



Universidade de São Paulo  
Escola de Engenharia de São Carlos  
Departamento de Engenharia Elétrica



# **REDES NEURAIS ARTIFICIAIS**

**Prof. Ivan Nunes da Silva**

**Rogério Saranz Oliani**

**São Carlos  
2005**

# SUMÁRIO

1 INTRODUÇÃO.....	1
1.1 CONCEITOS INICIAIS.....	1
1.1.1 Características .....	1
1.1.2 História das RNA .....	1
1.1.3 Potenciais Áreas de Aplicações.....	1
1.2 O NEURÔNIO BIOLÓGICO .....	2
1.3 O NEURÔNIO ARTIFICIAL .....	2
1.4 ANÁLISE MATEMÁTICA DO NEURÔNIO .....	4
2 ARQUITETURAS E TIPOS DE TREINAMENTO .....	5
2.1 PRINCIPAIS ARQUITETURAS.....	5
2.2 TIPOS DE TREINAMENTO .....	6
3 O PERCEPTRON.....	7
3.1 TREINAMENTO DO PERCEPTRON.....	7
4 O ADALINE E A REGRA DELTA.....	9
4.1 INTRODUÇÃO.....	9
4.2 A REGRA DELTA .....	10
4.3 INTERPRETAÇÃO GEOMÉTRICA (REGRA DELTA).....	10
4.4 ALGORITMO DO ADALINE.....	11
5 REDES PERCEPTRON MULTICAMADAS (PMC) .....	13
5.1 INTRODUÇÃO.....	13
5.2 ALGORITMO BACKPROPAGATION (REGRA DELTA GENERALIZADA) .....	14
5.3 PMC COMO CLASSIFICADORES .....	17
5.4 APROXIMADOR UNIVERSAL DE FUNÇÕES .....	18
5.5 IMPLEMENTAÇÃO DO PMC.....	19
5.6 ASPECTOS DA REDE PERCEPTRON .....	19
6 REDES RECORRENTES DE HOPFIELD.....	21
6.1 REDES DE HOPFIELD.....	21
6.2 A ESTABILIDADE DA REDE DE HOPFIELD .....	22
6.3 MEMÓRIAS ASSOCIATIVAS (APRENDER POR ASSOCIAÇÃO).....	23
6.4 ALGORITMO DA REDE DE HOPFIELD.....	24
7 SISTEMAS VARIANTES NO TEMPO E REDES NEURAIS (SISTEMAS DINÂMICOS) .....	25
7.1 REDES NEURAIS COM ATRASO DE TEMPO (TDNN).....	25
7.2 REDE PERCEPTRON RECORRENTE.....	26
7.3 SELECIONANDO A TOPOLOGIA .....	27
8 REDES DE FUNÇÕES DE BASE RADIAL .....	28
8.1 INTRODUÇÃO.....	28
8.2 TREINAMENTO DA RBF .....	29
8.3 COMPARAÇÃO ENTRE RBF E MLP .....	31
9 REDES DE KOHONEN E MAPAS AUTO-ORGANIZÁVEIS .....	32
9.1 CONSIDERAÇÕES INICIAIS .....	32
9.2 A ARQUITETURA DE KOHONEN .....	32
9.3 O PROCESSO DE APRENDIZADO COMPETITIVO.....	33
9.4 MAPA TOPOLÓGICO AUTO-ORGANIZÁVEL .....	33
9.5 ALGORITMO DE TREINAMENTO .....	34

10 REDE LVQ E COUNTER-PROPAGATION .....	35
10.1 CONSIDERAÇÕES INICIAIS .....	35
10.2 AS REDES LVQ.....	35
10.3 A REDE COUNTER-PROPAGATION .....	37
11 REDES ART (ADAPTIVE RESSONANCE THEORY).....	38
11.1 CONSIDERAÇÕES INICIAIS .....	38
11.2 A ARQUITETURA ART-1 .....	38
11.3 A OPERAÇÃO DA ART-1 .....	39
11.4 ALGORITMO COMPUTACIONAL.....	39
11.5 CONSIDERAÇÕES FINAIS .....	40

# 1 Introdução

## 1.1 Conceitos Iniciais

Redes Neurais Artificiais são modelos computacionais inspirados no cérebro humano e que possuem a capacidade de aquisição e manutenção do conhecimento (informações).

Em outras palavras, RNA podem ser definidas como um conjunto de unidades de processamento (“Neurônio”) que são interligadas por um grande número de interconexões (sistemas artificiais).

### 1.1.1 Características

As principais são:

- Aprender através de exemplos (Padrões de treinamento).
- Capacidade de se adaptar ou aprender.
- Capacidade de generalização (Principal).
- Agrupar ou organizar dados.
- Tolerância à falhas.
- Auto-organização.

### 1.1.2 História das RNA

Primeiras aplicações efetivas se deram a partir de 1987/88. Um breve histórico é dado a seguir:

- 1943 → Trabalho de McCulloch e Pitts.
- 1956 → Reunião no Dartmouth College.
- 1969 → Publicação do livro Perceptron.
- 1987 → Trabalhos de Rumelhart, Hopfield.
- Dias de hoje.

### 1.1.3 Potenciais Áreas de Aplicações

#### A-) Reconhecimento de padrões

Tarefa: Atribuir um padrão de entrada a uma das várias classes predefinidas.

Exemplo: Reconhecimento de imagens, de voz, de escrita, etc.

#### B-) Clustering / Categorização

Tarefa: Explorar semelhanças entre padrões e agrupar padrões.

Exemplo: Compressão de dados, Garimpagem de dados, Identificação de semelhanças, etc.

#### C-) Aproximação de função (Aplicação mais utilizada)

Tarefa: Encontrar uma estimativa “y” de função “f”.

Exemplo: Problemas de modelagem científica e de engenharia.

#### D-) Previsão / Estimulação

Tarefa: Dado um conjunto de exemplos  $\{y(t_1), y(t_2), \dots, y(t_N)\}$ , prever o valor “ $y(t_{N+1})$ ” no instante de tempo “ $t_{N+1}$ ”.

Exemplo: Previsão de tempo, de séries temporais, mercado financeiro, etc.

**E-) Otimização**

Tarefa: Minimizar ou maximizar uma função sujeito ou não a restrições.

Exemplos: Programação Linear, Programação não Linear, Otimização Combinatorial, Programação Dinâmica, etc.

**F-) Memórias Associativas (Endereçável pelo conteúdo)**

Tarefa: Recuperar item correto mesmo que a entrada seja parcial ou distorcida.

Exemplo: Processamento de imagens, de gráficos, caracteres, etc.

**G-) Controle**

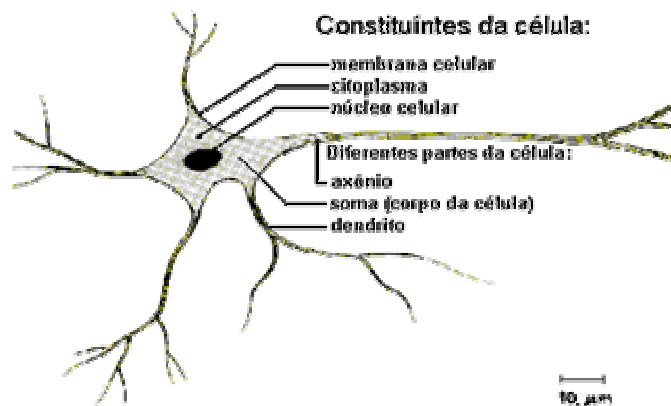
Tarefa: Gerar entrada de controle para que o sistema siga trajetória especificada pela referência.

Exemplo: Controle de processos, de Robôs, etc.

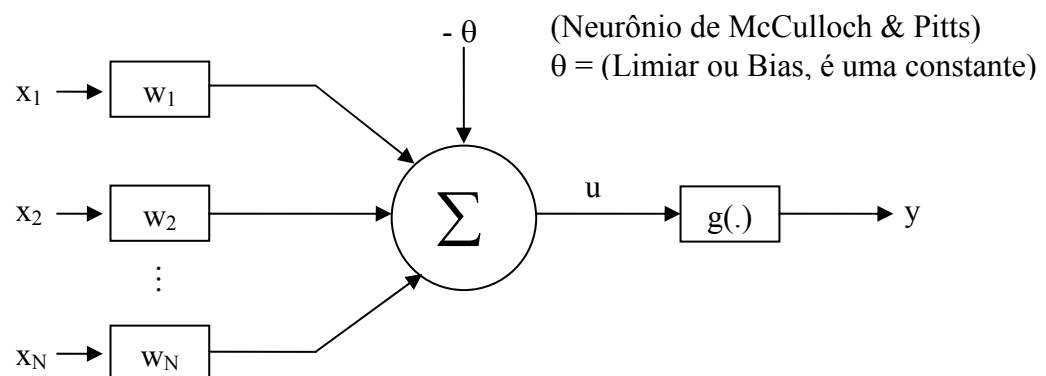
**1.2 O Neurônio Biológico**

O processamento de informações no cérebro humano é altamente complexo, não linear e paralelo.

O Neurônio é a célula fundamental do sistema nervoso cerebral.

**1.3 O Neurônio Artificial**

O modelo geral do neurônio artificial pode ser ilustrado por:



Onde:

$x_1, x_2, \dots, x_N$  são os sinais de entrada (informação que estão entrando no neurônio).

$w_1, w_2, \dots, w_N$  são os sinápticos.

$\theta$  é o limiar de ativação do neurônio.

$u$  é a saída do combinador linear.

$g(\cdot)$  é a função de ativação.

$y$  é o sinal de saída do neurônio.

Em termos matemáticos, tem-se:

$$u = \sum_{i=1}^N w_i * x_i - \theta$$

$$y = g(u)$$

$$\begin{cases} 2 \text{ entradas} = \text{reta} \\ 3 \text{ entradas} = \text{plano} \\ 4 \text{ ou mais} = \text{hiperplano} \end{cases}$$

Se não tiver o  $\theta$  a função vai estar sempre centrada na origem

A função de ativação “g(.)” processa o conjunto de entradas recebidas e o transforma em estado de ativação. Normalmente, o estado de ativação dos neurônios pode assumir os seguintes valores:

- Binários (0 e 1)
- Bipolares (-1 e 1)
- Reais  $\longrightarrow -1 \leq g(.) \leq 1$  ou  $\theta \leq g(.) \leq 1$

Função de ativação  $\rightarrow$  Limitar a saída do neurônio.

