, $0 \leq j \leq m$, deve refletir a função de distribuição de probabilidade do espaço de entrada Xp, $0 \leq p \leq n$, considerando relações de vizinhança.
- Por questões de visualização utilizam-se principalmente cadeias unidimensionais e grades bi- ou tridimensionais. Outras são possíveis.
- A "grade elástica" faz com que alterações em um neurônio também afetem os seus vizinhos. Quanto maior a distância do ponto alterado, menor sua influência.
- Conservação topológica no mapeamento dos vetores de entrada em $\Re^n$ sobre a grade de neurônios m-dimensional.

Aplicações:
Buscar e visualizar relações de similaridade no espaço de entrada. O mapeamento é calculado iterativamente através de um algoritmo de treinamento.

Algoritmo de Treinamento:
Compara-se o vetor de entrada $X = (X_1, X_2, ...X_n)$ paralelamente com todos os vetores de código $Wj = (W_1j,...Wnj)$. Em geral utiliza-se a métrica euclidiana:

$$||X - Wc|| = \min_j(||X - Wj||)$$

Wc é o neurônio vencedor, o que está mais próximo de X.
Em caso de vetores normalizados pode-se também utilizar o produto escalar:

$$<X|Wj> = \Sigma XiWij = net_j$$

onde o neurônio vencedor é o que apresenta o maior produto escalar.

Lei de Aprendizagem

$$Wj(t + 1) = Wj(t) + \eta(t)hcj(t)[X(t) - Wj(t)]$$

hcj(t) – função distância sobre a grade de neurônios (*neighborhood kernel*)
$\eta(t)$ - taxa de aprendizagem variante com o tempo $0 < \eta(t) < 1$

$$hcj(t) = h(||rc - rj||, t) = h(z, t) = h'(z, d)$$

rc, rj - vetores posição dos neurônios c e j na grade.
Quanto maior a distância do neurônio vencedor $z = ||rc - rj||$ $hcj(t) \rightarrow 0$.
Uma alternativa à variação com t é utilizar a distância d, com $d \rightarrow 0$ para $t \rightarrow \infty$ em h'(z,d).

Utilizando-se as relações de vizinhança parametrizadas em d, podem ser utilizadas as seguintes relações práticas em função de z = ||rc – rj||

$$\text{hgauss1}(z,d) = e^{-(z/d)^2}$$

$$\text{hcil}(z,d) = \begin{cases} 1 \ se \quad z < d \\ 0 \ \text{sen}\, ao \end{cases}$$

$$\text{hcone}(z,d) = \begin{cases} 1 - z/d \quad se \quad z < d \\ 0 \ \text{sen ao} \end{cases}$$

$$\text{hcos}(z,d) = \begin{cases} \cos\left(\frac{z}{d}\frac{\pi}{2}\right) \quad se \quad z < d \\ 0 \ \text{senao} \end{cases}$$

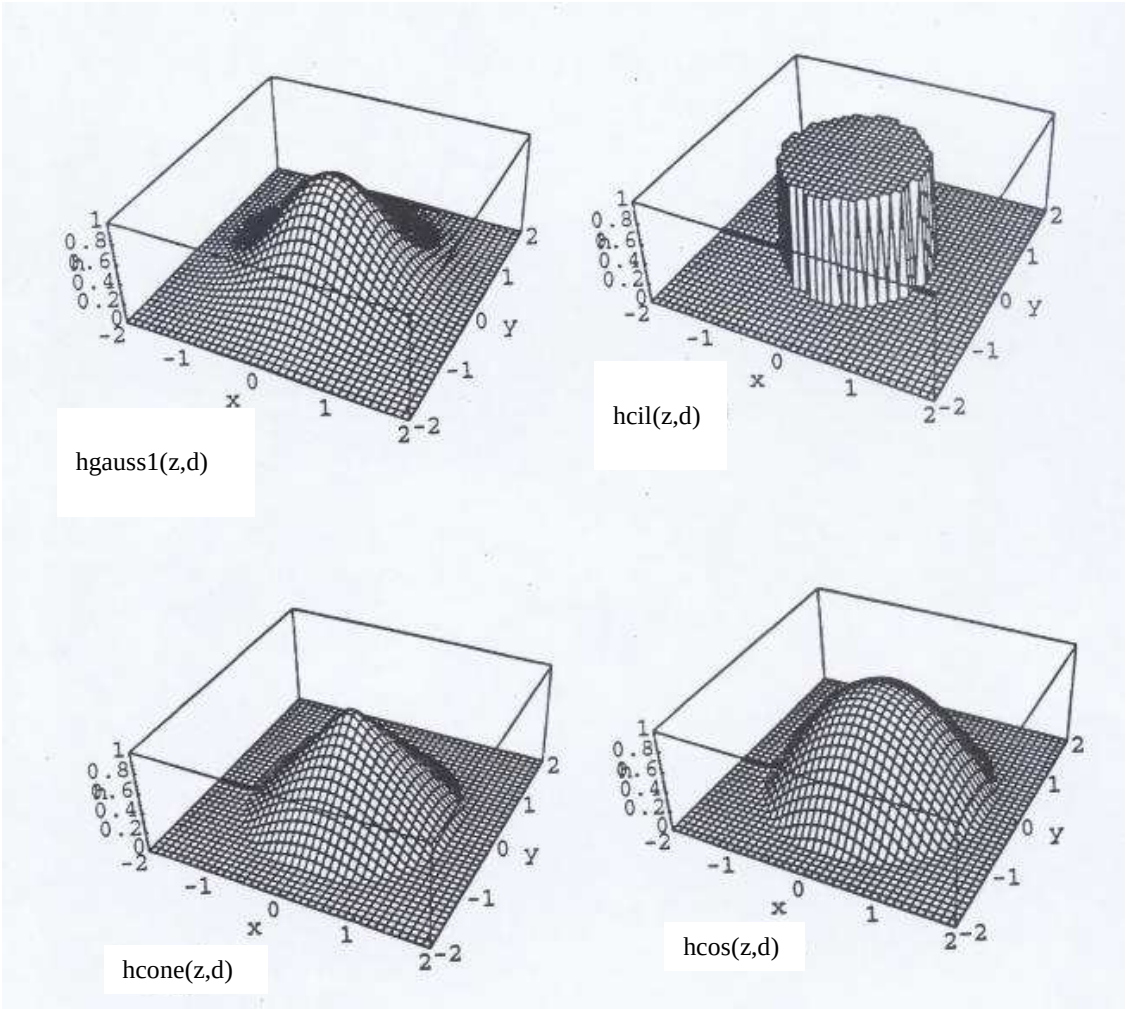


Figura 47 – Funções de vizinhança utilizadas no mapa de Kohonen.

Caso Unidimensional

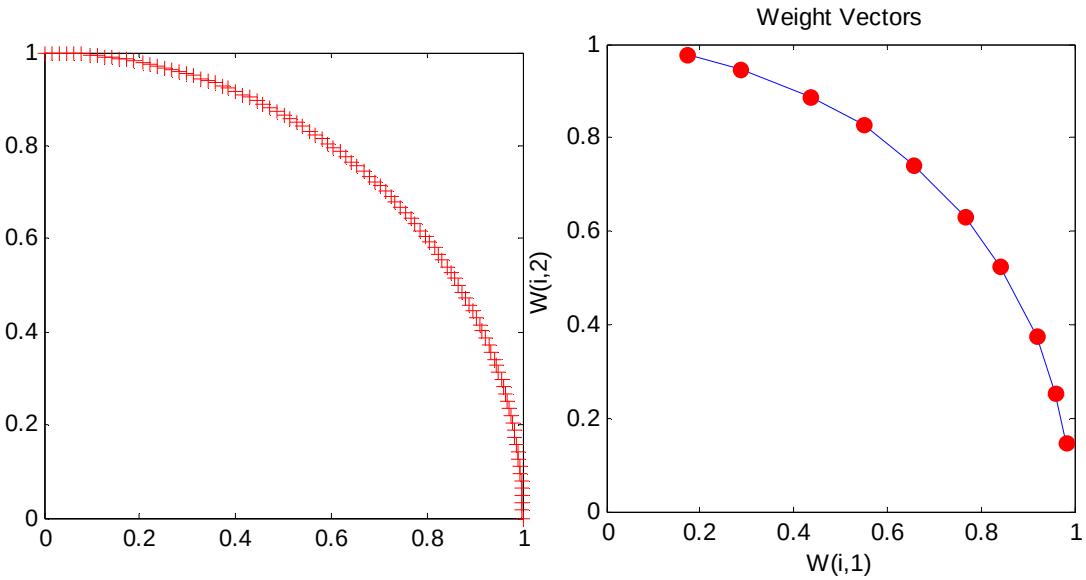


Figura 48 – Distribuição de entrada e mapa de Kohonen unidimensional resultante.

Caso Bidimensional

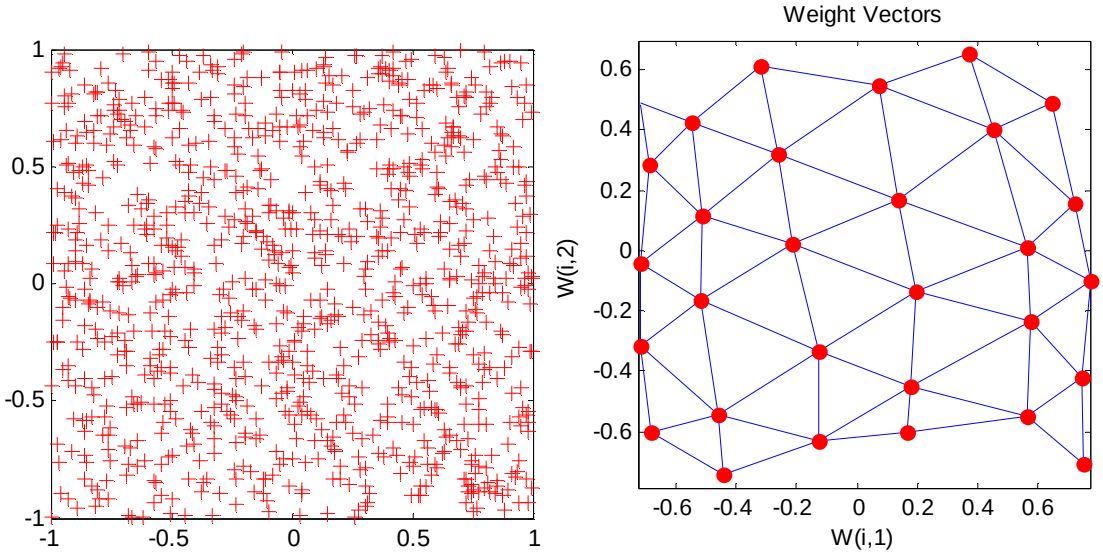


Figura 49 – Distribuição de entrada e mapa de Kohonen bidimensionl resultante.

