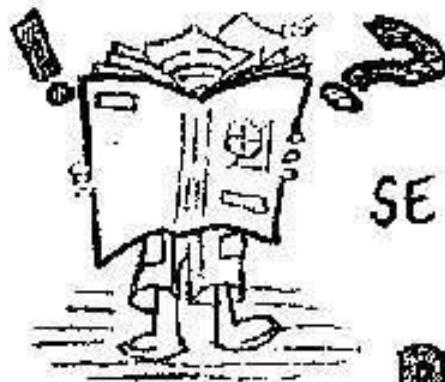


Sistemas Inteligentes e “soft computing”

Paulo Salgado

psal@utad.pt



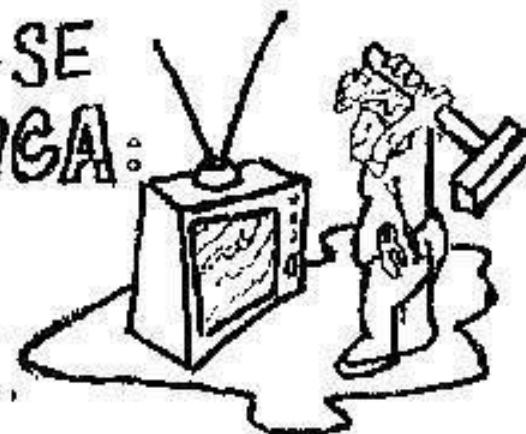


TEORIA É QUANDO
SE SABE TUDO E NADA FUNCIONA

PRÁTICA É QUANDO
TUDO FUNCIONA E NINGUÉM SABE PORQUÊ

NESTA FIRMA UNIU-SE
TEORIA À PRÁTICA:

NADA FUNCIONA
E NINGUÉM SABE PORQUÊ.



Conteúdo

- Sistemas Inteligentes
- Soft computing
- Áreas de Aplicação
- Redes Neuronais
- Fuzzy Logic (Lógica Difusa)
- Sistemas Evolutivos



Sistemas Inteligentes



Inteligência: Os Sistemas devem realizar operações sem significado.

Interpretar as informações.

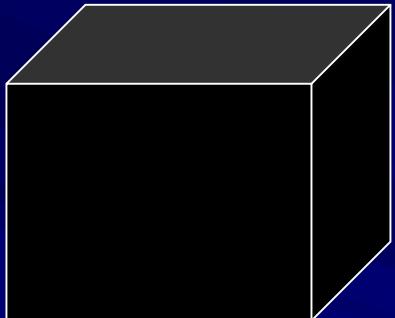
Compreender as relações entre fenómenos e objectos.

Aplicar os conhecimentos adquiridos a novas condições.

Objectivos dos sistemas inteligentes

- **Rotinas das necessidades humanas:** visão , processamento de linguagem, razão, aprendizagem, robótica.
- **Rotinas Artificiais:** jogos, matemática, lógica, programação.
- **Expert systems (ES)**

Aproximações tradicionais



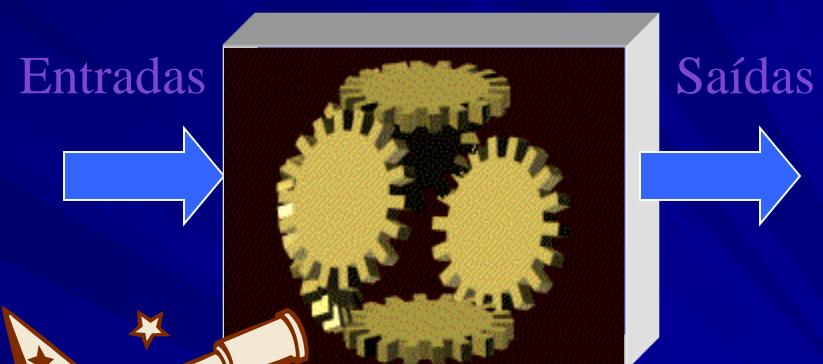
- **Modelos Matemáticos:**
Black boxes (caixa negra),
Físicos (baseados nas leis
da conservação da massa,
energia, ...).
- **Sistemas baseados em
Regras (disjuntas &
bivalente):**
Lista de regras numerosas.

Modelos

Black Box



Open Box



Soft computing (SC)



Objectivo:

Imitar a razão humana (linguística ou outra)

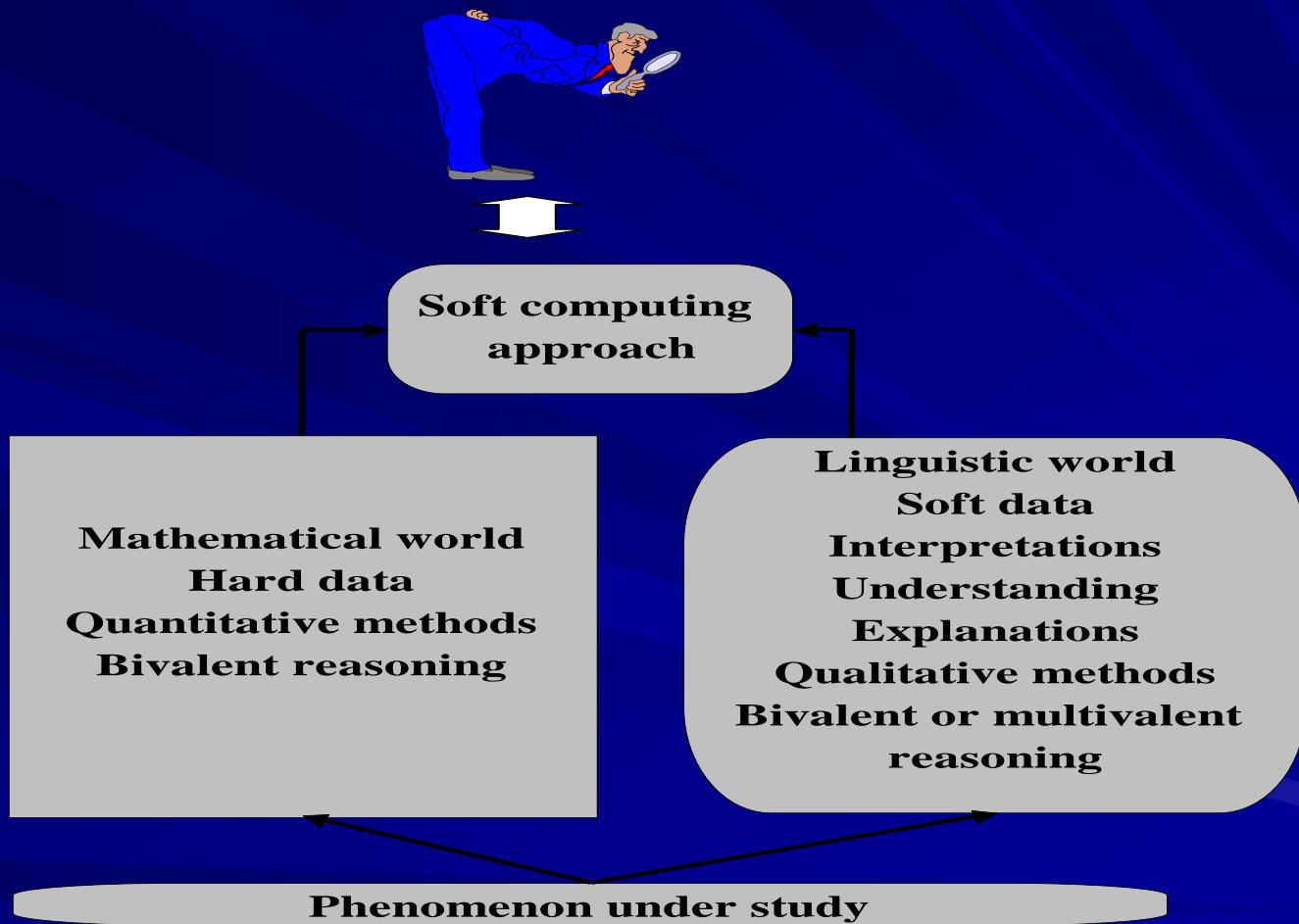
Áreas principais:

- Fuzzy systems
- Neural networks
- Evolutionary computing
- Probabilistic reasoning

Constituintes da SC

- Fuzzy systems => imprecisão
- Neural networks => aprendizagem
- Probabilistic reasoning => incerteza
- Evolutionary computing => optimizaçãopesquisa

SC: uma interface amigável



Vantagens da Soft Computing

- Modelos baseados na razão humana.
- Os modelos podem ser:
 - linguísticos
 - simples (não exaustivos),
 - comprehensíveis (opostos às black boxes),
 - rápidos de executar (computacionalmente),
 - bons do ponto de vista prático.

Investigação actuais do SC (Zadeh)

- **Computação com palavras**
- **Teoria da informação granulada**
(Theory of information granulation (TFIG))
- **Teoria da percepção computacional**

Possíveis tipos de dados & operações

■ Dados numéricos:

5, em torno de 5, 5 para 6, em torno de 5 para 6

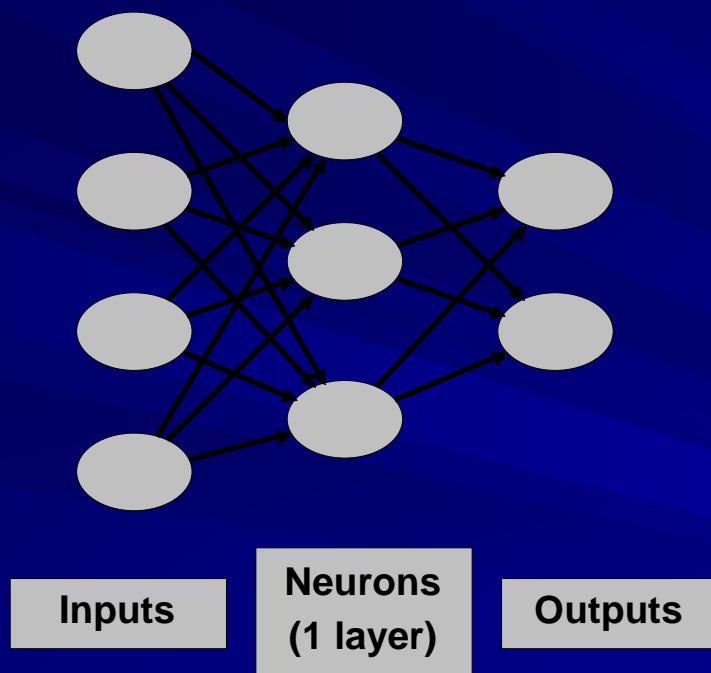
■ Dados Linguísticos:

barato, muito alto, não alto, médio ou mau

■ Funções & relações:

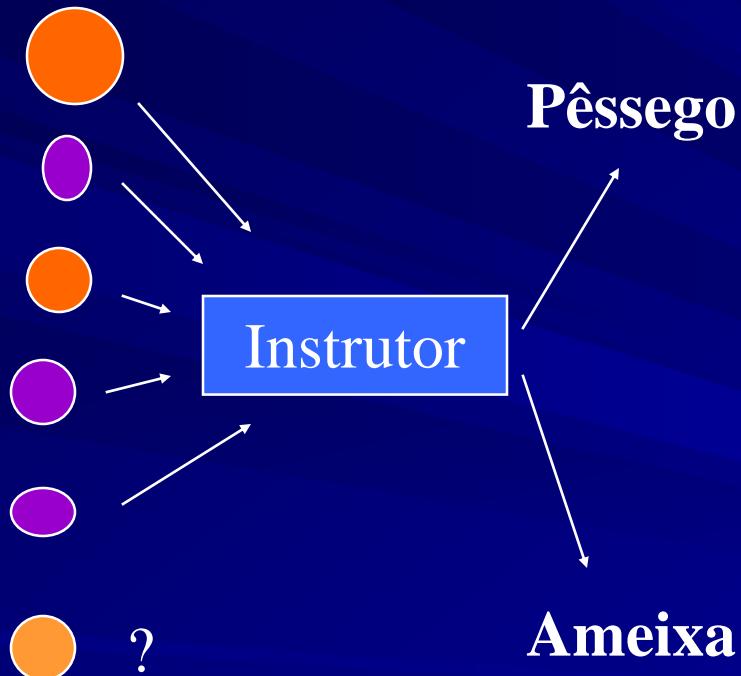
$f(x)$, em torno de $f(x)$, muito similar, muito maior

Neural networks (NN, 1940's)



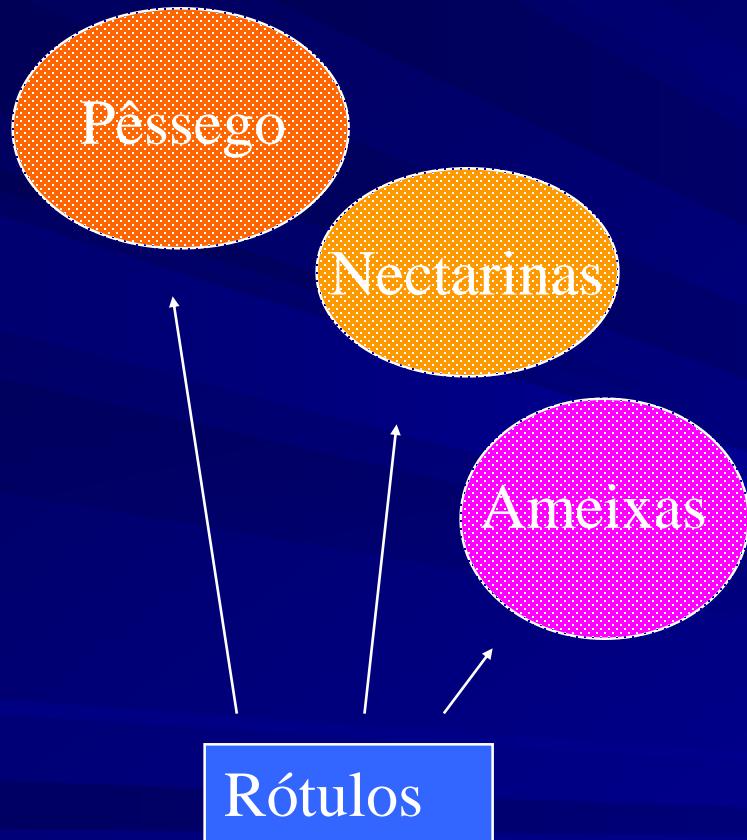
- Neural networks (redes neurais) oferecem um poderoso método para explorar, classificar, e identificar padrões nos dados.
- [Website do Matlab](#)
- Neurónio: $y = \sum w_i x_i$

Aprendizagem supervisionada



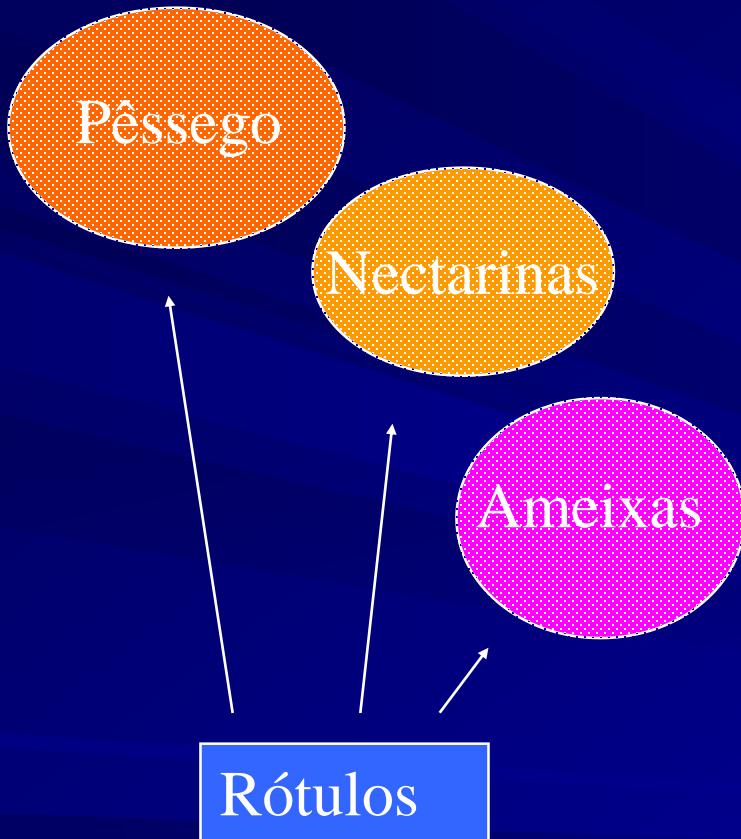
- Reconhecimento de Padrões baseados nos treinos de dados.
- Classificação supervisionada por um instrutor.
- Neural (disjunto or difuso), neuro-fuzzy e fuzzy models.

Aprendizagem não supervisionada



- Reconhecimento de Padrões baseado no treino de dados.
- Classificação baseada na estrutura dos dados (agrupamentos).
- Neural (disjunto ou difuso), neuro-fuzzy e fuzzy models.

Aprendizagem não supervisionada



- Self-organized maps (Kohonen).
- Fuzzy c-means (Bezdek).
- Subclust (Yager, Chiu).

Sistemas Difusos (Zadeh, 1960's)

- Lidam com entidades imprecisas em ambientes automatizados (computer environments)
- Baseiam-se na teoria dos conjunto difusos e na lógica difusa.
- A maior parte das aplicações são em controlo e tomada de decisões (decision making).

Aplicações: controlo



- **Industria pesada
(Matsushita, Siemens, Stora-Enso)**
- **Aplicações domesticas (Canon, Sony, Goldstar, Siemens)**
- **Automobilística (Nissan, Mitsubishi, Daimler-Chrysler, BMW, Volkswagen)**
- **Espacial (NASA) e militar**

Aplicações: robótica



Controlo
+
Planeamento
+
inteligência

Entertainment
robot AIBO

Fukuda's lab

Aplicações: negócios

- supplier evaluation for sample testing,
- customer targeting,
- sequencing,
- scheduling,
- optimizing R&D projects,
- knowledge-based prognosis,
- fuzzy data analysis
- hospital stay prediction,
- TV commercial slot evaluation,
- address matching,
- fuzzy cluster analysis,
- sales prognosis for mail order house,
- multi-criteria optimization etc.
- (source: FuzzyTech)

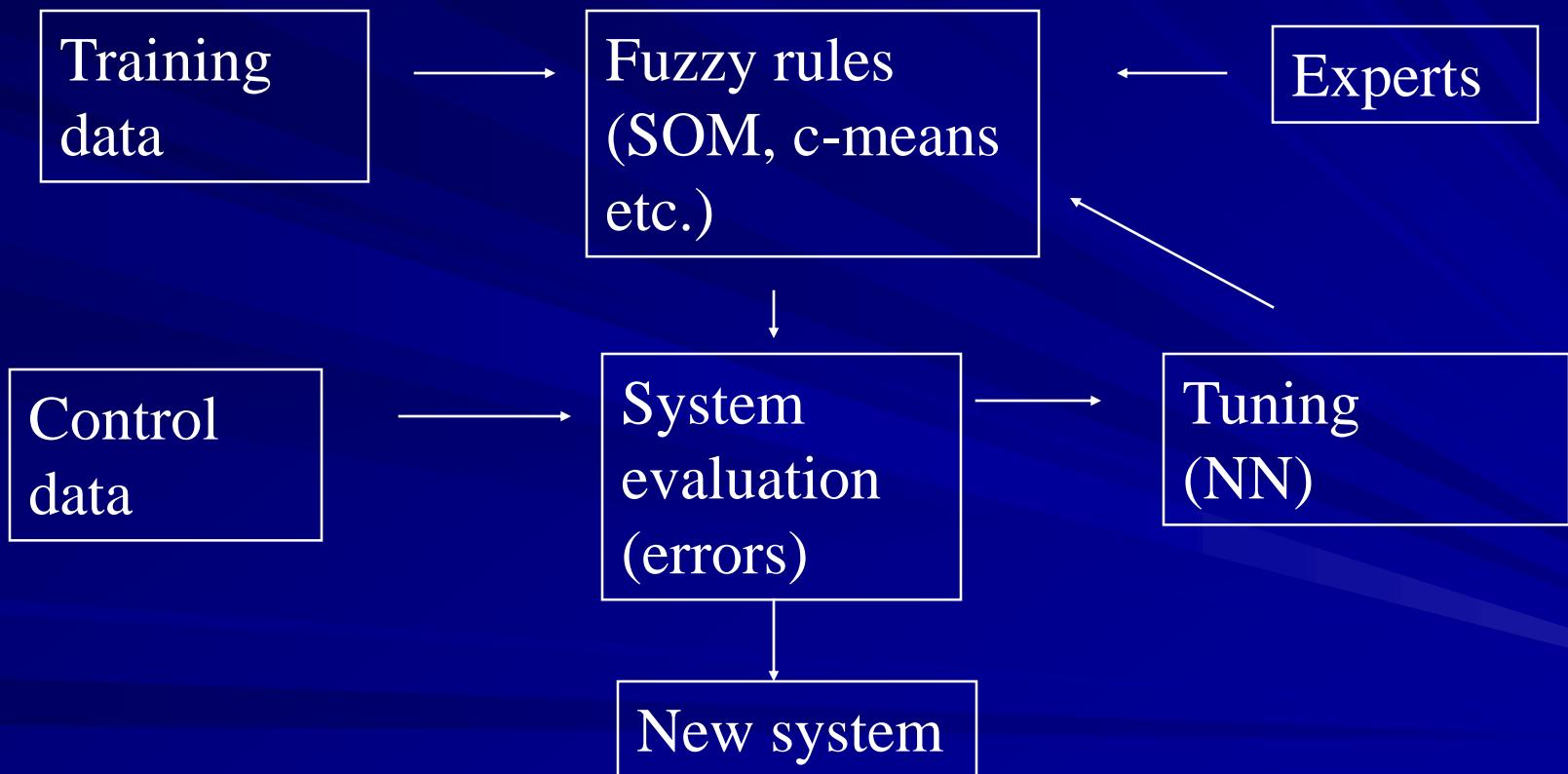
SC applications: finance

- Fuzzy scoring for mortgage applicants,
- creditworthiness assessment,
- fuzzy-enhanced score card for lease risk assessment,
- risk profile analysis,
- insurance fraud detection,
- cash supply optimization,
- foreign exchange trading,
- insider
- trading surveillance,
- investor classification etc.

Outras aplicações

- Estatística
- Ciências Sociais
- Ciência do ambiente
- Biologia
- Medicina

Construção dos sistemas (Neuro)-fuzzy



Referências

- J. Bezdek & S. Pal, Fuzzy models for pattern recognition (IEEE Press, New York, 1992).
- L. Zadeh, Fuzzy logic = Computing with words, IEEE Transactions on Fuzzy Systems, vol. 2, pp. 103-111, 1996.
- L. Zadeh, From Computing with Numbers to Computing with Words -- From Manipulation of Measurements to Manipulation of Perceptions, IEEE Transactions on Circuits and Systems, 45, 1999, 105-119.
- L. Zadeh, Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic, Fuzzy Sets and Systems 90/2 (1997) 111-127.
- H.-J. Zimmermann, Fuzzy set theory and its applications (Kluwer, Dordrecht, 1991).

Computação Evolutiva

Computação Evolutiva

- A Computação Evolutiva ou Computação Evolucionária é o nome dado a métodos computacionais inspirados nas teorias da evolução.
- A motivação para a construção de tais modelos computacionais surgiu de teorias de que a Natureza, através de seus recursos, resolveu problemas de complexidade, isto é, determinar quantidade de “recursos” para resolver “problemas”, de sobrevivência.
- Os Algoritmos Evolutivos, buscam tratar estruturas de objectos abstractos de uma população, como por exemplo, variáveis de um problema de optimização, dos quais são manipulados por operadores inspirados na evolução biologia (operadores genéticos), que objectivam a busca para a solução de um problema.

Algoritmos Genéticos (GA's)

Dado um processo ou método de *codificar* soluções de um problema na forma de cromossomas e dada uma função de desempenho que nos dá um valor de custo de qualquer cromossoma no contexto do problema, os GA's consistem nos seguintes passos:

Passo1: Iniciar a população de cromossomas



Passo2: Determinar o valor de desempenho de cada cromossoma



Passo3: Duplicar os cromossomas de acordo com os valores de desempenho e criar novos cromossomas por entrelaçamento com cromossomas actuais (i.e., mutação, recombinação)



Passo4: Apagar os membros da população indesejáveis



Passo5: Inserir novos cromossomas na população por forma a constituir uma nova população

Reprodução: reprodução é um processo na qual os indivíduos (cromossomas) são copiados de acordo com o seu valor de desempenho.

Esta operação é uma versão artificial da selecção natural

A função de desempenho $f(i)$ assinala para cada indivíduo da população o seu desempenho, em que altos valores representam um bom desempenho.

A função de desempenho pode ser não linear, não diferenciável, descontínua, função positiva.

Selecção: Uma selecção probabilística é executada baseada na aptidão dos indivíduos tais que os indivíduos melhores têm uma possibilidade aumentada de serem seleccionados.

Um indivíduo da população pode ser seleccionado mais que uma vez, assim como todos os indivíduos da população têm uma possibilidade de serem seleccionados para a reprodução. Há diversos esquemas para o processo de selecção: selecção por roda de roleta e suas extensões, técnicas de escalonamento, torneio, modelos do elitista, e métodos de *ranking*.

Selecção por roleta (*Roulette-wheel*):

1. Somar os valores de desempenho de todos os membros da população e chamar a este resultado de *desempenho total*.
2. Gerar um valor aleatório n , um número aleatório entre 0 e o *desempenho total*.
3. Retornar o primeiro elemento da população cujo desempenho que somados com os desempenhos precedentes dos membros da população seja maior ou igual a n .

Nº	String (Cromossoma)	Desempenho	% do total	Desempenho total
1	01110	8	16	8
2	11000	15	30	23
3	00100	2	4	25
4	10010	5	10	30
5	01100	12	24	42
6	00011	8	16	50

Número aleatório	26	2	49	15	40	36	9
Escolha do cromossoma	4	1	6	2	5	5	2

Métodos de selecção por Ranking:

Os métodos de selecção por Ranking requerem apenas a ordenação dos valores de desempenho dos vários indivíduos da população, podendo as funções de selecção dar resultados positivos e negativos. Os métodos de Ranking assinalam uma probabilidade de selecção do elemento i , P_i , baseados na sua posição dentro do ranking.

Método de ranking normalizado:

$$P[\text{selecção do indivíduo } i] = q'(1-q)^{r-1}$$

em que:

q = a probabilidade de selecionar o melhor indivíduo

r = o *rank* do indivíduo, em que o melhor bem valor 1.

P = tamanho da população

$$q' = \frac{q}{1 - (1-q)^P}$$

Selecção por torneio:

O método de selecção por torneio (*Tournament*) carece, tal como os métodos de ranking, dos valores de desempenho dos indivíduos e da sua lista ordenada. Todavia ele não assinala uma probabilidade de selecção para cada indivíduo.

O método de selecção por torneio selecciona aleatoriamente da população um conjunto de j indivíduos e insere o melhor deste conjunto na nova população.

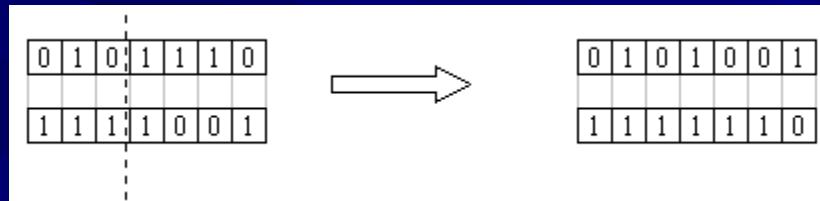
Este procedimento é repetido até que N indivíduos tenham sido seleccionados.

Crossover

Na natureza os filhos têm dois pais e recebem genes de ambos.

Seleccionar um par cromossomas (Pais) com uma probabilidade de crossover p_c

Um ponto de crossover é seleccionado aleatoriamente, então as *strings* são cruzadas



Generalização para Múltiplos pontos de cruzamento

Mutação

Reprodução e Crossover produz novas *strings* mas não introduz nenhuma nova informação na população.

Mutação é introduzida, com uma baixa probabilidade p_m , para inverter *bits* aleatoriamente escolhidos das *strings*.

Estes três operadores são aplicados repetidamente até que os filhos constituam por inteiro a nova população

A geração seguinte é constituída por filhos de três tipos:

- ◆ mutação depois do cruzamento
- ◆ cruzamento sem mutação
- ◆ nem cruzamento nem mutação

Parâmetros a especificar para os GA

n = tamanho da população

p_c = probabilidade de cruzamento

p_m = probabilidade de mutação

G = sobreposição de gerações ($G=1$ nenhuma sobreposição; $0 < G < 1$ sobreposição)

GA no planeamento de trajectórias

Shibata e Fukuda, “Coordinative behavior in evolutionary multi-agent system by genetic algorithm”

Proc. IEEE Int. Conf. Neural Networks, Vol. I, 209-214, San Francisco, 1993

Robô móvel

Strings começam por 0 e terminam com o 26.

1) Processo de selecção:

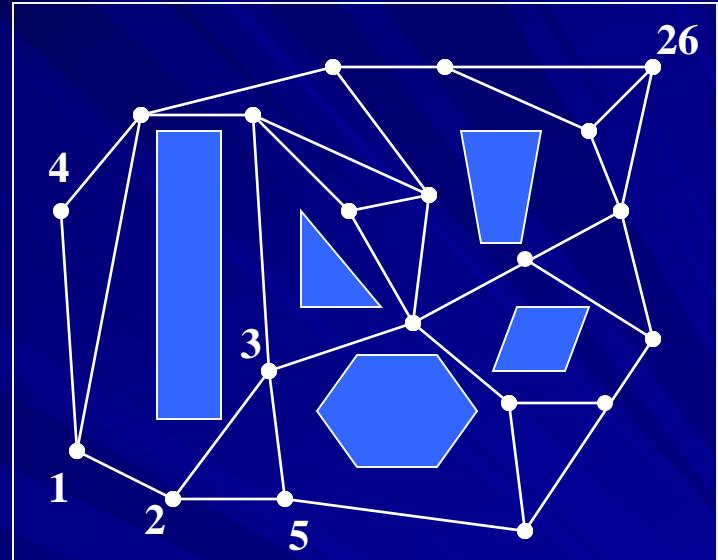
1º - Seleccionar uma *string* que tenha um desempenho eficiente (pelo processo de selecção descrito)

2º - Nessa *string* escolher um número aleatoriamente.