Eficiência Energética (Operações / Neurônio)

Biológico  $\rightarrow 10^{-16}\text{J}$

Artificial  $\rightarrow 10^{-6}\text{J}$

Velocidade de processamento (Operações / Neurônio)

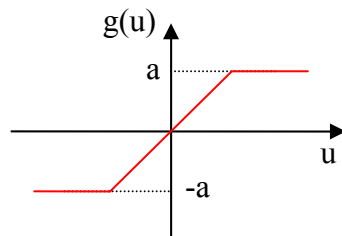
Artificial  $\rightarrow 10^{-9}\text{s}$  (seqüenciais)

Biológico  $\rightarrow 10^{-3}\text{s}$  (trabalha em paralelo)

As funções de ativação mais típicas são:

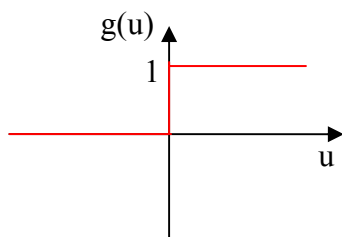
**a-) Função Rampa**

\* otimização



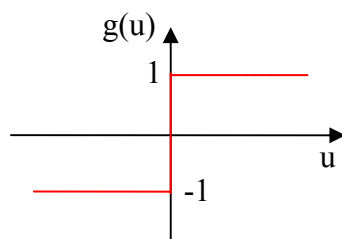
$$g(u) = \begin{cases} a, & \text{se } u > a \\ u, & \text{se } -a \leq u \leq a \\ -a, & \text{se } u < -a \end{cases}$$

**b-) Função Degrau (Função Limiar)**

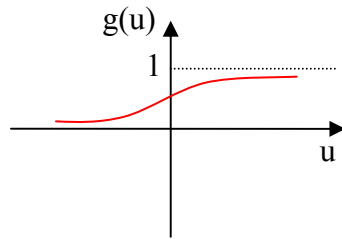


$$g(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ 0, & \text{se } u < 0 \end{cases}$$

**c-) Função Sinal (Função Degrau Bipolar)**

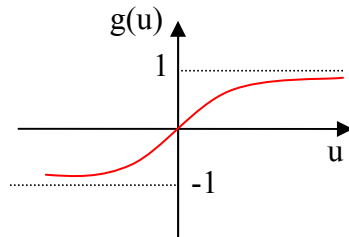


$$g(u) = \begin{cases} 1, & \text{se } u > 0 \\ 0, & \text{se } u = 0 \\ -1, & \text{se } u < 0 \end{cases}$$

**d-) Função Logística (Sigmóide)**

$$g(u) = \frac{1}{1 + e^{-\beta u}}$$

$\beta$  = inclinação da sigmóide  
Conforme altera o  $\beta$  altera a inclinação. Com o  $\beta$  muito grande parece o degrau.

**e-) Função Hiperbólica**

$$g(u) = \frac{1 - e^{-\beta u}}{1 + e^{-\beta u}}$$

$$g(u) = \text{TaH}\left(\frac{\beta u}{2}\right)$$

**1.4 Análise Matemática do Neurônio**

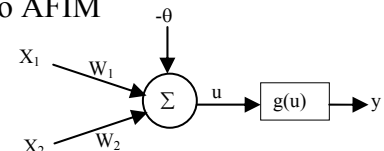
O neurônio de McCulloch, cujas saídas são binárias, é um caso particular de discriminador linear. A saída do neurônio, utilizando a função degrau, é dada por:

$$y = \begin{cases} 1, & \text{se } \sum w_i * x_i - \theta \geq 0 \\ 0, & \text{se } \sum w_i * x_i - \theta < 0 \end{cases}$$

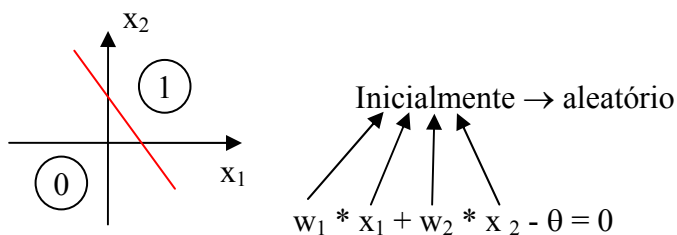
Minski & Pappert  
Na introduction to computer geometri

Para um neurônio com duas entradas, tem-se:

$$y = \begin{cases} 1, & \text{se } w_1 * x_1 + w_2 * x_2 - \theta \geq 0 \\ 0, & \text{se } w_1 * x_1 + w_2 * x_2 - \theta < 0 \end{cases} \quad \text{Transformação AFIM}$$



Ilustrando através do gráfico tem-se:



Assim o neurônio de McCulloch comporta-se como um classificador de padrões, que separa duas regiões linearmente separáveis.

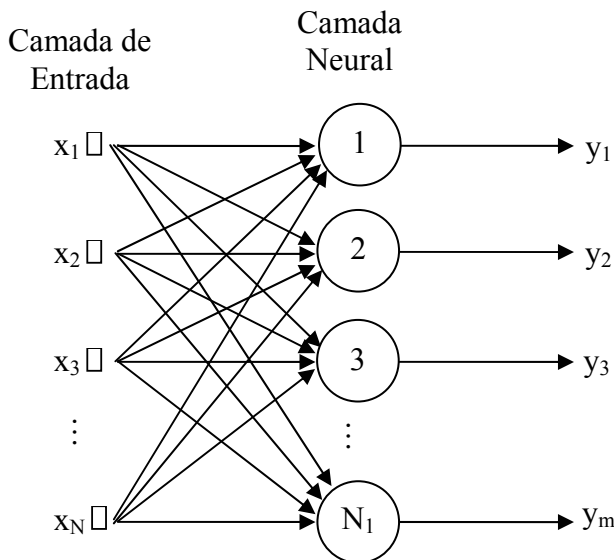
## 2 Arquiteturas e tipos de treinamento

### 2.1 Principais Arquiteturas

A arquitetura da RNA (maneira em que os neurônios estão arranjados) está fortemente relacionada com o algoritmo utilizado para treinar a rede. As principais arquiteturas são:

#### A-) Redes FeedForward (camada única) ( Realimentação para frente)

Neste tipo de rede, tem-se uma camada de entrada e uma única camada de neurônio que é a própria camada de saída.



##### Aplicações:

- Memória associativa
- Reconhecimento de padrões

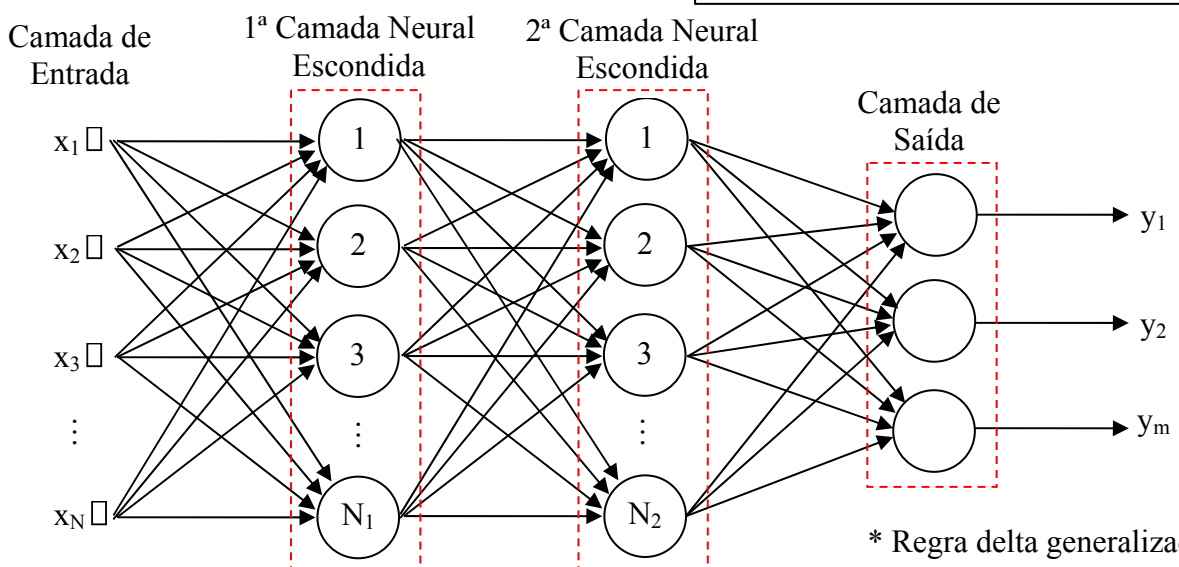
##### Tipos:

- Perceptron
- Adaline

#### B-) Redes FeedForward (Multicamadas)

Esse tipo de rede difere da anterior pela presença de uma ou mais camadas escondidas de neurônios.

Nº de saída é menor que de neurônios



\* Regra delta generalizada

Camada de entrada: onde os padrões (informações) são apresentados.

Camada escondida: onde é feito a maioria do processamento (extração de características).

Camada de saída: onde o resultado final é concluído e apresentado.



**Aplicações:**

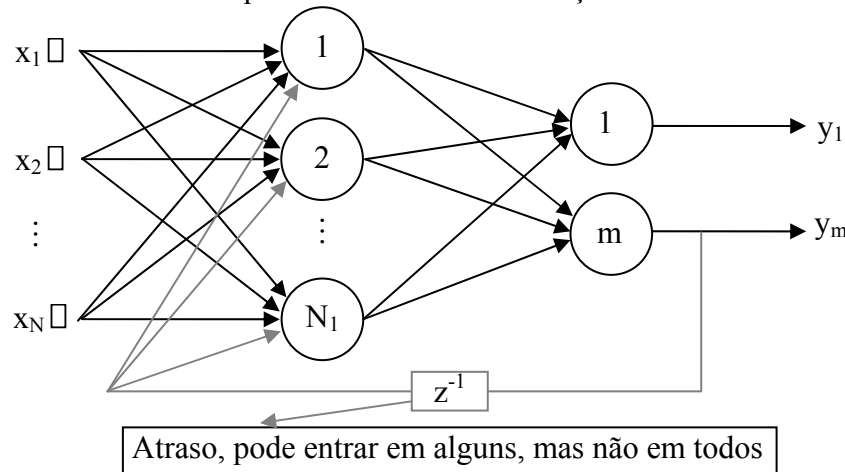
- Aproximador de funções
- Reconhecimento de padrões
- Identificação e controle

**Tipos:**

- Perceptron multicamadas
- Funções base radial

**C-) Redes Recorrentes**

São redes que contém retro-alimentação entre neurônios de camadas diferentes.