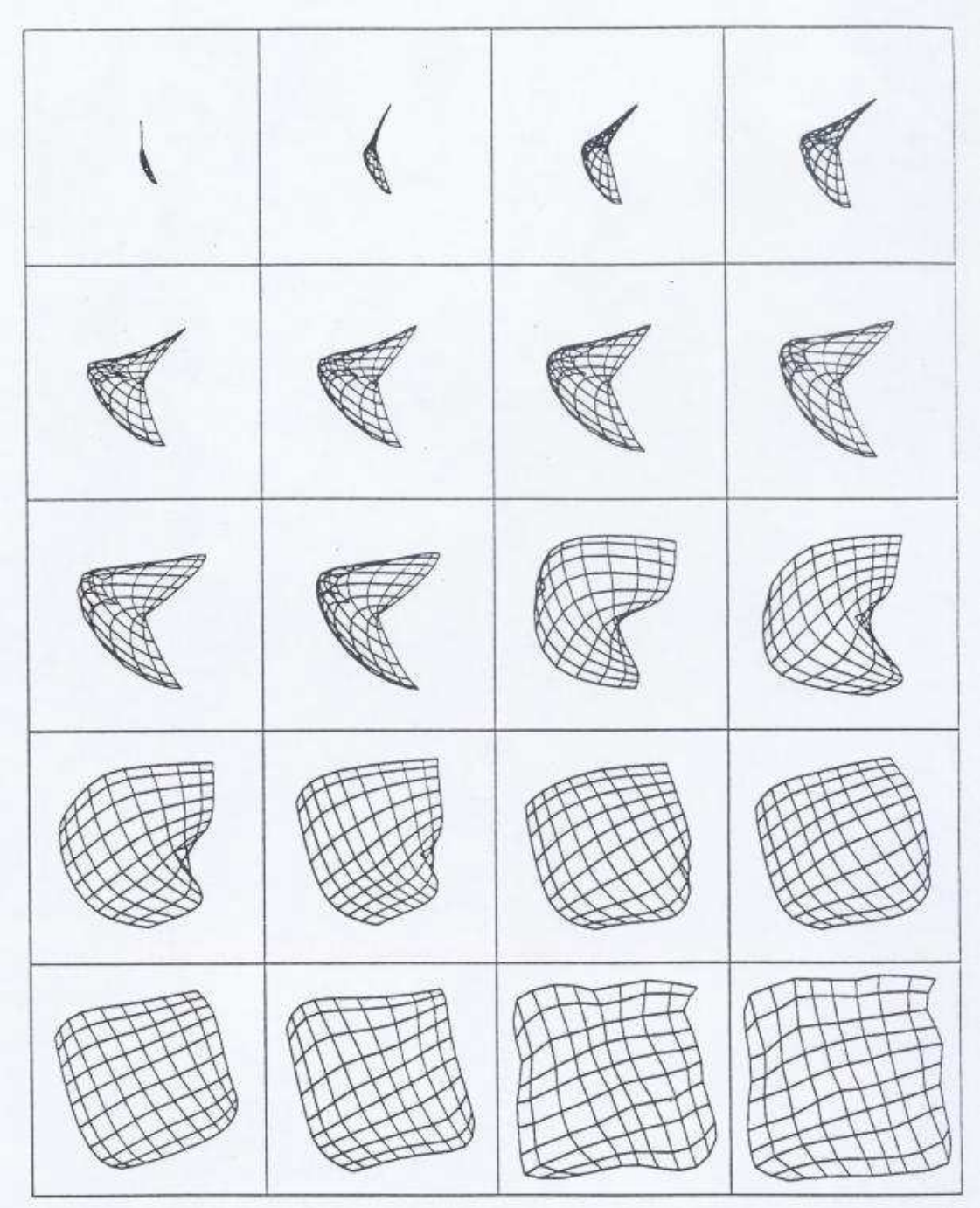


Figura 50 – Evolução de um mapa de Kohonen – (10 20 30 40), (50 60 70 80), (90 100 200 300), (400 500 600 700) e (800 1000 2000 3000) épocas (Software KNet – [Bayer91]).

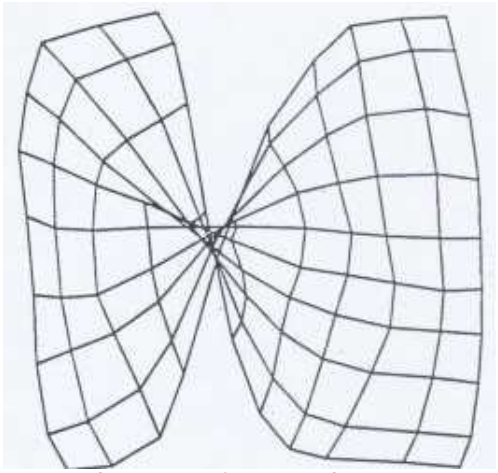


Figura 51 – Mapa auto-organizável com defeito topológico ("papel-de-balinha"), [Bayer91].

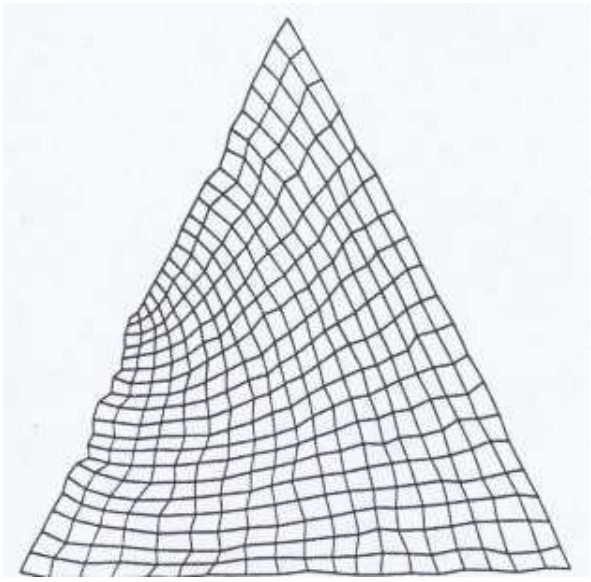


Figura 52 – Espaço de entrada com distribuição triangular aplicado a mapa quadrado. A distribuição da entrada é uniforme sobre o triângulo, [Bayer91].

Considerações práticas para a utilização de SOM's

- Estrutura da grade – quadrado é mais fácil de implementar que o hexágono proposto por Kohonen.
- Forma do mapa – Kohonen recomenda um retângulo em vez do quadrado para haver menos simetrias na orientação o mapa. Limite → mapa redondo!!
  Alguns autores propõem criação e eliminação de neurônios.
- Apresentação de padrões – tipicamente 10^4 –10^6 passos. Apresentação cíclica ou com permutações aleatórias dos vetores de entrada. Prática: cíclica.
- Reforço de casos raros importantes – replicar casos em vez de alterar $\eta(t)$.
- Escalonamento dos vetores de entrada – não é necessário, mas é recomendado por alguns autores.

2.23. Características típicas de Redes Neurais Artificiais

Concluindo esta seção destacam-se a seguir as principais características das redes neurais artificiais (RNAs).

Características Positivas

- **Capacidade de Aprendizado:**
  RNA não são programadas, mas treinadas com padrões de treinamento. Podem ser adaptadas através das entradas.
- **Paralelismo:**
  RNA são massivamente paralelas e são portanto muito bem adequadas para uma simulação/implementação em computação paralela.
- **Representação distribuída do conhecimento:**
  O conhecimento é armazenado de forma distribuída em seus pesos. O que aumenta muito a tolerância do sistema a falhas de neurônios individuais; permite o processamento paralelo.
- **Tolerância à falhas:**
  O sistema pode ser mais tolerante a falhas de neurônios individuais que algoritmos convencionais. A rede deve, no entanto, ser treinada para apresentar esta característica. Nem toda rede é automaticamente tolerante a falhas.
- **Armazenamento associativo da informação:**
  Para um certo padrão de entrada a RNA fornece o padrão que lhe é mais próximo. O acesso não é feito por endereçamento.
- **Robustez contra perturbações ou dados ruidosos:**
  Quando treinadas para tanto as redes neurais são mais robustas a padrões incompletos (ruidosos)

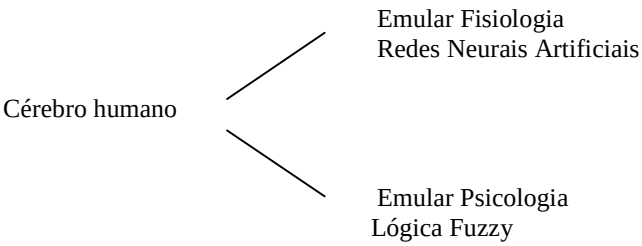
Características Negativas

- **Aquisição de conhecimento só é possível através de aprendizado:**
  Principalmente devido à representação distribuída é muito difícil (exceção: rede de Hopfield para problemas de otimização) introduzir conhecimento prévio em uma RNA. Isto é muito comum em sistemas de IA simbólicos.
- **Não é possível a introspecção:**
  Não é possível analisar o conhecimento ou percorrer o procedimento para a solução, como é possível com os componentes de explicação de sistemas especialistas.
- **Difícil dedução lógica (seqüencial):**
  É virtualmente impossível obter-se cadeias de inferência lógica com redes neurais.
- **Aprendizado é lento:**
  Principalmente redes completamente conectadas e quase todas as variantes dos algoritmos *backpropagation* são muito lentas.

3. Lógica “Fuzzy”

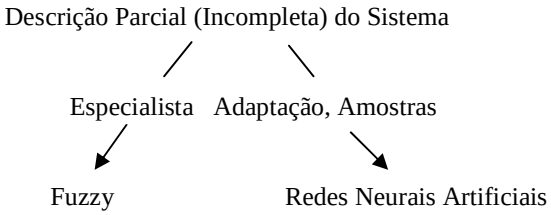
Obs: Em português: Lógica Difusa, Nebulosa.

A teoria dos conjuntos *fuzzy* foi proposta por Lotfi Zadeh em 1965. Por muito tempo permaneceu incompreendida. Em meados dos anos 80 Mamdani a utilizou para projetar controladores fuzzy. A partir daí houve um grande progresso da área, em especial com muitas aplicações reportadas do Japão.



A lógica fuzzy permite criar sistemas especialistas utilizando variáveis linguísticas para criar uma base de regras. Expressões linguísticas são típicas da natureza humana de tomar decisões. Por exemplo: "Se estiver *quente* vou ligar o ar condicinado no *máximo”*. *Quente* e *máximo* não significam um valor particular de temperatura e potência, mas podem assumir uma faixa considerável de valores. Pessoas diferentes também podem ter diferentes acepções para o mesmo conceito linguístico.

De acordo com a disponibilidade de um especialista ou de amostras de um sistema o paradigma Fuzzy ou RNA é mais indicado.



Sistemas fuzzy são sistemas baseados em conhecimento (sistemas especialistas). Um *expert* humano cria a base de conhecimento na forma de um banco de regras. Um usuário humano consulta o sistema especialista apresentado fatos à base de fatos. Conforme ilustrado na figura a seguir.