Parente 1: 0-1-2-3-6-9-11-15-16-21-24-26

3º - Escolher uma *string* que também tenha um desempenho eficiente:

Parente 2: 0-4-8-10-15-20-23-26



2) Processo de *crossover*

1º- Se as duas *strings* tiverem um mesmo número em comum -> Realizar o *crossover*.
Senão, escolher outra *string* (repetir o 3º passo)

Filho 1: 0-1-2-3-6-9-11-15-20-23-26

Filho 2: 0-4-8-10-15-16-21-24-26

2º- Verificar números repetidos

Filho 3: 0-1-2-3-6-9-11-15-10-7-11-17-22-25-26

Filho 3': 0-1-2-3-6-9-11-17-22-25-26

3) Processo de mutação

Este processo é realizado segundo uma probabilidade de mutação, que é baixa.
Uma posição em cada *string* é seleccionada e alterada por um numero aleatório.

4) Função de desempenho

$$F(q)=1/(d_{0-4}+d_{4-8}+\dots+d_{23-26})$$

- **Representação:**

Cada indivíduo ou cromossoma é realizado pela sequência de genes de um certo tipo de alfabeto:

Um alfabeto pode consistir em:

- dígitos binários (0 e 1)
- números de vírgula flutuante
- inteiros
- símbolos (isto é, A, B, C, D, ...)
- matrizes
- etc

• Números naturais → representação mais eficiente e produz melhores resultados

• Representação em valores reais é mais eficiente em termos de tempo de CPU que a representação binária

Operações Genéticas

Operações binárias

Mutação binária

$$x'_i = \begin{cases} 1 - x_i & ; \text{ se } U(0,1) < p_m \\ x_i & ; \text{ outros} \end{cases}$$

“Simples Crossover”

r= nº aleatório no intervalo [1,m], segundo uma distribuição de probabilidades uniforme

$$x'_i = \begin{cases} x_i & ; \text{ se } i < r \\ y_i & ; \text{ outros} \end{cases} \quad y'_i = \begin{cases} y_i & ; \text{ se } i < r \\ x_i & ; \text{ outros} \end{cases}$$

Operações sobre nº reais

Sejam a_i e b_i os limites inferiores e superior, respectivamente, de cada variável i

Escolher aleatoriamente uma variável j para mutação.

Seleccionar aleatoriamente uma variável j do cromossoma

Mutação uniforme

$$x'_i = \begin{cases} U(a_i, b_i) & ; \text{ se } i = j \\ x_i & ; \text{ outros} \end{cases}$$

Mutação de fronteira

$$x'_i = \begin{cases} a_i & ; \text{ se } i = j, r < 0.5 \\ b_i & ; \text{ se } i = j, r \geq 0.5 \\ x_i & ; \text{ outros} \end{cases}$$

Mutação não uniforme

$$\dot{x}_i = \begin{cases} x_i + (b_i - x_i) f(G) & ; \text{ se } r_1 < 0.5 \\ x_i + (a_i + x_i) f(G) & ; \text{ se } r_1 \geq 0.5 \\ x_i & ; \text{ outros} \end{cases}$$

Em que : r_1, r_2 = nº aleatórios (d.d.p uniforme) no intervalo [0,1]

$$f(G) = \left(r_2 \left(1 - \frac{G}{G_{\max}} \right) \right)^b$$

G = a geração actual

G_{\max} = o máximo nº de gerações

b = parâmetro factor forma

“Simples Crossover”

$$r = U(0,1)$$

$$\dot{x}_i = r \cdot x_i + (1-r) \cdot y_i$$

$$\dot{y}_i = (1-r) \cdot x_i + r \cdot y_i$$

Crossover heurístico

$$r = U(0,1)$$

$$\dot{x}_i = x_i + r \cdot (x_i - y_i)$$

$$\dot{y}_i = x_i$$

$$feasibility = \begin{cases} 1, & \text{if } x'_i \geq a_i, x'_i \leq b_i \quad \forall i \\ 0, & \text{outros} \end{cases}$$

Programação Genética (GP)

GA ⇒ ferramenta poderosa para encontrar os pontos óptimos do espaço para uma grande variedade de problemas.

Problema: a solução de muitos problemas não é meramente numérica mas uma função inteira, que é composta por funções primitivas (i.e., adição, subtracção, multiplicação, exp., ln, ...) e terminais (i.e., tempo, números reais, ...)

Programação genética (GP) fornece a via de pesquisa no espaço de todas as possíveis funções compostas por um certo número de terminais e funções primitivas, para encontrar a função que resolve o problema [Koza, 1992].

Programação Genética (GP)

PASSO 1

Gerar uma população inicial aleatória de programas compostos de funções primitivas e terminais do problema.



PASSO 2

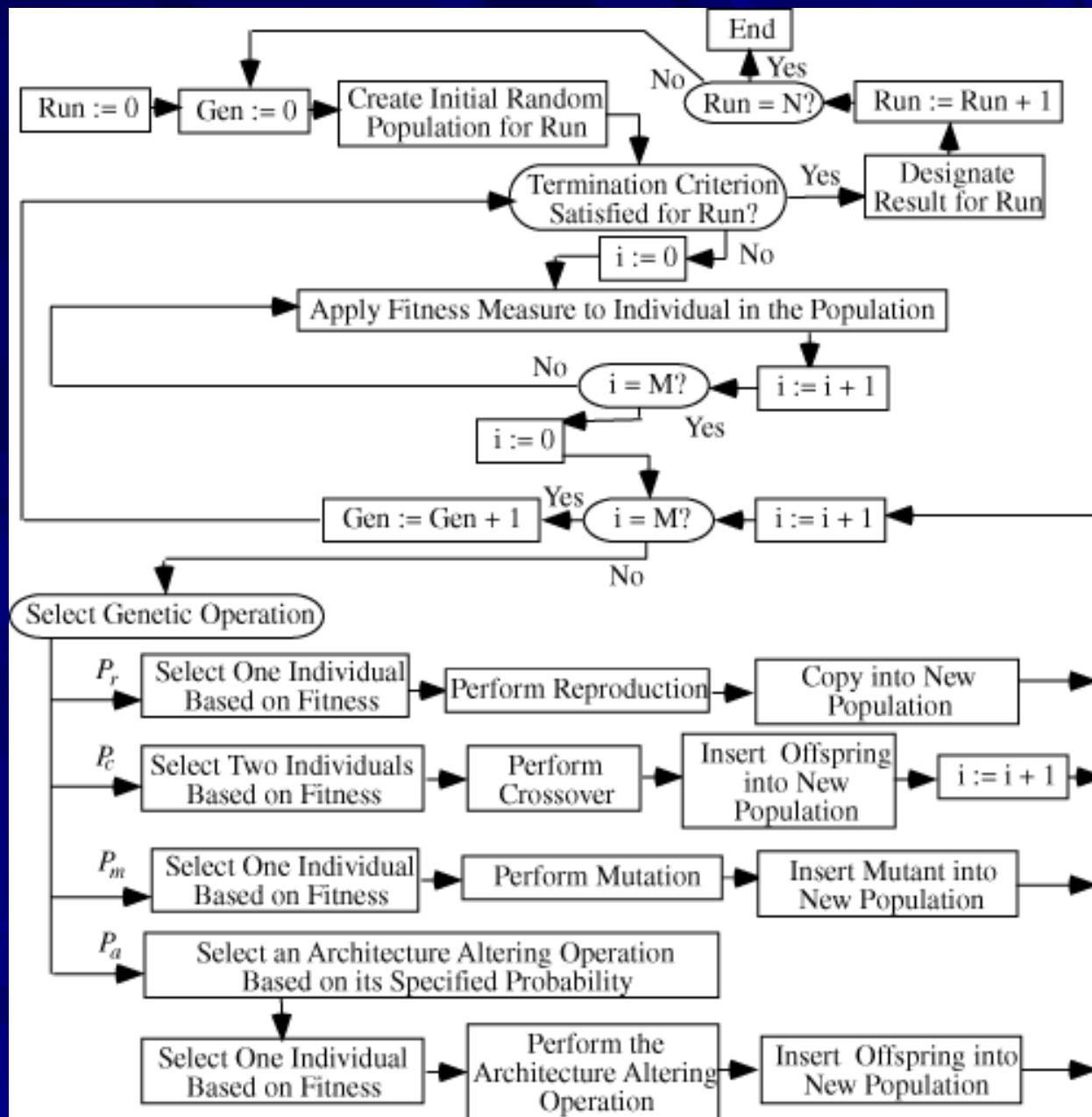
Realizar iterativamente os seguintes sub-passos até satisfazer o critério de paragem.

- A. Executar cada programa da população e assinalar o valor de desempenho.**
- B. Criar uma nova população de programas pela aplicação das seguintes operações genéticas**
 - (i) Reprodução:** copiar os programas existentes para a nova população.
 - (ii) Crossover:** Criar dois novos filhos programas da nova população pela recombinação genética, aleatoriamente escolhida de duas partes de programas existentes.
A operação genética de “crossover” opera sobre dois pais programas e produz dois filhos programas usando partes de cada um dos pais.
 - (iii) Mutação na população de programas**

PASSO 3

O melhor programa da população, produzido durante a execução do GP, é designado como o resultado do GP.





Representação em árvore

■ A forma em árvore é universal

■ Fórmula Aritmética

$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$

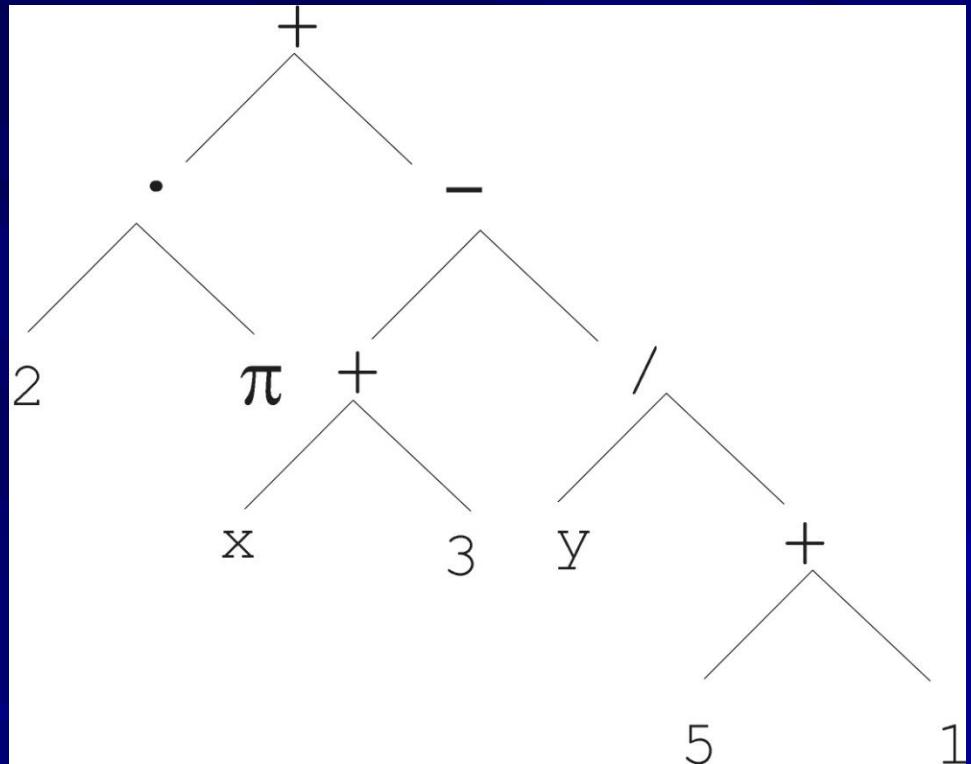
■ Fórmula lógica

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

■ Programa

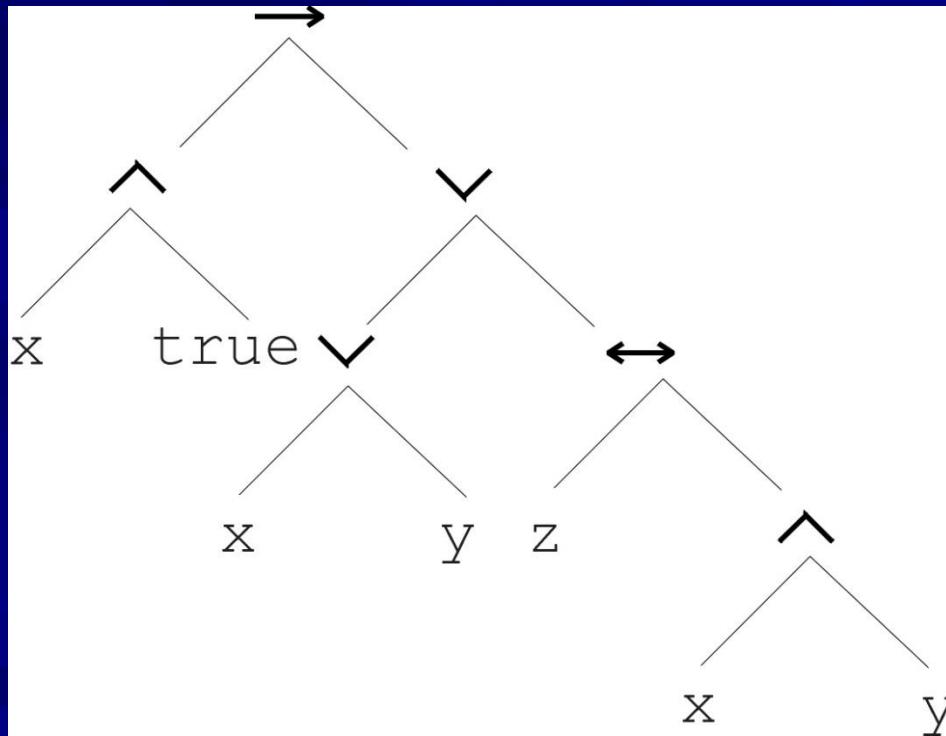
```
i = 1;  
while (i < 20)  
{  
    i = i + 1  
}
```

Representação em árvore

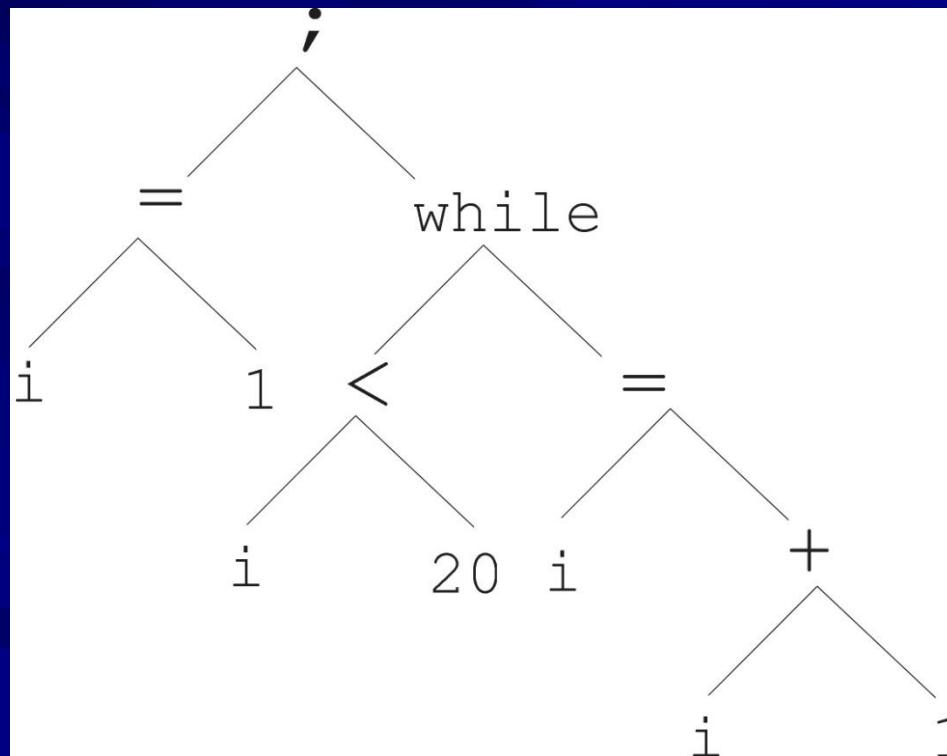


$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$



```
i = 1;  
while (i < 20)  
{  
    i = i + 1  
}
```



Representação em forma de árvore

- Os cromossomas em GA, ES, EP são estruturas lineares (*strings* de bits, *string* de inteiros, vectores de valores reais, permutações)
- Cromossomas em forma de árvore são estruturas não lineares
- Nos GA, ES, EP o tamanho do cromossoma é fixo
- O tamanho de uma árvore no GP pode variar em profundidade e largura

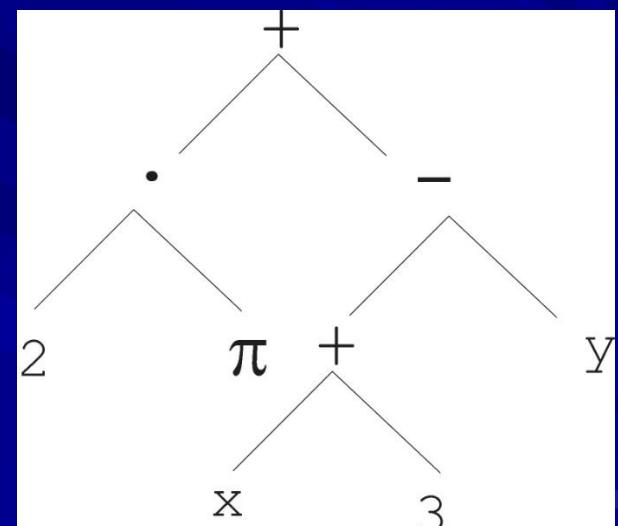
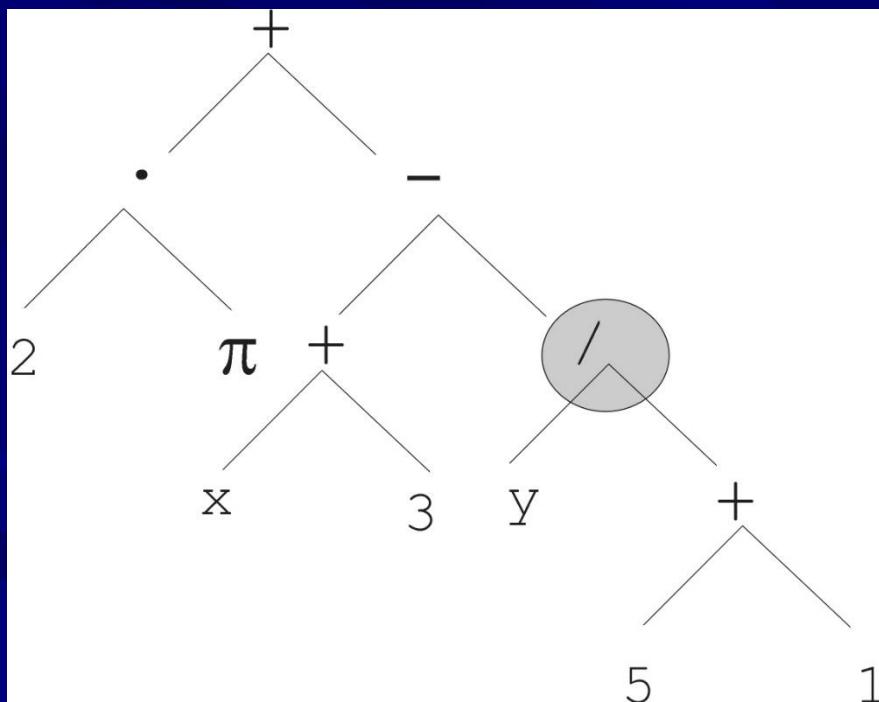
Representação

Expressões simbólicas pode ser definida por

- Conjunto Terminal T
- Conjunto de Funções F
- Adoptando a seguinte definição recorrente geral:
 1. Cada $t \in T$ uma expressão adequada
 2. $f(e_1, \dots, e_n)$ uma expressão adequada se $f \in F$,
 $\text{arity}(f)=n$ e e_1, \dots, e_n são expressões adequadas
 3. Não existem outras formas de expressões adequadas
- Em geral, as expressões no GP são não caracterizáveis
(propriedade fechada: para qualquer $f \in F$ podemos ter
qualquer $g \in F$ como argumentos)

Mutação

- Mutação mais utilizada: substituir uma sub-árvore escolhida aleatoriamente por uma outra sub-árvore gerada aleatoriamente

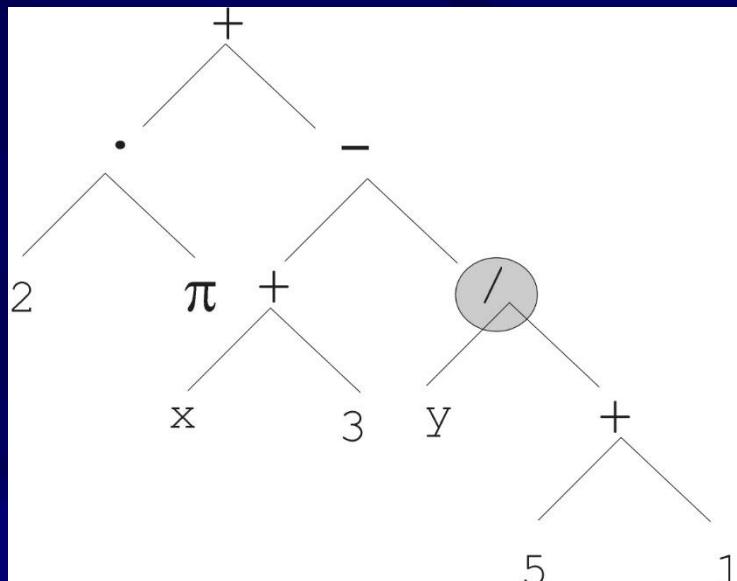


Mutação (cont.)

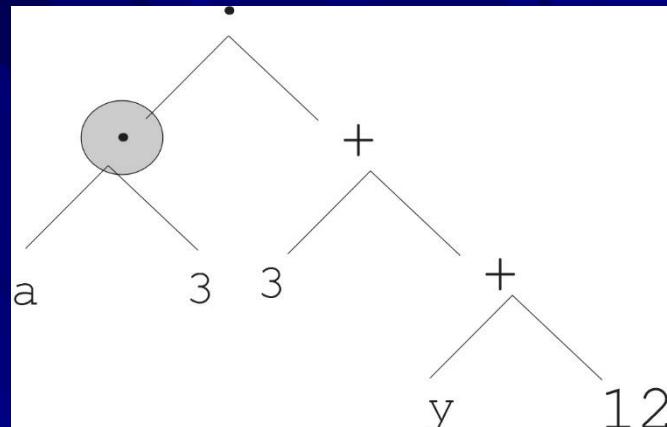
- Mutação tem dois parâmetros:
 - A probabilidade p_m da escolha da mutação versus recombinação
 - A probabilidade de escolher um ponto interno como raiz da sub-árvore a ser substituída
- É prudente o valor de p_m ser 0 (Koza'92) ou muito pequeno, tal como 0.05 (Banzhaf e outros. '98)
- O tamanho do filho pode exceder o tamanho do pai.

Recombinação

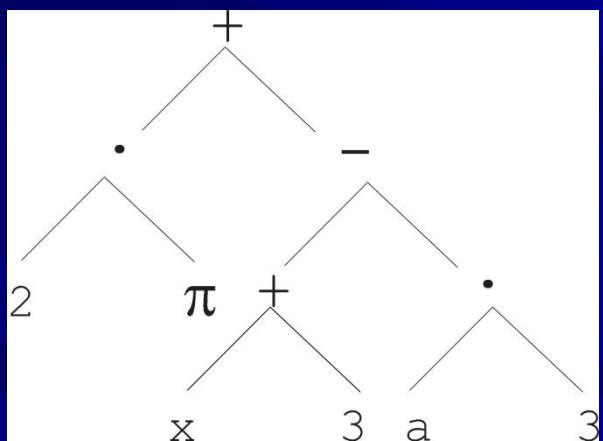
- Recombinação usual: trocar duas sub-árvore escolhidas aleatoriamente dos pais.
- A Recombinação tem dois parâmetros:
 - A Probabilidade p_c para escolher entre recombinação versus mutação
 - A Probabilidade de escolher um ponto interno dentro de cada cromossoma dos pais como pontos de cruzamento
- O tamanho dos descendentes pode exceder os tamanhos dos pais.



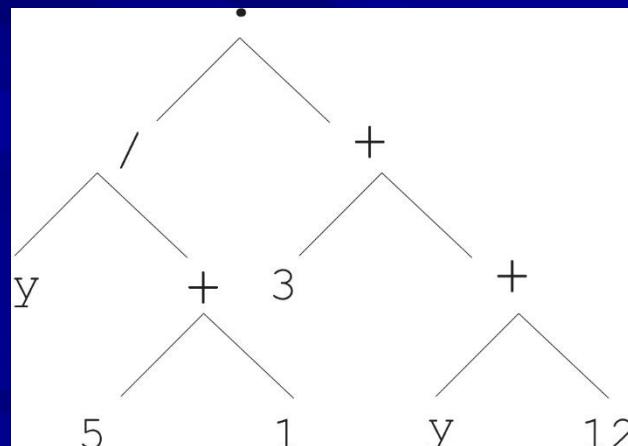
Pai 1



Pai 2



Filho 1



Filho 2

Selecção

- A selecção dos Pais é tipicamente proporcional ao desempenho
- Sobre-selecção em populações numerosas
 - ordenar população pelo desempenho e dividir em dois grupos:
 - 1º grupo: os x% melhores da população, 2º grupo os outros (100-x)%
 - 80% das operações de selecção escolhidos do grupo 1, 20% do grupo 2
 - para pop. size = 1000, 2000, 4000, 8000 x = 32%, 16%, 8%, 4%
 - motivação: aumentar eficiência das % através da regra do polegar
- Selecção dos sobreviventes:
 - Típica: esquema geracional
 - elitismo

Inicialização

- Profundidade inicial máxima da arvore D_{\max}
- Método completo (cada braço tem prof. = D_{\max}):
 - Nós à profundidade $d < D_{\max}$ são escolhidos aleatoriamente do conjunto de funções F
 - Nós à profundidade $d = D_{\max}$ são escolhidos aleatoriamente do conjunto de terminais T
- Método de crescimento (cada braço tem prof. $\leq D_{\max}$):
 - Nós à profundidade $d < D_{\max}$ são escolhidos aleatoriamente de $F \cup T$
 - Nós à profundidade $d = D_{\max}$ são escolhidos aleatoriamente de T
- Iniciar GP (tipicamente): metade da população inicial através do método Completo e do Crescimento

Exemplos de aplicações: regressão simbólica

- Dados alguns pontos em \mathbb{R}^2 , $(x_1, y_1), \dots, (x_n, y_n)$
- Encontrar a função $f(x)$ s.t. $\forall i = 1, \dots, n : f(x_i) = y_i$
- Possível solução GP:
 - Representação por $F = \{+, -, /, \sin, \cos\}$, $T = \mathbb{R} \cup \{x\}$
 - Desempenho é o erro
 - Todos os operadores standard
 - `pop.size = 1000`, inicialização metade/metade
 - Término: n “hits” ou o valor alto de desempenho (onde “hit” é se $|f(x_i) - y_i| < \varepsilon$)

Exemplos:

Variáveis: (x, y, z)

Funções primitivas: (+, *, -)

Terminais: (números reais)

Operação de Crossover:

$$(+(*z\ 0.1237)(-x\ 0.547)) \Leftrightarrow 0.123\ z + x - 0.547$$

$$(*(*zy)(+y(*0.725z))) \Leftrightarrow z\ y\ (y + 0.725\ z)$$

$$(+(+y(*0.725z))(-x\ 0.547)) \text{ e } (*(*zy)(*123z))$$

Problema Proposto

■ SUDOKU

Sudoku é um puzzle baseado na colocação lógica de números.

5	3			7				
6			1	9	5			
	9	8					6	
8			6					3
4		8		3				1
7			2					6
	6				2	8		
		4	1	9				5
		8			7	9		

Regra: Um número (ou símbolo) só pode aparecer uma vez em cada linha, coluna ou quadrante.

Objectivo: Implemente uma estratégia baseado nos GA's para a resolução do problema Sudoku.

Para o efeito, desenvolva apropriadas operações de:

- Codificação;
- Mutação;
- Cruzamento.

Swarm Intelligence

- Algoritmos em que os agentes actuam localmente realizando alguma interação com o grupo

Individualismo x Coletivo

- As ações individuais de cada agente somadas e a interação entre eles formam a solução do problema como um todo
- Algoritmos mais populares: Ant Colony Optimization e Particle Swarm Optimization

Travelling Salesman Problem (TSP)

- Problema do caixeiro viajante -



G. B. Dantzig, R. Fulkerson, and S. M. Johnson, *Solution of a large-scale traveling salesman problem*, Operations Research 2 (1954), 393-410.