**Aplicações:**

- Previsão / estimação
- Séries temporais
- Otimização
- Sistemas Dinâmicos

**Tipos:**

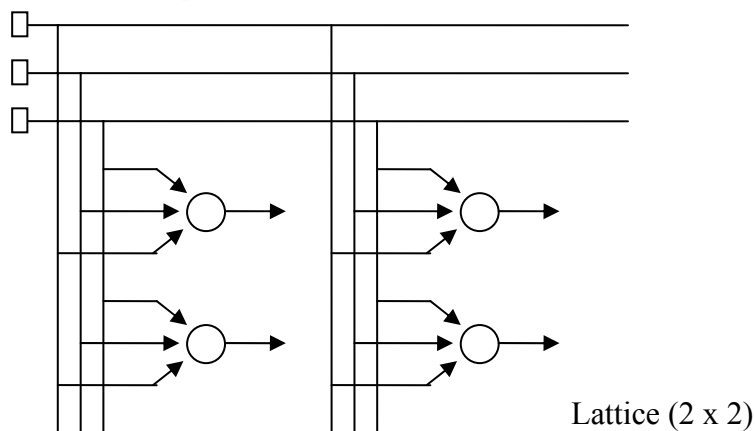
- Hopfield
- Perceptron com realimentação

**Treinamento:**

- Regra Delta
- Minimização

**D-) Estrutura Lattice (ou reticulada)**

Consiste de um ARRAY de neurônios de uma ou mais dimensões, os sinais de entrada são os mesmos para todos os neurônios.

**Aplicações:**

- Grafos, Clustering
- Aplicações que dependem de localização espacial dos neurônios visando retirada de características.

**Tipos:**

- Redes de Kohonen

**Treinamento:**

- Competitivo

Na realidade, um Lattice é uma rede FeedFoward cujos neurônios são arranjados em linhas e colunas

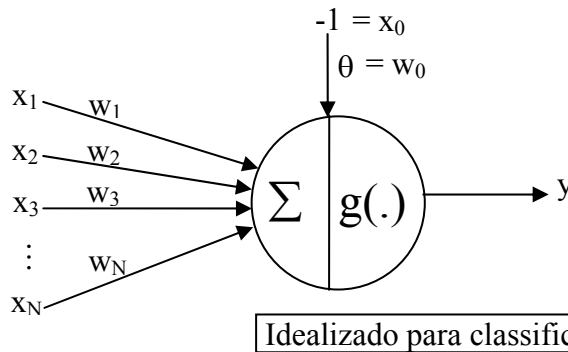
**2.2 Tipos de Treinamento**

O treinamento, o processo de aprendizado consiste em ajustar os pesos sinápticos e limiares de forma que a aplicação de um conjunto de entradas produza um conjunto de saídas desejadas. Podem, ser classificadas em:

- **Supervisionada:** A rede é treinada para fornecer a saída desejada a um estímulo de entrada específica.
- **Não supervisionada:** Não há uma saída específica em relação aos estímulos de entrada. A rede se auto-organiza em relação às particularidades do conjunto de entrada.

### 3 O Perceptron (Rosenblatt, 1949)

O perceptron é a forma mais simples de uma rede neural, e é utilizado na classificação de padrões que devem ser linearmente separáveis.



$$u = \sum_{i=1}^n w_i * x_i - 0$$

$$y = g(u)$$

$$u = \sum_{i=0}^n w_i * x_i$$

Na realidade o perceptron é uma rede FeedForward (camada única) e sua operação é baseada no modelo neural de McCulloch.

A classificação de padrões no perceptron possuem as seguintes características:

- **Entradas:** Reais.
- **Saída:** Bipolar (-1 e 1).
- **Função de ativação:** Função sinal.
- **Pesos:** Devem ser inicializados aleatoriamente.
- **Classes:** Devem ser linearmente separáveis.
- **Treinamento:** Supervisionado.

Cada entrada de  $w$  tem uma determinada saída.

#### 3.1 Treinamento do Perceptron

O treinamento do perceptron é realizado utilizando o princípio de aprendizado de Hebb(1949).

Sejam as variáveis e parâmetros:

$x(k)$  = vetor de entradas do padrão  $k$  =  $[-1, x_1(k), x_2(k), \dots, x_n(k)]$   $x_0(k) = -1$ .

$y$  = saída do neurônio.

$d(k)$  = saída desejada em relação ao padrão  $k$ .

$\eta$  = taxa de aprendizagem. ( $0 < \eta < 1$ ).

$w$  = vetor de pesos (conexões) da rede. ( $= [\theta \ w_1 * w_2 \ \dots \ w_n]$ ).

O ajuste dos pesos é feito utilizando a seguinte regra:

$$w \leftarrow w + \eta * (d(k) - y) * x(k)$$

$d(k)$	$y$	ação de $w$
1	-1	incrementa
-1	1	decrementa

Contexto neuro-biológico.  
↓  
The organization of Behavior (Hebs).

#### Algoritmo de treinamento

1-) Iniciar a conexão de  $w$  aleatoriamente

2-) Repita

erro  $\leftarrow$  “não existe”

para cada par de treinamento  $\{x(k), d(k)\}$  faça

$u \leftarrow x^T(k) * w$

$y \leftarrow \text{sinal}(u)$

Se  $d(k) \neq y$   
Então  $w \leftarrow w + \eta * (d(k) - y) \cdot x(k)$   
erro  $\leftarrow$  “Existe”  
Fim para  
Até erro = “não existe”

**Algoritmo de Teste (Classificação)**

- 1-) Apresentar padrão  $x$  a ser reconhecido
- 2-) Determinar a saída  $y$
- 3-) se  $y = 1$   
então  $x \in C_1$   
senão  $x \in C_2$

**Época de treinamento**

Apresentação total de todos os padrões.

## 4 O Adaline e a Regra Delta

### 4.1 Introdução

O Adaline (Adaptive Linear Element, chamado mais tarde de Adaptive Linear Neuron), idealizado por Widrow e Hoff (1960), é uma Rede Neural que utiliza um algoritmo supervisionado para minimizar o erro entre as entradas e saídas.

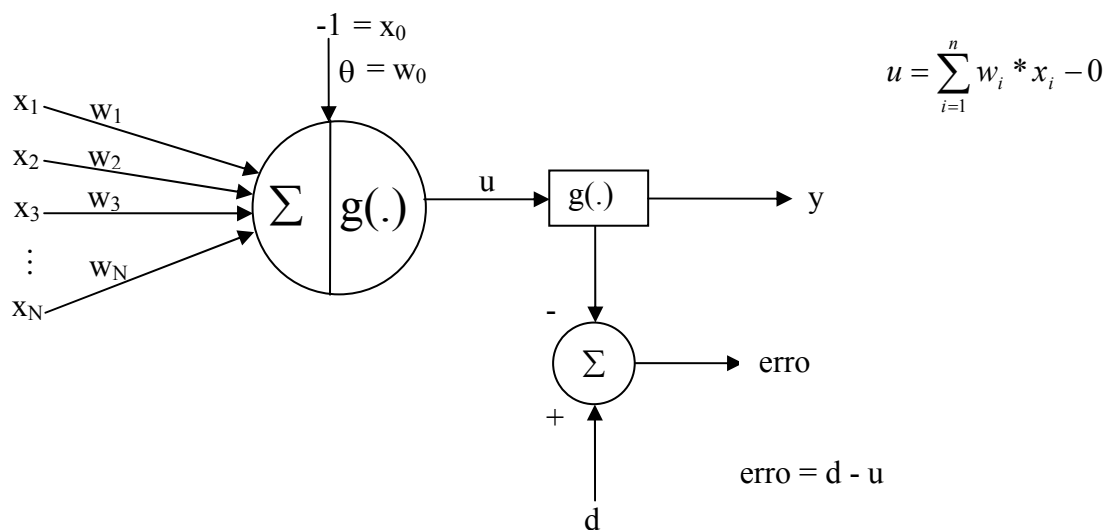
As principais contribuições do Adaline são:

I-) Invenção do algoritmo de treinamento conhecido como regra delta (Precursor da Regra Delta Generalizada).

II-) Pesquisas e aplicações pioneiras.

III-) Aplicações em processamento de sinais desde 1960 até hoje.

A arquitetura básica do Adaline é constituída por:



Onde:

u é a saída do combinador linear.

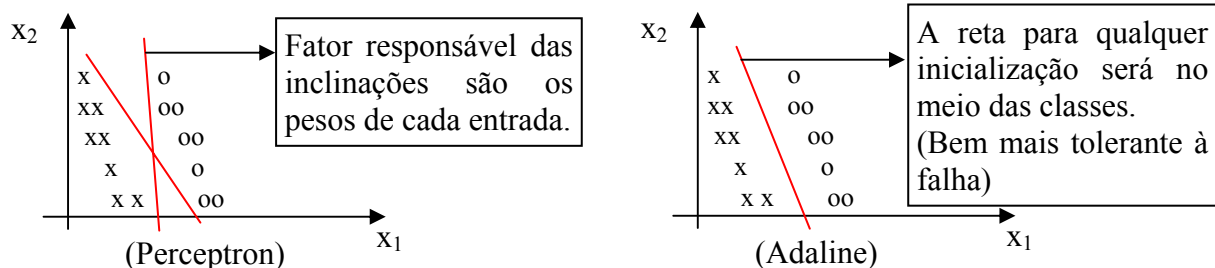
y é a saída do neurônio.

d é a saída desejada.

g(.) é a função degrau

O ajuste dos pesos no algoritmo de aprendizagem (regra delta) do Adaline é feita minimizando a diferença entre a saída desejada (d) e a saída do combinador linear (u). Esta metodologia torna o sistema mais robusto (tolerante) em relação às flutuações dos ruídos nos sinais de entradas.

O Adaline funciona como um filtro que separa duas classes linearmente separáveis.



## 4.2 A Regra Delta

A regra delta, utilizada no aprendizado do adaline, faz a adaptação (ajuste) dos pesos da rede usando a minimização do erro quadrático entre (u) e (d).

Na realidade, pretende-se determinar um “w” ótimo tal que o erro quadrático (e) sobre todo o conjunto de entradas seja o mínimo possível, ou seja:

$$E(w^*) \leq E(w), \forall w \in R^{(i)}$$

Para “p” padrões de entradas (de treinamento), a função erro quadrático é definida por:

$$E(w) = \frac{1}{2} \sum_{k=1}^p (d(k) - u(k))^2$$

$$E(w) = \frac{1}{2} \sum_{k=1}^p (d(k) - (\sum x_i(k) * w_i - \theta))^2 = \frac{1}{2} \sum_{k=1}^p (d(k) - (w^T * x(k) - \theta))^2$$

A minimização do erro é feita com a utilização do vetor gradiente ( $\nabla$ ) de “E(w)” em relação a “w”, a seja:

$$\nabla_E(w) = \frac{\partial E(w)}{\partial w}$$

$$\nabla_E(w) = \sum_{k=1}^p [(d(k) - (w^T * x(k) - \theta)) * (-x(k))]$$

$$\nabla_E(w) = - \sum_{k=1}^p [(d(k) - u(k)) * x(k)]$$

Logo, a adaptação (ajuste) dos pesos deve ser efetuada na direção oposta ao gradiente:

$$\Delta w = -\eta * \nabla_E(w)$$

$$\Delta w = -\eta \sum_{k=1}^p [(d(k) - u(k)) * x(k)]$$

$$w(t+1) = w(t) + \eta \sum_{k=1}^p [(d(k) - u(k)) * x(k)]$$

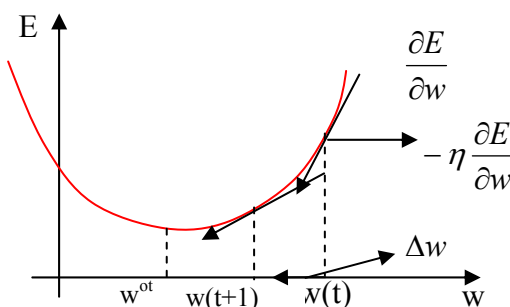
Pode-se atualizar “w” discretamente após a apresentação de cada padrão de treinamento “k”, ou seja:

$$w(t+1) = w(t) + \eta (d(k) - u(k)) * x(k), k = 1$$

$$w \leftarrow w + \eta * (d(k) - u(k)) * x(k)$$

## 4.3 Interpretação Geométrica (Regra Delta)

O método de ajuste dos pesos (Regra Delta) pode ser interpretada geometricamente da seguinte forma:



O  $\eta$  não pode ser grande por causa da estabilidade em convergir para o ótimo.