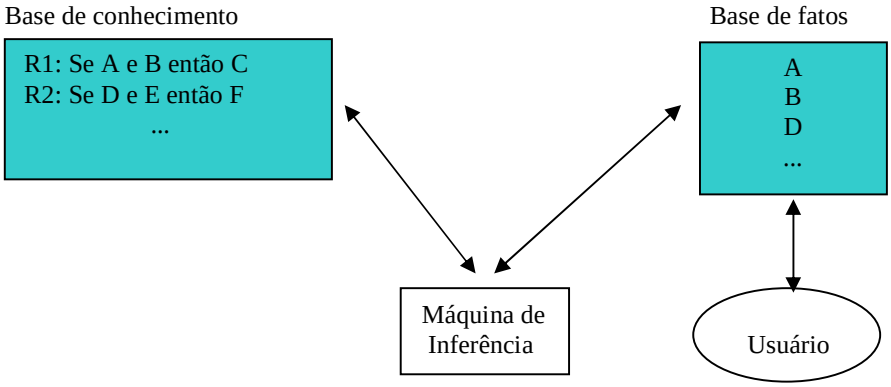


Figura 53 – Sistema especialista.

A máquina de inferência deduz informações novas ao comparar fatos com as premissas das regras.

A	fato:	x é A,
A→B	regra:	se x é A então y é B
	_____	
B	conseqüência:	y é B

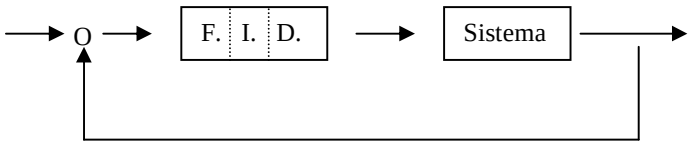
fato:	Os tomates estão vermelhos
regra:	**Se** os tomates estão vermelhos **então** estão maduros
_____	
conseqüência:	Os tomates estão maduros

Exemplos de sistemas em que se utiliza lógica fuzzy:

- Estação de Tratamento de Água Nbg
- Guiagem Automática BMW
- Ar condicionado
- Máquina Fotográfica-Autofoco
- Máquina de lavar
- Aspirador de pó
- Japão – grande quantidade de patentes
- "Controle Inteligente"

Em aplicações na engenharia trabalha-se com números (temperatura, pressão, força etc), i.é., as variáveis são contínuas. Por outro lado a inferência fuzzy utiliza variáveis linguísticas. Para que um sistema fuzzy possa ser utilizado em engenharia faz-se necessário converter números (valores exatos) em variáveis lingísticas e vice-versa.

Um controlador realimentado baseado em lógica fuzzy (*Controlador Inteligente*) teria a seguinte estrutura, onde F. I. D. significam Fuzzyficação, Inferência e Defuzzyficação.



Um sistema *fuzzy* permite implementar controladores não-lineares, em que as regras de funcionamento são obtidas de especialistas.

3.1. Lógica Fuzzy - Perspectiva histórica

Alguns pesquisadores contribuiram significativamente para a evolução da lógica booleana (0´s e 1´s, binária) para a lógica multivalorada.

~ 1930, Lukasiewicz : {0,1/2,1}, [0,1]

1937, Black : Função de Pertinência

1965, Lotfi Zadeh : "Fuzzy Sets"
Teoria dos Conjuntos Multivalentes

~ 1988, Produtos Comerciais : "terceira onda" de interesse

3.2. Função de Pertinência

Na lógica clássica utiliza-se a função indicadora bivalente $I_A(x) = \begin{cases} 1 \text{ se } x \in A \\ 0 \text{ se } x \notin A \end{cases}$

Uma função indicadora multivalente leva ao conceito de "função de pertinência".

$\mu_A : x \rightarrow [0,1]$

Permite operar com conjuntos fuzzy, com os seguintes operadores pontuais:

$I_{A \cap B}(x) = \min (I_A(x) , I_B(x))$

$I_{A \cup B}(x) = \max (I_A(x) , I_B(x))$

$I_{Ac}(x) = 1 - I_A(x)$

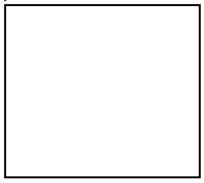
$A \subset B$ sse $I_A(x) \leq I_B(x) \quad \forall \; x \in X$

m A(x) $\rightarrow$ mede o grau de pertinência de x ao conjunto A
$I_A(x) \rightarrow$ assertivas em cálculo bivalente
$m_A(x) \rightarrow$ assertivas em lógica multivalente (contínua)

Conjuntos Fuzzy como pontos em cubos

Os vértices de um quadrado podem ser considerados como valores booleanos. Todos os demais pontos no interior e nas arestas do quadrado são valores válidos em lógica fuzzy.

$\{x_2\} = (0\ 1)$ $\qquad$ $X = (1,1)$



$\phi = (0\ 0)$ $\qquad$ $\{x_1\} = (1\ 0)$

$X = \{ x_1 \ x_2 \}$ Subconjuntos não-fuzzy : { (0 0) ; (0 1) ; (1 0) ; (1,1)}

3.3. Lógica Fuzzy como generalização da lógica binária

- Lógica binária : A ou não-$\overline{A}$ (A)
  Lei da não contradição : não (A e não-A)

- Lógica Fuzzy : graus de indeterminação
  Graus de ocorrência de eventos ou relações

O Princípio da Incerteza de Heisenberg (Mecânica Quântica) estabeleceu limites à precisão com a qual podemos observar fenômenos físicos. A lógica fuzzy generaliza a lógica binária formalizando a imprecisão característica do raciocínio humano.

3.4. Máquina de Inferência

A máquina de inferência fuzzy segue os seguintes passos para obter o resultado da inferência para um conjunto de fatos:

i) fatos com premissas (antecedentes)
ii) grau de compatibilidade de cada regra
iii) crença em cada regra
iv) agregação

Para a agregação quatro métodos se tornaram populares:

a) Método clássico de Mamdani
b) Método clássico de Larsen
c) Método clássico de Tsukamoto
d) Método clássico de Takagi-Sugeno

3.5. Um primeiro exemplo de controle fuzzy

Considere um sistema de ar condicionado para um ambiente de escritório. Deseja-se levar em conta a temperatura e a humidade relativa do ar para acionar o aparelho com economia de energia e conforto. São criadas funções de pertinência para cada variável envolvida.

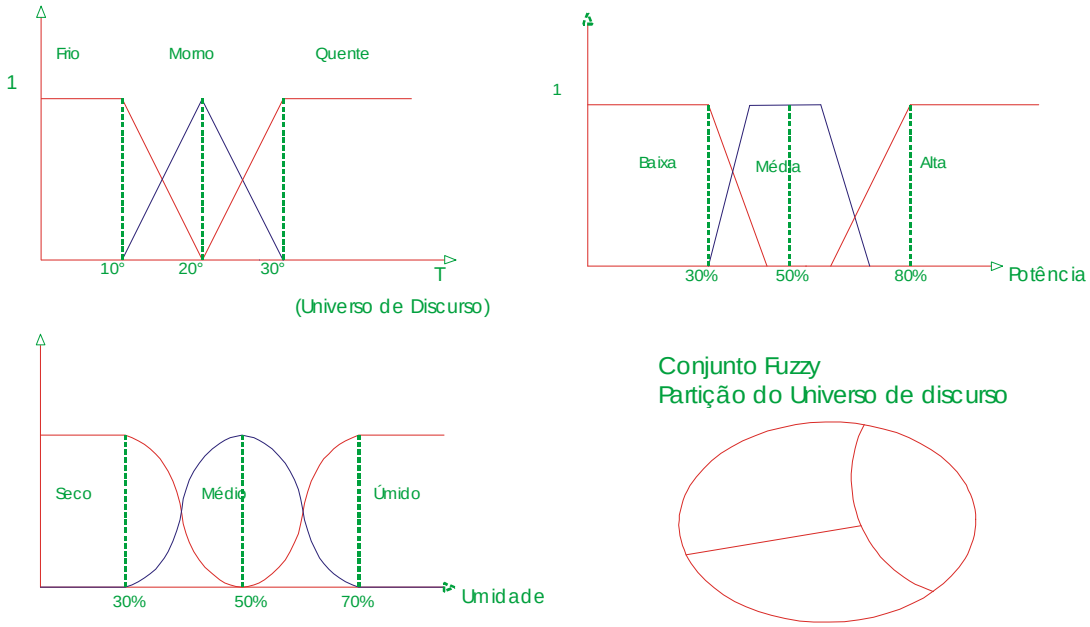


Figura 54 – Funções de pertinência utilizadas para um controle de temperatura.

O seguinte conjunto de regras poderia ser utilizado para o controle fuzzy.

Banco de regras

- Se T é Frio e U é Seco então P é Baixa
- Se T é Quente e U é Úmido então P é Alta
- Se T é Morno e U é Médio então P é Alta
- Se T é Quente e U é Seco então P é Média
- .....

A Inferência é o processo de se obter a saída fuzzy a partir dos valores linguísticos. É preciso transformar o valor T=28º C em uma variável fuzzy (fuzzyficação). O mesmo se dá para a umidade. De acordo com a combinação das regras de são ativadas obtém se o valor de saída.