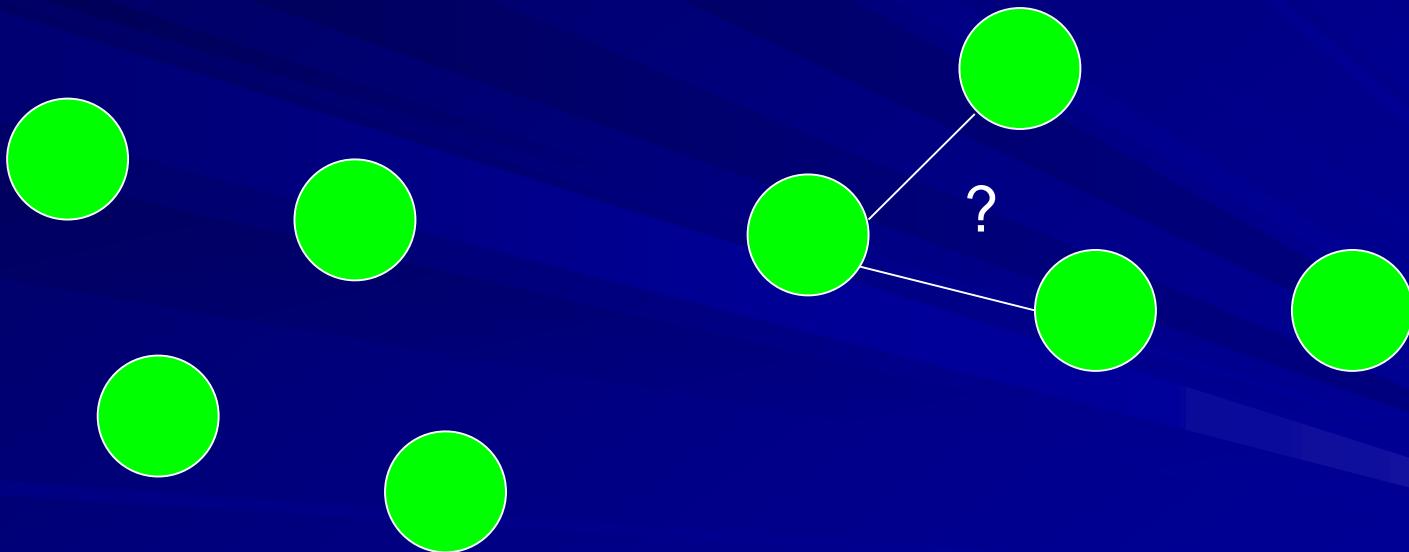
Um caixeiro viajante deve partir da sua cidade, visitar “n” cidades diferentes e retornar à origem.

Questão a resolver:

Qual a seqüência de cidades que o caixeiro deve percorrer de modo a que gaste o menor tempo possível ou percorra a menor distância?

Problema do caixeiro viajante

A solução não é nada intuitiva, para um número modesto de cidades!!!!!!



Problema do caixeiro viajante

- No exemplo anterior temos 8 cidades. Isso dá um total de $(8-1)! = 5040$ combinações possíveis. Fácil de resolver!
- Se aumentarmos para 20 cidades, teríamos $(20-1)! = 121645100408832000$. Não é tão simples!!!

– Problema do Caixeiro Viajante –

- Para um problema típico com 100 cidades, teríamos $(100-1)! = 9 \times 10^{155}$ combinações possíveis!!!!
- Com um computador de 10THz (que consegue processar 10^{12} combinações por ciclo) levaríamos 9×10^{143} ciclos para completar essa tarefa, 9×10^{131} segundos que é igual a 3×10^{136} anos. O universo tem aproximadamente $13,7 \times 10^9$ anos.

Ant Colony Optimization



Goss, S., S. Aron e J. L. Deneubourg. Self-organized shortcuts in the Argentine ant.
Naturwissenschaften, v.76, p.579-581. 1989.

Ant Colony Optimization



Ant Colony Optimization

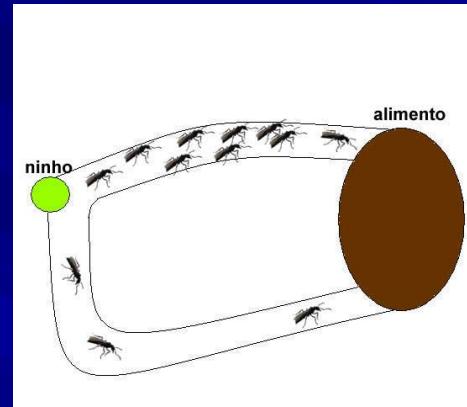
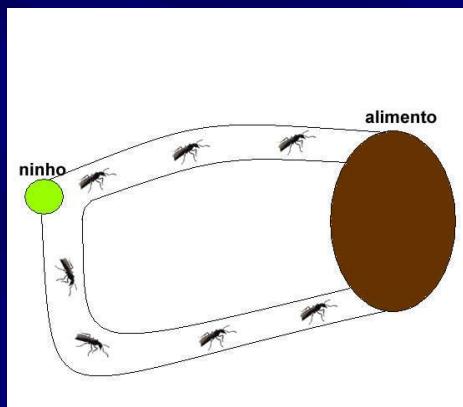


- Resultado da observação do comportamento das formigas na busca pelos alimentos.
- Inicialmente cada formiga segue um caminho aleatório
- Após algum tempo elas tendiam a seguir um único caminho, considerado óptimo.
- Cada formiga utiliza uma comunicação indirecta para informar as outras de quanto foi óptimo o caminho que ela escolheu
- Essa informação está ligada à intensidade de uma substância chamada “feromôna”, que as formigas vão largando no seu trajecto.

Ant Colony Optimization

Foi colocado um ninho de formigas em um aquário com uma fonte de alimentos na outra ponta.

Para chegar até esse alimento foram criados dois caminhos, sendo um maior que o outro.



As formigas que escolheram o caminho mais curto fazem o trajecto de ida e volta mais rapido do que as outras. Estas acabavam depositando uma maior quantidade de feromônio nesse caminho em relação ao outro caminho alternativo, para um mesmo instante de tempo.

A dado momento a intensidade do feromônio no caminho mais curto será tão mais alto que quase todas as formigas seguirão por esse trajecto.

Ant Colony Optimization

Em 1992, Dorigo percebeu que as formigas resolviam um problema muito similar ao TSP e, inspirado nesse comportamento, resolveu modelá-lo computacionalmente e verificar como se comportava em algumas instâncias conhecidas do problema.

Ant Colony Optimization

```
While it < max_it do,  
    for each ant do,  
        build_solution();  
    endfor  
    update_pheromone();  
endwhile
```

Ant Colony Optimization build_solution()

Para construir a solução cada formiga utiliza iterativamente uma função probabilística para decidir se incluirá ou não determinada aresta na solução.

$$p_{i,j}^k(t) = \begin{cases} \frac{\tau_{i,j}^\alpha(t) \cdot \eta_{i,j}^\beta}{\sum_{j \in J^k} \tau_{i,j}^\alpha(t) \cdot \eta_{i,j}^\beta} & \text{se } j \in J^k \\ 0 & \text{c. c.} \end{cases}$$

Onde:

J^K é a lista de vértices não visitados;

$\tau_{i,j}$ é a quantidade de feromônio na aresta (i,j) ;

$\eta_{i,j}$ é a informação de qualidade dessa aresta (geralmente determinada pelo inverso da distância)

α e β são parâmetros que definem o grau de importância de τ e η respectivamente.

Ant Colony Optimization

update_pheromone()

Para atualizar a trilha de feromônio nas arestas é calculada inicialmente a quantidade a ser depositada em cada uma das proporcional a qualidade das soluções que elas pertencem

$$\tau_{i,j}(t + 1) = (1 - \rho) \cdot \tau_{i,j}(t) + \rho \cdot \Delta\tau_{ij}$$

$$\Delta\tau_{i,j} = \begin{cases} 1/f(S) & \text{se } (i,j) \in S \\ 0 & \text{c. c.} \end{cases}$$

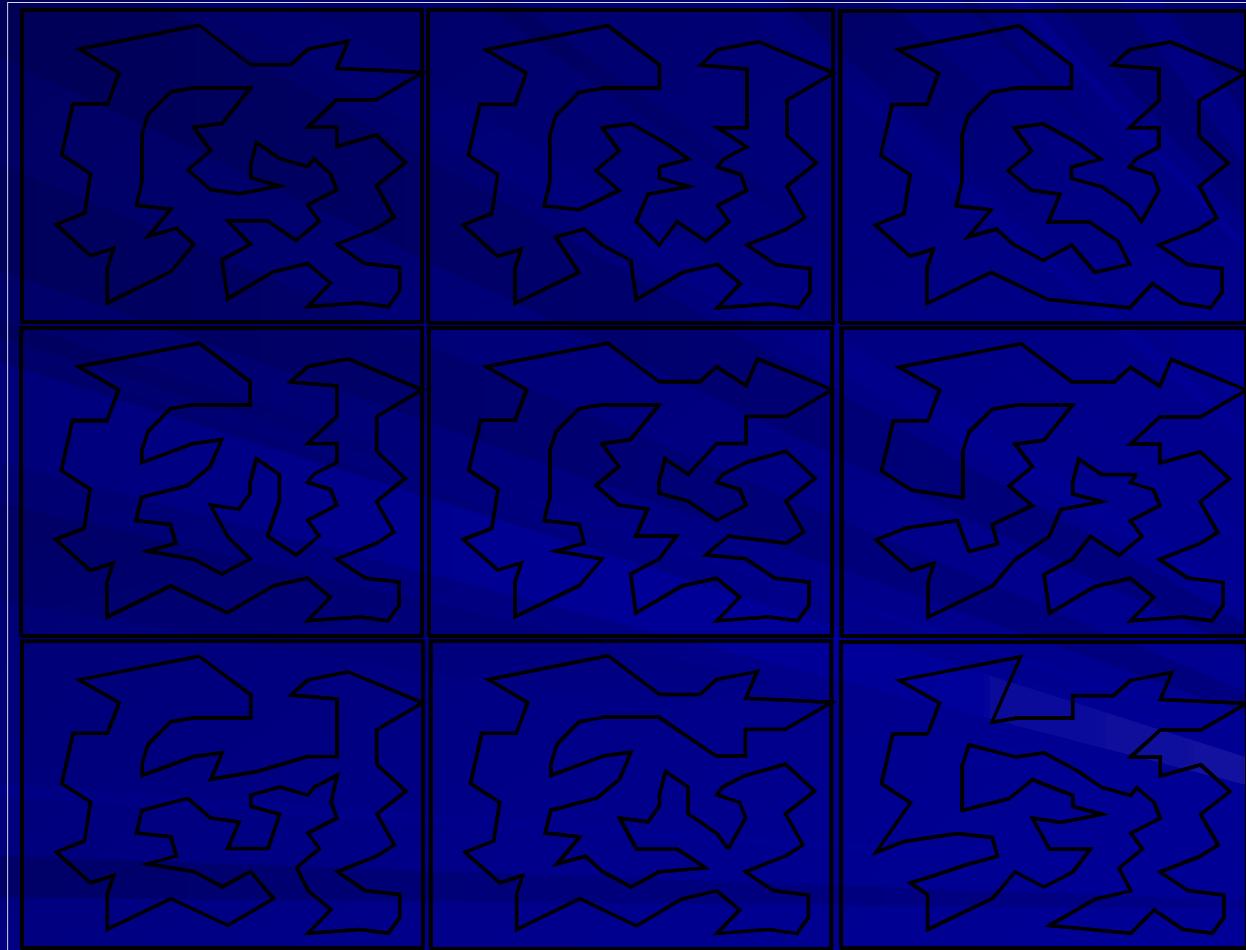
Onde:

ρ é a taxa de evaporação do feromônio;

$\Delta\tau_{i,j}$ é a quantidade de feromônio que será depositada na aresta (i,j) ;

$f(S)$ é o custo total da solução “S”

Aplicações TSP



Cada vértice representa uma cidade

Aplicações TSP

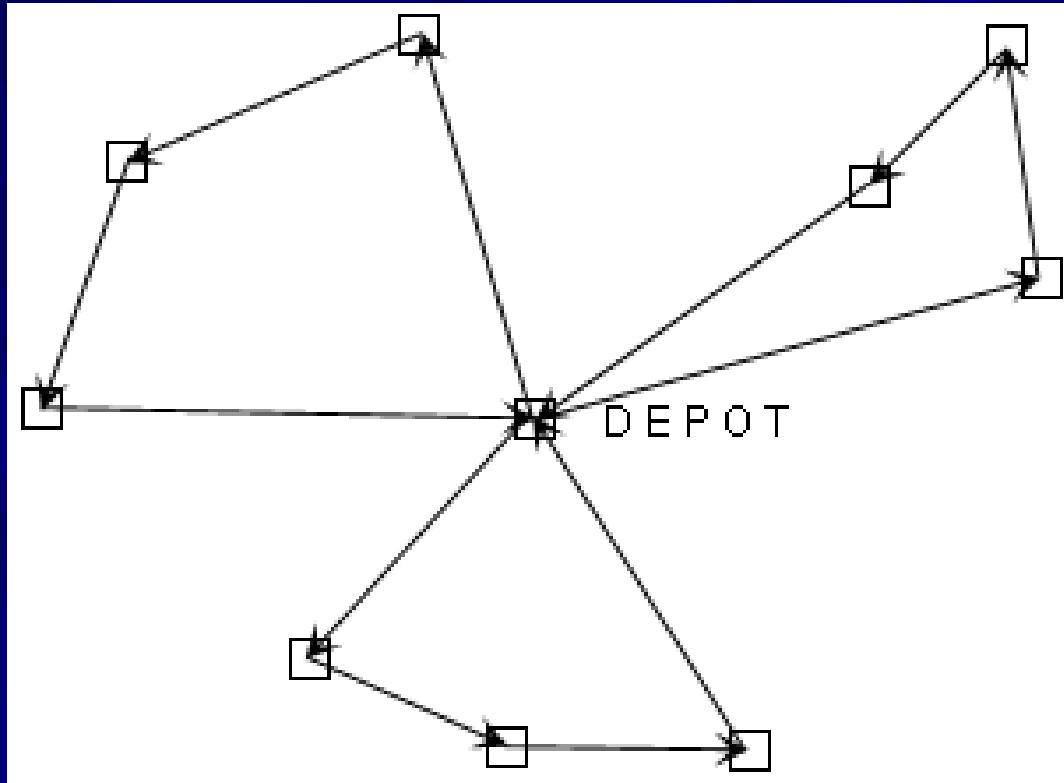
Dorigo M., V. Maniezzo & A. Colorni (1996). Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41

Colorni A., M.Dorigo, F.Maffioli, V. Maniezzo, G. Righini, M. Trubian (1996). Heuristics from Nature for Hard Combinatorial Problems. International Transactions in Operational Research, 3(1):1-21.

Dorigo M. & L.M. Gambardella (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1):53-66. (Also Technical Report TR/IRIDIA/1996-5, IRIDIA, Université Libre de Bruxelles.)

Dorigo M. & L.M. Gambardella (1997). Ant Colonies for the Traveling Salesman Problem. BioSystems, 43:73-81. Also Technical Report TR/IRIDIA/1996-3, IRIDIA, Université Libre de Bruxelles.

Aplicações Vehicle Routing



É formado por vários problemas de TSP partindo sempre de um ponto central

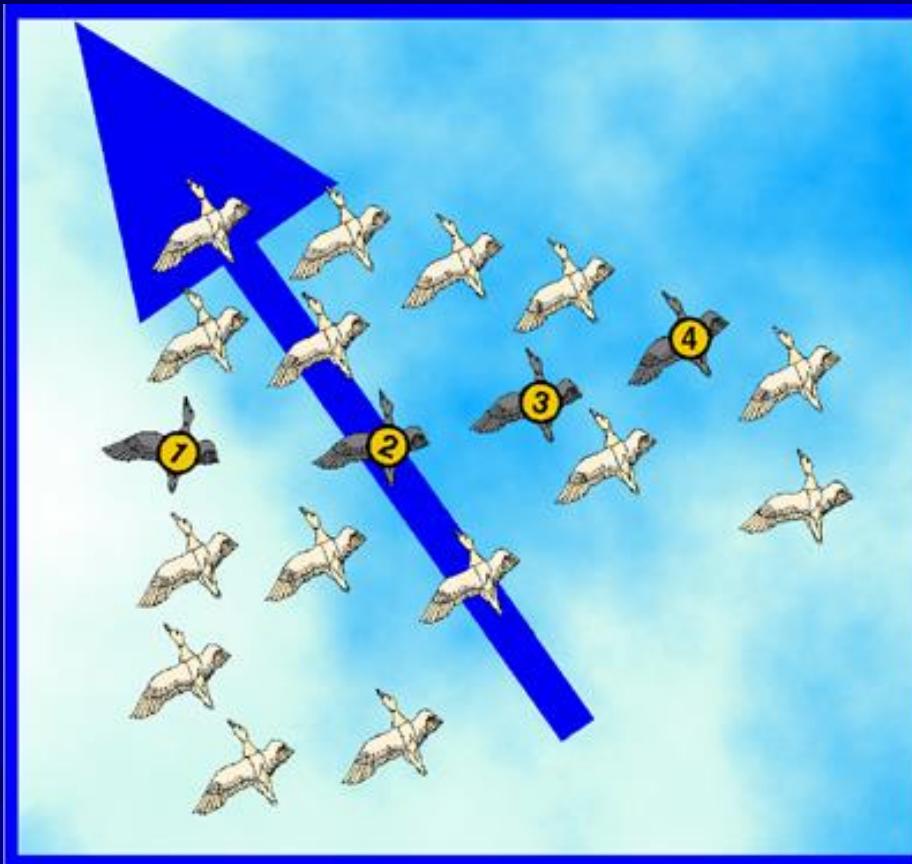
Aplicações Rota de veículos (Vehicle Routing)

Bullnheimer B., R.F. Hartl and C. Strauss (1999). **An Improved Ant system Algorithm for the Vehicle Routing Problem.** Paper presented at the Sixth Viennese workshop on Optimal Control, Dynamic Games, Nonlinear Dynamics and Adaptive Systems, Vienna (Austria), May 21-23, 1997, to appear in: *Annals of Operations Research* (Dawid, Feichtinger and Hartl (eds.): Nonlinear Economic Dynamics and Control, 1999.

Bullnheimer B., R.F. Hartl and C. Strauss (1999). **Applying the Ant System to the Vehicle Routing Problem.** In: Voss S., Martello S., Osman I.H., Roucairol C. (eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer:Boston.

Bullnheimer B. (1999). **Ant Colony Optimization in Vehicle Routing.** Doctoral thesis, University of Vienna, January 1999.

Particle Swarm Optimization - PSO



Particle Swarm Optimization - PSO

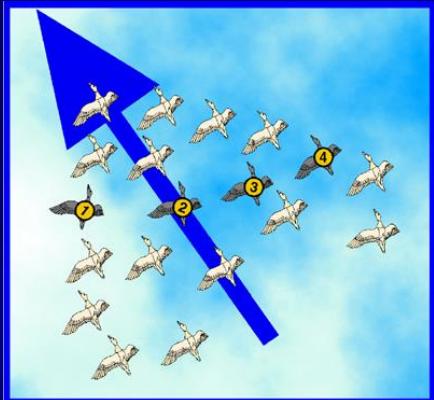


Craig Reynolds percebeu que a movimentação de bandos de pássaros e cardumes de peixes eram sincronizados sem existir um controle central.

Ele então criou um modelo do movimento de bandos de pássaros composto apenas por 4 regras:

- **Separação**: para evitar que cada pássaro colidisse com um outro
- **Alinhamento**: para fazer com que cada pássaro seguisse a mesma direção de seus vizinhos
- **Coesão**: para que cada pássaro seguisse a mesma posição de seus vizinhos
- **Desvio**: para que cada pássaro desvisasse de obstáculos a frente

Particle Swarm Optimization - PSO



Em 1995, James Kennedy and Russell Eberhart seguiram o modelo de Reynolds para aplicar em problemas de otimização contínua.

Para isso eles utilizaram as regras de alinhamento e coesão e criaram um sistema de partículas denominado Particle Swarm Optimization (PSO)

Particle Swarm Optimization - PSO

Regra de atualização do posicionamento das partículas:

$$p = p + v$$

com:

$$v = v + c1 * \text{rand} * (\text{pBest} - p) + c2 * \text{rand} * (\text{gBest} - p)$$

onde

p = posição da partícula

v = direção de caminhada

c1 = importância da informação local

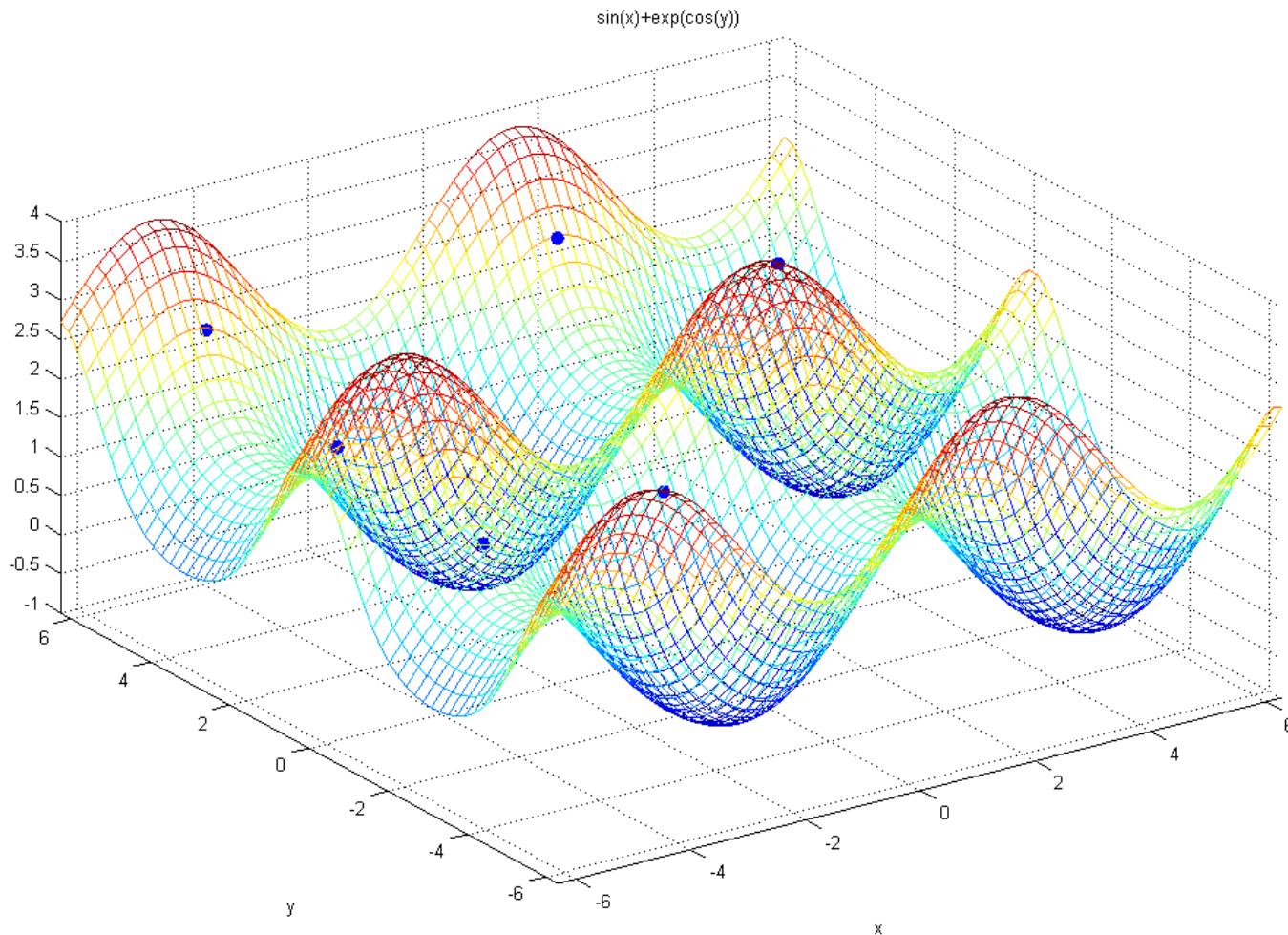
c2 = importância da informação global

pBest = melhor posição do passado dessa partícula

gBest = melhor posição global dos vizinhos dessa partícula

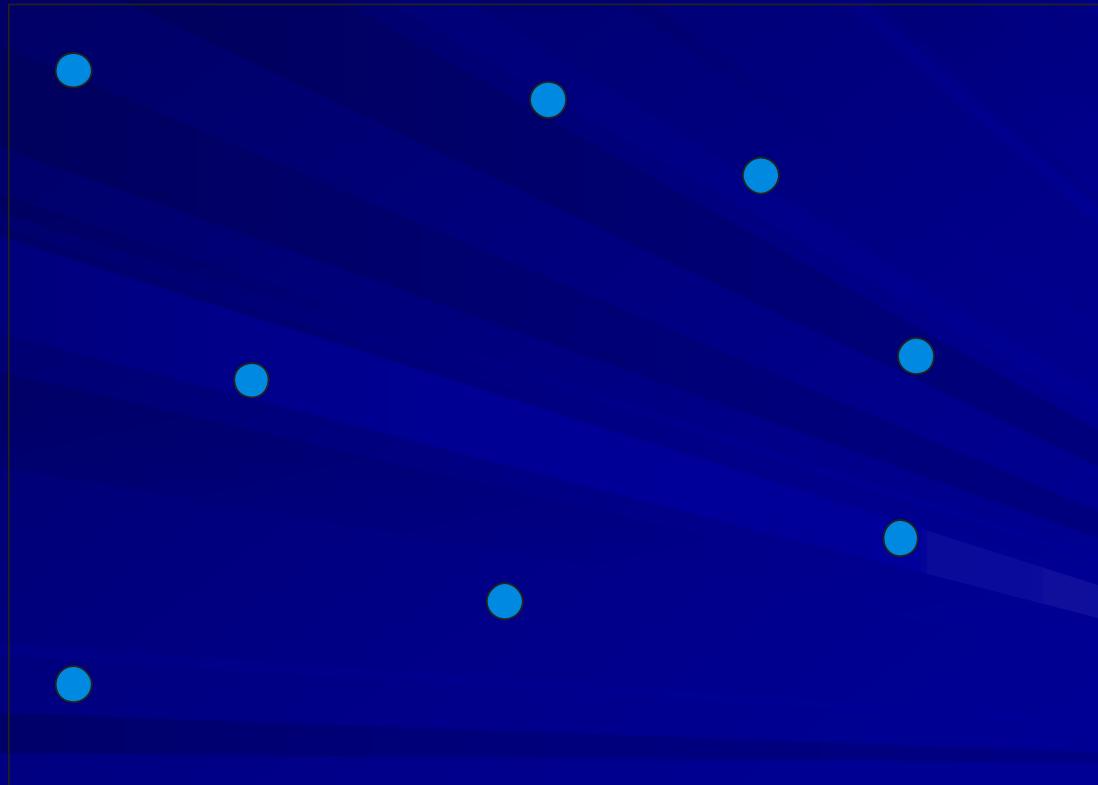
rand = variável aleatória

Particle Swarm Optimization - PSO



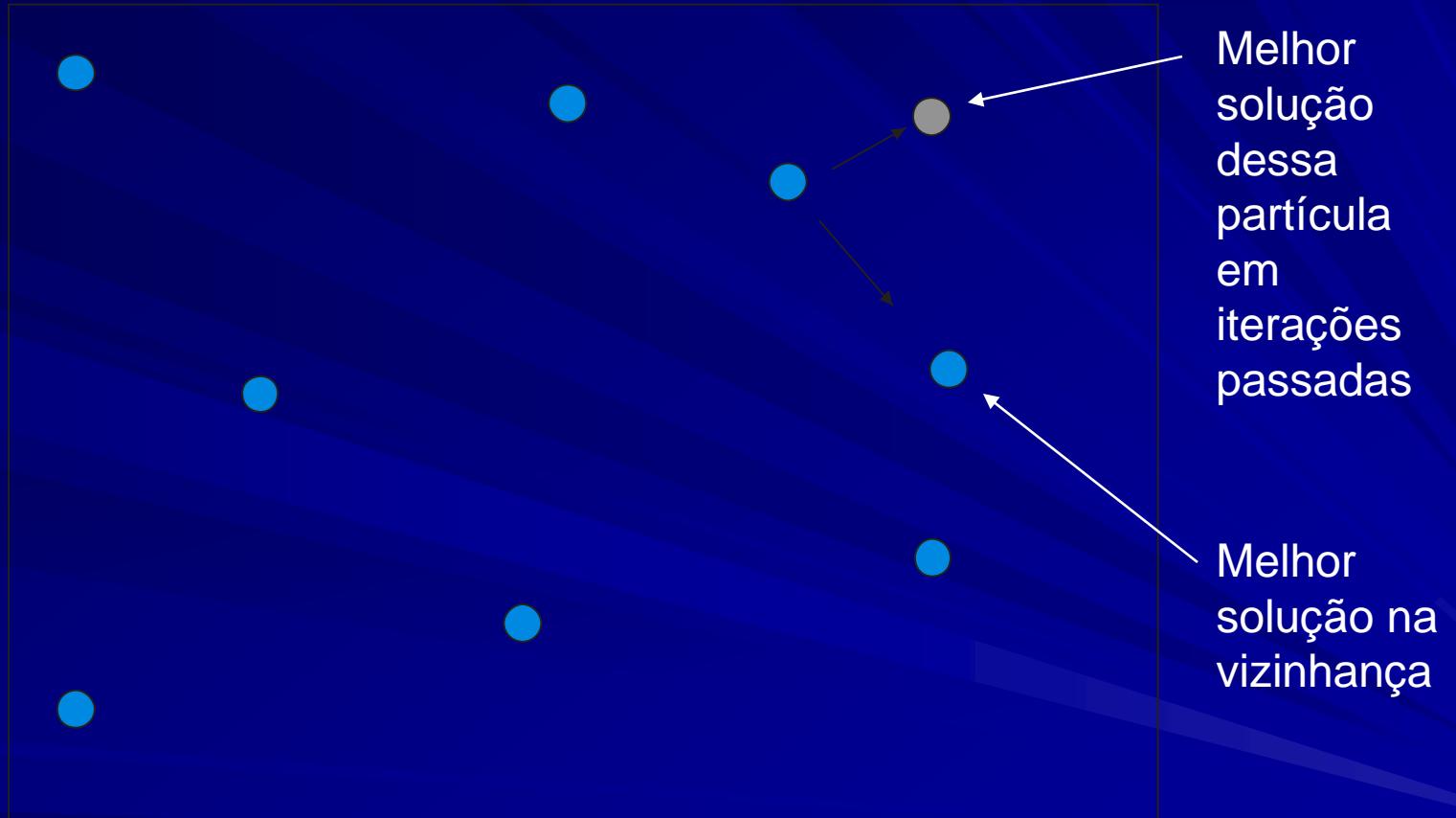
Particle Swarm Optimization - PSO

Nas ilustrações a seguir o gráfico anterior estará mapeado em um ambiente 2D para melhor visualização.