- I-) Os pesos são alterados iterativamente ao longo da superfície de erro.
- II-) O método sempre procura o mínimo.
- III-) A alteração é feita na direção oposta de  $\nabla e$

#### 4.4 Algoritmo do Adaline

As características do treinamento do Adaline são:

- Entradas: reais.
- Saídas do combinador: reais.
- Saídas do neurônio: Bipolares.
- Função de ativação: Degrau Bipolar.
- Pesos: inicializados aleatoriamente.
- Treinamento: supervisionado.

Sejam as variáveis e parâmetros:

$$x(k) = [-1, x_1(k), x_2(k), \dots, x_N(k)]^T$$

$d(k)$  = saída desejada em relação ao padrão  $k$ .

$u(k)$  = saída do combinador em relação ao padrão  $k$ .

$\eta$  = taxa de aprendizagem ( $0 < \eta < 1$ ).

$$w = [\theta, w_1, w_2, \dots, w_N]^T$$

O ajuste de pesos é efetuado utilizando a regra delta:

$$w(t+1) = w(t) + \eta * (d(k) - u(k)) * x(k)$$

O critério de parada é estipulado em função do erro quadrático médio definido,

$$eqm(t) = \frac{1}{p} \sum_{k=1}^p (d(k) - u(k))^2$$

O algoritmo converge quando o “eqm” entre duas épocas sucessivas for suficientemente pequena, ou seja:

$$Eqm(t+1) - eqm(t) \leq \varepsilon \quad \text{Onde } \varepsilon \text{ representa a precisão}$$

##### Algoritmo de treinamento

- 1-) Inicializar vetor  $w$  aleatoriamente
- 2-)  $EQM\_ANT \leftarrow BIG\_M \rightarrow \boxed{\text{inf}}$
- 3-)  $EQM\_ATUAL \leftarrow EQM \rightarrow \boxed{\text{valor de retorno da função}}$
- 4-) Enquanto  $(EQM\_ATUAL - EQM\_ANT) > \varepsilon$ 
  - $EQM\_ANT \leftarrow EQM\_ATUAL$
  - Para cada par  $(x(k), d)$  faça
    - $u \leftarrow x^T(k) * w$
    - $w \leftarrow w + \eta * (d(k) - u) * x(k)$
  - fim\_para
  - $EQM\_ATUAL \leftarrow EQM$
- Fim\_enquanto
- 5-) Imprima ( $w$ )
- 6-) Fim

O teste de novos padrões (não pertencentes ao conjunto de treinamento) pode ser realizado da seguinte forma:

**Algoritmo de teste (classificação)**

- 1-) Apresentar padrão  $x(k)$
- 2-)  $u \leftarrow x^T(k) * w$
- 3-)  $y \leftarrow \text{Degrau}(u)$
- 4-) se  $y = 1$ 
  - então  $x(k) \in \text{classe A}$
  - senão  $x(k) \in \text{classe B}$
- 5-) Fim

## 5 Redes Perceptron Multicamadas (PMC) (MultLayer Perceptron (MLP))

### 5.1 Introdução

As redes perceptron multicamadas são redes feedforward constituída por:

- 1 camada de entrada
- Pelo menos uma camada neural escondida
- 1 camada neural de saída

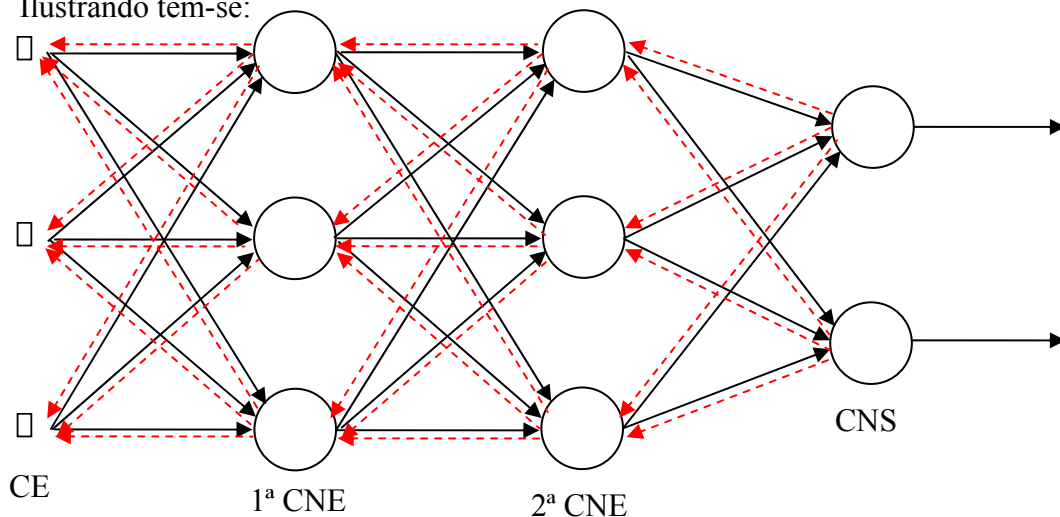
(Rede mais utilizada)  
Treinamento supervisionado

O processo de treinamento destas redes é realizado através do algoritmo “Backpropagation”, conhecido também como Regra Delta Generalizada. Este algoritmo constitui-se de dois passos principais, ou seja.

**I-) Passo Forward:** um padrão é aplicado nas entradas da rede e as informações são propagadas camada a camada até as suas saídas.

**II-) Passo Backward:** A partir das saídas, calcula-se o erro que será propagado (De Volta) objetivando o ajuste da matriz “w”.

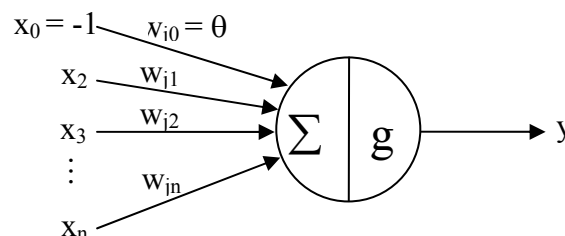
Ilustrando tem-se:



Quanto mais camadas, as mais próximas da entrada não contribuem muito na rede

Onde as setas contínuas ilustram o passo Forward, sendo que as pontilhadas o passo Backward.

Cada neurônio da rede é constituído da seguinte forma:



Onde  $g(\cdot)$  deve ser uma função contínua e diferenciável em todo o seu domínio.

As principais aplicações das redes Perceptron são:

**A-) Aproximador universal de funções**

**B-) Reconhecimento de padrões**

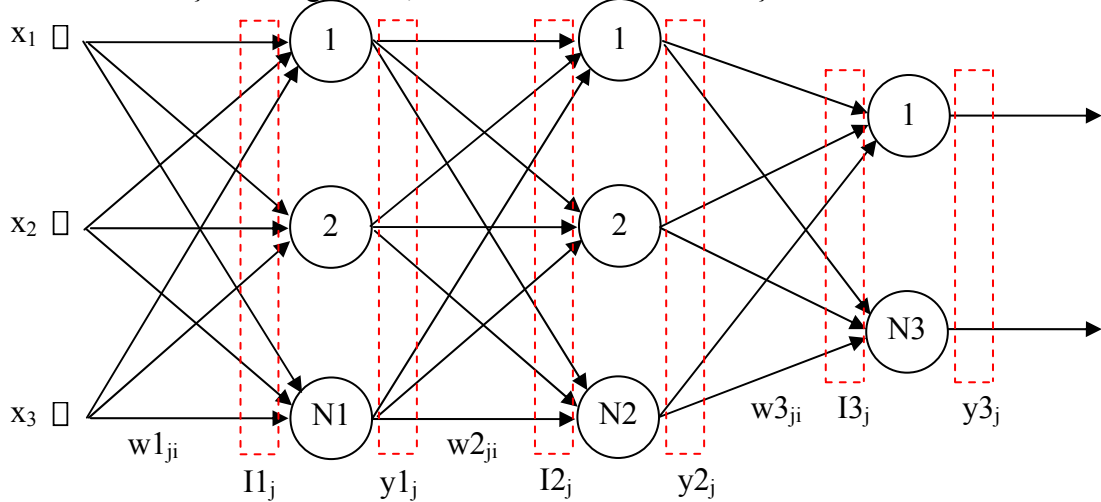
- Fronteiras não lineares
- Conjuntos conexos e não conexos
- Conjuntos desconexos



- C-) Previsão de séries temporais
- D-) Identificação de sistemas e controle
- E-) Processamento de imagem

## 5.2 Algoritmo Backpropagation (Regra Delta Generalizada)

Para a derivação do algoritmo, assume-se a seguinte notação.



Onde:

$wl_{ji}$  é o peso sináptico conectando o  $j$ -ésimo neurônio da camada “ $l$ ” ao  $i$ -ésimo neurônio da camada  $(l-1)$ .

$I1_j$  é a entrada da camada do  $j$ -ésimo neurônio da camada “ $l$ ”.

$$I1_j = \sum_{i=0}^N w1_{ji} * x_i$$

$$I2_j = \sum_{i=0}^N w2_{ji} * y1_i$$

$$I3_j = \sum_{i=0}^N w3_{ji} * y2_i$$

$y1_i$  é a saída do  $j$ -ésimo neurônio da camada “ $l$ ”.

$$y1_j = g(I1_j)$$

$$y2_j = g(I2_j)$$

$$y3_j = g(I3_j)$$

As funções sinais de erro são definidas por:

– **Erro quadrático:** (referente à camada de saída)

$$E(k) = \frac{1}{2} \sum_{j=1}^{N3} (d_j(k) - y3_j(k))^2 \{ \text{relativo ao } k - \text{ésimo padrão} \}$$

– **Erro quadrático médio:**

$$E_M = \frac{1}{p} \sum_{k=1}^p E(k) \{ \text{relativo aos " } p \text{ " padrões de treinamento} \}$$

O objetivo do processo de aprendizagem é ajustar os parâmetros livres (matriz  $w$ ) da rede a fim de minimizar “ $E_M$ ”.