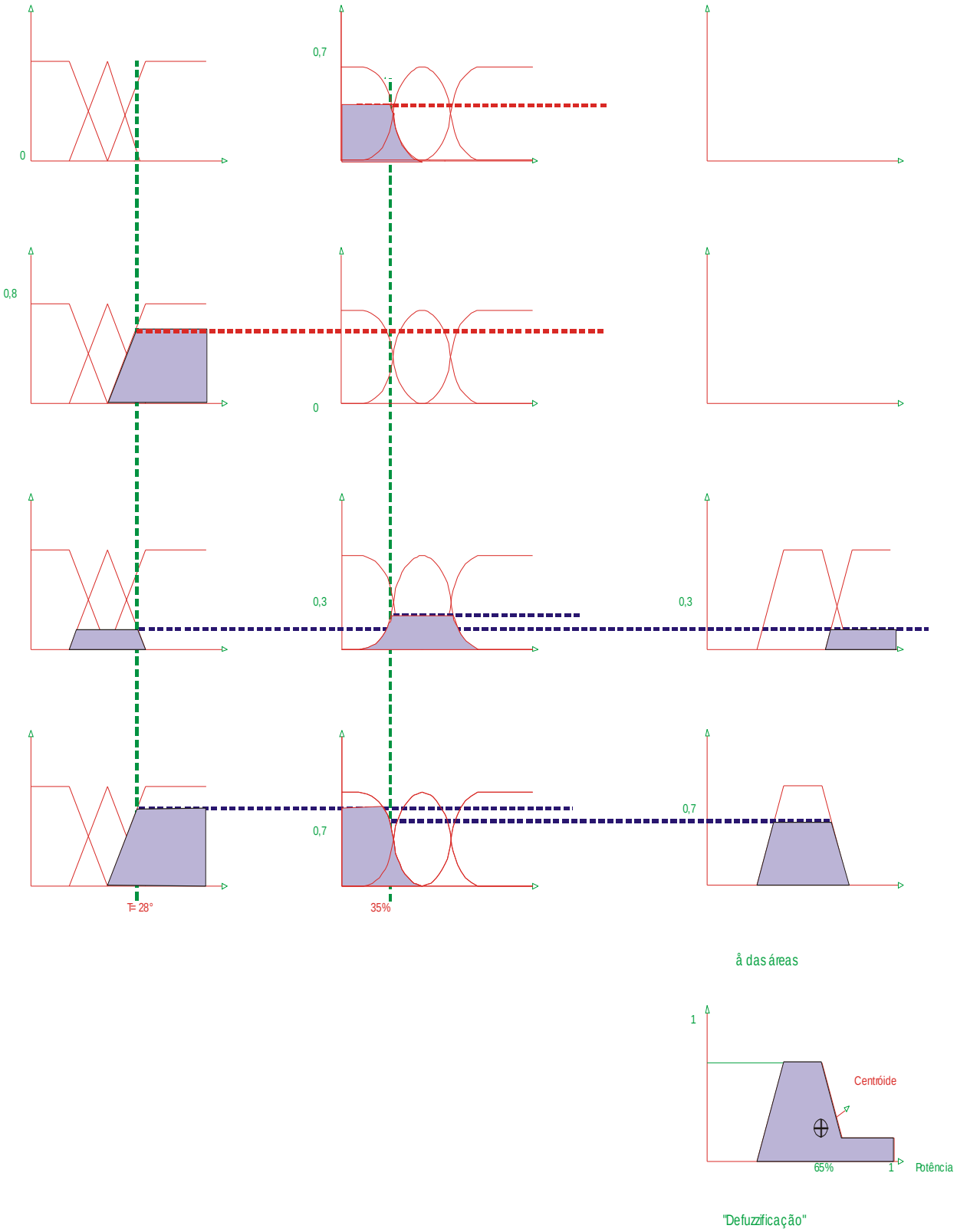


Figura 55 – Inferência fuzzy. A temperatura 28º C e umidade relativa 35% levam pelas avaliação das regras a uma potência de 65% do ar condicionado.

3.6. Paradoxos bivalentes e média fuzzy

A lógica fuzzy trata de forma natural os paradoxos, que em lógica clássica não podem ser resolvidos. Alguns exemplos de paradoxos:

- O mentiroso de Creta está mentindo quando afirma que:
  "Todas as pessoas de Creta mentem? "
  Se ele mente, está dizendo a verdade?

- O barbeiro de Russel
  " Faço a barba de todos os homens que não se barbeiam ".

- Político
  " Não acreditem no que digo ".

Tabela verdade bivalente para o paradoxo

$t(\hat{S}) = 1 - t(S)$
$t(S) = 1 - t(S)$

Se S = true; se $t(S) = 1 \rightarrow 1 = 0$
se $t(S) = 0 \rightarrow 0 = 1$

Interpretação Fuzzy / Multivalente

$2\,t(s) = 1 \rightarrow t(s) = ½$

O "paradoxo" se reduz a "meia-verdade" em lógica fuzzy. Geometricamente ele ocupa o ponto central do hipercubo unidimensional

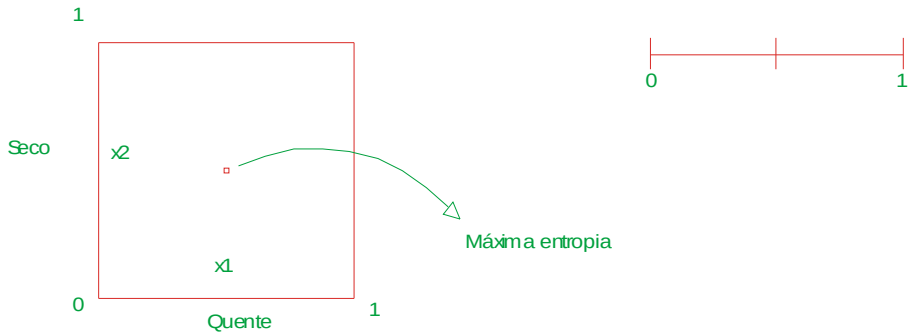
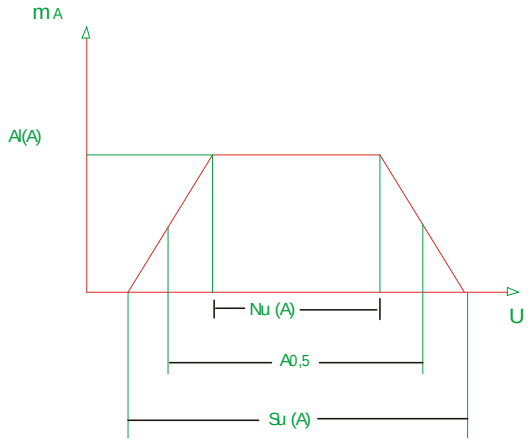


Figura 56 – Representação do paradoxo como o ponto de máxima entropia no quadrado.

3.7. Caracterização de Conjuntos Nebulosos

Os conjuntos fuzzy podem ser caracterizados por alguns parâmetros.



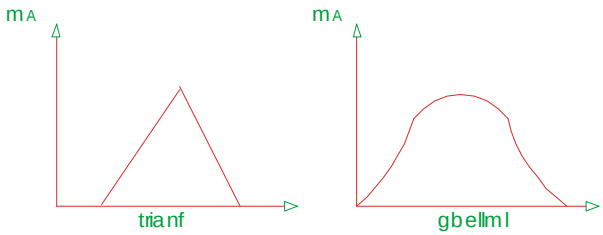


Figura 57 – Caracterização de conjuntos fuzzy.

- $A_x \rightarrow$ corte x
- Núcleo
- Suporte
- Altura

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

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

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

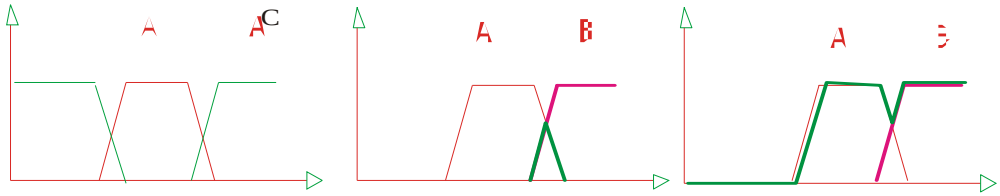


Figura 58 - Complemento, Interseção e União de conjuntos fuzzy.

3.8. Propriedades

Os operadores min e max, complemento, união e interseção satisfazem as seguintes propriedades, como na teoria clássica:

▪ Involução	$(A^C)^C = A$		
▪ Comutatividade	$A \cup B = B \cup A$		$A \cap B = B \cap A$
▪ Associatividade	$A \cup (B \cup C) = (A \cup B) \cup C$		$A \cap (B \cap C) = (A \cap B) \cap C$
▪ Distributividade	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$		$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
▪ Idempotência	$A \cup A = A$		$A \cap A = A$
▪ Absorção	$A \cup (A \cap B) = A$		$A \cap (A \cup B) = A$
▪ Identidade	$A \cup \Phi = A$		$A \cap \Omega = A$
▪ Absorção por Ω e Φ	$A \cup \Omega = \Omega$		$A \cap \Phi = \Phi$
▪ Lei de De Morgan	$(A \cap B)^C = A^C \cup B^C$		$(A \cup B)^C = A^C \cap B^C$

Porém:

▪ $A \cap A^C \neq \Phi$	Não satisfaz lei da não-contradição
▪ $A \cup A^C \neq \Omega$	Não satisfaz lei do terceiro excluído
▪ $A \cup (A^C \cap B) \neq A \cup B$	Não satisfaz absorção do complemento
▪ $A \cap (A^C \cup B) \neq A \cap B$	Não satisfaz absorção do complemento

3.9. Operadores principais da interseção, união e complemento

As seguintes Normas Triangulares são mais utilizadas para implementar a interseção, união e complemento de conjuntos fuzzy. Dependendo da aplicação uma pode ser mais interessante que as outras.

t-norma	t-conorma	Negação	Nome
min (a,b)	max (a,b)	1 – a	Zadeh
a . b	a + b – ab	1 – a	probabilista
max (a + b –1, 0)	min (a + b, 1)	1 – a	Lukasiewicz
a, se b = 1	a, se b = 0	1 – a	Weber
b, se a = 1	b, se a = 0	1 – a	Weber
0	1 senão	1 – a	Weber

Na figura a seguir ilustra-se a utilização da Interseção e da União por estes quatro métodos.

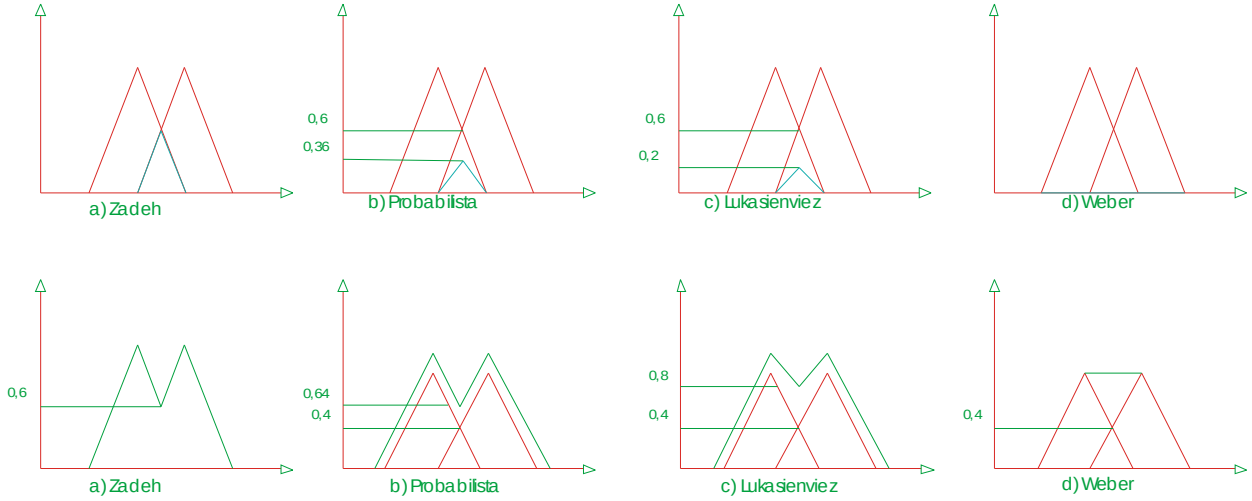


Figura 59 – Ilustração das principais t-normas e t-conormas

3.10. Cálculo Sentencial em Lógica Fuzzy

Na lógica clássica, os valores verdade das proposições (cálculo sentencial) são obtidos pelas seguinte tabela verdade (“modus ponens” – modo afirmativo).