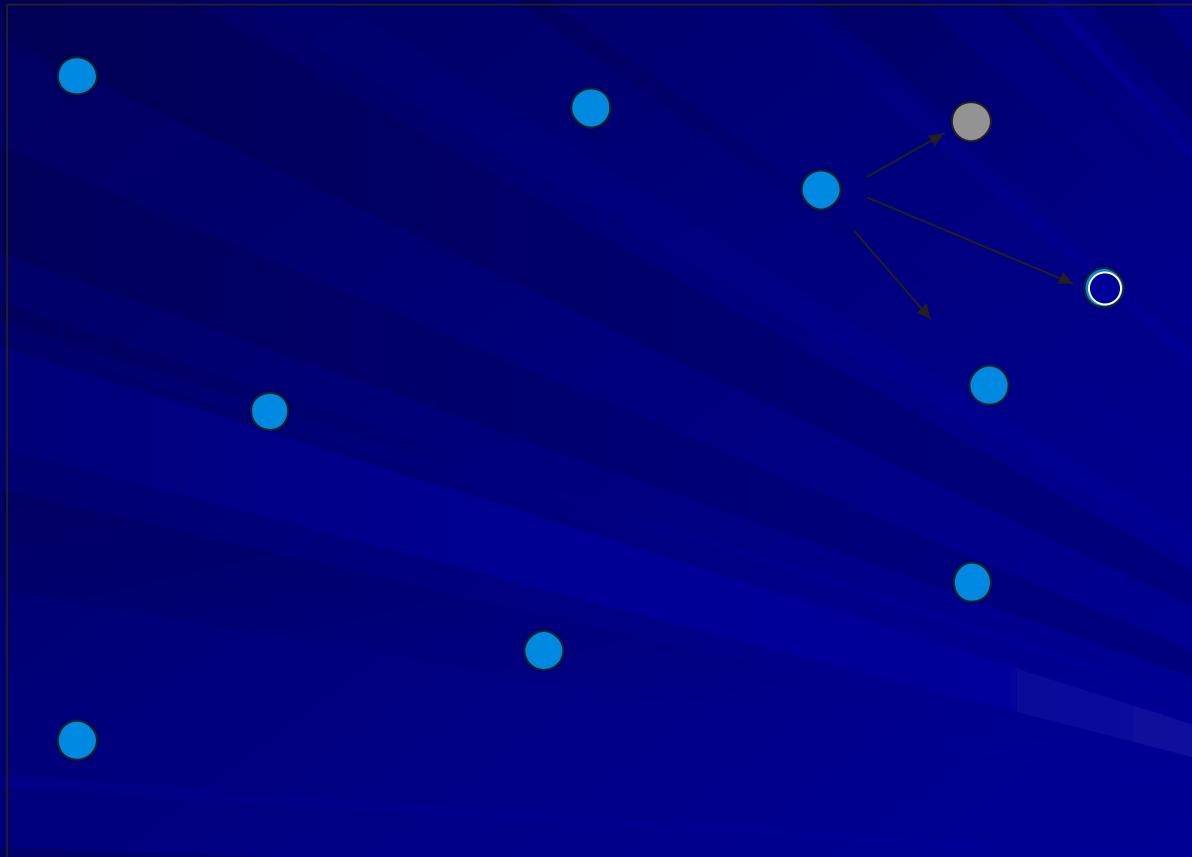
Inicialmente no algoritmo, diversas partículas são espalhadas aleatoriamente no espaço de busca.

Particle Swarm Optimization - PSO



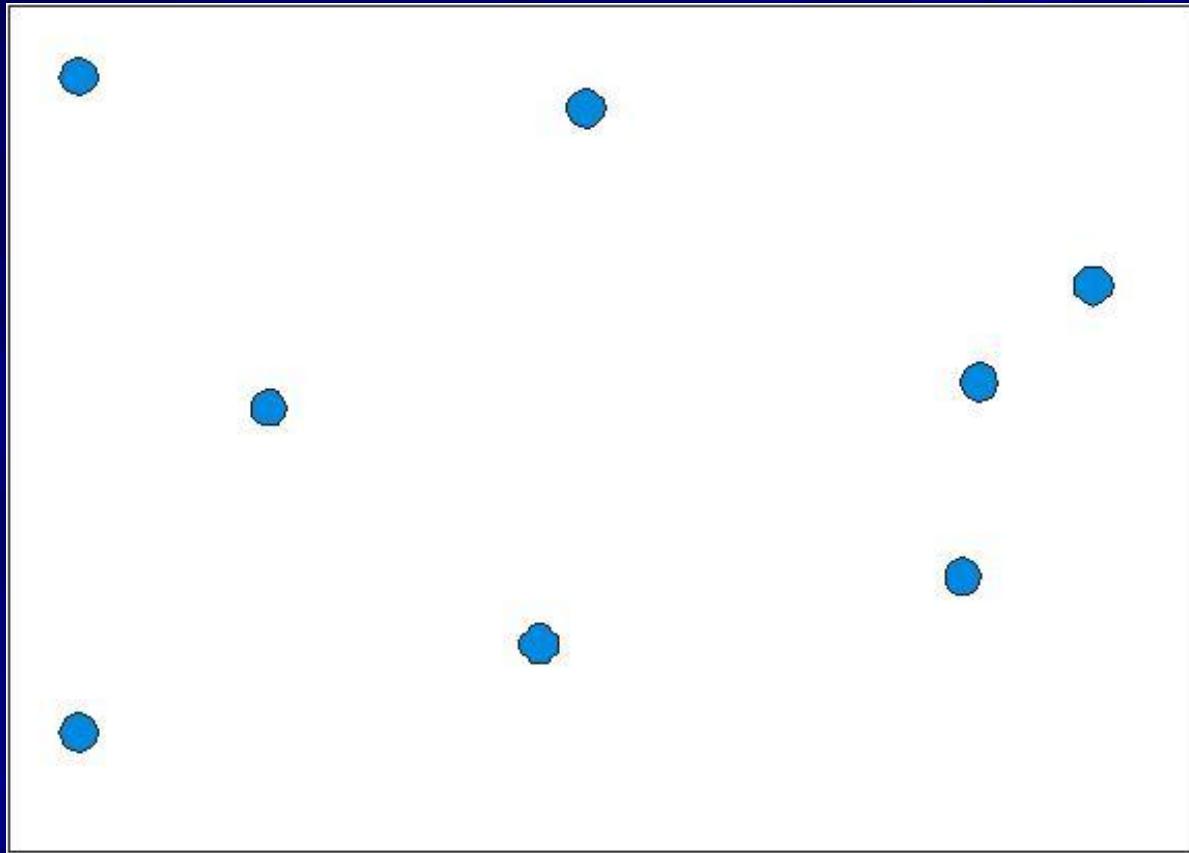
Iterativamente cada partícula utiliza a informação de sua melhor posição no passado (em cinza) e da melhor posição atual entre seus vizinhos.

Particle Swarm Optimization - PSO



E então ela se move em uma combinação linear desses dois vetores, com pesos diferentes, em uma nova posição.

Particle Swarm Optimization - PSO



Particle Swarm Optimization - PSO

Eberhart, R. C. and Kennedy, J. A new optimizer using particle swarm theory.
Proceedings of the Sixth International Symposium on Micromachine and Human Science,
Nagoya, Japan. pp. 39-43, 1995

Kennedy, J. and Eberhart, R. C. Particle swarm optimization. Proceedings of IEEE
International Conference on Neural Networks, Piscataway, NJ. pp. 1942-1948, 1995

Kannan, S., Slochanal, S. M. R., Subbaraj, P., and Padhy, N. P., "Application of particle
swarm optimization technique and its variants to generation expansion planning problem,"
Electric Power Systems Research, vol. In Press, Corrected Proof 2004.

Onwubolu, G. C. and Clerc, M., "Optimal path for automated drilling operations by a new
heuristic approach using particle swarm optimization," *International Journal of Production
Research*, vol. 4 pp. 473-491, 2004.

Soft Computing no futuro

Soft Computing e os métodos convencionais podem ser usados em conjunto.



Referencias

- J. Bezdek & S. Pal, Fuzzy models for pattern recognition (IEEE Press, New York, 1992).
- L. Zadeh, Fuzzy logic = Computing with words, IEEE Transactions on Fuzzy Systems, vol. 2, pp. 103-111, 1996.
- L. Zadeh, From Computing with Numbers to Computing with Words -- From Manipulation of Measurements to Manipulation of Perceptions, IEEE Transactions on Circuits and Systems, 45, 1999, 105-119.
- L. Zadeh, Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic, Fuzzy Sets and Systems 90/2 (1997) 111-127.
- H.-J. Zimmermann, Fuzzy set theory and its applications (Kluwer, Dordrecht, 1991).

Redes Neuronais

Método Repropagação

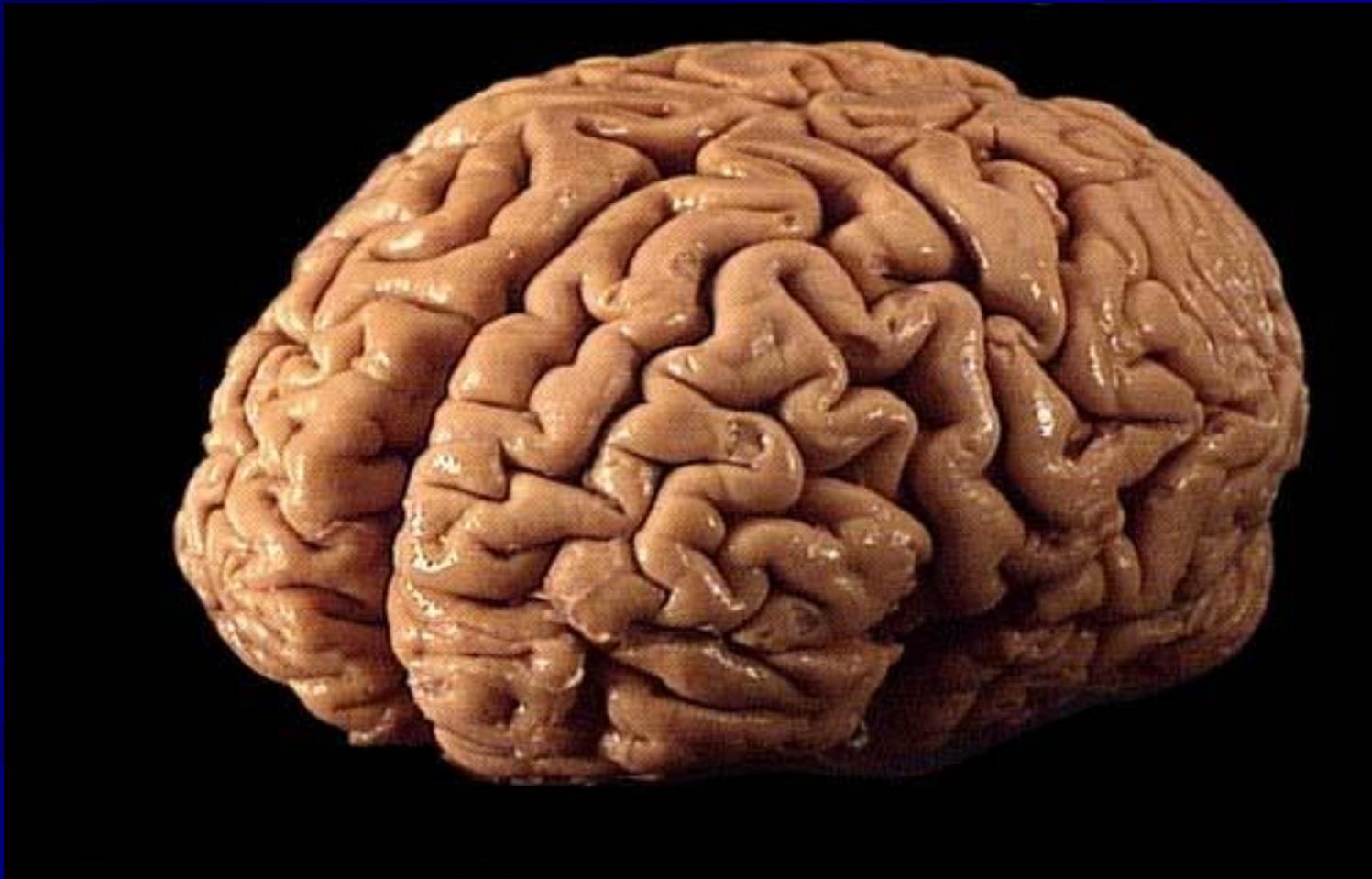
Paulo Salgado

UTAD

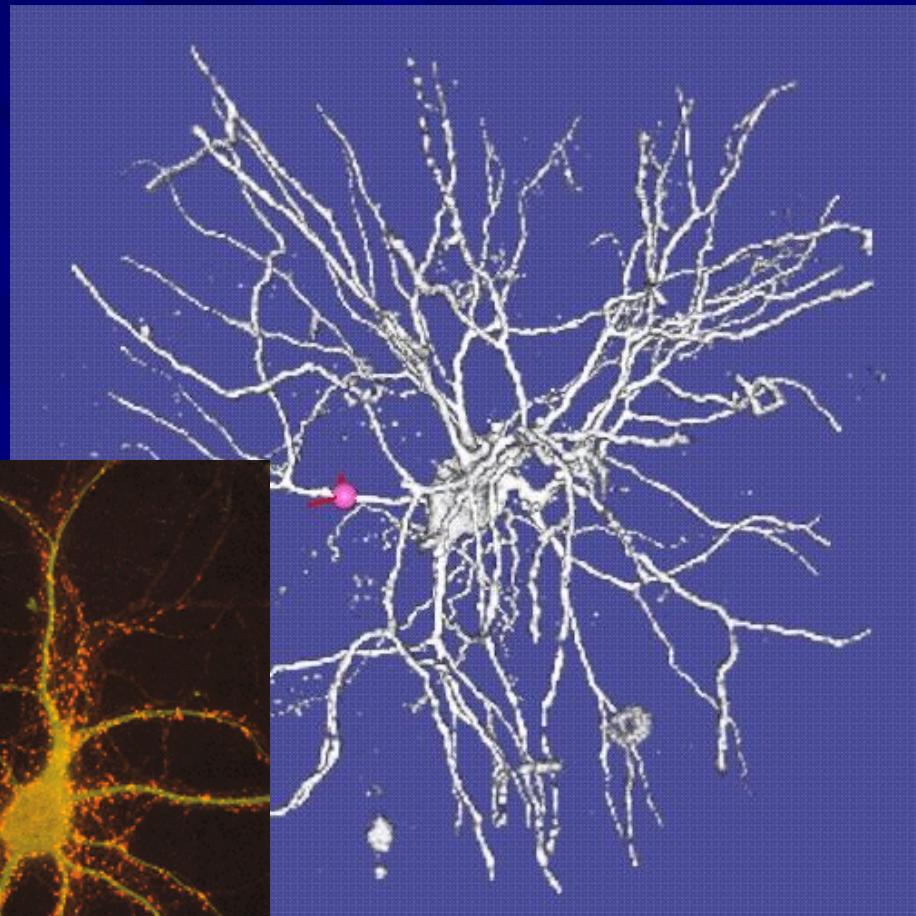
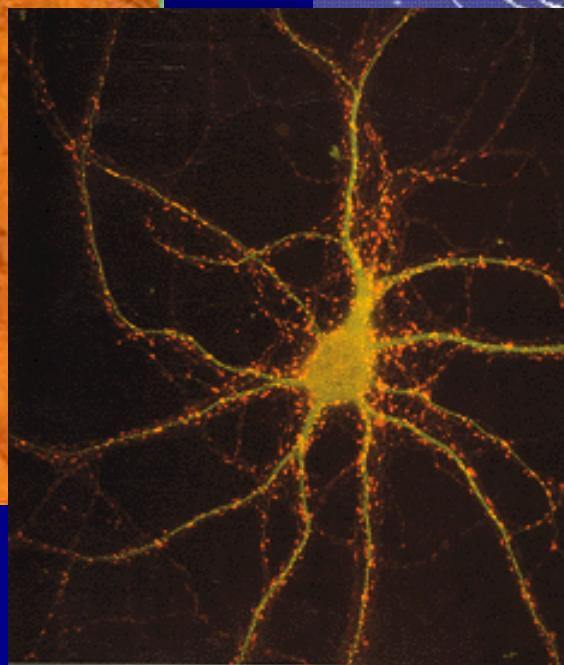
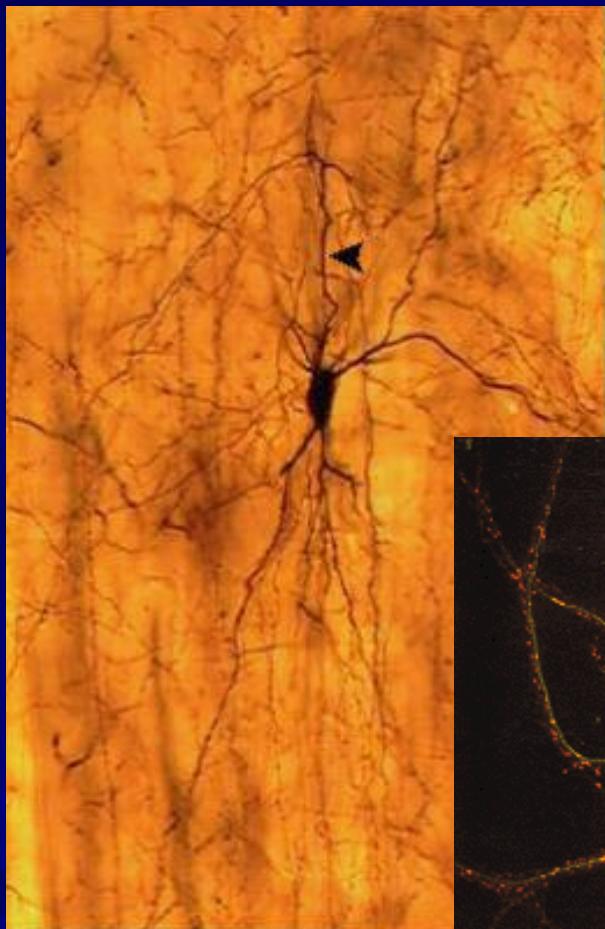
Índice

- Perceptrons
- Aprendizagem
- Redes Multi-camada.
- Método de Backpropagation
- Bias, Overfitting e 1^a paragem
- (Exemplos: Reconhecimento facial)

Cérebro Humano



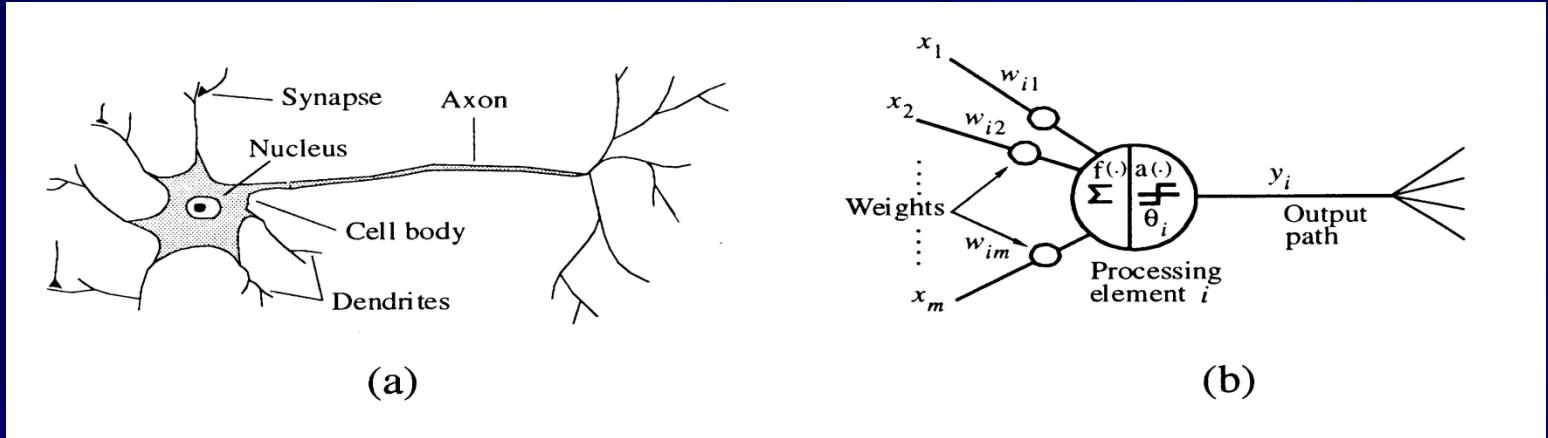
Neurónios



Aprendizagem Humana

- Numero de neurónios: $\sim 10^{11}$
- Ligações por neurónio: $\sim 10^4$ to 10^5
- Tempo de processamento: ~ 0.001 segundo
- Tempo de reconhecimento de uma cena: ~ 0.1 segundo

100 etapas de inferência não parecem ser muitas!



Partes constituintes do Neurónio:

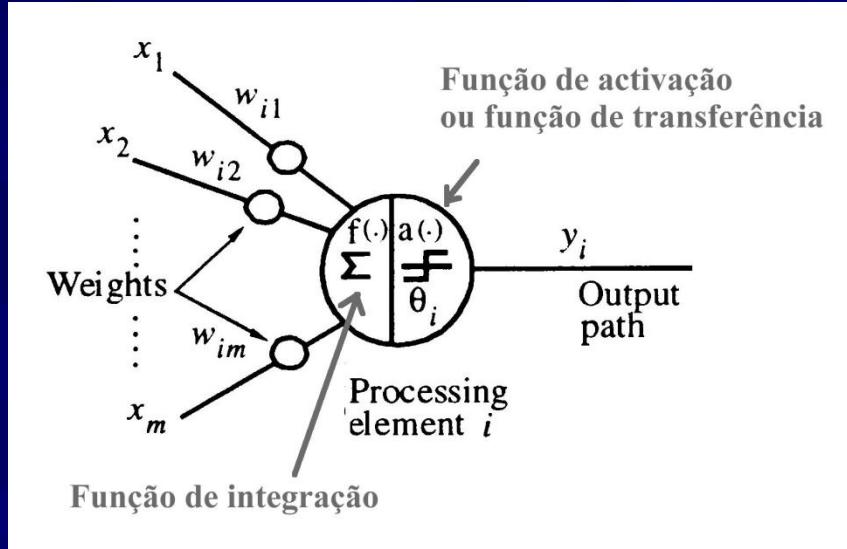
- Núcleo da célula (*cell body*) ou soma.
- *Dendrites*
- *Axon*
- Sinapses (são *excitadores/inibidores* que deixam passar impulsos que causam o disparo/inibição do neurónio receptor).

NN's são constituídas por uma grande número de elementos de processamento (nós ou unidades) fortemente interligadas, que usualmente operam em paralelo e estão configuradas numa arquitectura regular.

Em resumo, as NN é uma estrutura de processamento de informação **paralela** com as seguintes propriedades:

1. O seu modelo matemático é inspirada nos neurónios.
2. Consiste num largo número de elementos de processamento, cujas interligações são pesadas.
3. As suas ligações (pesadas) retêm o conhecimento.
4. Um *elemento de processamento* responde dinamicamente aos estímulos de entrada e a resposta depende unicamente da informação em si localizada; isto é, os sinais de entrada chegam aos elemento de processamento através das ligações e dos seus pesos.
5. Possui a habilidade de aprender, relembrar-se e generalizar a partir dos dados de treino, pelo assinalar ou ajuste dos pesos das ligações.
6. Apenas de forma colectiva demonstra poder computacional. Um único neurónio não contém informação específica (propriedade \Rightarrow *representação distribuída*).

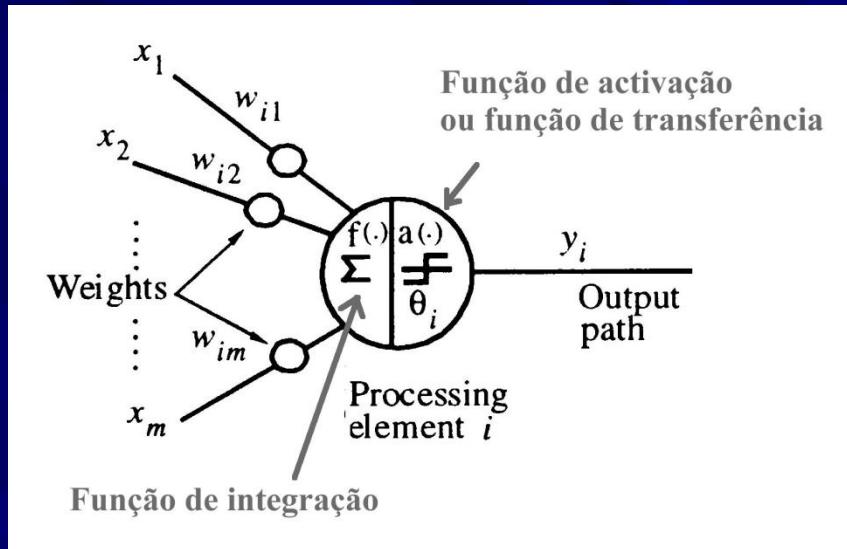
Elemento de processamento



Função de integração

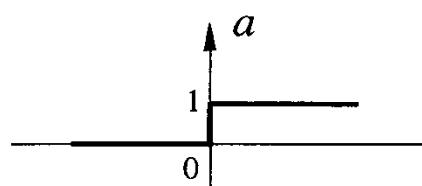
Função de integração linear	Função esférica
$f_i \triangleq net_i = \sum_{j=1}^m w_{ij} \cdot x_j - \theta_i$	$f_i = \rho^{-2} \sum_{j=1}^m (x_j - w_{ij})^2 - \theta_i$ $\rho \rightarrow \text{raio} ; w_{ij} \rightarrow \text{centro da esfera.}$
Função quadrática	Função polinomial ou <i>sigma-pi</i> ($\Sigma\Pi$)
$f_i = \sum_{j=1}^m w_{ij} \cdot x_j^2 - \theta_i$	$f_i = \sum_{j=1}^m \sum_{k=1}^m w_{ijk} x_j x_k + x_j^{\alpha_j} + x_k^{\alpha_k} - \theta_i$

Elemento de processamento (EP)

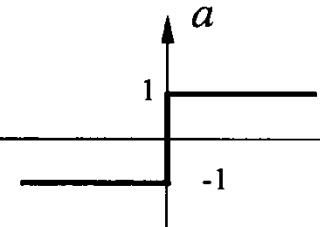


Função de activação

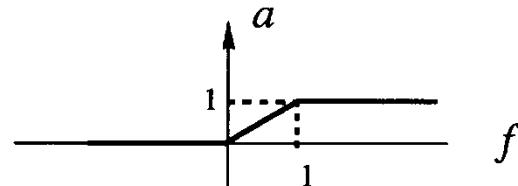
Função degrau	Hard limiter (função threshold)	Função rampa
$a(f) = \begin{cases} 1 & ; \text{ se } f \geq 0 \\ 0 & ; \text{ se } f < 0 \end{cases}$	$a(f) = \text{sgn}(f) = \begin{cases} 1 & ; \text{ se } f \geq 0 \\ -1 & ; \text{ se } f < 0 \end{cases}$	$a(f) = \begin{cases} 1 & ; \text{ se } f \geq 1 \\ f & ; \text{ se } 0 \leq f < 1 \\ 0 & ; \text{ se } f < 0 \end{cases}$
Função sigmoid unipolar	Função sigmoid bipolar	
$a(f) = \frac{1}{1 + e^{-\lambda f}}$	$a(f) = \frac{2}{1 + e^{-\lambda f}} - 1$	



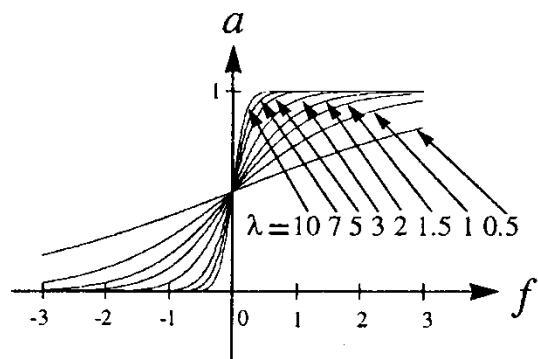
(a)



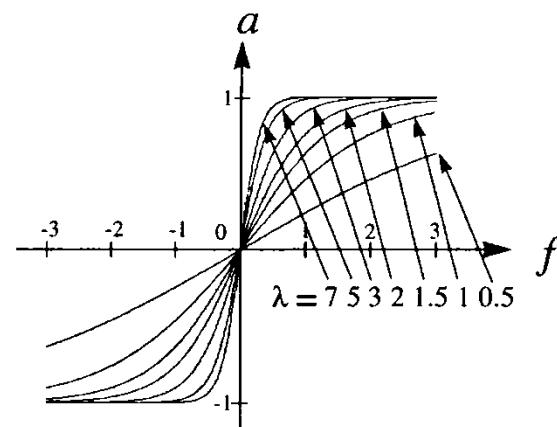
(b)



(c)



(d)



(e)

Ligações/Estrutura

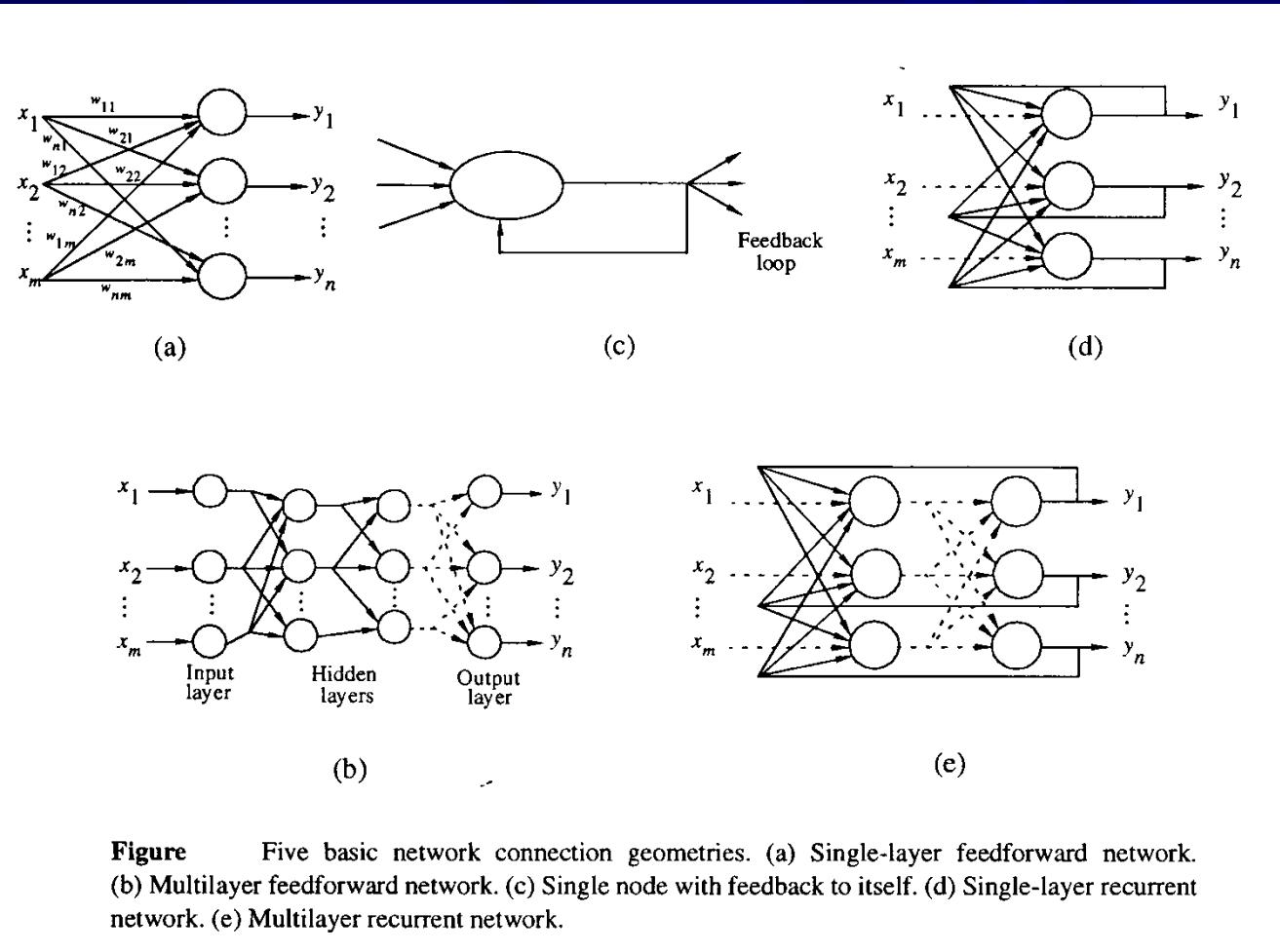


Figure Five basic network connection geometries. (a) Single-layer feedforward network. (b) Multilayer feedforward network. (c) Single node with feedback to itself. (d) Single-layer recurrent network. (e) Multilayer recurrent network.