O ajuste pode ser feito em dois casos como descrito a seguir:

**Caso 1 – Neurônios da camada de saída**

Similar ao processo de ajuste do adaline (regra delta) utilizando a regra de diferenciação em cadeia, tem-se:

$$\nabla E = \frac{\partial E}{\partial w_{3ji}} = \frac{\partial E}{\partial y_{3j}} * \frac{\partial y_{3j}}{\partial I_{3j}} * \frac{\partial I_{3j}}{\partial w_{3ji}} \quad (1)$$

mas:

$$\frac{\partial I_{3j}}{\partial w_{3ji}} = y_{2i} \quad (2)$$

$$\frac{\partial y_{3j}}{\partial I_{3j}} = g'(I_{3j}) \quad (3)$$

$$\frac{\partial E}{\partial y_{3j}} = -(d_j - y_{3j}) \quad (4)$$

Substituindo (2), (3), e (4) em (1) obtém-se:

$$\frac{\partial E}{\partial w_{3ji}} = -(d_j - y_{3j}) * g'(I_{3j}) * y_{2i}$$

Logo, o ajuste deve ser feito em direção oposta ao gradiente, ou seja:

$$\Delta w_{3ji} = -\eta * \frac{\partial E}{\partial w_{3ji}}$$

$$\Delta w_{3ji} = \eta * \delta_{3j} * y_{2i}$$

onde:

$$\delta_{3j} = (d_j - y_{3j}) * g'(I_{3j})$$

ou ainda:

$$w_{3ji}(t+1) = w_{3ji}(t) + \eta * \delta_{3j} * y_{2i}$$

$$w_{3ji} \leftarrow w_{3ji} + \eta * \delta_{3j} * y_{2i}$$

**Caso 2 – Neurônios das camadas escondidas**

Nos neurônios das camadas escondidas não temos as saídas desejadas correspondentemente o sinal de erro para um neurônio escondido deve ser determinado em termos de sinais de erro de todos os neurônios aos quais o neurônio escondido está conectado.

Para a segunda camada tem-se:

$$\nabla E = \frac{\partial E}{\partial w_{2ji}} = \frac{\partial E}{\partial y_{2j}} * \frac{\partial y_{2j}}{\partial I_{2j}} * \frac{\partial I_{2j}}{\partial w_{2ji}} \quad (5)$$

Mas:

$$\frac{\partial I_{2j}}{\partial w_{2ji}} = y_{1i} \quad (6)$$

$$\frac{\partial y_{2j}}{\partial I_{2j}} = g'(I_{2j}) \quad (7)$$

$$\frac{\partial E}{\partial y_{2j}} = \sum_{k=1}^{N3} \frac{\partial E}{\partial I_{3k}} * \frac{\partial I_{3k}}{\partial y_{2j}} = \sum_{k=1}^{N3} \frac{\partial E}{\partial I_{3k}} * \frac{\partial (\sum_{k=1}^{N3} w_{3kj} * y_{2j})}{\partial y_{2j}} = \sum_{k=1}^{N3} \frac{\partial E}{\partial I_{3k}} * w_{3kj} = -\sum_{k=1}^{N3} \delta_{3k} * w_{3kj} \quad (8)$$

Substituindo (6), (7) e (8) em (5), tem-se:

$$\frac{\partial E}{\partial w_{2_{ji}}} = \left( - \sum_{k=1}^{N3} \delta 3_k * w_{3_{kj}} \right) * g'(I2_j) * g'(I2_j) * y1_i$$

Logo, o ajuste  $w_{2_{ji}}$  deve ser efetuado na direção oposta ao gradiente:

$$\Delta w_{2_{ji}} = -\eta * \frac{\partial E}{\partial w_{2_{ji}}} \quad \Delta w_{2_{ji}} = \eta * \delta 2_{ji}$$

Onde:

$$\delta 2_{ji} = g'(I2_j) * \sum_{k=1}^{N3} (\delta 3_k * w_{3_{kj}})$$

Ou ainda:

$$w_{2_{ji}}(t+1) = w_{2_{ji}}(t) + \eta * \delta 2_{ji} * y1_i$$

$$w_{2_{ji}} \leftarrow w_{2_{ji}} + \eta * \delta 2_{ji} * y1_i$$

Para a 1ª camada:

$$\nabla E = \frac{\partial E}{\partial w_{1_{ji}}} = \frac{\partial E}{\partial y1_j} * \frac{\partial y1_j}{\partial I1_j} * \frac{\partial I1_j}{\partial w_{1_{ji}}} \quad (9)$$

mas:

$$\frac{\partial I1_j}{\partial w_{1_{ji}}} = X_i \quad (10)$$

$$\frac{\partial y1_j}{\partial I1_j} = g'(I1_j) \quad (11)$$

$$\frac{\partial E}{\partial y1_j} = \sum_{k=1}^{N2} \frac{\partial E}{\partial I2_k} * \frac{\partial I2_k}{\partial y1_j} = \sum_{k=1}^{N2} \frac{\partial E}{\partial I2_k} * \frac{\partial (\sum_{k=1}^{N2} w_{2_{kj}} * u1_j)}{\partial y1_j} = \sum_{k=1}^{N2} \frac{\partial E}{\partial I2_k} * w_{2_{kj}} = - \sum_{k=1}^{N2} \delta 2_k * w_{2_{kj}} \quad (12)$$

Substituindo (10), (11) e (12) em (9), tem-se:

$$\frac{\partial E}{\partial w_{1_{ji}}} = \left( - \sum_{k=1}^{N2} \delta 2_k * w_{2_{kj}} \right) * g'(I1_j) * x_i$$

O ajuste deve ser feito na direção oposta ao gradiente, ou seja:

$$\Delta w_{1_{ji}} = -\eta * \frac{\partial E}{\partial w_{1_{ji}}} = \eta * \delta 1_j * x_i$$

Onde:

$$\delta 1_j = g'(I1_j) * \left( \sum_{k=1}^{N2} \delta 2_k * w_{2_{kj}} \right)$$

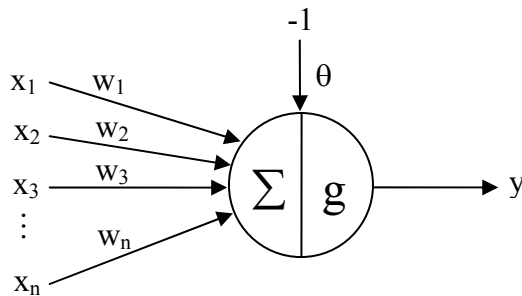
Ou ainda:

$$w_{1_{ji}}(t+1) = w_{1_{ji}}(t) + \eta * \delta 1_j * x_i$$

$$w_{1_{ji}} \leftarrow w_{1_{ji}} + \eta * \delta 1_j * x_i$$

### 5.3 PMC como classificadores

A rede PMC pode ser utilizada como classificadores de padrões em várias situações. A partir do modelo matemático do neurônio, tem-se:



$$u = \sum_{i=1}^n w_i * x_i - \theta$$

$$y = g(u)$$

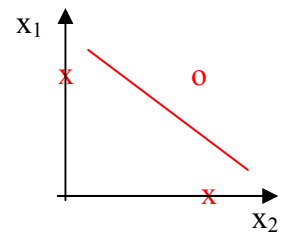
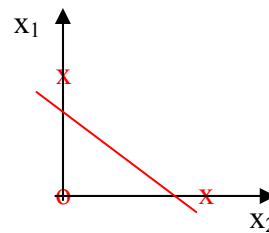
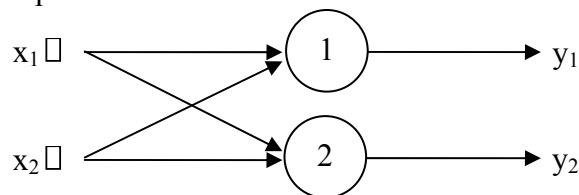
Por simplicidade de ilustração, considera-se que o neurônio possui duas entradas ou seja:

$$y = \begin{cases} 1, & \text{se } w_1 * x_1 + w_2 * x_2 - \theta \geq 0 \\ 0, & \text{se } w_1 * x_1 + w_2 * x_2 - \theta < 0 \end{cases}$$

As três situações que se tem para o PMC são:

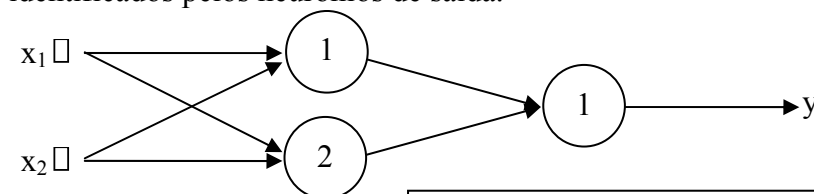
#### I-) Uma Camada

Cada neurônio separa o espaço de decisão em dois subconjuntos linearmente separáveis.

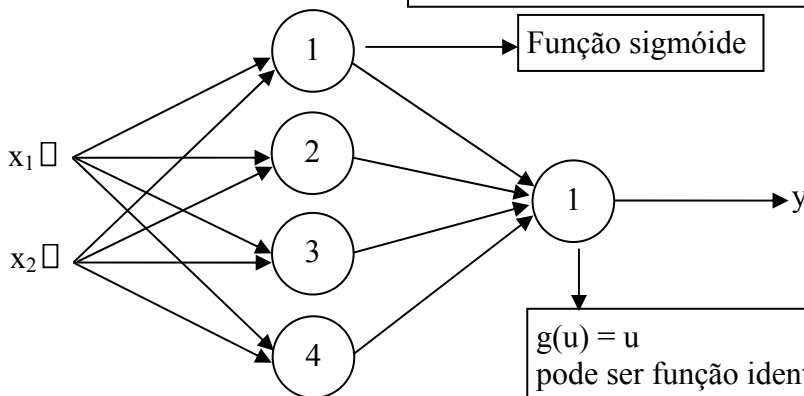
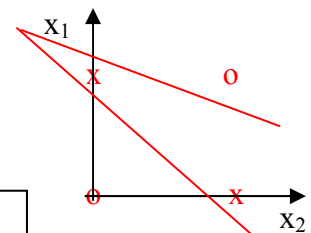


#### II-) Duas Camadas

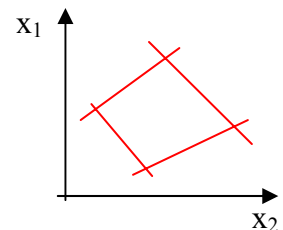
Os neurônios da segunda camada executam uma combinação linear (lógica) das regiões de decisão. Neste caso, padrões que estão dentro de regiões conexas, podem ser identificados pelos neurônios de saída.