A	B	$\neg$ A	A $\wedge$ B	A $\vee$ B	A $\rightarrow$ B
0	0	1	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	1	1	1

Quando as informações são imprecisas, a máquina de inferência implementa o assim chamado raciocínio aproximado. A lógica Fuzzy implementa o raciocínio aproximado no contexto dos conjuntos fuzzy ("modus ponens generalizado").

fato:	A`	Os tomates estão muito vermelhos
regra:	A→B	Se os tomates estão vermelhos então estão maduros
conseqüência	B`	Os tomates estão muito maduros

$\neg A = n(A)$ n – negação
$A \wedge B = T(A,B)$ T – t -norma
$A \vee B = S(A,B)$ S – t-conorma
$A \rightarrow B = I(A,B)$ I – implicação

Operadores de implicação

$I : [0,1]^2 \rightarrow [0,1]$, $\mu_A : X \rightarrow [0,1]$, $\mu_B : Y \rightarrow [0,1]$

$\mu_{A\rightarrow B}(x,y) = I(\mu_A(x), \mu_B(y))$

"Se <premissa> então <conclusão>"

Implicação	Nome
max (1-a,b)	Kleene-Dimes
min(1-a+b,1)	Lukasiewicz
min(a.b)	Mamdani
a.b	Larsen
…	

Raciocínio fuzzy baseado em composição Max-Min

Definição: Sejam A, A' e B conjuntos fuzzy em X, X e Y respectivamente. Assuma que a inplicação fuzzy $A \rightarrow B$ é expressa pela relação fuzzy R sobre XxY, então o conjunto fuzzy B' é induzido "x é A'" e a regra fuzzy "se x é A então y é B" é definida por:

$\mu_{B'}(y) = \max_x \min [\mu_{A'}(x), \mu_R(x,y)]$
$= V_x [\mu_{A'}(x) \wedge \mu_R(x,y)]$
ou seja, $B' = A' \circ R = A' \circ (A \rightarrow B)$.

- Uma regra com um antecedente

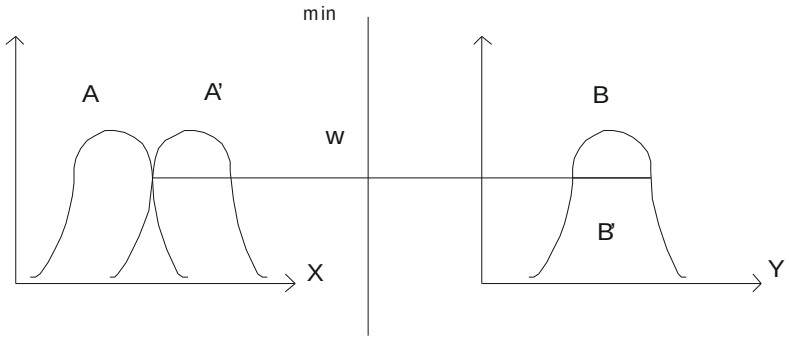


Figura 60 - Raciocinio fuzzy para uma regra e um antecedente.

$\mu_{B'}(y) = [V_x(\mu_{A'}(x) \wedge \mu_A(x))] \wedge \mu_B(y)$
$= \omega \wedge \mu_B(y)$

$\omega \rightarrow$ grau de petinência ao máximo de $\mu_{A'}(x) \wedge \mu_A(x)$.

• Uma Regra fuzzy com dois antecedentes

"Se x é A e y é B então z é C." AxB → C

Fato: x é A` e y é B`
Regra: se x é A e y é B então z é C
Conclusão: z é C`

$\mu_R(x,y,z) = \mu_{(AxB)xC}(x,y,z)$
$= \mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)$
C` = (A` x B`) o (AxB → C)

$\mu_{C'}(z) = V_{x,z}[\mu_{A'}(x) \wedge \mu_{B'}(y)] \wedge [\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)]$
$= \{V_x[\mu_{A'}(x) \wedge \mu_A(x)]\} \wedge \{V_y[\mu_{B'}(y) \wedge \mu_B(y)]\} \wedge \mu_C(z)$
ω_1 ω_2

$\mu_{C'}(z) = \omega_1 \wedge \omega_2 \wedge \mu_C(z)$

$\omega_1, \omega_2 \rightarrow$ graus de pertinência das respectivas regras.

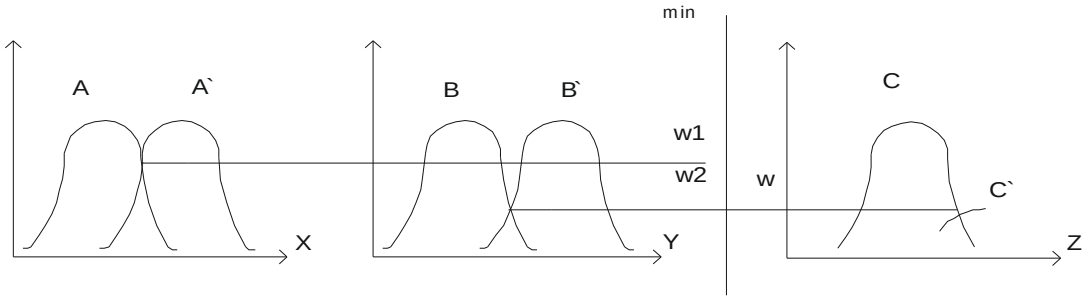


Figura 61 - Raciocinio fuzzy para uma regra e dois antecedentes.

• Múltiplas regras fuzzy com múltiplos antecedentes

Fato: x é A` e y é B`
Regra 1: Se x é A_1 e y é B_1 então z é C_1
Regra 2: Se x é A_2 e y é B_2 então z é C_2
Conclusão: z é C`

$R_1 = A_1 \times B_1 \rightarrow C_1$
$R_2 = A_2 \times B_2 \rightarrow C_2$

Como o operador de composição Max-min o é distributivo sobre o operador υ:

C` = (A`x B`) o (R_1 υ R_2)
= [(A`x B`) o R_1] υ [(A`x B`) o R_2]
= C_1` υ $C_2$`

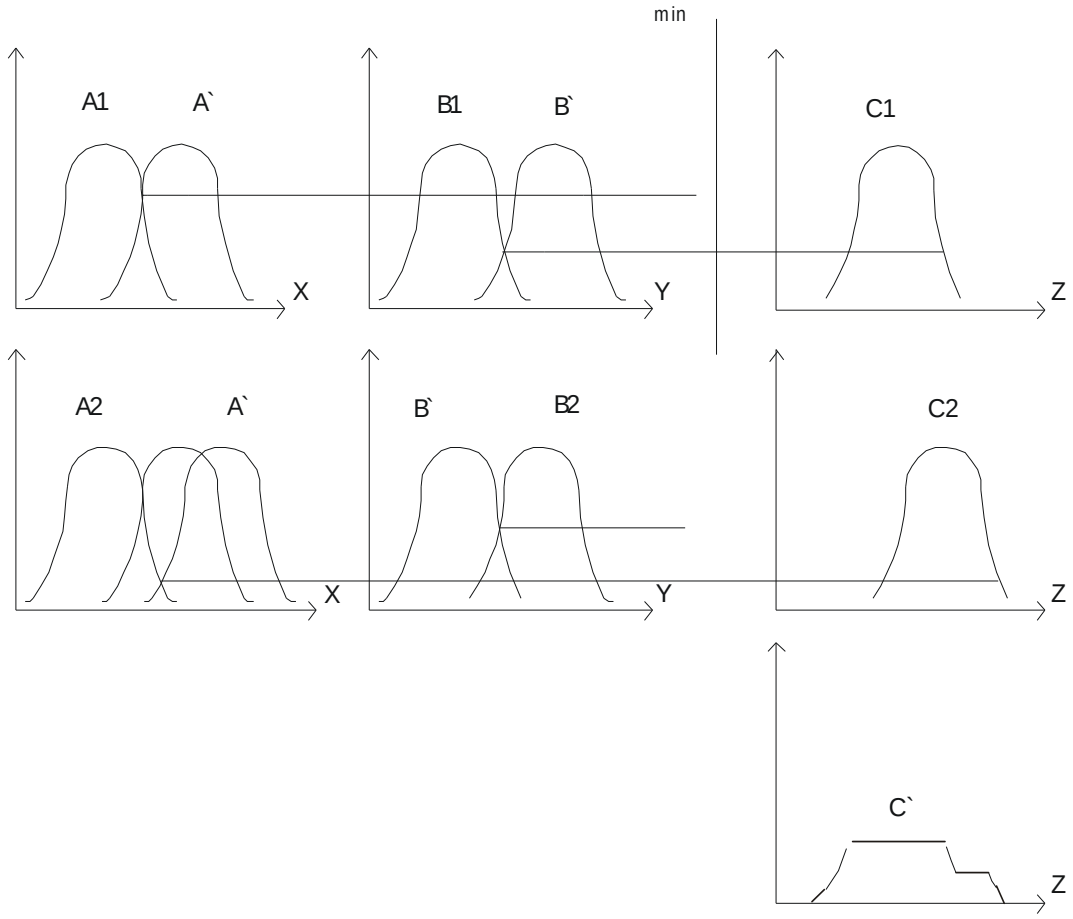


Figura 62 - Raciocinio fuzzy para duas regra com dois antecedentes.