Regras de aprendizagem



n EP - cada EP com m pesos

Matriz de pesos ou matriz de ligações

em que

$$\mathbf{W} \triangleq \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_n^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & & w_{nm} \end{bmatrix}$$

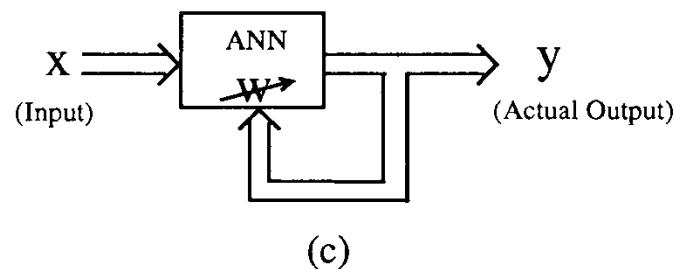
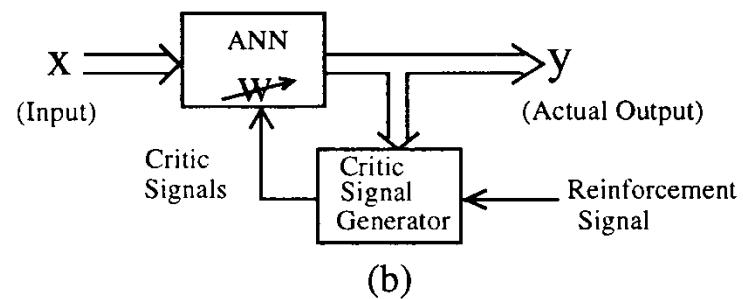
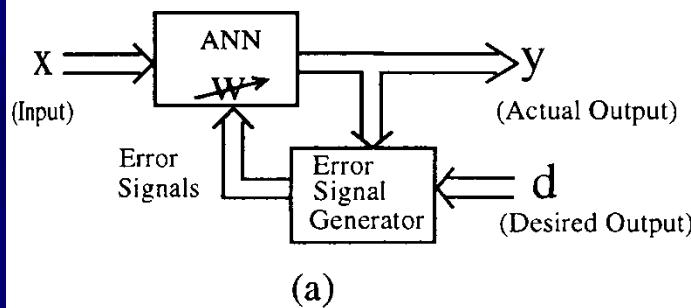
$w_{ij} \rightarrow$ peso da ligação entre do j^{esimo} EP (nó fonte) para o i^{esimo} EP (nó destino).

Aprendizagem paramétrica

(a) Supervisionada

(b) reforçada
(reinforcement learning)

(c) Não supervisionada



Regra de aprendizagem supervisionada

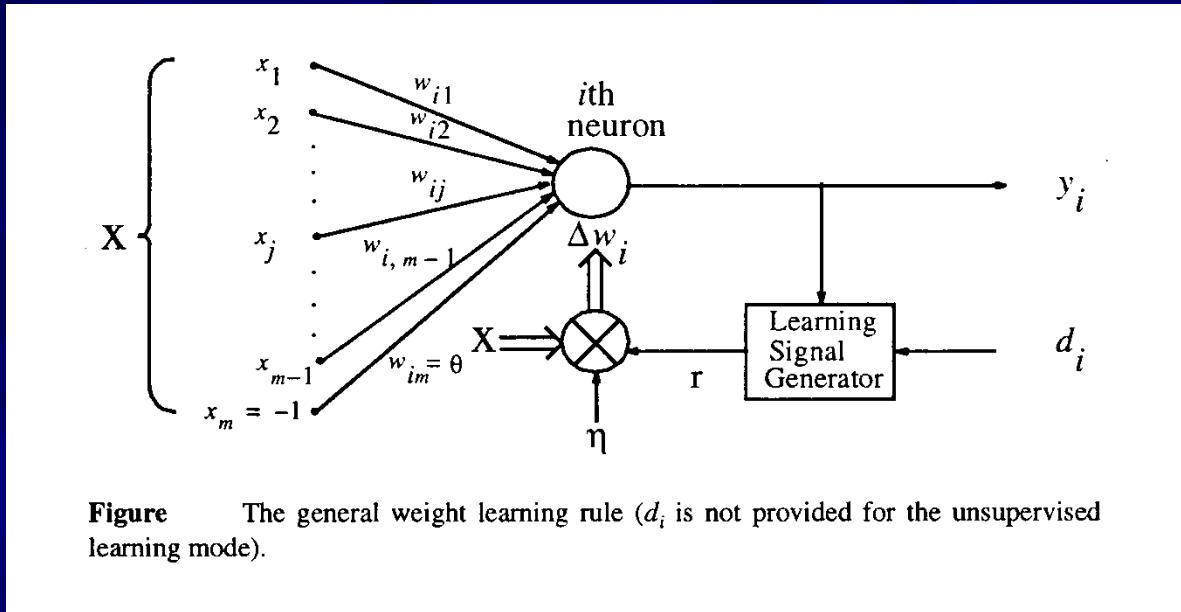


Figure The general weight learning rule (d_i is not provided for the unsupervised learning mode).

$$r = f_r(\mathbf{w}_i, \mathbf{x}, d_i) \quad \text{Sinal de supervisão}$$

$$\frac{d\mathbf{w}_i(t)}{dt} = \eta \cdot r \cdot \mathbf{x}(t) \quad \mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} + \eta \cdot r \cdot \mathbf{x}^{(t)}$$

Perceptron

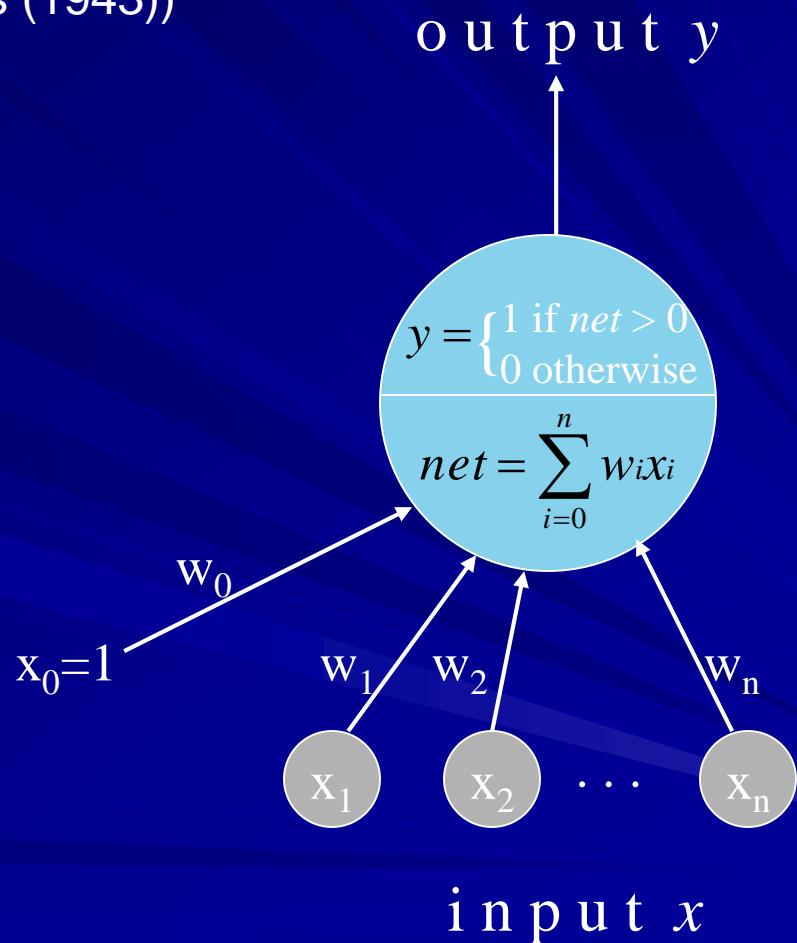
Modelo Matemático (McCulloch e Pitts (1943))

$$y_i(t+1) = a \left(\sum_{j=1}^m w_{ij} \cdot x_j(t) - \theta_i \right)$$



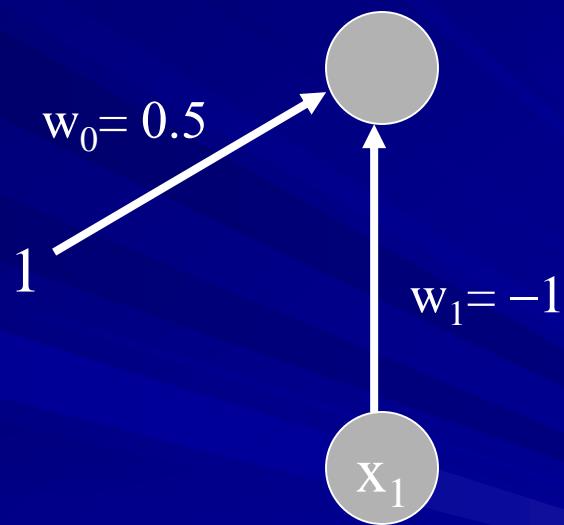
$$a(\text{net}) = \begin{cases} 1 & ; \text{ se } \text{net} \geq 0 \\ 0 & ; \text{ outros} \end{cases}$$

Pode realizar operações lógicas elementares como :
NOT, OR e AND.



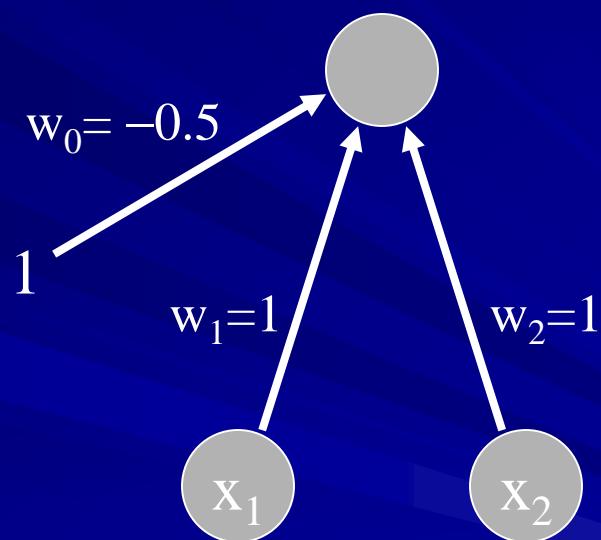
Negação

input x1	output
0	1
1	0



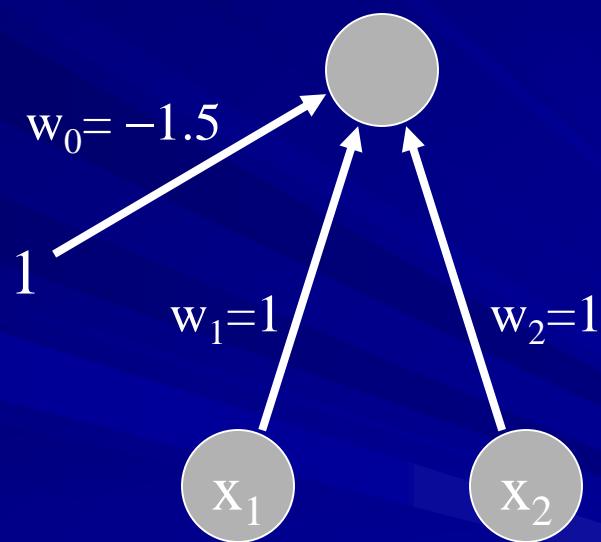
OR

input x1	input x2	output
0	0	0
0	1	1
1	0	1
1	1	1



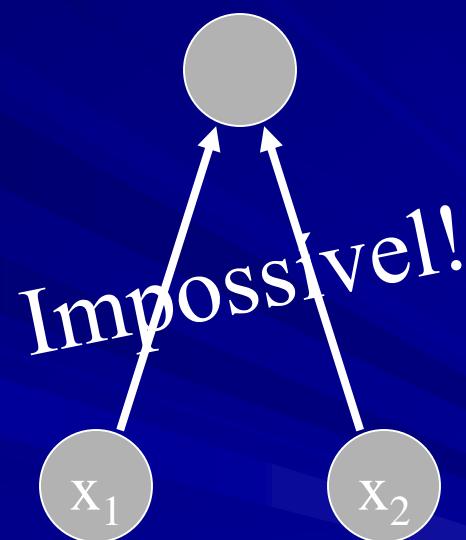
AND

input x1	input x2	output
0	0	0
0	1	0
1	0	0
1	1	1

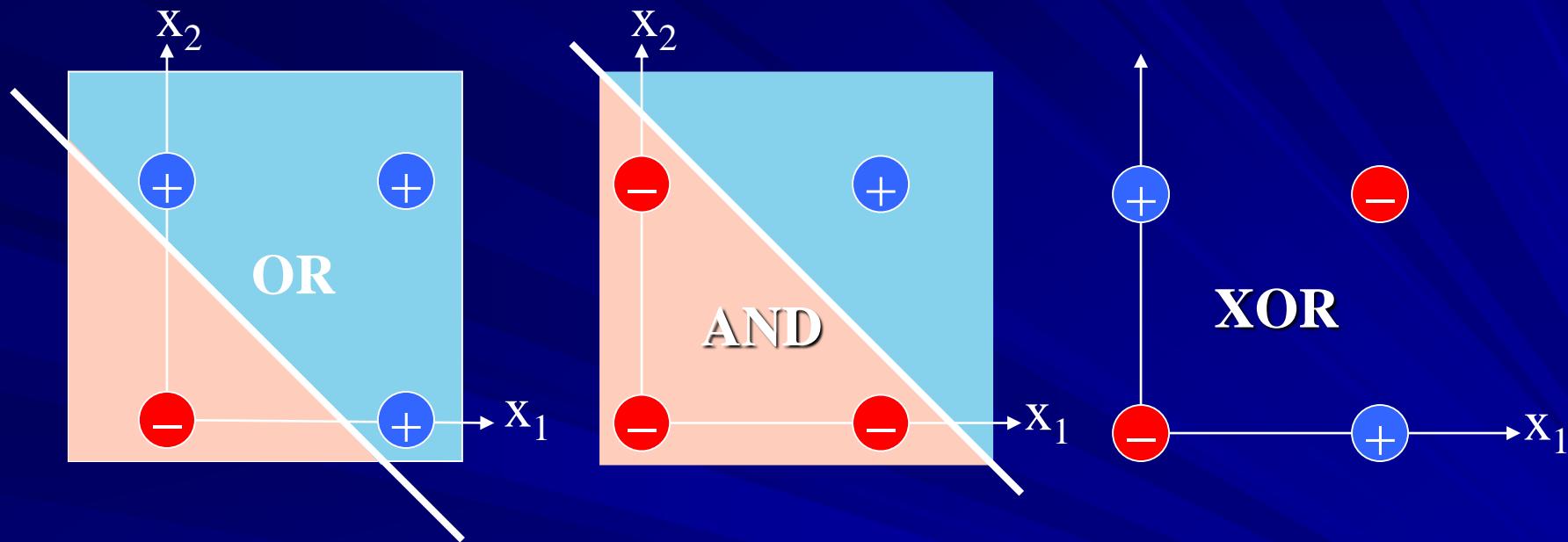


XOR

input x1	input x2	output
0	0	0
0	1	1
1	0	1
1	1	0



Separabilidade linear



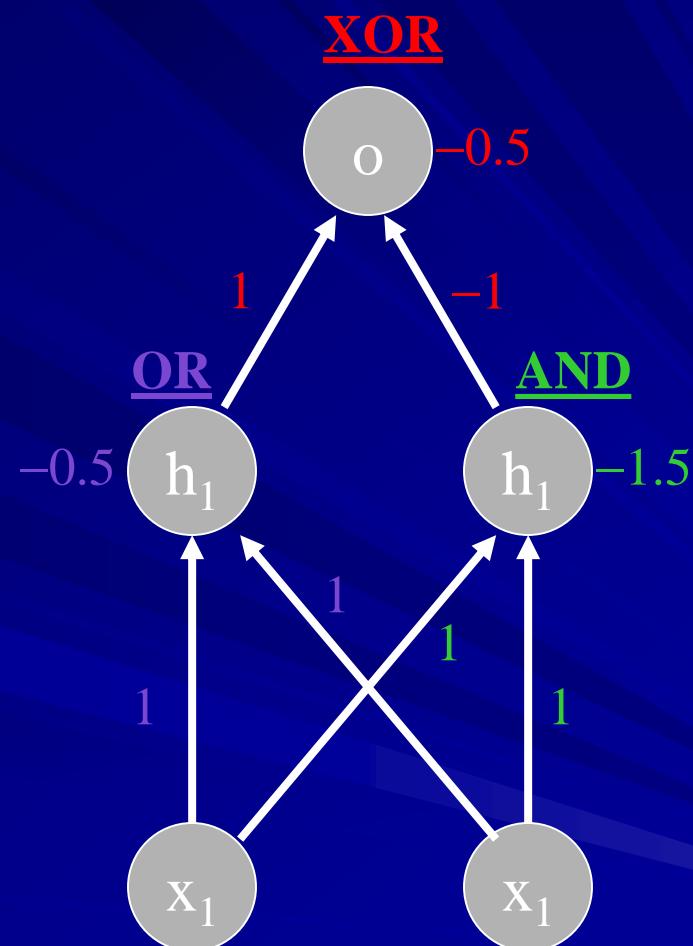
O problema de separação linear por um *perceptron* só é realizável se for possível encontrar os vectores de pesos \mathbf{w}_i , $i=1,2, \dots, n$, tais que:

$$\mathbf{w}_i^T \mathbf{x} > 0 \quad \text{para cada } \mathbf{x} \text{ com saída desejada} = +1$$

$$\mathbf{w}_i^T \mathbf{x} < 0 \quad \text{para cada } \mathbf{x} \text{ com saída desejada} = -1$$

XOR

input x1	input x2	output
0	0	0
0	1	1
1	0	1
1	1	0



Regra de aprendizagem do Perceptron

$$w_i \leftarrow w_i + \Delta w_i$$

$$r \triangleq d_i - y_i$$

saída do perceptron

saída desejada

$$\Delta w_i = \eta \ r \ x_i$$

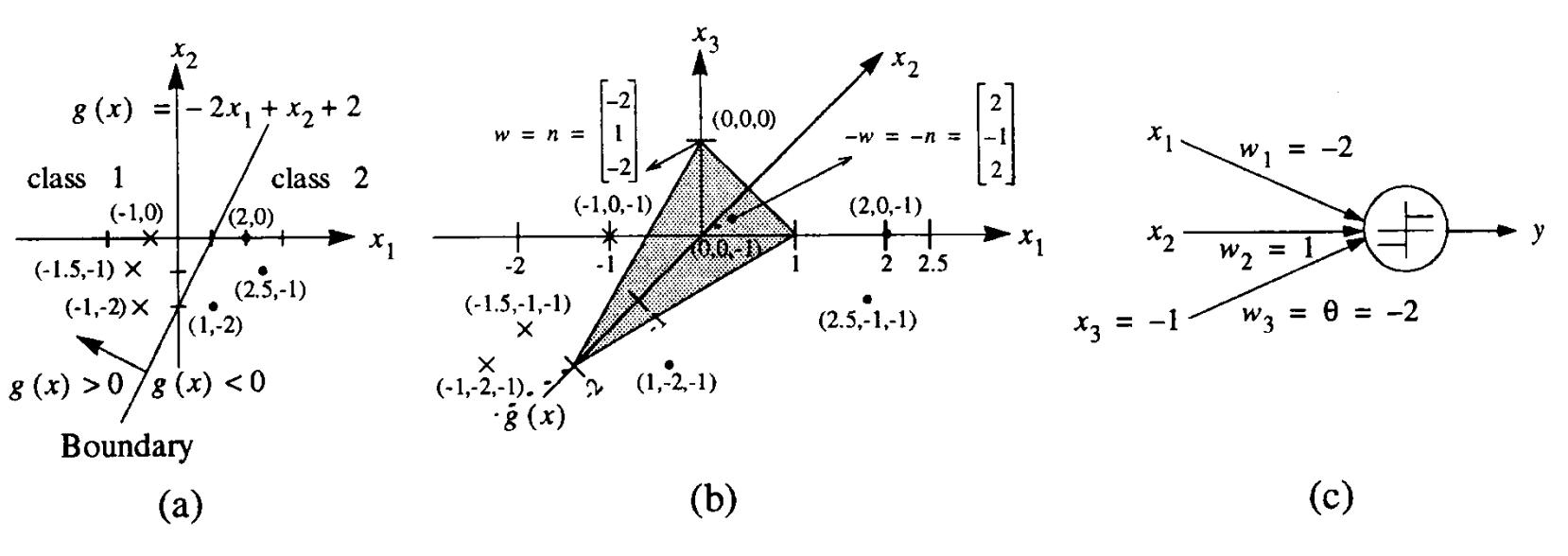
incremento step size entrada
 Sinal de erro

$$\Delta w_{ij} = \eta \cdot [d_i - \text{sgn}(w_i^T x)] \cdot x_j = \begin{cases} 2\eta d_i x_j & \text{se } y_i \neq d_i \\ 0 & \text{outros} \end{cases}$$

Exemplo:

$$\left\{ \begin{bmatrix} -1, 0 \end{bmatrix}^T, \begin{bmatrix} -1.5, -1 \end{bmatrix}^T, \begin{bmatrix} -1, -2 \end{bmatrix}^T \right\} : \text{Classe 1}$$

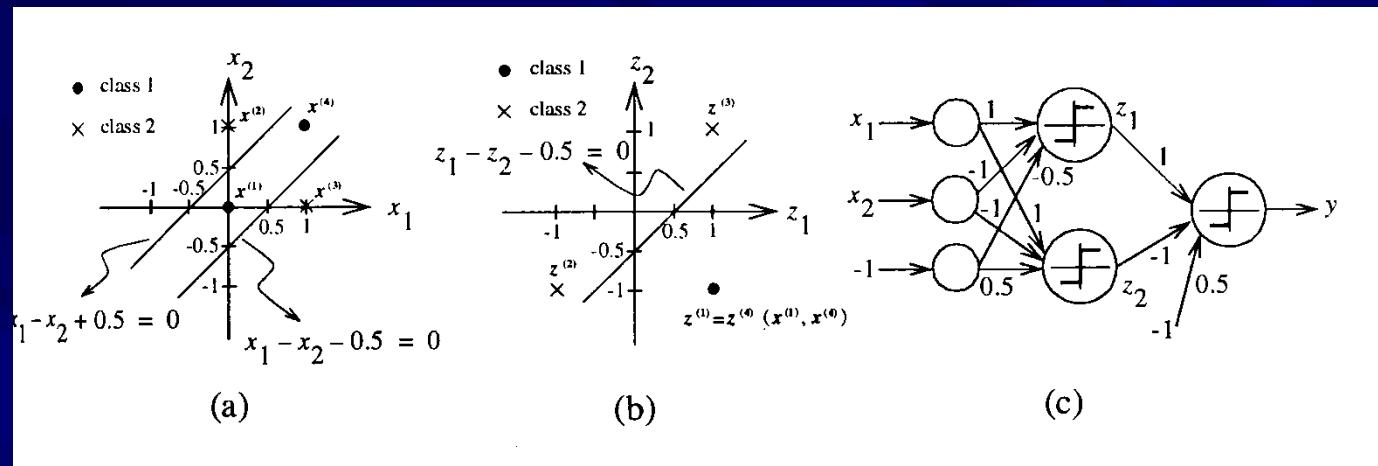
$$\left\{ \begin{bmatrix} 2, 0 \end{bmatrix}^T, \begin{bmatrix} 2.5, -1 \end{bmatrix}^T, \begin{bmatrix} 1, -2 \end{bmatrix}^T \right\} : \text{Classe 2}$$



Convergência, Se...

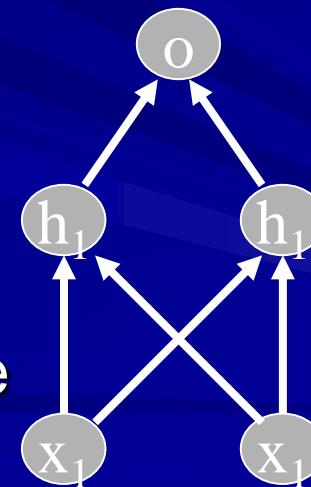
- ... os dados de treino forem linearmente separáveis
- ... tamanho do passo η suficientemente pequeno
- ... sem unidades escondidas (“hidden”)

Redes neurais *feedforward* multicamada

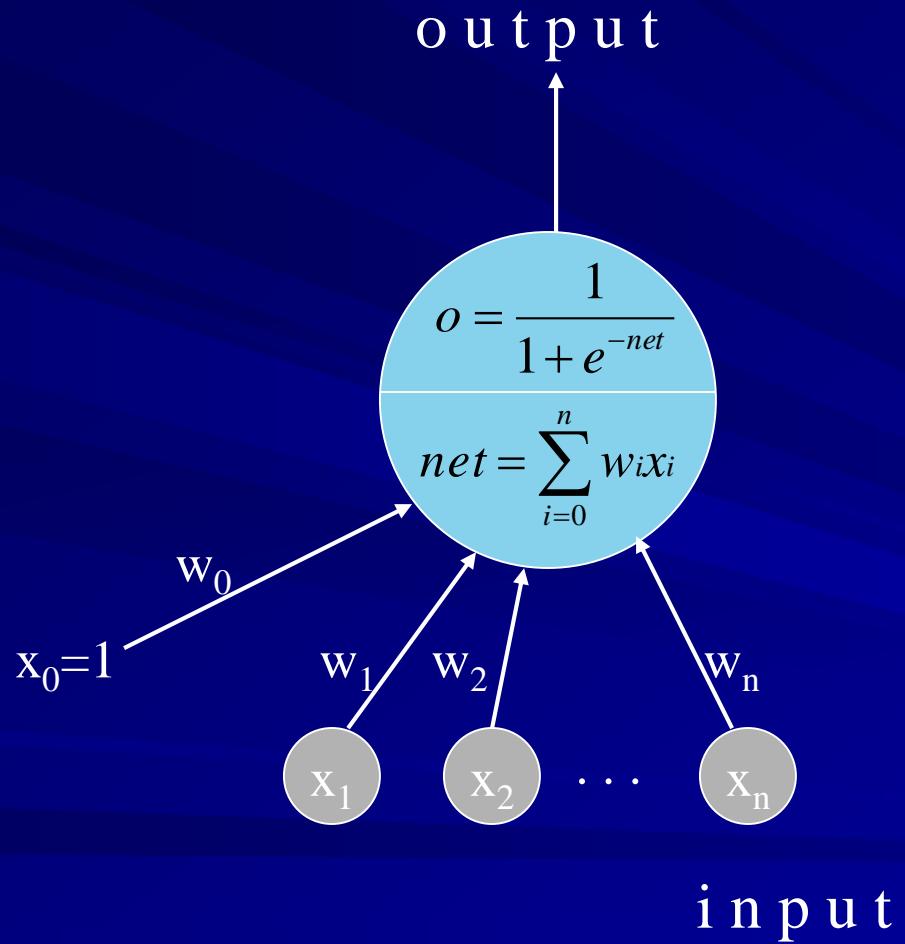


Como treinar redes
Multi-Layer Perceptrons?