Pode fazer:  
- Seleção  
- classificação de padrões convexos

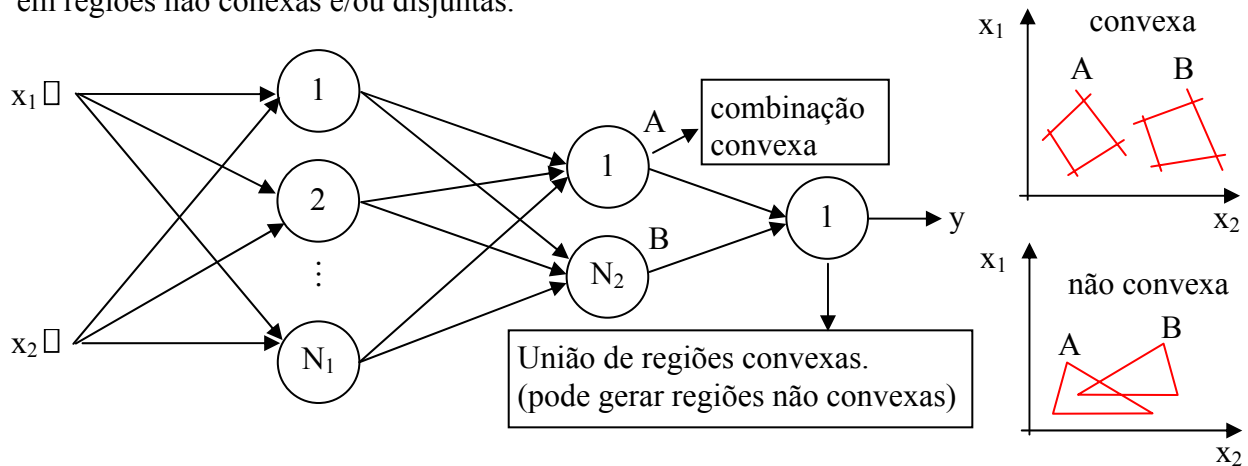


$g(u) = u$   
pode ser função identidade



### III-) Três Camadas

Nesta rede não há restrições de conectividade. Os neurônios da 3ª camada recebem como entradas um grupo de regiões conexas, e realizam uma combinação lógica que pode resultar em regiões não conexas e/ou disjuntas.



## 5.4 Aproximador Universal de Funções (CyBenko, 1989)

Uma rede PMC treinada com o algoritmo “Backpropagation” é capaz de implementar qualquer mapeamento não-linear que relacione as entradas e saídas.

Baseado no teorema de chadman (Kolmogorov)

### Teorema da aproximação universal

Dado que “g(.)” é limitada, monótona crescente e contínua. Então qualquer função contínua arbitrária “f” pode ser implementada por uma rede PMC com uma camada neural escondida de N neurônios, ou seja:

$$F(x_1, x_2, \dots, x_N) = \sum_{j=1}^N x_j * g\left(\sum_{i=1}^N w_{ij} - x_j - \theta\right)$$

Cujo erro de aproximação é dado por:

$$|F(x_1, x_2, \dots, x_N) - f(x_1, x_2, \dots, x_N)| < \varepsilon$$

Oferece a condição de existência

Nas equações acima, tem-se:

**Neurônios escondidos** → Saídas, implementam funções sigmóides.

**Neurônios de saída** → executa uma combinação linear das saídas dos neurônios escondidos. ( $x_1, x_2, \dots, x_n$  Definem os coeficientes)

O teorema da aproximação universal não fornece a quantidade exata de neurônios da camada escondida. Em certos casos, torna-se necessárias a utilização de duas ou mais camadas escondidas.

### Domínio da aproximação

A rede PMC poderá ser utilizada como um aproximador de funções somente para aqueles novos dados de entrada que estejam dentro do domínio de definição do conjunto de treinamento da qual a mesma foi treinada, ou seja:

$$X_1^{new} \in [X_1^{min}, X_1^{max}]$$

$$X_2^{new} \in [X_2^{min}, X_2^{max}]$$

(...)

$$X_F^{new} \in [X_F^{min}, X_F^{max}]$$

conjunto de treinamento

## 5.5 Implementação do PMC

Sejam as variáveis e parâmetros:

$$\mathbf{x}(k) = [-1, x_1(k), x_2(k), \dots, x_N(k)]^T$$

$$\mathbf{d}(k) = [d_1(k), d_2(k), \dots, d_{N_s}(k)]^T$$

1-) Inicializar todas as matrizes  $\mathbf{W}$  aleatoriamente

2-)  $EQM\_ANT \leftarrow BIG\_M$ ;  $EQM\_ATUAL \leftarrow E_M$

3-) Enquanto  $|EQM\_ATUAL - EQM\_ANT| > \varepsilon$

$EQM\_ANT \leftarrow EQM\_ATUAL$

Para cada par  $\{\mathbf{x}(k), \mathbf{d}(k)\}$  faça:

$$I_1 \leftarrow w_1 * x(k); y_1 \leftarrow \text{Sigmoid}(I_1)$$

$$I_2 \leftarrow w_2 * y_1; y_2 \leftarrow \text{Sigmoid}(I_2)$$

$$I_3 \leftarrow w_3 * y_2; y_3 \leftarrow \text{Sigmoid}(I_3)$$

Determinar  $\delta_3$ ; ajustar  $w_3$

Determinar  $\delta_2$ ; ajustar  $w_2$

Determinar  $\delta_1$ ; ajustar  $w_1$

Fim para

$EQM\_ATUAL \leftarrow E_M$

Fim enquanto

4-) Fim

$N_s \rightarrow$  número de neurônios de saída  
 $E_M \rightarrow$  erro médio

### Derivação

$$g'(x) = g(x) * (1 - g(x))$$

### Teste (aproximador funcional)

1-) Apresentar padrão  $\mathbf{x}(k)$

2-) Determinar  $y_3$  (caso de teste)

3-) Valor da função  $f$  em  $\mathbf{x}(k)$  é igual a  $y_3$

### Teste (classificação)

1-) Apresentar padrão  $\mathbf{x}(k)$

2-) Determinar  $y_3$

3-) Caso  $y_3 = C_1 \rightarrow$  Classe 1

$y_3 = C_2 \rightarrow$  Classe 2

(...)

## 5.6 Aspectos da rede perceptron

Apresenta-se a seguir algumas técnicas que auxiliam para incrementar o desempenho do treinamento do PMC.

### A-) Validação cruzada (cross validation)

Metodologia utilizada para testar o poder de generalização de uma PMC. Neste contexto, deve-se seguir os passos adiantes:

I-) Particionar o conjunto de dados disponíveis em 2 subconjuntos:

- **conjunto de treinamento**  $\rightarrow$  utilizado para treinar a rede (80 a 90% do total).

- **conjunto de teste**  $\rightarrow$  utilizado para avaliar se a rede está generalizando de forma satisfatória (10 a 20% do total) (over fitting).

II-) Utilizar o mesmo conjunto de treinamento para ajustar todas as topologias candidatas.

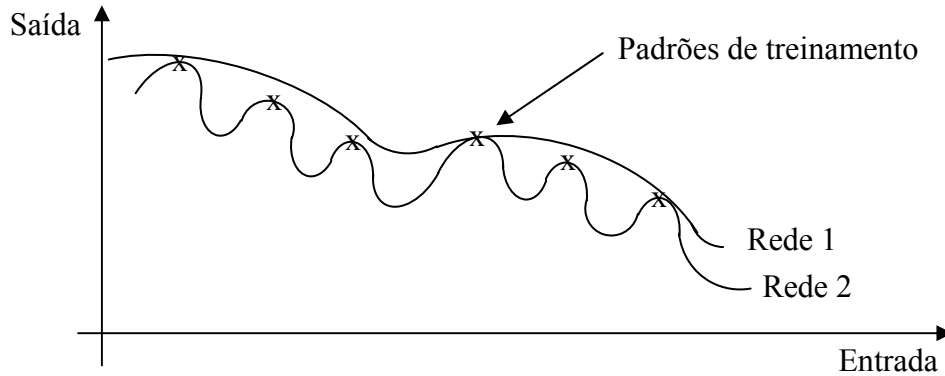
III-) Após o treinamento, utilizar o conjunto de teste para avaliar e escolher (validar) o melhor modelo (aquele que generaliza melhor).

Muitos dados usar 90%,  
poucos usar 80%

**Obs1:** Em certos casos, uma determinada topologia pode ter desempenho satisfatório no treinamento, mas durante a fase de validação (aplicação do conjunto de teste) a rede produz resultados insatisfatórios (não consegue generalizar).

**Obs2:** O aumento de neurônios e de camadas não significa que a rede irá generalizar melhor.

**Obs3:** Para duas topologias que estão generalizando com o mesmo grau de precisão, deve-se optar por aquilo com menor número de neurônios.



### B-) Inserção de termo de momento

A velocidade do algoritmo Backpropagation pode ser aumentado (sem perigo de instabilidade) através da inclusão de um termo de “momento” ( $\alpha$ ), ou seja:

$$\Delta w_{ji}(t+1) = \alpha * \Delta w_{ji}(t) + \eta * d_j * y_i$$

Ou ainda:

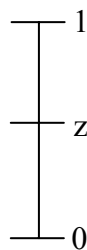
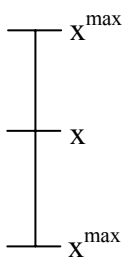
$$w_{ji}(t+1) = w_{ji}(t) + \alpha * (w_{ji}(t) - w_{ji}(t-1)) + \eta * d_j * y_i$$

Inserção de “ $\alpha$ ” não deixa a rede instável

### C-) Normalização dos dados

As variáveis referentes às entradas da rede devem ser normalizadas para a faixa  $[0, 1]$  se estivermos utilizando a função sigmóide ou então, para  $[-1, 1]$  se utilizarmos a tangente hiperbólica.

$$x \in [x^{\min}, x^{\max}] \Leftrightarrow z \in [0, 1]$$



$$z = \frac{x - x_{\max}}{x_{\max} - x_{\min}}$$

$$x = x_{\min} + z(x_{\max} - x_{\min})$$

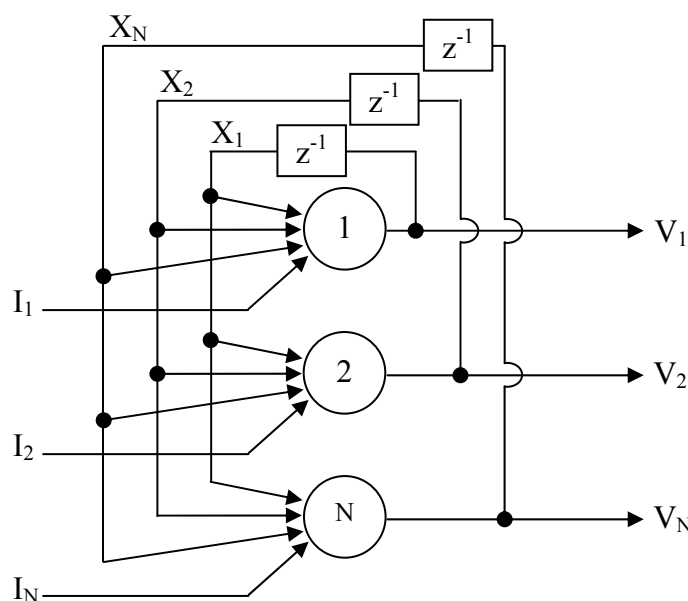
## 6 Redes Recorrentes de Hopfield (1986, J. Hopfield)

Redes Recorrentes são aquelas em que a saída da rede pode ser conectada às suas entradas. Esta arquitetura (com realimentação) possui as seguintes características:

- são dinâmicas
- possuem memória
- facilidade de implementação em hardware

### 6.1 Redes de Hopfield

A rede de Hopfield é composta por uma única camada, onde todos os neurônios são completamente interconectados. Isto é, todos os neurônios da rede são conectados a todos os outros em si próprio (todas as saídas da rede realimentam todas as suas entradas) ilustrando, tem-se:



Completamente retro-alimentada  
muito utilizado principalmente  
em otimização.