- Inferência fuzzy com A` "crisp" (exato)

Para valores numéricos de entrada os seguintes modelos podem ser utilizados para a inferência fuzzy:

-Modelo de Mamdani
-Modelo de Sugeno
-Modelo de Tsukamoto

O modelo de Mamdani para inferência fuzzy é o mais utilizado. Existem duas variantes:

⇒ min-max
⇒ prod-max

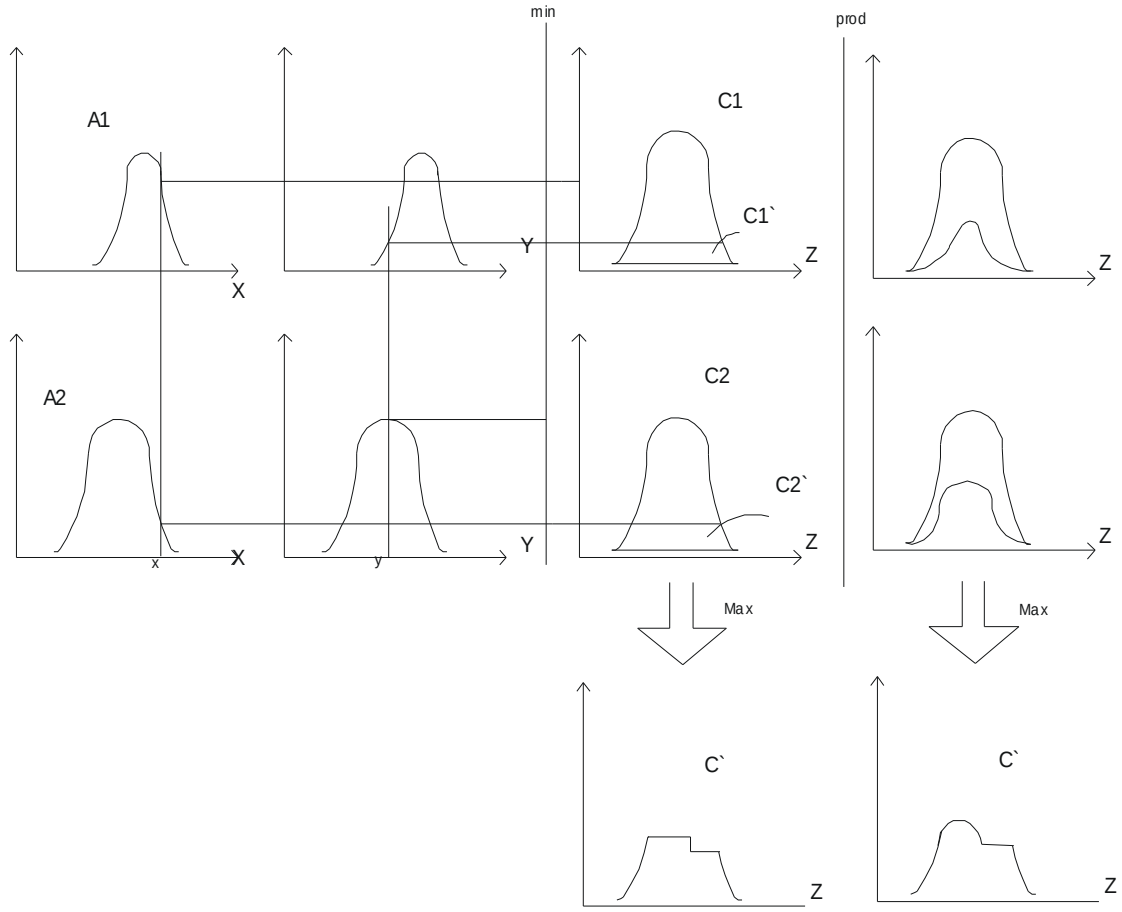


Figura 63 - Raciocinio fuzzy para fatos numéricos. Utilização de min-max e prod-max.

$$z_{CoA} = \frac{\int_z \mu_{C'}(z) z dz}{\int_z \mu_{C'}(z) dz}$$

Esquemas de defuzzyficação

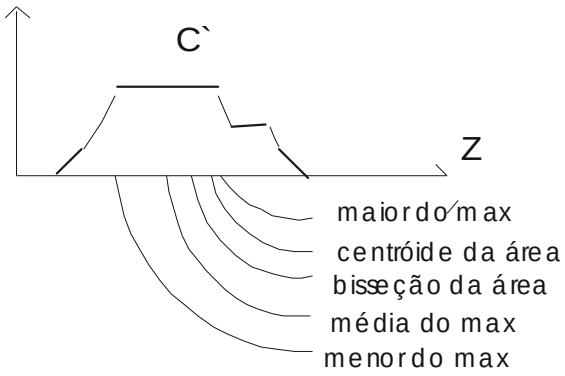


Figura 64 – Esquemas de defuzzyficação.

Modelo de Sugeno para inferência fuzzy

Se x é A y é B, então z = f(x,y)

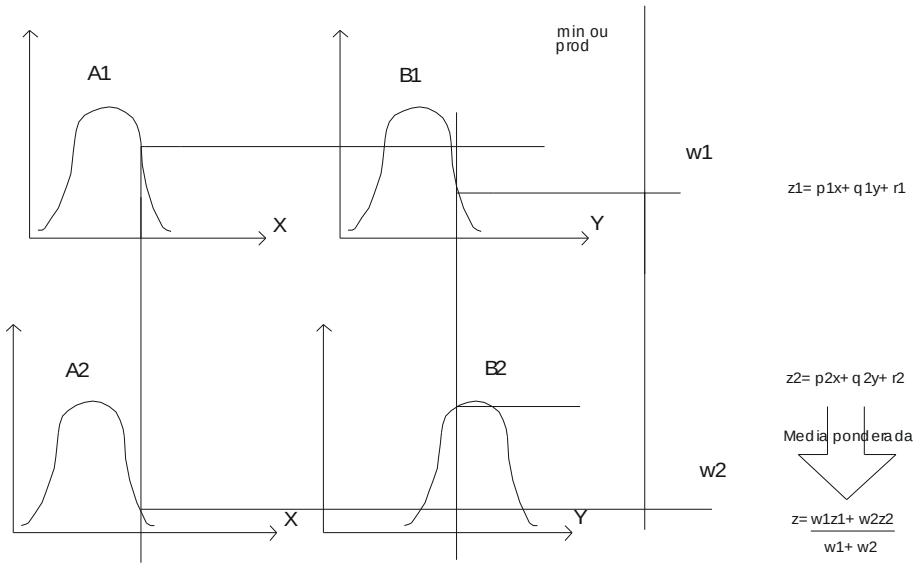


Figura 65 – Modelo de Sugeno para inferência fuzzy.

Se f(x,y) é um polinômio de 1ª ordem, o sistema é dito ser um modelo de inferência de Sugeno de 1ª ordem.

Modelo de Tsukamoto para inferência fuzzy

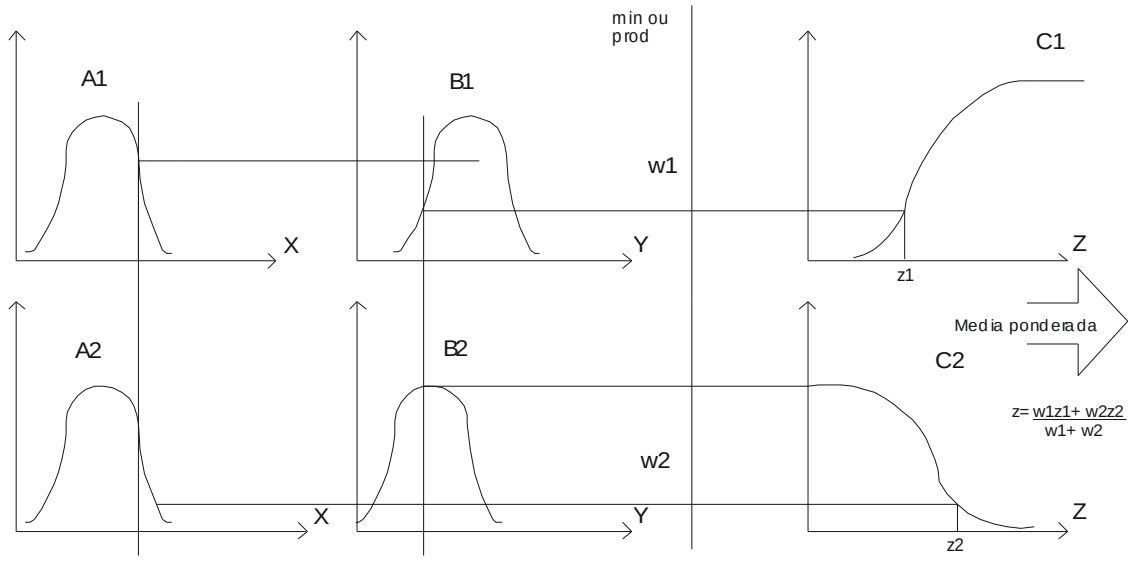
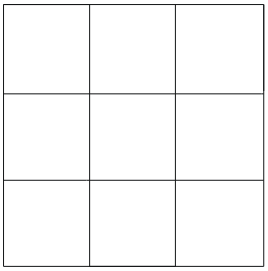


Figura 66 – Modelo de Tsukamoto para inferência fuzzy.

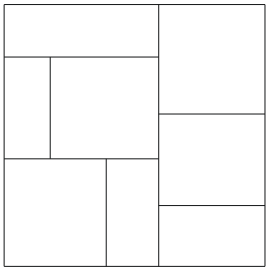
O conseqüente de cada regra é um conjunto fuzzy com função de pertinência monotônica.

3.11. Estilos de partição de espaço para modelos fuzzy

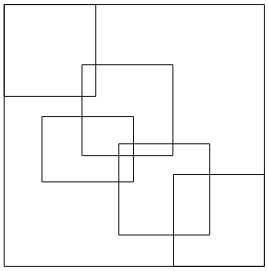
Idéia básica: “dividir para conquistar”.
Cada regra fuzzy abrange uma característica local do espaço de entrada.



Partição em grade

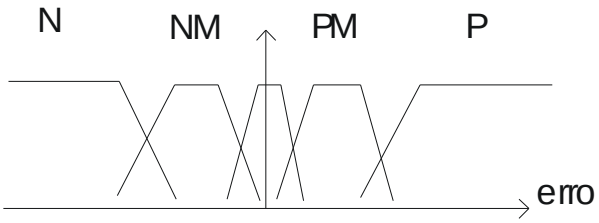


Partição em árvore

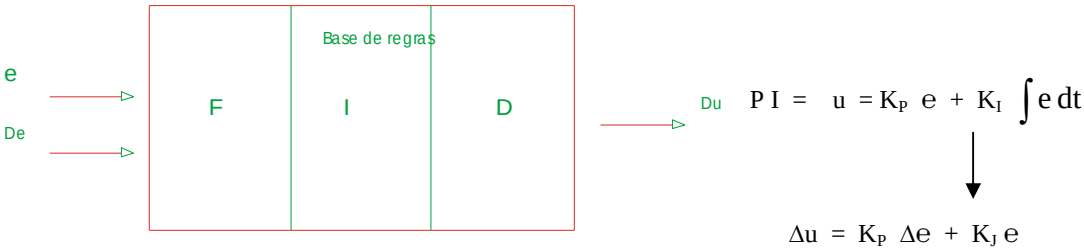


Partição distribruída

- Exemplo: controlador fuzzy proporcional



- Exemplo: Controlador Fuzzy PI



$$PI = u = K_P\, e + K_I \int e\, dt$$

$$\Delta u = K_P\, \Delta e + K_I\, e$$

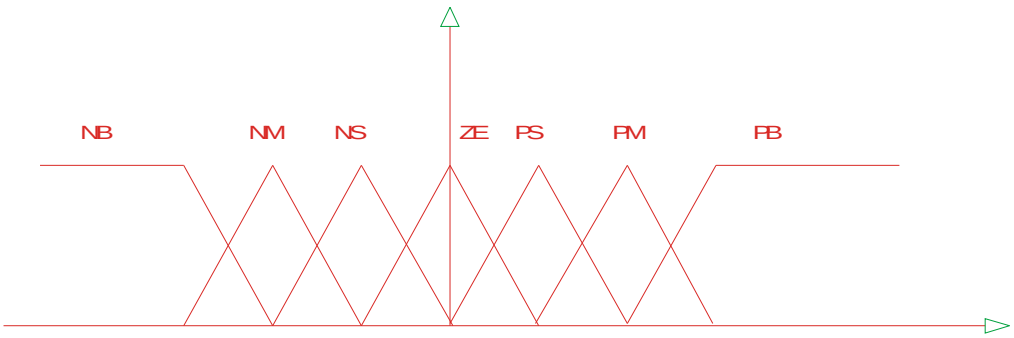


Figura 67 – Partição do Universo de Discurso (erro) para um Controlador Fuzzy PI

3.12. ANFIS: Adaptive Neuro-Fuzzy Inference System

Uma das possibilidades mais interessantes em sistemas inteligentes é combinar as características interessantes de RNA com a Lógica Fuzzy. O Sistema ANFIS permite criar um conjunto de regras que são trainadas com os dados da aplicação. Tem-se assim a componente de explicação dos sistemas especialistas combinada com a característica de aprendizagem dos sistemas neurais.