- Método do gradiente descendente

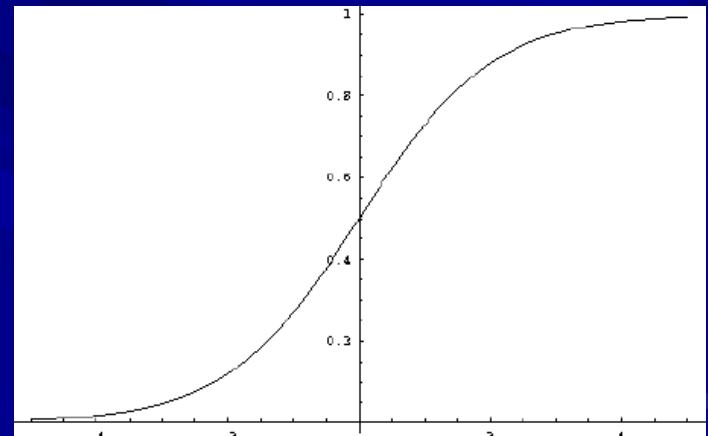


Função Sigmoidal



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Método de Gradiente Descendente

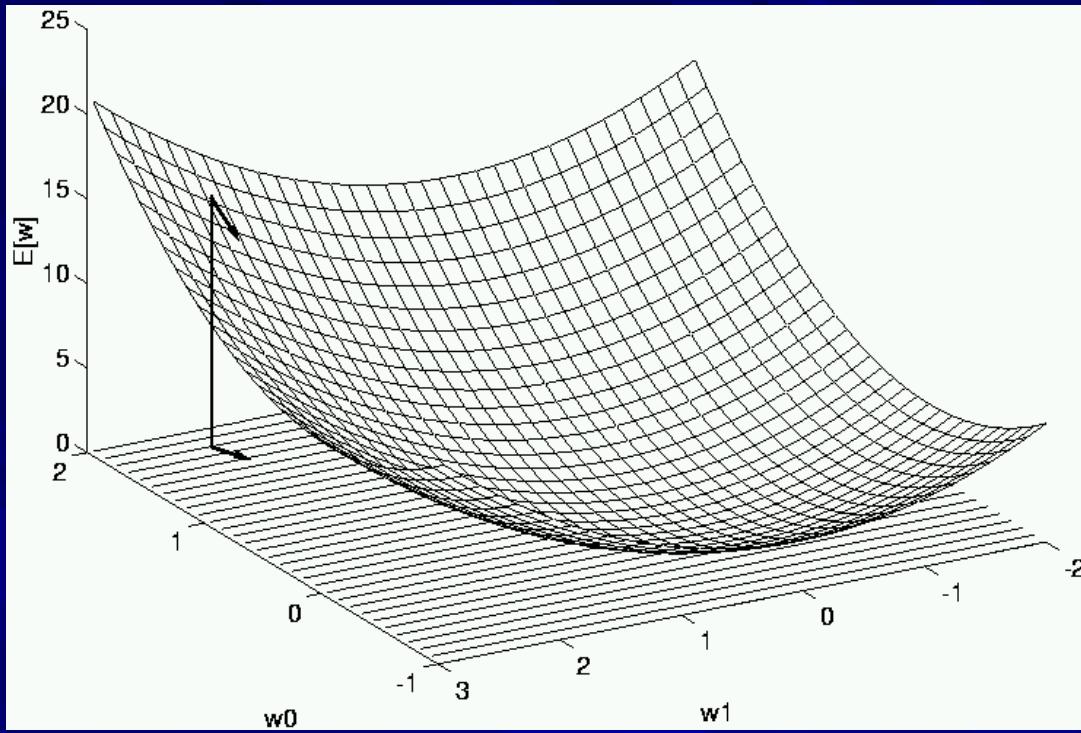
- Adaptar os pesos w_i que minimiza a soma dos erros quadráticos

$$E[\vec{w}] = \frac{1}{2} \sum_{k \in D} (d_k - y_k)^2$$



$D = \text{Dados de treino}$

Gradiente Descendente



Gradiente:

$$\nabla E[\vec{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Regra de treino: $\Delta \vec{w} = -\eta \nabla E[\vec{w}]$ $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

Gradiente Descente (uma camada)

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_k (d_k - y_k)^2 \\ &= \frac{1}{2} \sum_k \frac{\partial}{\partial w_i} (d_k - y_k)^2 \\ &= \frac{1}{2} \sum_k 2(d_k - y_k) \frac{\partial}{\partial w_i} (d_k - y_k) \\ &= \sum_k (d_k - y_k) \frac{\partial}{\partial w_i} (d_k - \sigma(\vec{w} \cdot \vec{x}_k)) \\ &= \sum_k (d_k - y_k) \left(-\frac{\partial}{\partial w_i} \sigma(\vec{w} \cdot \vec{x}_k) \right) \\ &= -\sum_k (d_k - y_k) \sigma(\vec{w} \cdot \vec{x}_k) (1 - \sigma(\vec{w} \cdot \vec{x}_k)) x_{i,k}\end{aligned}$$

Batch Learning

- Iniciar cada peso w_i com um pequeno valor aleatório
- Repetir até : $\Delta w_i = 0$

Para cada exemplo de treino k fazer

$$y_k \leftarrow \sigma(\sum_i w_i x_{i,k})$$

$$\Delta w_i \leftarrow \Delta w_i + \eta (d_k - y_k) \sigma_k (1 - \sigma_k) x_{i,k}$$

$$w_i \leftarrow w_i + \Delta w_i$$

Aprendizagem Incremental (Online)

- Iniciar cada peso w_i com um pequeno valor aleatório
- Repetir até : $\Delta w_i = 0$

Para cada exemplo de treino k fazer

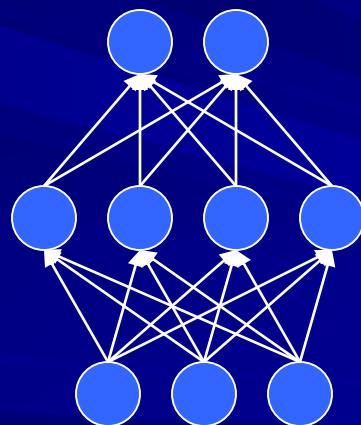
$$o_d \leftarrow \sum_i w_i x_{i,k}$$

$$\Delta w_i \leftarrow \Delta w_i + \eta (d_k - y_k) \sigma_k (1 - \sigma_k) x_{i,k}$$

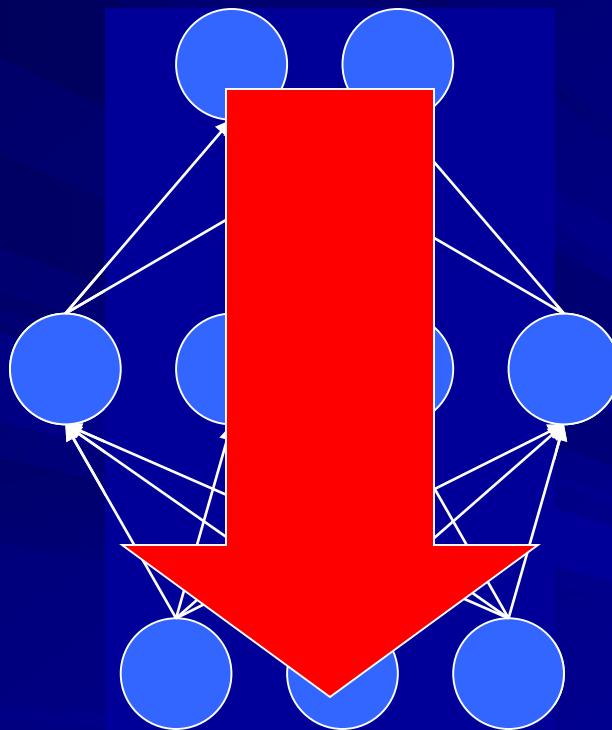
$$w_i \leftarrow w_i + \Delta w_i$$

Algoritmo de Retropropagação

- Generalização para múltiplas camadas e múltiplas unidades de saída.



Backpropagation Algorithm



“activações”

“erros”

Dados um par de dados entrada saída $(x^{(k)}, d^{(k)})$, o algoritmo de back-propagation realiza-se em duas fases:

1º O vector de entrada $x^{(k)}$ é propagado da camada de entrada para a de saída e, como resultado deste fluxo directo dos dados, é produzido a saída $y^{(k)}$.

2º O sinal de erro resultante da diferença entre $d^{(k)}$ e $y^{(k)}$ é *back-propagated* da camada de saída para as camadas precedentes, adaptando os seus pesos.

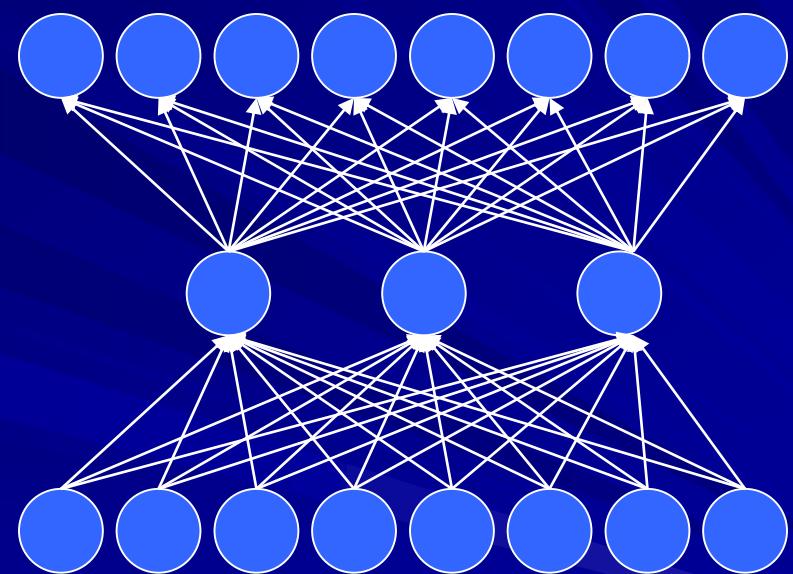
Algoritmo Backpropagation

- Iniciar os pesos com valores pequenos e aleatórios
- Para cada exemplo de treino fazer
 - Para cada unidade escondida h :
$$o_h = \sigma(\sum_i w_{hi} x_i)$$
 - Para cada unidade de saída k :
$$o_k = \sigma(\sum_h w_{kh} x_h)$$
 - Para cada unidade de saída k :
$$\delta_k = o_k (1 - o_k) (t_k - o_k)$$
 - Para cada unidade escondida h :
$$\delta_h = o_h (1 - o_h) \sum_k w_{hk} \delta_k$$
- Actualizar cada peso da rede w_{ij} :

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \quad \text{com} \quad \Delta w_{ij} = \eta \delta_j x_{ij}$$

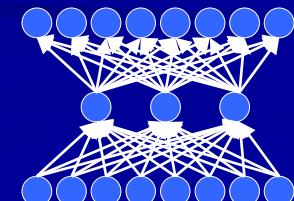
Exemplo de aprendizagem

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

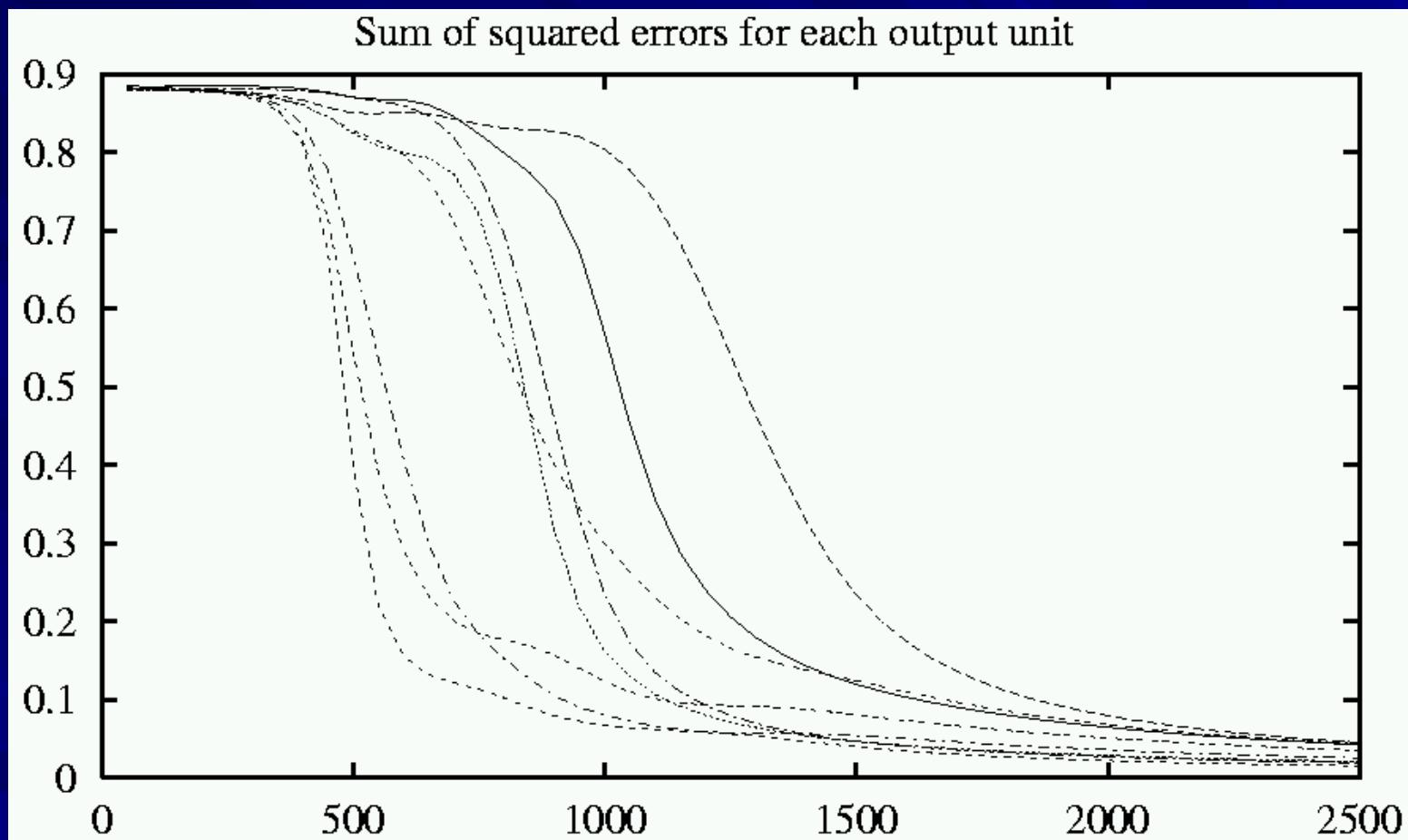


Aprendizagem das camadas escondidas

Input				Output
10000000	→	.89 .04 .08	→	10000000
01000000	→	.01 .11 .88	→	01000000
00100000	→	.01 .97 .27	→	00100000
00010000	→	.99 .97 .71	→	00010000
00001000	→	.03 .05 .02	→	00001000
00000100	→	.22 .99 .99	→	00000100
00000010	→	.80 .01 .98	→	00000010
00000001	→	.60 .94 .01	→	00000001

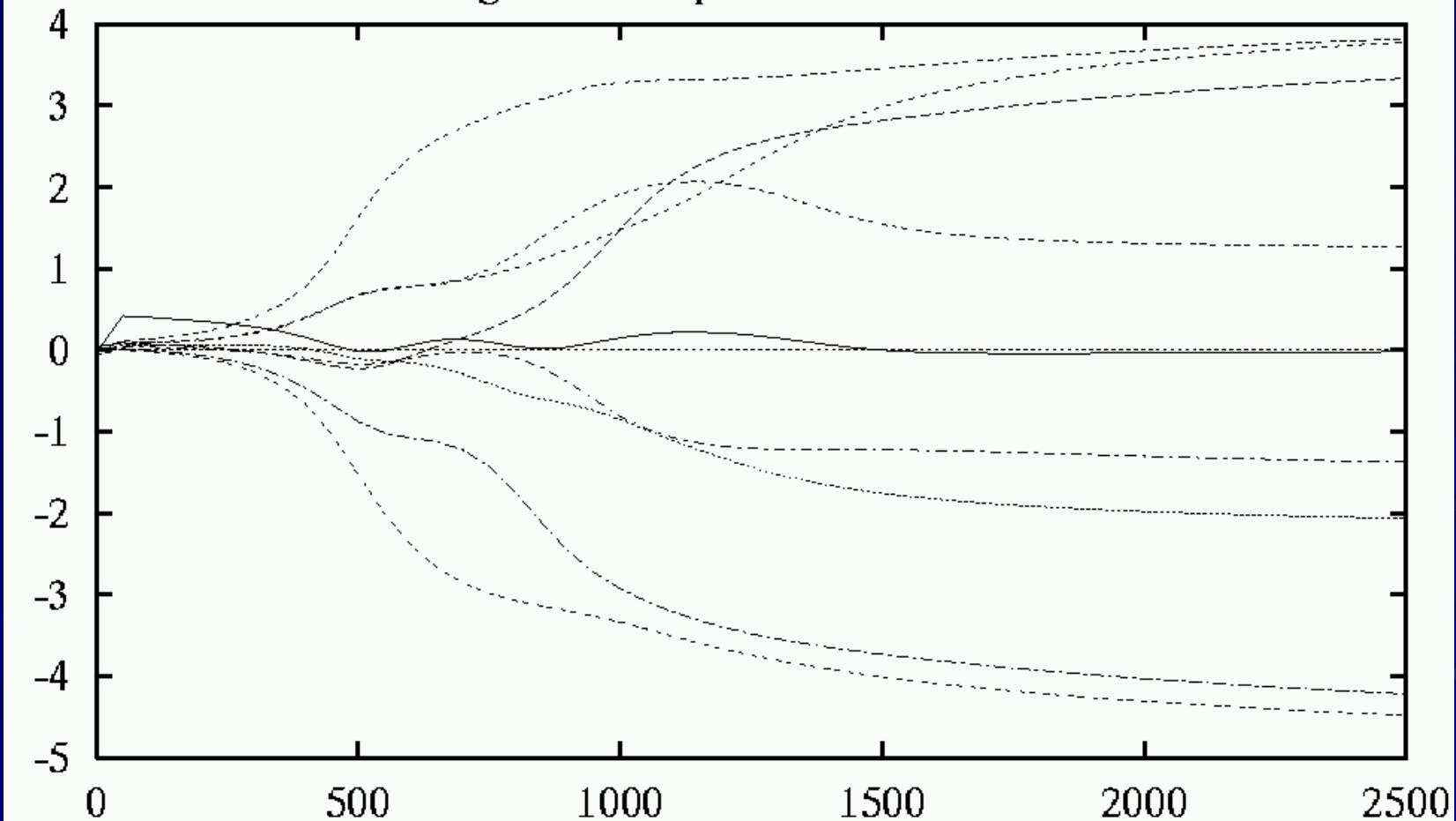


Erro de Treino



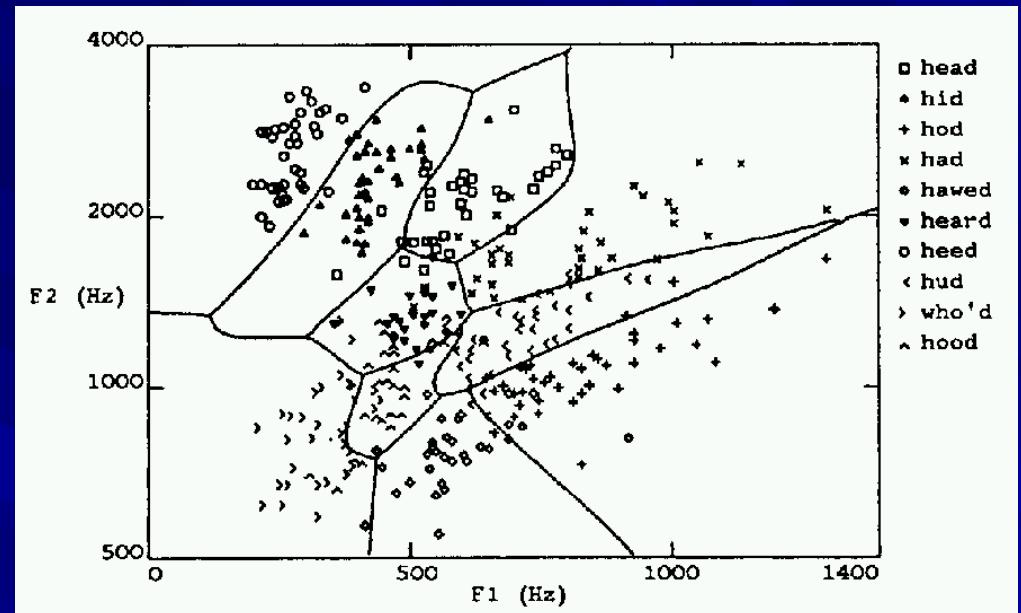
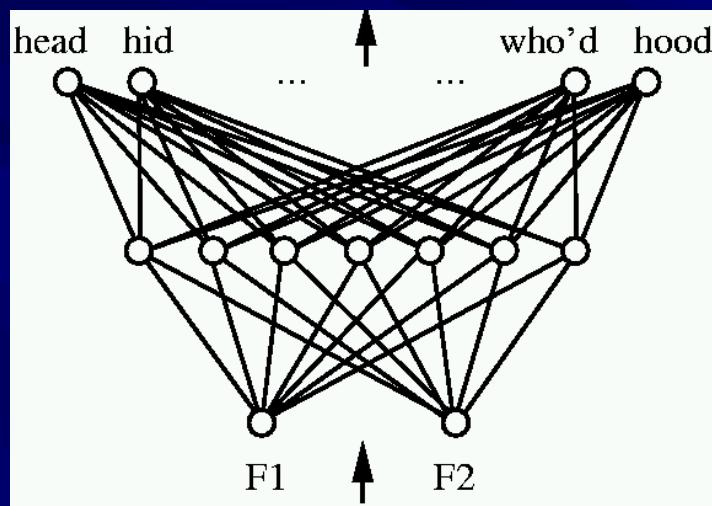
Pesos

Weights from inputs to one hidden unit



Teorema: A arquitectura de uma rede neuronal multicamada com pelo menos uma camada intermédia (*hidden*) usando funções de activação compacta e funções de integração linear ou polinomial pode virtualmente aproximar qualquer (medida de Borel) função de interesse a qualquer desejado grau de exactidão, desde que existam suficiente número de unidades nas camadas intermédias. A função $a: \mathfrak{M} \rightarrow [0,1]$ (ou $[-1,1]$) é uma função compacta se ela é não decrescente, $\lim_{\lambda \rightarrow \infty} a(\lambda) = 1$, e $\lim_{\lambda \rightarrow -\infty} a(\lambda) = 0$ (ou -1), em que λ é um parâmetro da função.

Reconhecimento de Voz

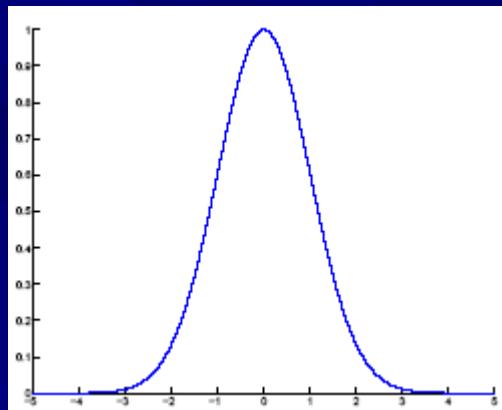


[Haung/Lippman 1988]

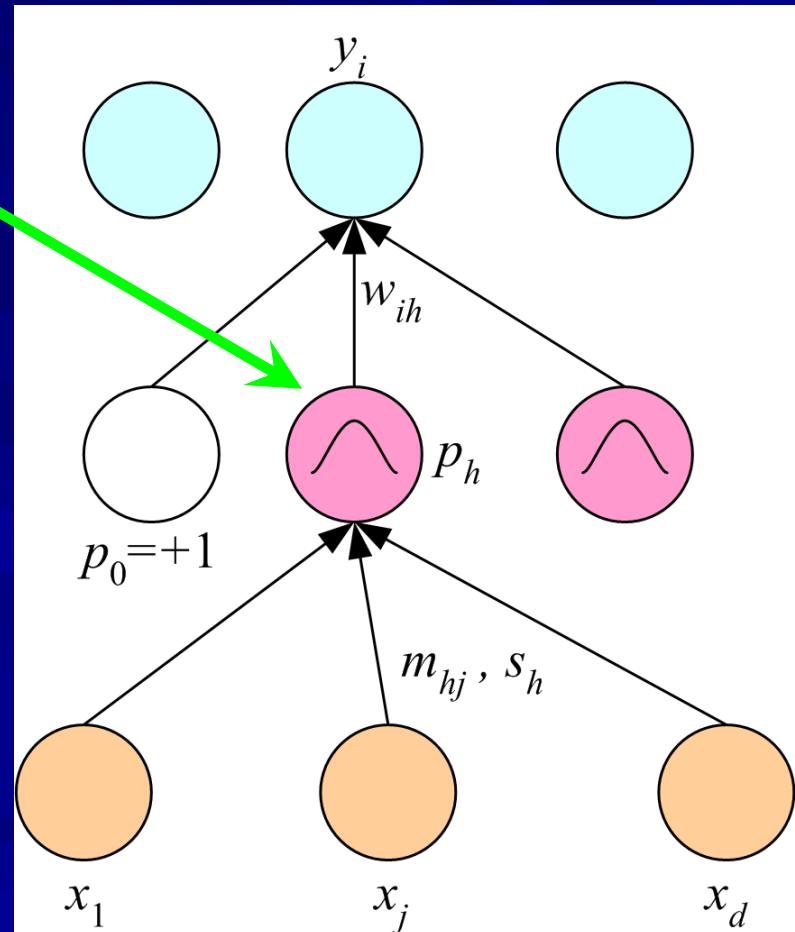
Radial-Basis Functions - RBF

■ Funções radiais

$$p_h^t = \exp\left[-\frac{\|\mathbf{x}^t - \mathbf{m}_h\|^2}{2s_h^2}\right]$$



$$y^t = \sum_{h=1}^H w_h p_h^t + w_0$$



Breve Descrição da Rede RBF

- É constituído por 3 camadas (*input, hidden, output*)
- A camada de entrada liga a rede ao exterior
- Na entrada de cada neurónio (*hidden layer*), é calculada a distância entre o centro do neurónio e o vector de entrada
- A camada interna são constituídos por neurónios com funções radiais.
- A camada de saída é linear. Fornece a resposta da rede agregando as saídas dos neurónios da camada intermédia.