#### Utilização:

- otimização linear
- otimização não linear
- otimização combinatorial
- Programação dinâmica
- Memórias associativas

A equação dinâmica de cada neurônio é dada por:

$$\dot{u}_j(t) = \sum_{i=1}^N w_{ji} * V_i(t) + I_j^b, j = 1 \dots N \quad (1)$$

$$V_j(t) = g(\dot{u}_j(t))$$

(2) —————> versão em tempo contínuo

Onde:

$\dot{u}_j(t)$  é o estado interno do neurônio j.

$V_j(t)$  é a saída do neurônio j.

$X_j(t)$  é a entrada do neurônio j.

$W_{ji}$  é o peso sináptico entre os neurônios j e i.

$I_j^b$  é o limiar aplicado ao neurônio j.

Desta forma, as respostas da rede de Hopfield são dinâmicas, isto é:

I-) Aplica-se um conjunto de entradas “x”.

II-) As saídas (V) são calculadas e retro-alimentadas às entradas.

III-) A saída é então recalculada e o processo é repetido iterativamente.

IV-) Essas iterações produzem mudanças nas saídas cada vez menores até que todas as saídas tornam-se constantes.



Interpretando as equações (1) e (2) tem-se:

- para  $t = t_0 \rightarrow$  entrada  $x(t)$  gera  $V(t_0)$ .
- para  $t = t_1 \rightarrow$  entrada  $x(t_1) = V(t_0)$  gera  $V(t_1)$ .
- para  $t = t_2 \rightarrow$  entrada  $x(t_2) = V(t_1)$  gera  $V(t_2)$ .
- para  $t = t_N \rightarrow$  rede estabilizada  $\{V(t_N + 1) = V(t_N)\}$ .

Uma versão em tempo discreto para as equações (1) e (2) é dada por:

$$u_j(k) = \sum_{i=1}^N w_{ji} * V_i(k) + I_j^b \quad (3)$$

$$V(k+1) = g(u_j(k)) \quad (4) \longrightarrow \boxed{\text{versão em tempo discreto}}$$

Ou ainda, na forma matricial:

$$u(k) = w * V(k) + I^b$$

$$V(k+1) = g(u(k)) \quad (\text{versão em tempo discreto})$$

## 6.2 A estabilidade da rede de Hopfield

Para analisar a estabilidade e evolução desta rede, Hopfield definiu uma função de energia (função de Lyapunov) que está associada à sua dinâmica, ou seja:

$$E(t) = \frac{1}{2} V(t)^T * w * V(t) - V(t) * I^b \quad (5)$$

A estabilidade assintótica pode ser demonstrada através da aplicação do segundo método de Lyapunov, ou seja, deve-se mostrar que (de baixo de certas condições) as derivadas temporais de “E(t)” são não crescentes ( $\dot{E}(t) \leq 0$ ). A partir de (5), tem-se:

$$\dot{E}(t) = \frac{dE(t)}{dt} = (\nabla_V E(k))^T * \dot{V}(k) \quad (6)$$

Impondo a condição de que W seja simétrico, obtém-se

$$\nabla_V E(k) = -w * V(k) - I^b$$

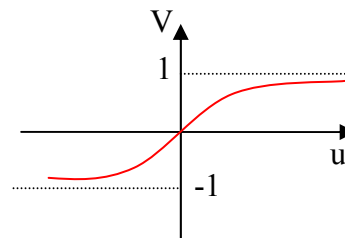
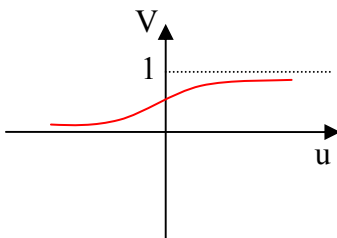
A partir de (2) conclui-se que:

$$\nabla_V E(k) = -\dot{u}(t) \quad (7)$$

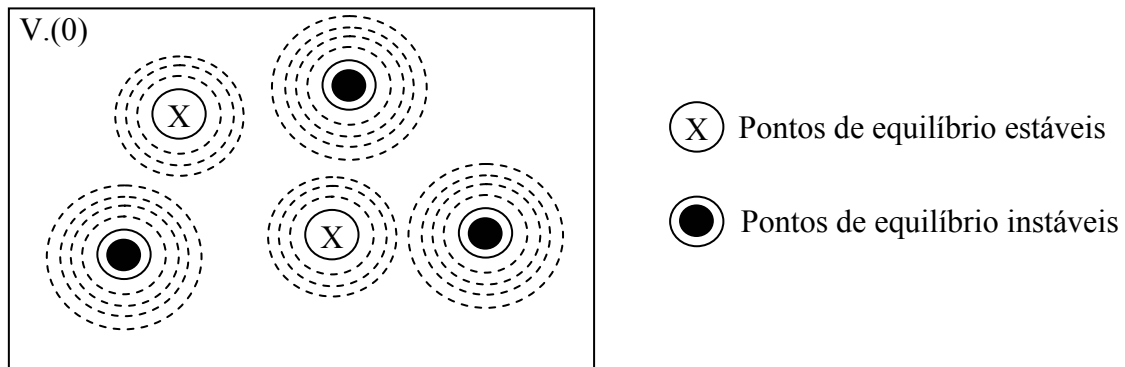
Logo, substituindo (7) em (6):

$$\dot{E}(t) = -\dot{u}(t)^T * \dot{V}(t) = -\sum_{j=1}^N \dot{u}_j(t) * \dot{V}_j(t) = -\sum_{j=1}^N \dot{u}_j(t) * \frac{\partial V_j(t)}{\partial \dot{u}_j(t)} * \frac{\partial \dot{u}_j(t)}{\partial t} = -\sum_{j=1}^N (\dot{u}_j(t))^2 * \frac{\partial V_j(t)}{\partial \dot{u}_j(t)}$$

Como “ $(\dot{u}_j(t))^2$ ” é sempre positivo, precisa-se verificar apenas que  $\partial V_j(t) / \partial \dot{u}_j(t) \geq 0$ , para se mostrar que “ $\dot{E}(t) \leq 0$ ”. Isso é trivialmente verdadeiro desde que a função “g(.)” seja monótona crescente. Utilizando a função sigmóide ou tangente hiperbólica tem-se:



Portanto, considerando a condição que “w” é simétrico, os resultados anteriores estabelecem que dado qualquer conjunto de condições iniciais “V(0)”, a rede convergirá para um ponto de equilíbrio estável que representará a solução final para o problema considerado. Ilustrando tem-se:



### Projetando Redes de Hopfield

O projeto de redes de Hopfield consiste na obtenção, de forma explícita, da matriz “w” e do vetor  $I^b$  que corresponde a alguma aplicação específica. Em suma, tem-se:

- A rede de Hopfield dispensa treinamento.
- A matriz “w” deve ser simétrica.
- A função de Energia associada à aplicação deve ter a forma da equação (5).
- Normalmente, “w” e “ $I^b$ ” são derivados a partir de especificações de “E(t)”.

A operação da rede de Hopfield pode ser resumida da seguinte forma::

- I-) Definir a matriz “w” e o vetor  $I^b$ .
- II-) Apresentar um vetor de entrada “x”, ou seja;  $V(0) = x$ .
- III-) Iteragir a rede até a convergência para um ponto de equilíbrio.
- IV-) A solução do sistema é dado pelo ponto de equilíbrio (problema).

### 6.3 Memórias Associativas (aprender por associação)

Uma das aplicações da rede de Hopfield são as memórias associativas ou memórias endereçáveis pelo conteúdo. A função das memórias associativas é recuperar um padrão previamente armazenado nela, dado uma apresentação incompleta ou duvidosa do padrão.

A forma mais simples de determinar a matriz “w” e “ $I^b$ ” para a memória é através do método do produto externo, ou seja:

Dado “p” vetores  $Z$ (memórias) com “N” dimensão (neurônios) tem-se:

$$w = \frac{1}{N} \sum_{k=1}^P Z(k) * Z(k)^T; \quad e \quad I^b = 0$$

(postulado generalizado de HEBB)

Para as memórias associativas, a matriz “w” deve possuir diagonal nula ( $w_{kk} = 0$ ). Reescrevendo a equação (8), tem-se:

$$w = \frac{1}{N} \sum_{k=1}^P Z(k) * Z(k)^T - \frac{P}{N} * I \quad \boxed{\text{Matriz identidade}}$$

#### Obs:

w é simétrico → estabilidade garantida.

$Z(k)$  formado com elementos -1 e 1.

$g(\cdot)$  é a função TGH ou sinal.

A capacidade de armazenamento de padrões nas memórias associativas são definidas por:

Para 99% de acertos com até no máximo 40 % de bits ruidosos:  $P_{MAX} \approx \frac{N}{2 \ln N}$

Para 100% de aceros:  $P_{MAX} \approx \frac{N}{4 \ln N}$  ln → logaritmo neperiano

### 6.4 Algoritmo da Rede de Hopfield

- 1-) Definir  $w$  e  $I^b$
- 2-) Apresentar padrão ruidoso  $x$ .
- 3-)  $V\_ATUAL \leftarrow x$
- 4-)  $V\_ANT \leftarrow \{\text{valores aleatórios}\}$
- 5-) Enquanto ( $V\_ATUAL \neq V\_ANT$ ) faça
  - $V\_ANT \leftarrow V\_ATUAL$
  - $u \leftarrow w * V\_ANT + I^b$
  - $V\_ATUAL \leftarrow TGH(u) \text{ ou } \text{sinal}(u)$
- Fim enquanto
- 6-) Imprimir ("Resposta: ",  $V\_ATUAL$ )
- 7-) Fim

$\beta$  grande 100 ou 500

## 7 Sistemas variantes no tempo e Redes Neurais (sistemas dinâmicos)

Sistemas variantes no tempo são aqueles cujo comportamento é modelado em função do tempo. Geralmente, estes sistemas são dinâmicos e não-lineares, podendo ser descritos por uma série temporal.