Para o modelo Fuzzy de Sugeno de 1ª ordem:

R_1: Se x é A e y é B, então $f_1 = p_1x + q_1y + r_1$
R_2: Se x é A_2 e y é B_2, então $f_2 = p_2x + q_2y + r_2$

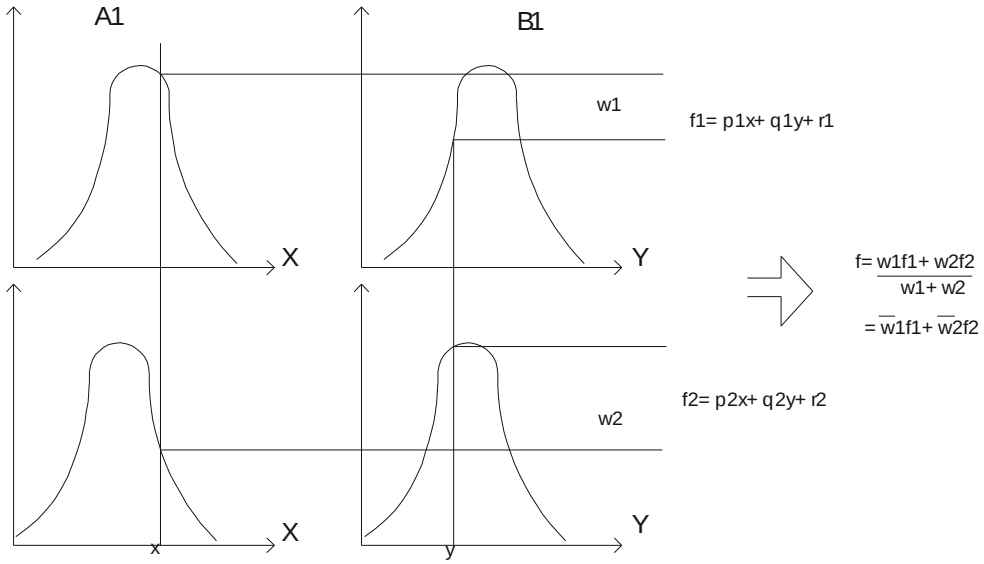


figura 68 – Modelo de Sugeno com duas regras.

Tem a seguinte arquitetura ANFIS equivalente:

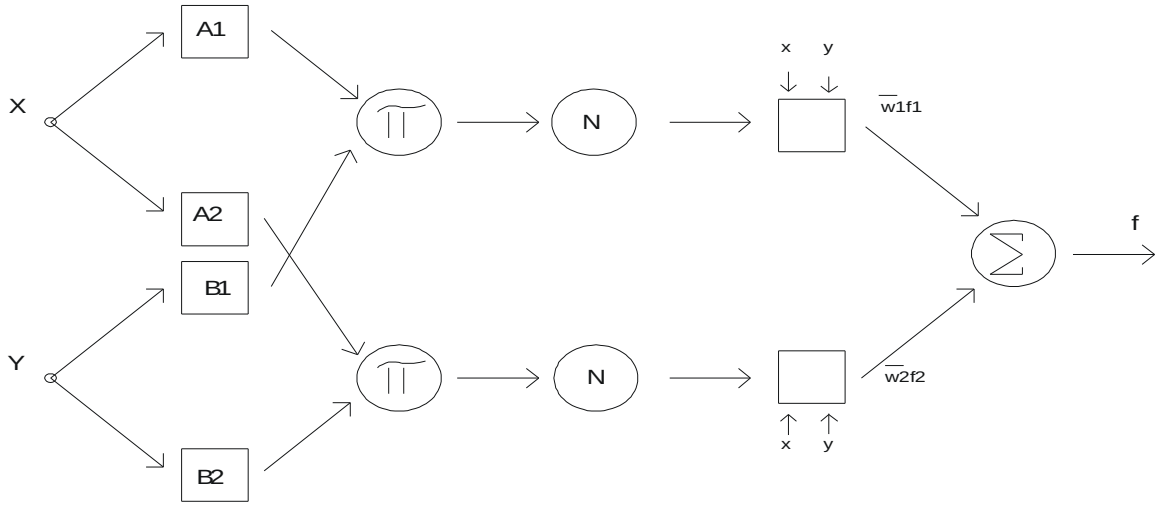


Figura 69 – Arquitetura ANFIS para o modelo de Sugeno.

Rede "feed-forward" Adaptativa → bloco adaptativo → bloco fixo

Camada 1: Nós adaptativos.

$$O_{1,i} = \mu_{Ai}(x), \text{ para } i = 1, 2.$$

$$O_{1,i} = \mu_{B\,i-2}(y), \text{ para } i = 3, 4.$$

Onde Ai pode ser caracterizada pela função seno generalizada

$$\mu_{Ai}(x) = \frac{1}{1 + [(x-ci)^2/(ai)^2]^{bi}}$$

{ai, bi, ci} é o conjunto de parâmetros das premissas.

Camada 2: Nós fixos.

$$O_{2,i} = \omega_i = \mu_{Ai}(x) \times \mu_{B\,i}(y), \; i = 1, 2$$

$$\downarrow$$

T-Norma

- disparo de cada regra.

Camada 3: Nós fixos.

$$O_{3,i} = \omega_i \text{ (médio)} = \frac{\omega_i}{\omega_1 + \omega_2}, \qquad i = 1,2$$

- disparo normalizado de cada regra.

Camada 4: Nós adaptativos.

$$O_{4,i} = \frac{\omega_i}{\omega_1 + \omega_2} \cdot f_i = \frac{\omega_i}{\omega_1 + \omega_2} \cdot (p_i x + q_i y + r_i), \qquad i = 1,2$$

{ p_i, q_i, r_i} é o conjunto de parâmetros das conseqüências.

Camada 5: Nó fixo.

$$O_{5,I} = \Sigma_i \frac{\omega_i}{\omega_1 + \omega_2} \cdot f_i = \frac{\Sigma_i\, \omega_i f_i}{\Sigma_i\, \omega_i}$$

Da mesma podem ser utilizados os modelos de Mamdani ou Tsukamoto para implementar uma ANFIS.

- Algoritmo de Aprendizado Híbrido

  1 – Fixar os parâmetros das premissas.
  ⇒ Saída é combinação linear dos parâmetros das conseqüências.

$$f = \frac{\omega_1}{\omega_1 + \omega_2} . f_1 + \frac{\omega_2}{\omega_1 + \omega_2} . f_2$$
$$= \frac{1}{\omega_1 + \omega_2} . (\omega_1 x p_1 + \omega_1 y q_1 + \omega_1 r_1 + \omega_2 x p_2 + \omega_2 y q_2 + \omega_2 r_2)$$

  ⇒ Parâmetros das conseqüências identificáveis pelo método dos mínimos quadrados.

  2 – "Back-propagtion" dos sinais de erro para adaptar os parâmetros das premissas.
  ⇒ Método do gradiente descendente.

	Passo Forward	Passo Backward
Parâmetros Premissas	Fixos	Gradiente descendente
Parâmetros Conseqüências	Estimativa MMQ	Fixos
Sinais	Nós de saída	Sinais de erro

Na toolbox *Fuzzy Logic* do MatLab a função ANFIS implementa o *Adaptive Neuro-Fuzzy Inference System.*

4. Ferramentas Computacionais

Existem atualmente diversas ferramentas computacionais que podem ser utilizadas para implementar sistemas inteligentes. Dependendo da aplicação uma ou outra pode ser mais interessante.

Para o ambiente Unix o sistema SNNS (Suttgart Neural Network Simulator), desenvolvido pelo IPVR da Universidade de Stuttgart e mantido pela Universidade de Tübingen - Alemanha, implementa uma grande variedade de redes e é bastante difundido entre a comunidade acadêmica. É de domínio público e pode ser obtido em: http://www-ra.informatik.uni-tuebingen.de/SNNS/

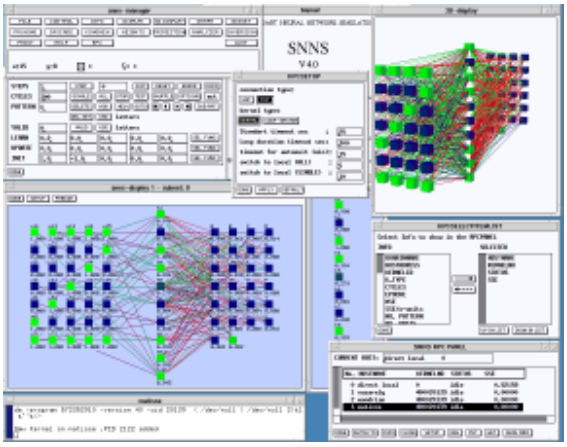
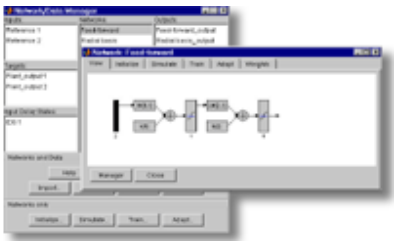


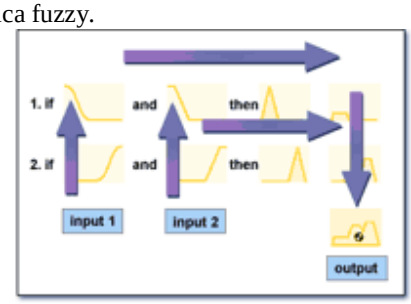
Figura 70 – Suttgart Neural Network Simulator.

O sistema XFuzzy para Unix foi desenvolvido pelo Instituto de Microelectrónica de Sevilla – Espanha e pode ser obtido em http://www.imse.cnm.es/Xfuzzy/download.htm

Atualmente o MatLab® (http://www.mathworks.com) com as suas Toolboxes *Neural Network* e *Fuzzy Logic* tanto para ambiente Windows Microsoft como para estações de trabalho Unix / Linux é a ferramenta predominante para aplicações de sistemas inteligentes. O MatLab/Simulink permite um tratamento integrado do problema, onde a rede neural ou a lógica fuzzy são um dos blocos da simulação do processo completo. E isto é decisivo, pois, em geral o pré- e pós-processamento dos sinais a serem tratados pelos sistemas inteligentes demandam muito mais esforço de engenharia que a definição e treinamento da RNA / Sistema Fuzzy.