Treino da RBF

- Aprendizagem híbrida:
 - Centros e desvios da primeira camada:
Método k -means não supervisionado
 - Pesos da segunda camada:
Método supervisionado (gradiente-descente ou RLS)
- Totalmente supervisionado
- (Broomhead and Lowe, 1988; Moody e Darken, 1989)

Método de Gradiente descendente

$$E\left(\left\{\mathbf{m}_h, s_h, w_{ih}\right\}_{i,h} | \times\right) = \frac{1}{2} \sum_t \sum_i (r_i^t - y_i^t)^2$$

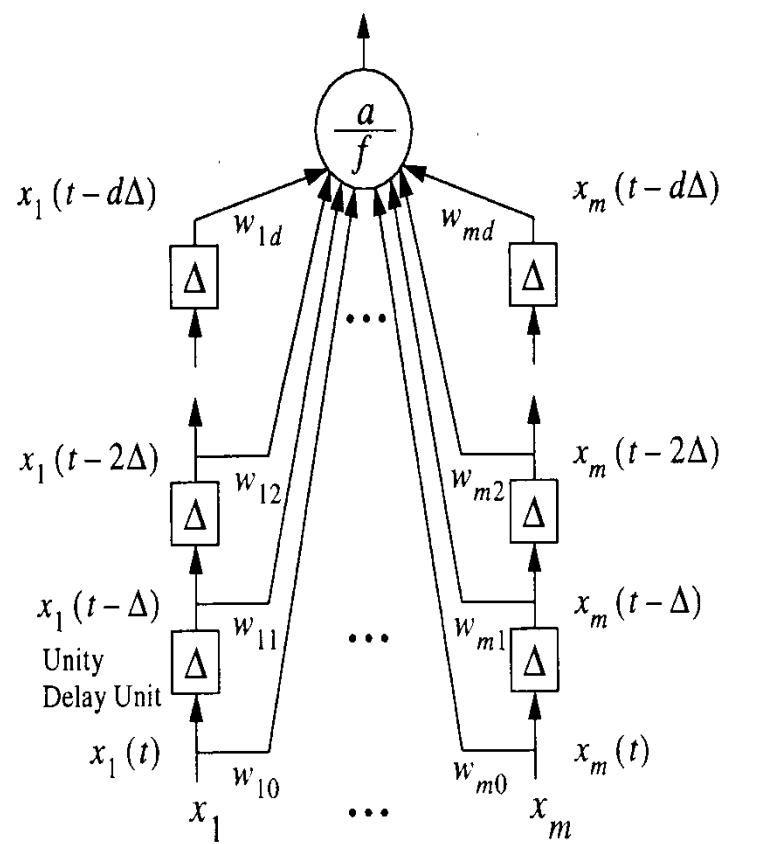
$$y_i^t = \sum_{h=1}^H w_{ih} p_h^t + w_{i0}$$

$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_i^t) p_h^t$$

$$\Delta m_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) w_{ih} \right] p_h^t \frac{(x_j^t - m_{hj})}{s_h^2}$$

$$\Delta s_h = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) w_{ih} \right] p_h^t \frac{\|\mathbf{x}^t - \mathbf{m}_h\|^2}{s_h^3}$$

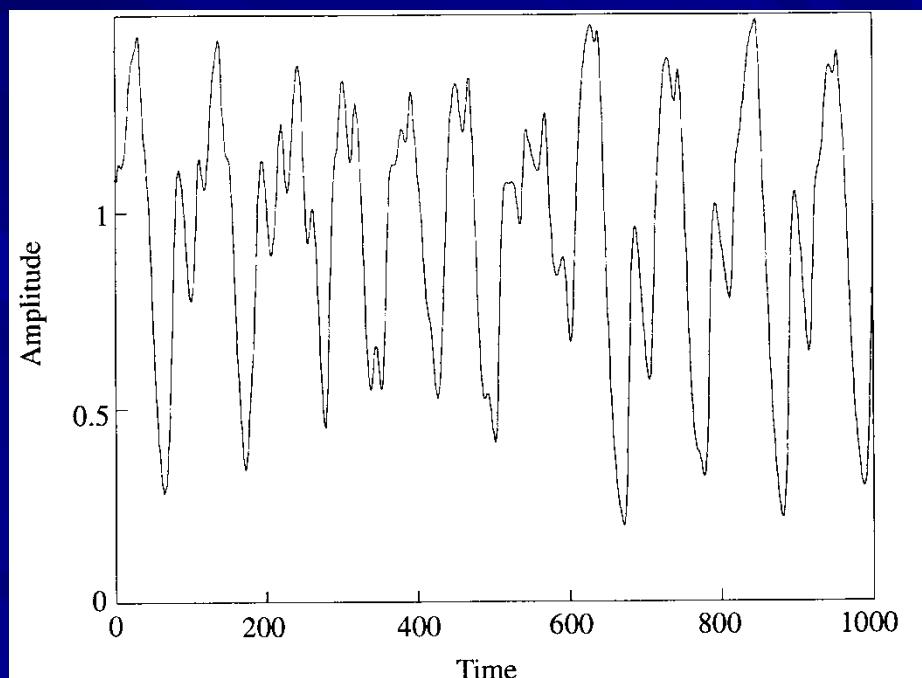
Redes neurais com atrasos temporais



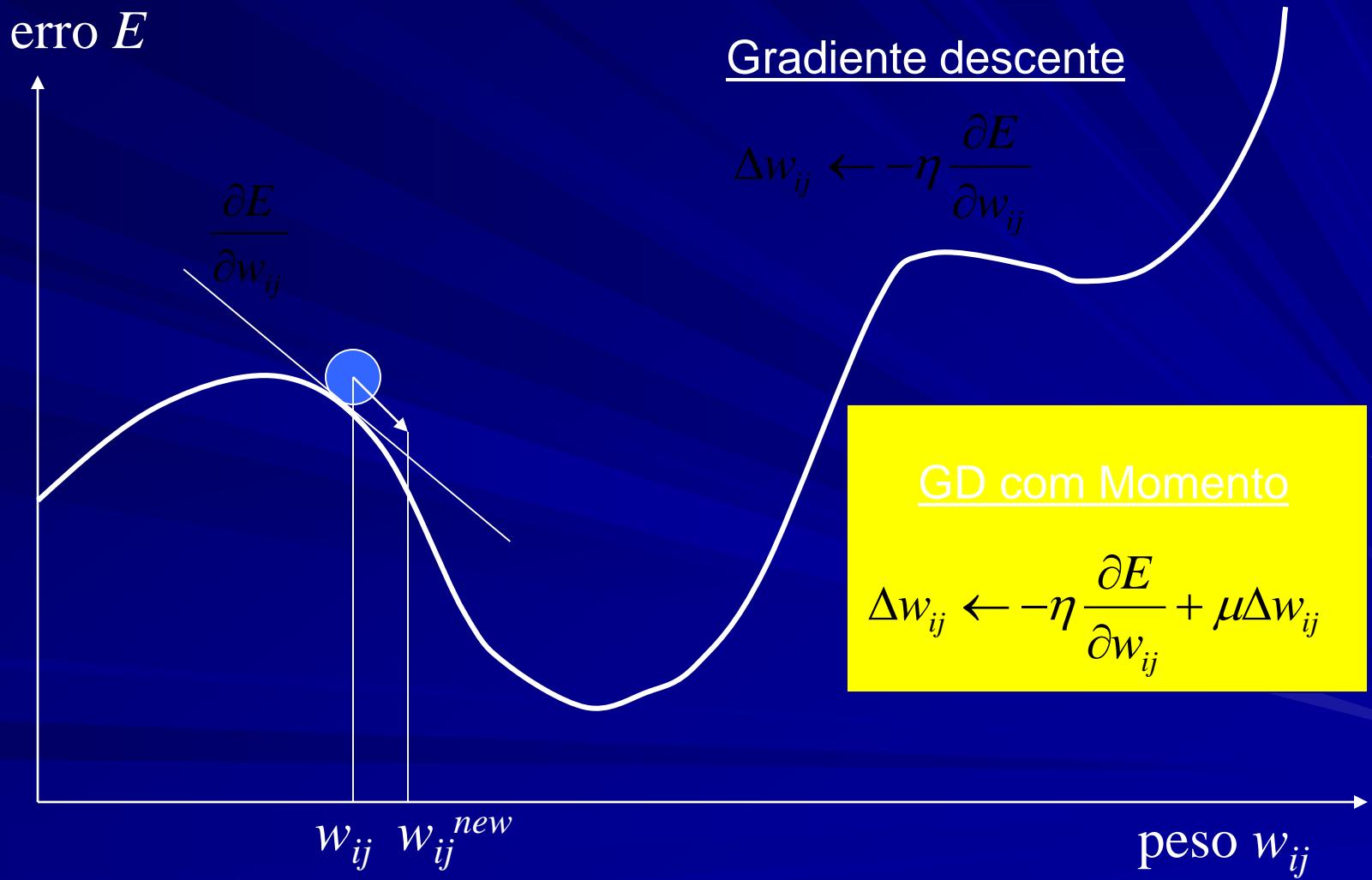
Série caótica de Mackey-Glass

$$\frac{\partial x(t)}{\partial t} = \frac{a \cdot x(t-\tau)}{1 + x^{10}(t-\tau)} - b \cdot x(t)$$

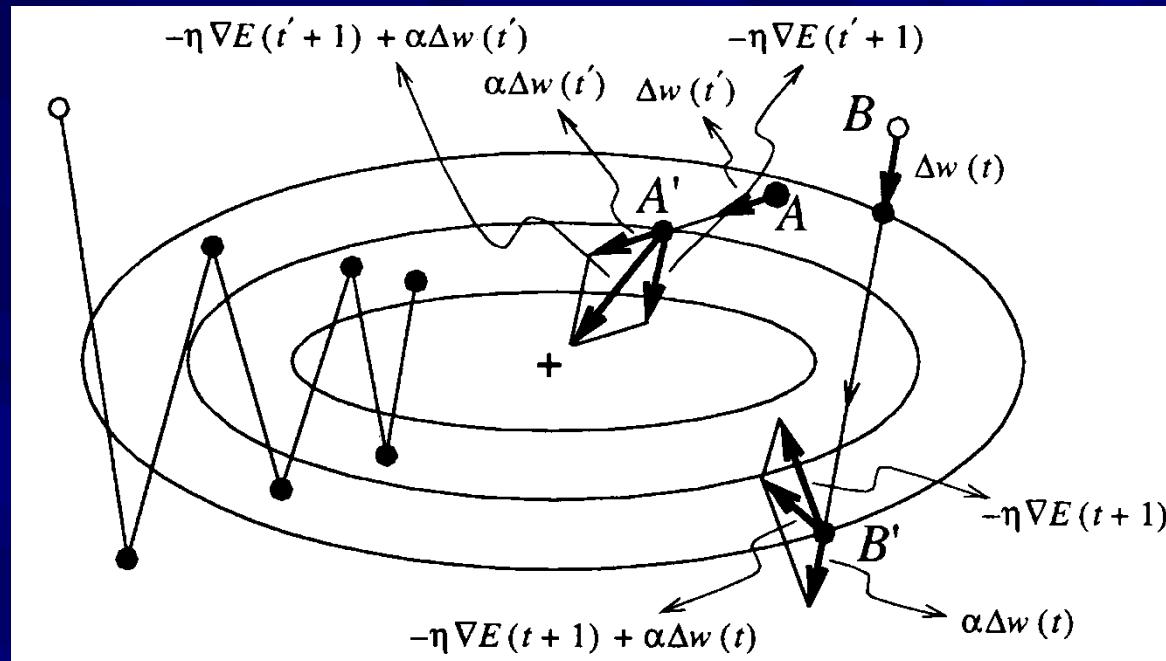
$a=0.2$, $b=0.1$ e $\tau=17$



GD com momento

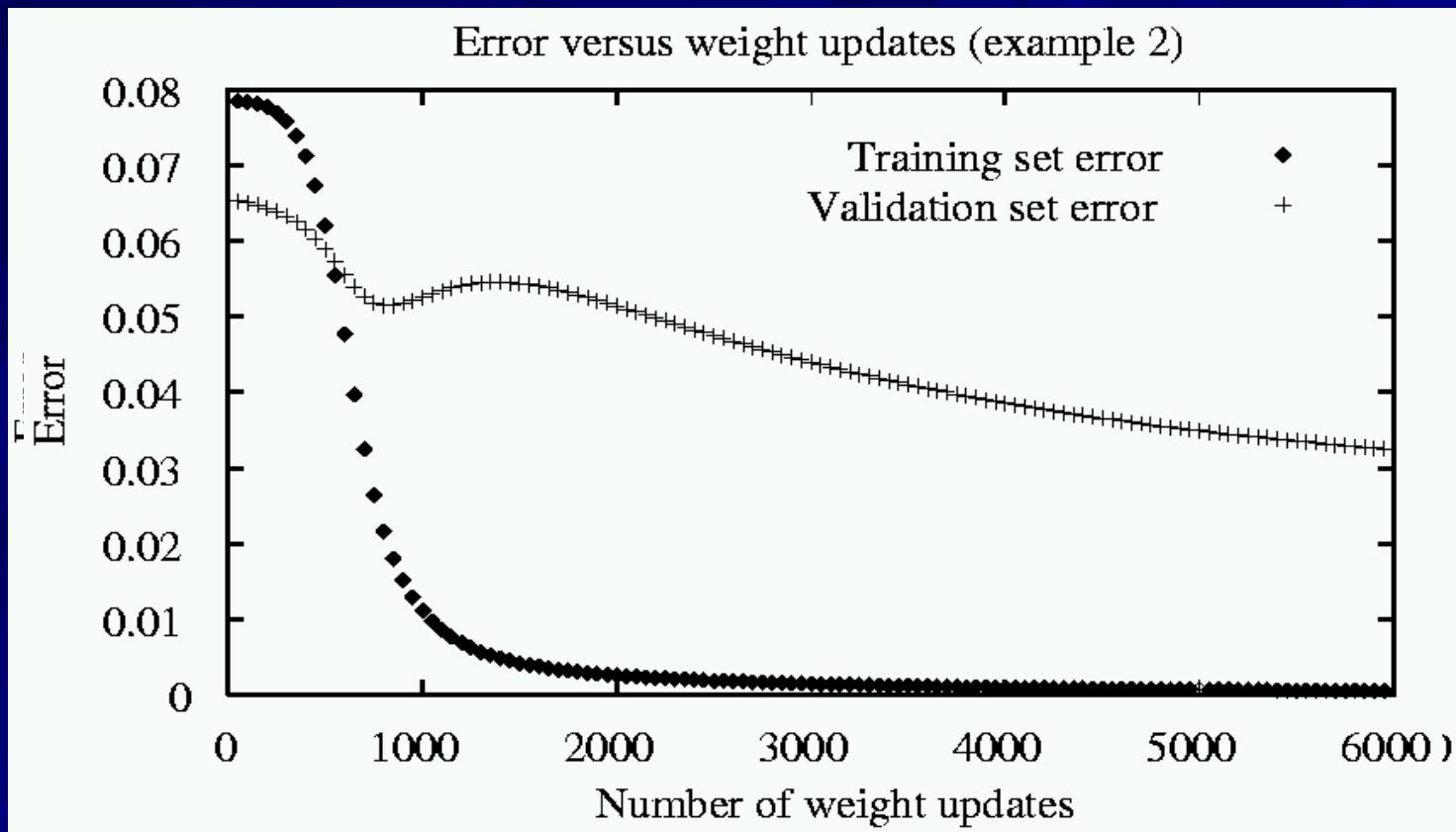


$$\Delta w(t) = \eta \nabla E(t) + \alpha \Delta w(t-1)$$



- Pode a estrutura cair num mínimo local
- Pesos podem divergir
- ...mas trabalhar bem na prática

Problema do Overfitting



(1^a paragem) Parar aprendizagem quando da validação começa a subir.

Estimador de Mínimos Quadrados

$y = \theta_1 \cdot f_1(\mathbf{u}) + \theta_2 \cdot f_2(\mathbf{u}) + \cdots + \theta_n \cdot f_n(\mathbf{u}) \rightarrow$ Regressão linear

$$\begin{cases} f_1(\mathbf{u}_1) \cdot \theta_1 + f_2(\mathbf{u}_1) \cdot \theta_2 + \cdots + f_n(\mathbf{u}_1) \cdot \theta_n = y_1, \\ f_1(\mathbf{u}_2) \cdot \theta_1 + f_2(\mathbf{u}_2) \cdot \theta_2 + \cdots + f_n(\mathbf{u}_2) \cdot \theta_n = y_2, \\ \vdots \qquad \qquad \qquad \vdots \qquad \qquad \vdots \\ f_1(\mathbf{u}_m) \cdot \theta_1 + f_2(\mathbf{u}_m) \cdot \theta_2 + \cdots + f_n(\mathbf{u}_m) \cdot \theta_n = y_m, \end{cases}$$

$$\mathbf{A} \cdot \boldsymbol{\theta} = \mathbf{y} \qquad \qquad (\mathbf{A} - \text{matriz } m \times n)$$

$$\mathbf{A} = \begin{bmatrix} f_1(\mathbf{u}_1) & \cdots & f_n(\mathbf{u}_1) \\ \vdots & \ddots & \vdots \\ f_1(\mathbf{u}_m) & \cdots & f_n(\mathbf{u}_m) \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad (n \times 1) \qquad \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \quad (m \times 1)$$

Se $m = n$ $\boldsymbol{\theta} = \mathbf{A}^{-1} \cdot \mathbf{y}$

Se $m > n$ $\mathbf{A} \cdot \boldsymbol{\theta} + \mathbf{e} = \mathbf{y}$

Solução, para $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ que minimiza a **soma do erro quadrático**, definido por:

$$E(\boldsymbol{\theta}) = \sum_{i=1}^m (y_i - \mathbf{a}_i^T \cdot \boldsymbol{\theta})^2 = \mathbf{e}^T \cdot \mathbf{e} = (\mathbf{y} - \mathbf{A} \cdot \boldsymbol{\theta})^T \cdot (\mathbf{y} - \mathbf{A} \cdot \boldsymbol{\theta})$$

Solução, para $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ que minimiza a **soma do erro quadrático**, definido por:

$$\mathbf{A}^T \mathbf{A} \hat{\boldsymbol{\theta}} = \mathbf{A}^T \mathbf{y}$$

Se $(\mathbf{A}^T \mathbf{A})^{-1}$ existe

$$\hat{\boldsymbol{\theta}} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y}$$

Solução, para $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ que minimiza a **soma do erro quadrático pesado**, definido por:

$$E_W(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{A} \cdot \boldsymbol{\theta})^T \cdot \mathbf{W} \cdot (\mathbf{y} - \mathbf{A} \cdot \boldsymbol{\theta})$$

$$\text{Se } (\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1} \text{ existe} \implies \hat{\boldsymbol{\theta}} = (\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{W} \mathbf{y}$$

Exemplo:

Experiência	Força (N)	Deslocamento (in)
1	1.1	1.5
2	1.9	2.1
3	3.2	2.5
4	4.4	3.3
5	5.9	4.1
6	7.4	4.6
7	9.2	5.0

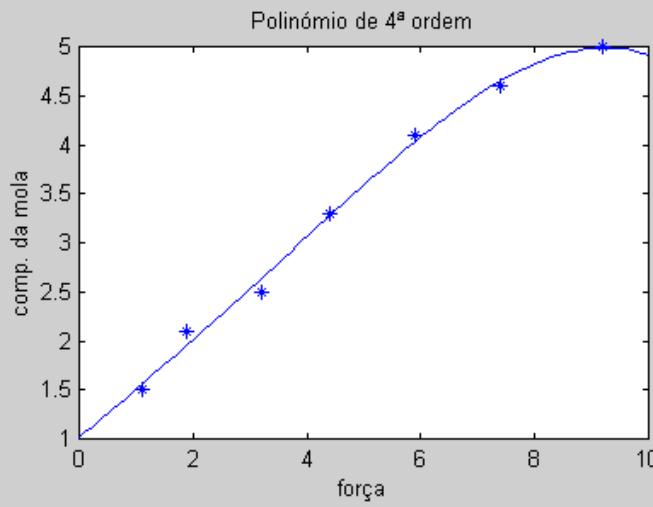
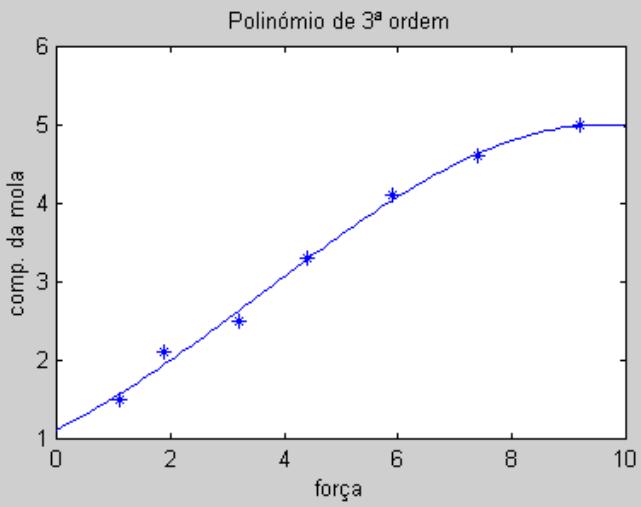
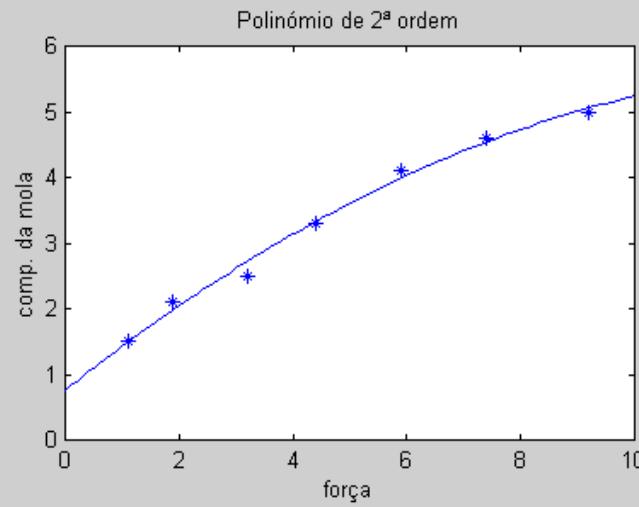
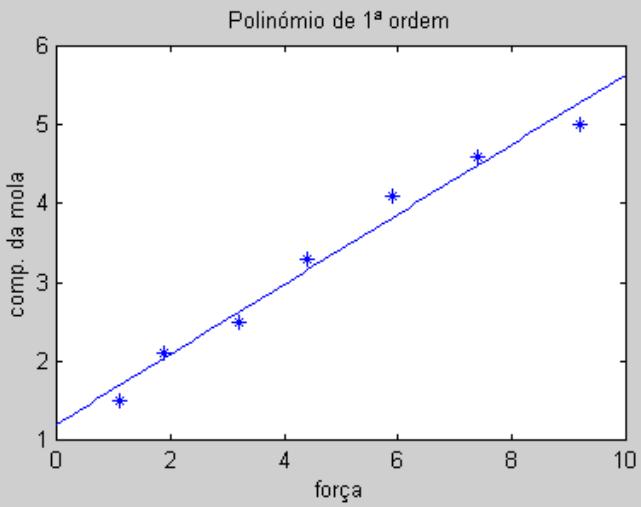
Lei de Hooke $l = k_1 f + k_o$

$$\underbrace{\begin{bmatrix} 1.1 & 1 \\ 1.9 & 1 \\ 3.2 & 1 \\ 4.4 & 1 \\ 5.9 & 1 \\ 7.4 & 1 \\ 9.2 & 1 \end{bmatrix}}_{\mathbf{A}} + \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 2.1 \\ 2.5 \\ 3.3 \\ 4.1 \\ 4.6 \\ 5.0 \end{bmatrix} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y} = \begin{bmatrix} 0.44 \\ 1.20 \end{bmatrix}$$

$$l = k_2 f^2 + k_1 f + k_o$$

$$l = k_3 f^3 + k_2 f^2 + k_1 f + k_o$$

$$l = k_4 f^4 + k_3 f^3 + k_2 f^2 + k_1 f + k_o$$



Estimador de Mínimos Quadrados Recursivos

$$\text{Se } (\mathbf{A}^T \mathbf{A})^{-1} \text{ existe} \quad \theta_k = \underbrace{\left(\mathbf{A}^T \cdot \mathbf{A} \right)^{-1}}_{\mathbf{P}_k} \cdot \mathbf{A}^T \cdot \mathbf{y} \quad \mathbf{P}_k = (\mathbf{A}^T \cdot \mathbf{A})^{-1}$$

$$\theta_{k+1} = \underbrace{\left(\begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix}^T \cdot \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix} \right)^{-1}}_{\mathbf{P}_{k+1}} \cdot \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix}^T \cdot \begin{bmatrix} \mathbf{y}_k \\ y_{k+1}^T \end{bmatrix}$$

$$\mathbf{P}_{k+1}^{-1} = \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix}^T \cdot \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix} = \mathbf{A}_k^T \mathbf{A}_k + \mathbf{a}_{k+1} \mathbf{a}_{k+1}^T$$

• • •

$$\theta_{k+1} = \theta_k + \mathbf{P}_{k+1} \mathbf{a}_{k+1} \left(y_{k+1}^T - \mathbf{a}_{k+1}^T \theta_k \right)$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{a}_{k+1} \mathbf{a}_{k+1}^T \mathbf{P}_k}{1 + \mathbf{a}_{k+1}^T \mathbf{P}_k \mathbf{a}_{k+1}}$$

Valores inicial de \mathbf{P}_0 e θ_0 :

$$\mathbf{P}_n = (\mathbf{A}_n^T \cdot \mathbf{A}_n)^{-1}$$

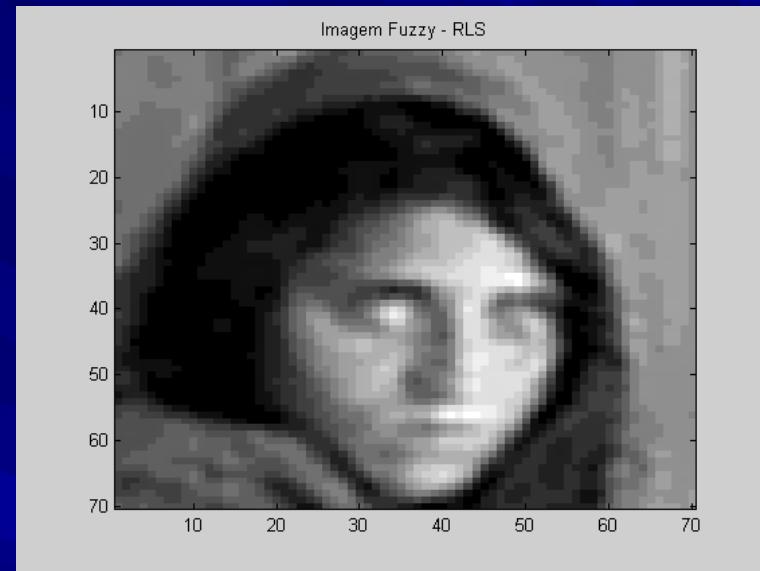
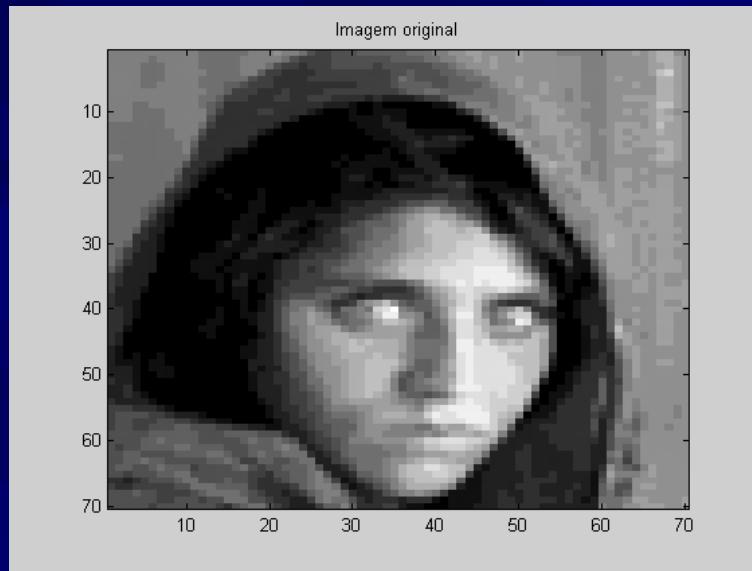
$$\theta_n = \mathbf{P}_n \cdot \mathbf{A}_n^T \cdot y_n$$

recomeçar no ponto n+1

$$\mathbf{P}_0 = \alpha \mathbf{I} \quad \alpha \text{ um valor muito grande}$$

$$\theta_0 = \mathbf{0}$$

Estimador de Mínimos Quadrados Recursivos Aplicados a uma Rede RBF



4900 pixels

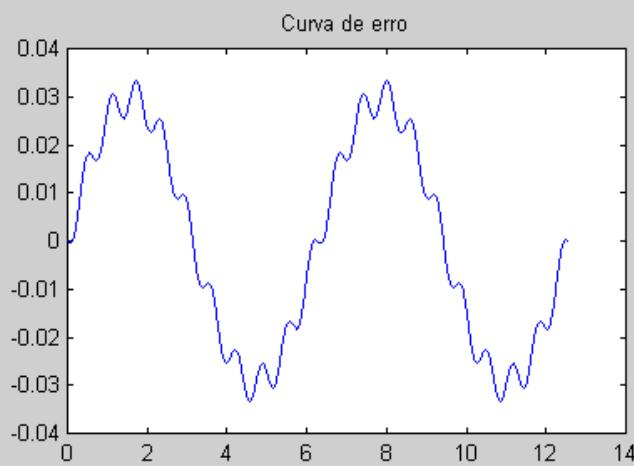
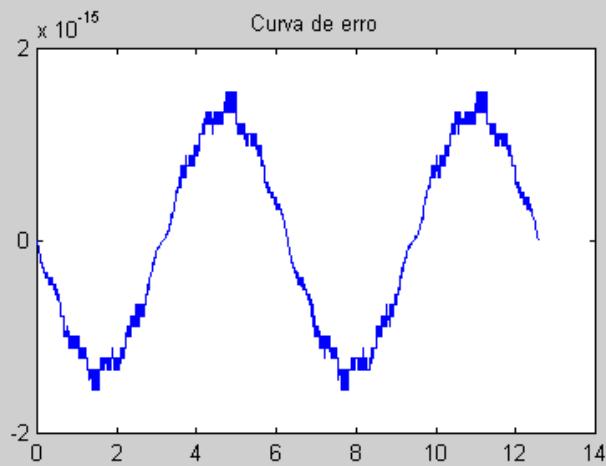
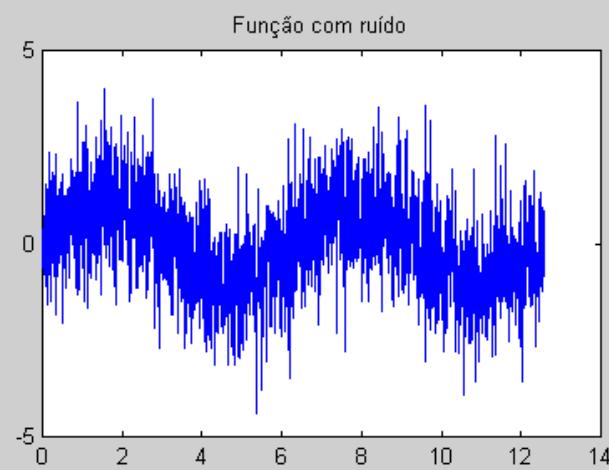
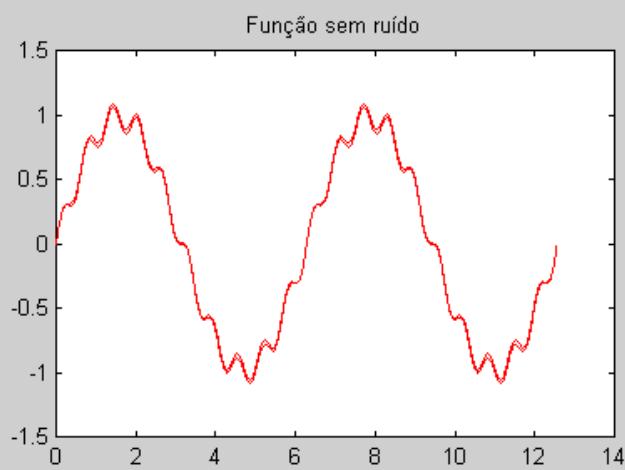
840 parâmetros estimados

Exemplos:

$$y_m = 1 \cdot \sin(t) + 0,1 \cdot \sin(10t)$$

```
% Modelo  
t=(0:0.001:2) .* 2*pi;  
np=size(t,1);  
ym=sin(t)+0.1*sin(10*t);  
  
figure(1);  
subplot(2,2,1);  
plot(t,ym);  
title('Função sem ruído');  
  
% Adicionamento de ruído branco  
subplot(2,2,2);  
yr=ym+randn(np,1);  
  
plot(t,yr);  
title('Função com ruído');  
  
% Obtenção do modelo  
  
A=[sin(t) sin(10*t)];  
  
K=A\ym % dados sem ruído  
Kr=A\yr % dados com ruído
```

```
y_sim=A*K;  
  
subplot(2,2,1);  
hold on;  
plot(t,y_sim, 'r');  
hold off;  
  
subplot(2,2,3);  
erro=(ym-y_sim);  
plot(t,erro);  
title('Curva de erro');  
  
y_sim=A*Kr;  
  
subplot(2,2,1);  
hold on;  
plot(t,y_sim, 'r');  
hold off;  
  
subplot(2,2,4);  
erro=(ym-y_sim);  
plot(t,erro);  
title('Curva de erro');
```



Exemplo:

$$y_m(t) = V_0 \cdot t^m \cdot e^{-\frac{t}{h}} \cdot \cos(w_c t + \theta)$$

$$y_m(t) = u_1(t) \cdot \cos(w_c t) + u_2(t) \cdot \sin(w_c t)$$

Para um intervalo de tempo pequeno, podemos considerar u_1 e u_2 constantes, nesse intervalo.