O ambiente SciLab (http://www-rocq.inria.fr/scilab/) é um pacote científico de domínio público que tem entre outros pacotes de redes neurais e de lógica fuzzy.





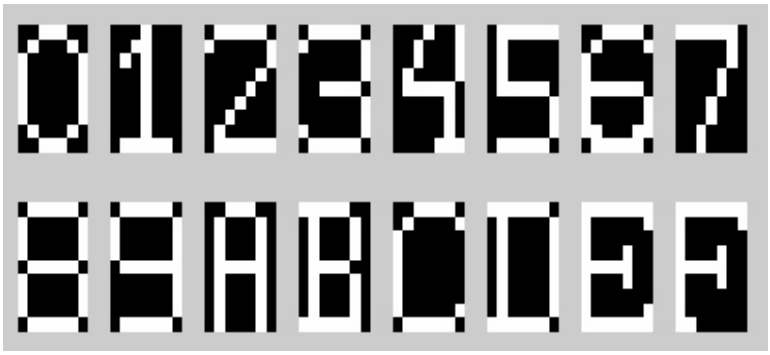
5. Exemplos de Aplicações em Engenharia

A seguir serão apresentadas brevemente três aplicações que ilustram o potencial dos sistemas inteligentes para tratar problemas de engenharia "complexos".

5.1. Reconhecimento Óptico de Caracteres (OCR)

Rede Perceptron Multicamada com 63 entradas e 16 saídas para reconhecimento de caracteres hexadecimais.

Vetores de treinamento:



Padrão de Entrada com 20% de ruído:

Resultado:

Apenas o dígito "0" foi classificado incorretamente.

5.2. Identificação de Falhas em Estruturas Mecânicas

DAMAGE DETECTION USING AN HIBRID FORMULATION BETWEEN CHANGES IN CURVATURE MODE SHAPES AND NEURAL NETWORK.

Miguel Genovese, Adolfo Bauchspiess, José L.V. de Brito,Graciela N. Doz

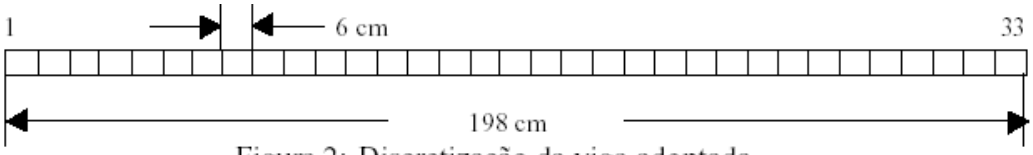
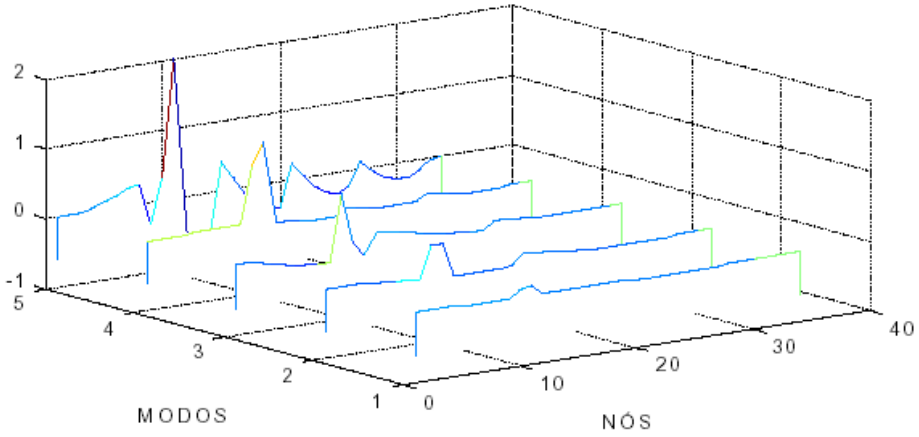


Figura 2: Discretização da viga adoptada



Método da Alteração na Curvatura aplicado a viga com dano de 20 % no elemento

Tabela 1: Freqüências das Vigas com e sem dano

Freqüências (Hz)	Viga Sem dano	Redução 20% inércia no elemento No. 10.
Primeira	67.76	67.48
Segunda	184.22	182.72
Terceira	354.01	352.62
Quarta	570.26	569.59
Quinta	825.83	821.93

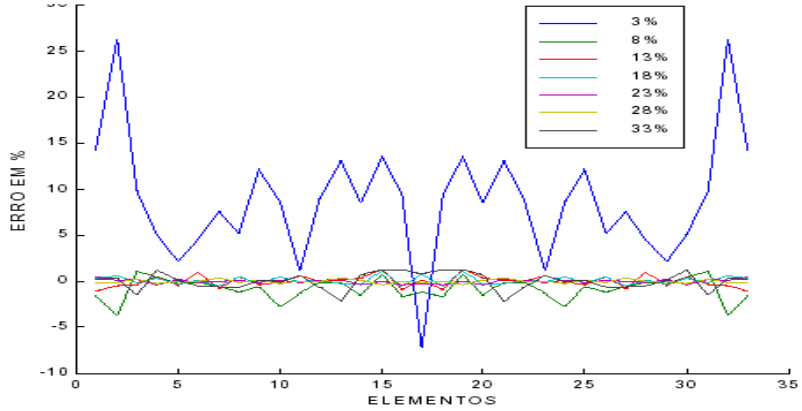


Figura 5: Erro pontual cometido na quantificação pela rede neural quando submetida ao conjunto de treinamento.

5.3. Controle Fuzzy de Processo de Nível de Líquidos

O objetivo é controlar o nível de líquido em três reservatórios acoplados.

$$A\frac{dh_1}{dt} = q_{i1} + signal(h_3 - h_1)k\sqrt{|h_3 - h_1|} - k\sqrt{h_1},$$

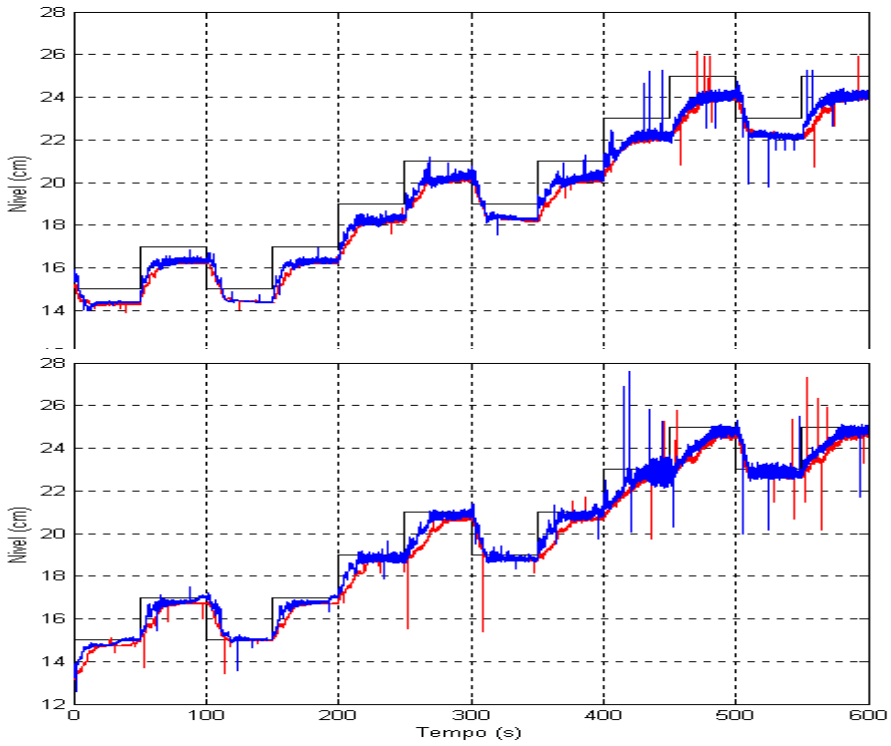
$$A\frac{dh_2}{dt} = q_{i2} + signal(h_3 - h_2)k\sqrt{|h_3 - h_2|} - k\sqrt{h_2},$$

$$A\frac{dh_3}{dt} = -signal(h_3 - h_1)k\sqrt{|h_3 - h_1|} - signal(h_3 - h_2)k\sqrt{|h_3 - h_2|}$$

Não-Linear Acoplado Multivariável

Controle PI

Controle Fuzzy



Conclusões

A teoria de sistemas inteligentes é ainda relativamente recente, porém diversas aplicações comerciais mostram o potencial desta nova área do conhecimento.

Não se tem a expectativa de que RNA ou Lógica Fuzzy venham a substituir as técnicas convencionais. Elas são um novo paradigma para lidar com sistemas complexos.

O aprendizado é um aspecto fundamental na sobrevivência dos seres vivos. A utilização desta em problemas de engenharia permite tratar problemas de engenharia de forma promissora.

A aplicação de lógica fuzzy gera um sistema especialista. Portanto só um especialista no assunto em questão pode definir regras que reflitam a “expertise” deste. Não basta definir um conjunto de regras para que o sistema opere satisfatoriamente. As regras devem refletir o conhecimento matemático (não-linear) subjacente ao problema em questão.

A visão crítica dos resultados obtidos por sistemas inteligentes continua sendo fundamental para toda aplicação. A validação do sistema inteligente é indispensável, antes que seja empregado numa certa aplicação.

Bibliografia

1. Haykin, S.: Redes Neurais: Princípios e Prática. 2.ed. Porto Alegre, Bookman, 2001.
2. Nascimento Jr. C.L.: Yoneyama, T.: Inteligência Artificial em Controle e Automação, Edgard Blücher, 2000
3. Shaw, I. S., Simões, M. G.: Controle e Modelagem Fuzzy, Edgard Blücher, São Paulo, 1999
4. Kasabov, N.K.: *Foundations of Neural Network, Fuzzy Systems and Knowledge Engineering*, MIT-Press, 1996
5. Loesch, C.; Sari, S.: *Redes Neurais Artificiais – Fundamentos e Modelos*, Editora da FURB, 1996
6. Kovács, Z.L.: *Redes Neurais Artificiais – Fundamentos e Aplicações*, Edição Acadêmica, 1996
7. Kovács, Z.L.: *O Cérebro e a sua Mente – Uma Introdução à Neurociência Computacional*, Edição Acadêmica, 1997
8. CD-ROM - V Escola de Redes Neurais, ITA, 1999
9. Kröse, B., van der Smagt, P: An Introduction to Neural Networks – www.robotic.dlr.de/Smagt/books/
10. MathWorks® - Neural Network Toolbox - User Manual
11. MathWorks® - Fuzzy Logic Toolbox - User Manual
12. Internet www – Muitos *sites* com programas em java.