Solução: dividir a serie temporal em pequenos intervalos, e em cada um deles determinar as constantes u_1 e u_2

$$y_m(k_i) = u_1^i \cdot \cos(w_c k_i T) + u_2^i \cdot \sin(w_c k_i T)$$

Tomando 2 $N+1$ amostras, para $k_i = k-N, \dots, k+N$

```

% Modelo
t=(1/100:1/100:10)';
% vector tempo

np=size(t,1);

m=1.1;
h=1.5;

ym=t.^m.*exp(-t./h).*cos(2*pi*10*t+pi/6);
% Sinal original

Env=t.^m.*exp(-t./h);
% Envolvente

figure(1);
subplot(2,2,1);
plot(t,ym);
title('Função sem ruído');

subplot(2,2,2);
plot(t,Env);
title('Envolvente');

```

```

% Obtenção do modelo

A=[cos(2*pi*10*t) sin(2*pi*10*t)];

step=np./100;

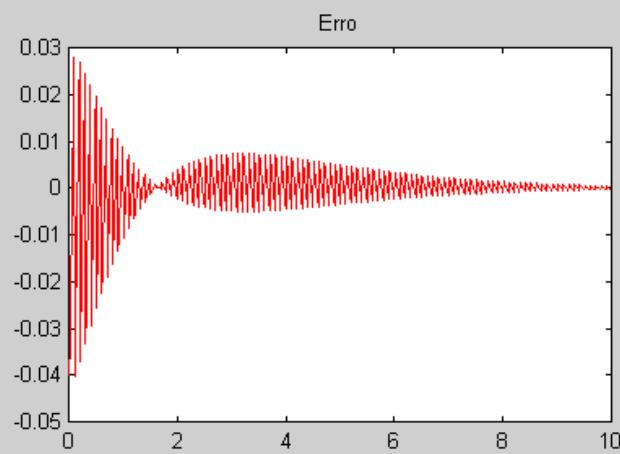
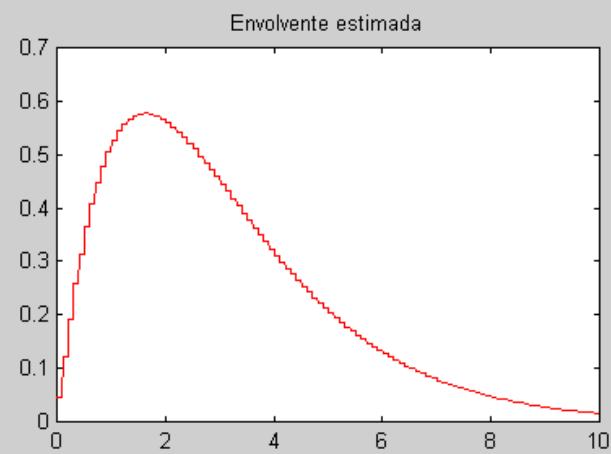
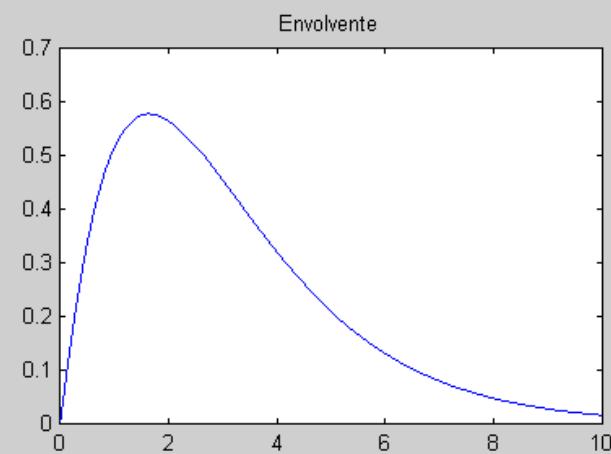
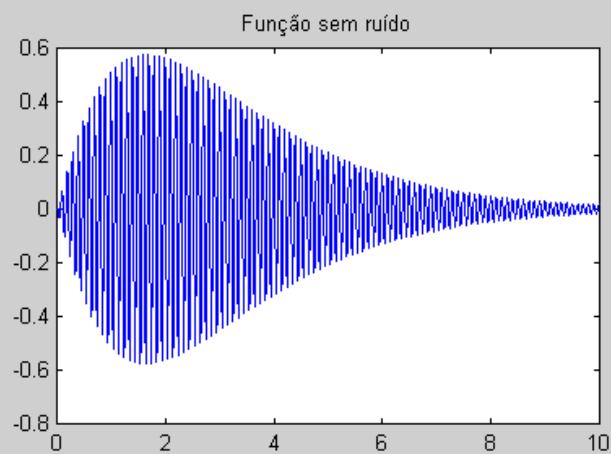
Envz=zeros(np,1);

for(i=1:100)
    ti=(1+(i-1).*step:i.*step)';
    Ui=A(ti,:)\ym(ti)
    % parametros obtido dos dados sem ruído
    Envz(ti)=sqrt(Ui'*Ui).*ones(step,1);
end

subplot(2,2,3);
plot(t,Envz,'r');
title('Envolvente estimada');

subplot(2,2,4);
plot(t,Env-Envz,'r');
title('Erro');

```



Estimador de Mínimos Quadrados Recursivos

Para sistemas variantes no tempo

$$\text{Se } (\mathbf{A}^T \mathbf{A})^{-1} \text{ existe} \quad \boldsymbol{\theta}_k = \underbrace{(\mathbf{A}^T \cdot \mathbf{A})^{-1}}_{\mathbf{P}_k} \cdot \mathbf{A}^T \cdot \mathbf{y} \quad \mathbf{P}_k = (\mathbf{A}^T \cdot \mathbf{A})^{-1}$$

Se $k > n^o$ de parâmetros e os dados continham informação valiosa, então $(\mathbf{A}^T \mathbf{A})$ é geralmente uma matriz definida positiva, e à medida que k tende para infinito $\frac{1}{k} \times (\mathbf{A}^T \mathbf{A})$ converge para uma matriz constante não singular.

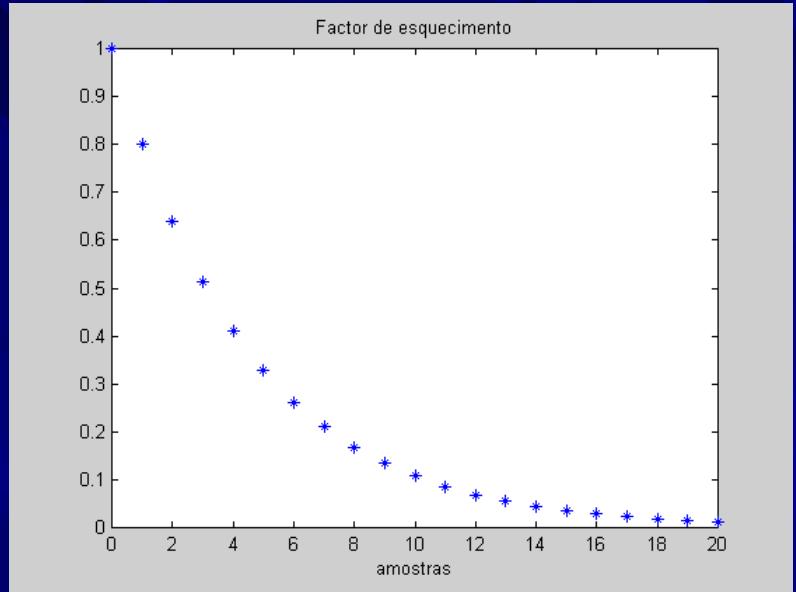
$$\lim_{k \rightarrow \infty} \mathbf{P}_k = \lim_{k \rightarrow \infty} \frac{1}{k} \left(\frac{1}{k} \mathbf{A}^T \cdot \mathbf{A} \right)^{-1} = 0$$



$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{P}_{k+1} \mathbf{a}_{k+1} \left(\mathbf{y}_{k+1}^T - \mathbf{a}_{k+1}^T \boldsymbol{\theta}_k \right)$$

Factor de esquecimento, λ

$$Forget(m) = \lambda^m, \quad \text{para } 0 < \lambda \leq 1$$



$$E_W(\boldsymbol{\theta}) = \sum_{i=1}^m \lambda^{m-i} (y_i - \mathbf{a}_i^T \boldsymbol{\theta})^2 = (\mathbf{y} - \mathbf{A} \cdot \boldsymbol{\theta})^T \cdot \mathbf{W} \cdot (\mathbf{y} - \mathbf{A} \cdot \boldsymbol{\theta})$$

$$\mathbf{W} = \begin{bmatrix} \lambda^{m-1} & 0 & \cdots & 0 \\ 0 & \lambda^{m-2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

$$\mathbf{Se}(\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1} \text{ existe} \implies \hat{\boldsymbol{\theta}} = (\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{W} \mathbf{y}$$

$$\mathbf{Se}(\mathbf{A}^T \mathbf{A})^{-1} \text{ existe} \quad \boldsymbol{\theta}_k = \underbrace{(\mathbf{A}^T \cdot \mathbf{A})^{-1}}_{\mathbf{P}_k} \cdot \mathbf{A}^T \cdot \mathbf{y} \quad \mathbf{P}_k = (\mathbf{A}^T \cdot \mathbf{A})^{-1}$$

$$\boldsymbol{\theta}_{k+1} = \underbrace{\left(\begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix}^T \cdot \begin{bmatrix} \lambda \mathbf{W} & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix} \right)^{-1}}_{\mathbf{P}_{k+1}} \cdot \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix}^T \cdot \begin{bmatrix} \lambda \mathbf{W} & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{y}_k \\ \mathbf{y}_{k+1}^T \end{bmatrix}$$

$$\mathbf{P}_{k+1}^{-1} = \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix}^T \cdot \begin{bmatrix} \lambda \mathbf{W} & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{A}_k \\ \mathbf{a}_{k+1}^T \end{bmatrix} = \lambda \mathbf{A}_k^T \mathbf{W} \mathbf{A}_k + \mathbf{a}_{k+1} \mathbf{a}_{k+1}^T$$

• • •

$$\theta_{k+1} = \theta_k + \mathbf{P}_{k+1} \mathbf{a}_{k+1} \left(y_{k+1}^T - \mathbf{a}_{k+1}^T \theta_k \right)$$

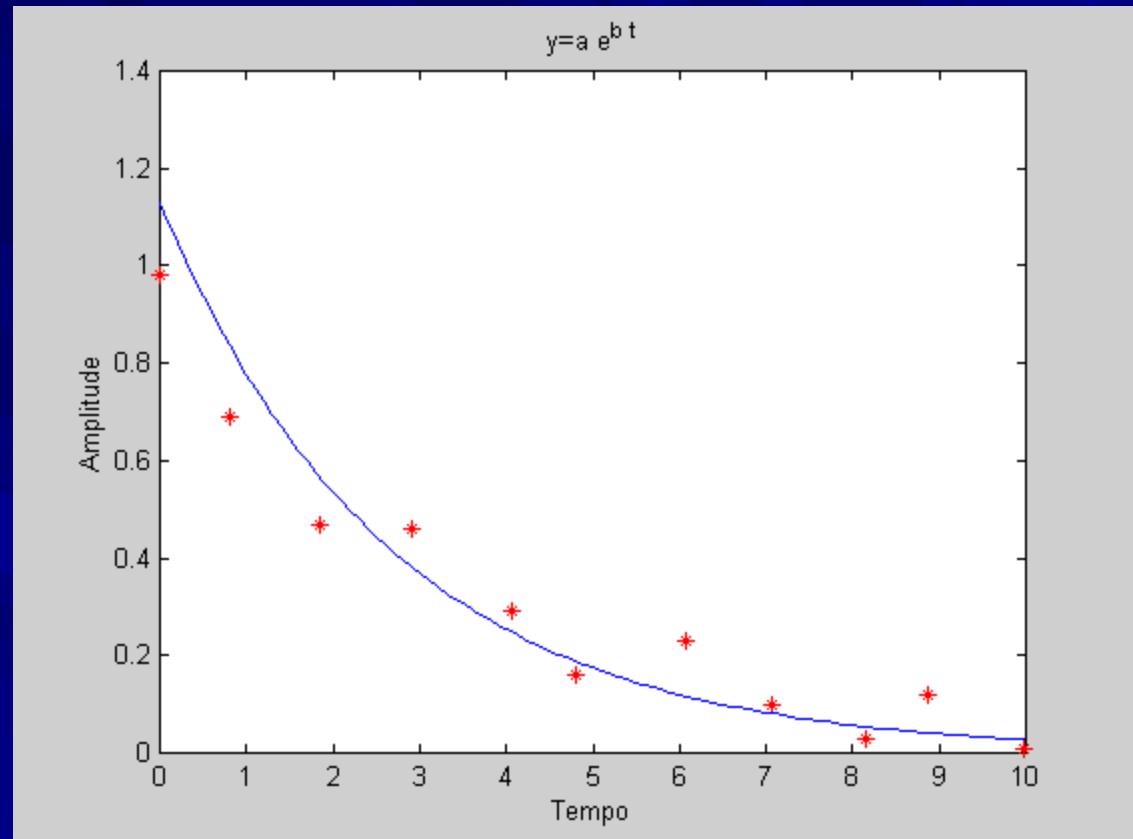
$$\mathbf{P}_{k+1} = \frac{1}{\lambda} \left(\mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{a}_{k+1} \mathbf{a}_{k+1}^T \mathbf{P}_{k+1}}{\lambda + \mathbf{a}_{k+1}^T \mathbf{P}_k \mathbf{a}_{k+1}} \right)$$

Proposta: Aplicar este método à detecção de envolvente do problema da página 22.

Exemplo:

$$y = a \cdot e^{b \cdot t}$$

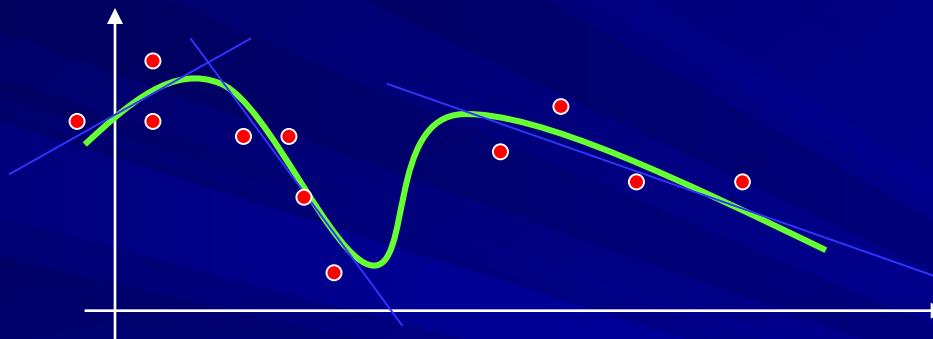
t_i	y_i
0	0.98
0.80	0.69
1.81	0.47
2.90	0.46
4.06	0.29
4.81	0.16
6.07	0.23
7.06	0.10
8.15	0.03
8.87	0.12
9.98	0.01



$$\ln y = \ln a + b \cdot t$$

Aprendizagem local

- Dividir o espaço de entrada em regiões locais de aprendizagem simples (constant/linear)



Não supervisionado: Competitiva, “online clustering”

Supervisionado: Radial-basis func, combinação com especialistas

Competitive Learning

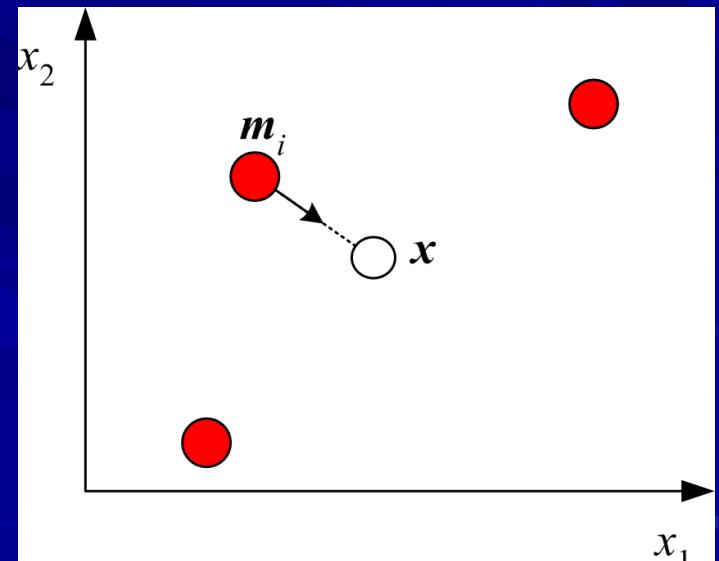
$$E\left(\left\{\mathbf{m}_i\right\}_{i=1}^k | \times\right) = \sum_t \sum_i b_i^t \|\mathbf{x}^t - \mathbf{m}_i\|$$

$$b_i^t = \begin{cases} 1 & \text{if } \|\mathbf{x}^t - \mathbf{m}_i\| = \min_l \|\mathbf{x}^t - \mathbf{m}_l\| \\ 0 & \text{outros} \end{cases}$$

$$k\text{-means: } \mathbf{m}_i = \frac{\sum_t b_i^t \mathbf{x}^t}{\sum_t b_i^t}$$

Batch k -means:

$$\Delta m_{ij} = -\eta \frac{\partial E^t}{\partial m_{ij}} = \eta b_i^t (x_j^t - m_{ij})$$



Initialize $\mathbf{m}_i, i = 1, \dots, k$, for example, to k random \mathbf{x}^t

Repeat

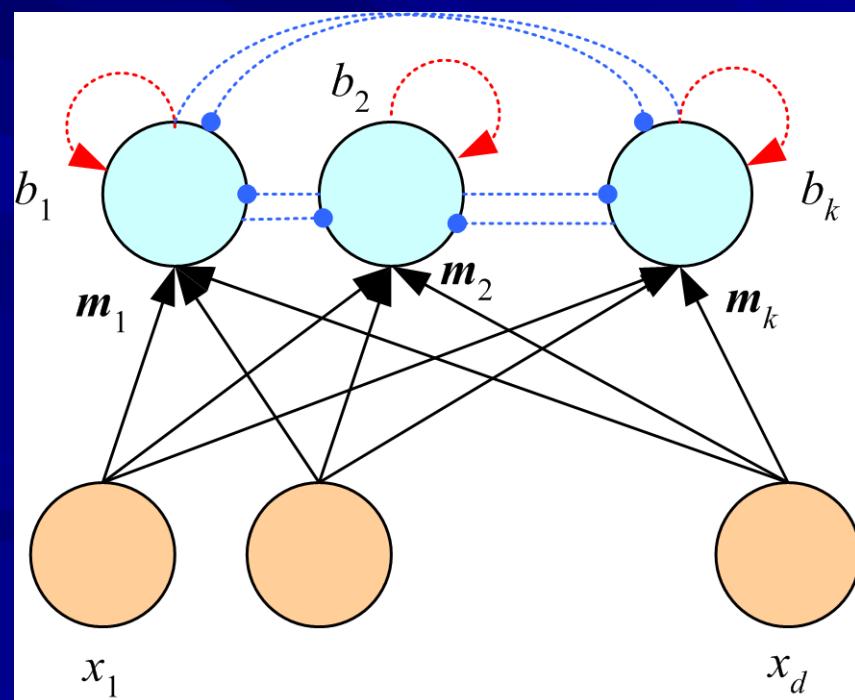
For all $\mathbf{x}^t \in \mathcal{X}$ in random order

$$i \leftarrow \arg \min_j \|\mathbf{x}^t - \mathbf{m}_j\|$$

$$\mathbf{m}_i \leftarrow \mathbf{m}_i + \eta(\mathbf{x}^t - \mathbf{m}_j)$$

Until \mathbf{m}_i converge

*Winner-take-all
network*



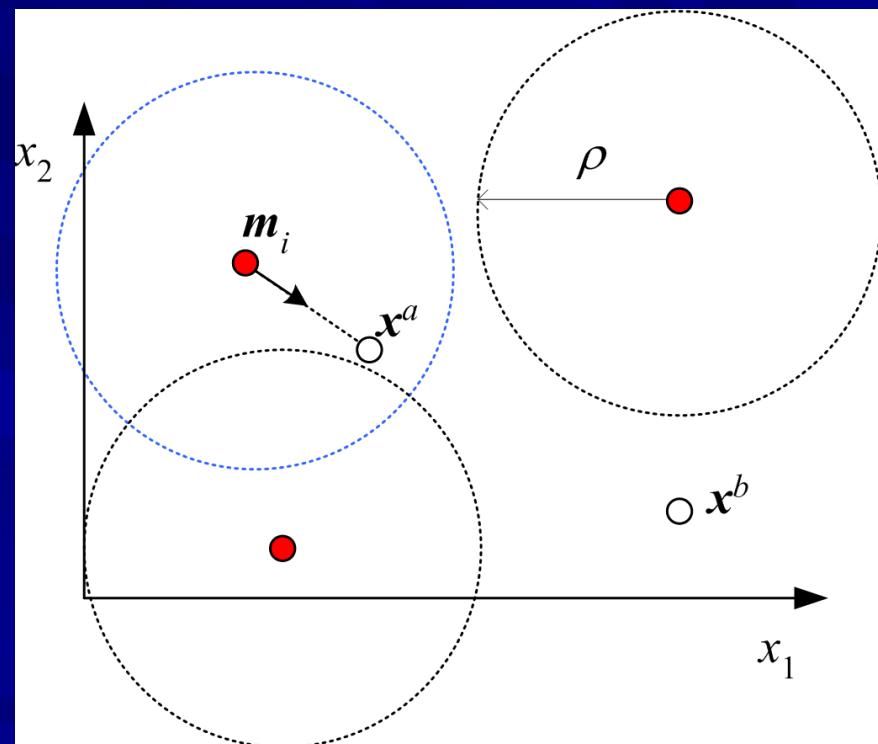
Adaptive Resonance Theory

- Incremental; soma um novo agrupamento se não está coberto; definido pela vigilância, ρ

$$b_i^t = \|\mathbf{x}^t - \mathbf{m}_i\| = -\min_{l=1}^k \|\mathbf{x}^t - \mathbf{m}_l\|$$

$$\begin{cases} \mathbf{m}_{k+1} \leftarrow \mathbf{x}^t & \text{se } b_i > \rho \\ \Delta \mathbf{m}_i = \eta (\mathbf{x}^t - \mathbf{m}_i) & \text{outros} \end{cases}$$

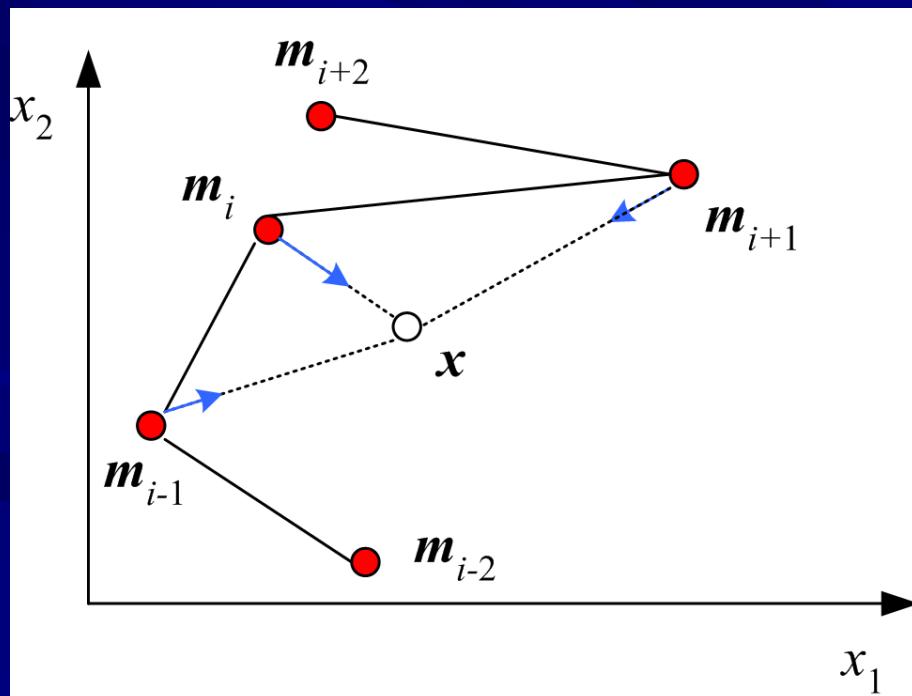
(Carpenter and Grossberg, 1988)



Self-Organizing Maps

- As unidades têm uma vizinhança definida; \mathbf{m}_i está “entre” \mathbf{m}_{i-1} e \mathbf{m}_{i+1} , e todos são simultaneamente adaptados
- Mapa a uma dimensão:

(Kohonen, 1990)



$$\Delta \mathbf{m}_l = \eta e(l, i) (\mathbf{x}^t - \mathbf{m}_l)$$
$$e(l, i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(l-i)^2}{2\sigma^2}\right]$$

Algoritmo EM

- Em cálculo estatístico, um algoritmo de maximização de expectativa (EM) tem como objetivo o encontrar estimativas da probabilidade máxima para os parâmetros dos modelos probabilísticos, onde o modelo depende das variáveis latentes não observáveis.
- O EM alterna entre executar uma etapa da expectativa (E), que calcula uma expectativa da probabilidade incluindo as variáveis latentes como se fossem observadas, e uma etapa do maximização (M), que calcula as estimativas da probabilidade máxima dos parâmetros, que maximizam a probabilidade prevista encontrada na etapa E. Os parâmetros encontrados na etapa de M são usados então começar uma outra etapa E, e o processo é repetido.

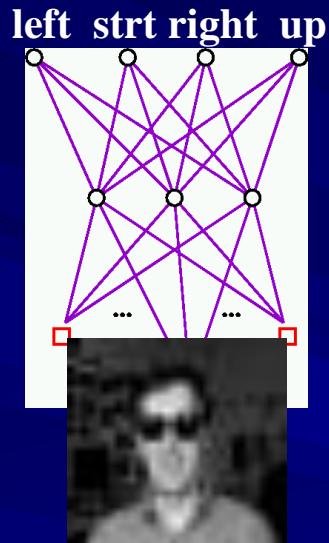
EM no treino de RBF

■ E-step: $f_h^t \equiv p(\mathbf{r}|h, \mathbf{x}^t)$

■ M-step:

$$\mathbf{m}_h = \frac{\sum_t f_h^t \mathbf{x}^t}{\sum_t f_h^t}$$
$$s_h = \frac{\sum_t f_h^t (\mathbf{x}^t - \mathbf{m}_h)(\mathbf{x}^t - \mathbf{m}_h)^T}{\sum_t f_h^t}$$
$$w_{ih} = \frac{\sum_t f_h^t r_i^t}{\sum_t f_h^t}$$

ANNs no reconhecimento de Faces



Imagens de entrada típicas

Pose da cabeça(1-of-4):

Reconhecimento da face (1-of-20):

90% accuracy

90% accuracy