A estimação/predição de valores atuais e futuros da série podem ser calculada em função de valores passados, ou seja:

$$x(t) = f(x(t-1), x(t-2), \dots, x(t-p))$$

$$\begin{aligned} p = 3 &\rightarrow \text{precisão de 3 valores passados} \\ p = n &\rightarrow \text{precisão de n valores passados} \end{aligned}$$

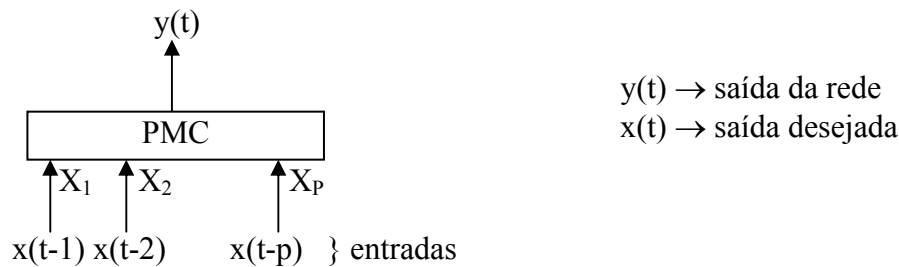
Onde “p” é a ordem do preditor/estimador.

A seguir, aborda-se duas arquiteturas de RNA que são utilizadas nos sistemas variantes no tempo.

### 7.1 Redes Neurais com atraso de tempo (TDNN)

A TDNN (Time-Delay Neural Network) é uma rede perceptron multicamadas cujas saídas são reaplicadas ao longo do tempo.

Dado um conjunto de dados observáveis “x(1), x(2), ..., x(N)”, onde “N” é o comprimento total da série. Uma TDNN pode ser utilizada para mapear o modelo físico associado ao sistema conforme a ilustração seguinte:



Desta forma a rede funciona como um estimulador de um passo à frente de ordem “p”, isto é, estima a saída “x(t)” dado “p” valores anteriores, onde o valor atual “x(t)” atua como a saída desejada da função desconhecida “f”.

Então, a TDNN é treinada para minimizar o erro quadrático do erro da predição dado por:

$$E(t) = x(t) - y(t), \quad p + 1 \leq t \leq N$$

#### Exemplo

Treinar uma TDNN (ordem 2) para estimular valores futuros de um processo. Representado pela seguinte série temporal:

$$x(t) = \begin{bmatrix} 0.11 & 0.32 & 0.53 & 0.17 & 0.98 & 0.67 \end{bmatrix}$$

t = 1   t = 2   t = 3   t = 4   t = 5   t = 6

O conjunto de treinamento para “p = 2” será então dada por:

	X <sub>1</sub>	X <sub>2</sub>	d
t = 3	x(2)	x(1)	x(3)
t = 4	x(3)	x(2)	x(4)
t = 5	x(4)	x(3)	x(5)
t = 6	x(5)	x(4)	x(6)

}

Ordem 2  
(p = 2)

3 ≤ t ≤ 6

t = 6      p = 2

nº linhas = t - p = 6 - 2 = 4

Para “ $p = 3$ ”, o conjunto seria dado por:

	$X_1$	$X_2$	$X_3$	$d$
$t = 4$	$x(3)$	$x(2)$	$x(1)$	$x(4)$
$t = 5$	$x(4)$	$x(3)$	$x(2)$	$x(5)$
$t = 6$	$x(5)$	$x(4)$	$x(3)$	$x(6)$

Ordem 3  
( $p = 3$ )

$4 \leq t \leq 6$

$$\begin{matrix} t = 6 & p = 3 \\ \text{n}^\circ \text{linhas} = t - p = 6 - 3 = 3 \end{matrix}$$

Depois de treinada a rede, estimula-se valores futuros do processo  $\{x(7), x(8), \dots\}$ , ou seja:

$$\begin{matrix} \text{Saída} & \text{Entrada} \\ x(7) = y(7) \rightarrow & [x(6) \ x(5)]^T \\ x(8) = y(8) \rightarrow & [x(7) \ x(6)]^T \\ x(9) = y(9) \rightarrow & [x(8) \ x(7)]^T \end{matrix} \left. \vphantom{\begin{matrix} \text{Saída} \\ x(7) = y(7) \rightarrow \\ x(8) = y(8) \rightarrow \\ x(9) = y(9) \rightarrow \end{matrix}} \right\} \begin{matrix} \text{Ordem 2} \\ (p = 2) \end{matrix}$$

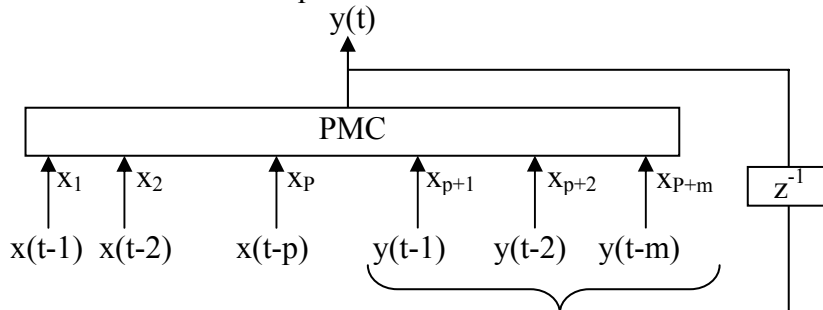
$$\begin{matrix} \text{Saída} & \text{Entrada} \\ x(7) = y(7) \rightarrow & [x(6) \ x(5) \ x(4)]^T \\ x(8) = y(8) \rightarrow & [x(7) \ x(6) \ x(5)]^T \\ x(9) = y(9) \rightarrow & [x(8) \ x(7) \ x(6)]^T \end{matrix} \left. \vphantom{\begin{matrix} \text{Saída} \\ x(7) = y(7) \rightarrow \\ x(8) = y(8) \rightarrow \\ x(9) = y(9) \rightarrow \end{matrix}} \right\} \begin{matrix} \text{Ordem 3} \\ (p = 3) \end{matrix}$$

## 7.2 Rede Perceptron Recorrente

Nesta arquitetura, a saída da rede é realimentada às entradas. Esta rede é capaz de modelar qualquer sistema que possa ser expresso por:

$$x(t) = f(x(t-1), x(t-2), \dots, x(t-p), y(t-1), y(t-2), \dots, y(t-m))$$

Onde “ $m$ ” é a quantidade de saídas a ser realimentada.



Onde:

$x(t)$  é a saída desejada.  
 $y(t)$  é a saída da rede.

### Utilização

- Identificação
- Controle
- Sistemas dinâmicos

Variações: Elman realimenta as entradas também  
Jordan tem mais saídas e realimenta todas

### Exemplo

Treinar uma rede perceptron recorrente para o seguinte processo:

$$x(t) = [ \ 0.17 \ 0.11 \ 0.25 \ 0.29 \ 0.58 \ 0.32 \ 0.75 \ ]$$

Utilizando  $p = 3$  e  $m = 2$ , tem-se:

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$d$
$t = 4$	$x(3)$	$x(2)$	$x(1)$	0	0	$x(4)$
$t = 5$	$x(4)$	$x(3)$	$x(2)$	$y(4)$	0	$x(5)$
$t = 6$	$x(5)$	$x(4)$	$x(3)$	$y(5)$	$y(4)$	$x(6)$
$t = 7$	$x(6)$	$x(5)$	$x(4)$	$y(6)$	$y(5)$	$x(7)$

Após o treinamento, estima-se os valores futuros:

$$\begin{array}{ccc} \text{Saída} & & \text{Entrada} \\ x(8) = y(8) \rightarrow & [x(7) \ x(6) \ x(5) \ y(7) \ y(6)]^T \\ x(9) = y(9) \rightarrow & [x(8) \ x(7) \ x(6) \ y(8) \ y(7)]^T \end{array}$$

### 7.3 Selecionando a topologia

A escolha da topologia é feita geralmente da seguinte forma:

#### A-) TDNN

- Se a operação do processo requerer uma ordem de predição pequena.
- Se o processo não requerer referências a valores prévios da série.

#### B-) Rede perceptron recorrente

- Se não tivermos qualquer idéia da ordem de predição.
- Se não há necessidade de referenciar valores prévios da série.

#### C-) Considerações práticas

Se a topologia escolhida não converge, tem-se as seguintes opções:

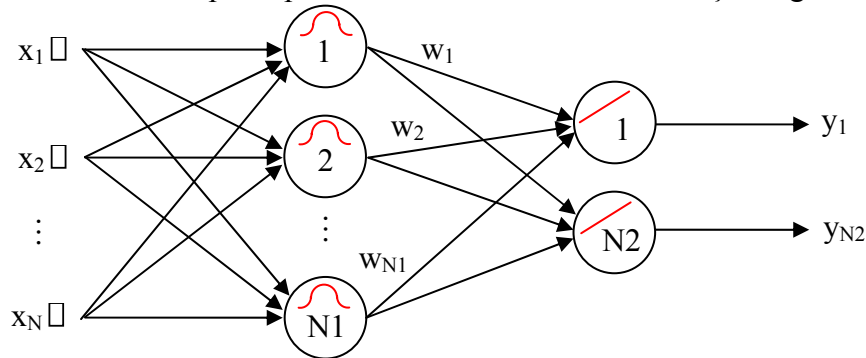
1. Incrementar a ordem de “p”.
2. Incrementar o número de neurônios.
3. Incrementar o número de camadas escondidas.

## 8 Redes de funções de base Radial

### 8.1 Introdução

Assim como no PMC, as RBF (Radial Bases Function – RBF) são arquiteturas de redes neurais que são normalmente utilizadas na aproximação de função e classificação de padrões em espaços com decisão muito grande.

Uma rede RBF é composta por 3 camadas conforme a ilustração seguinte:



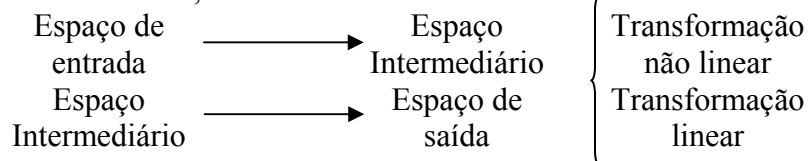
As camadas possuem as seguintes características:

1ª camada → entrada dos padrões.

2ª camada → neurônios escondidos, com função de ativação de base radial.

3ª camada → neurônios de saída possuem função de ativação linear.

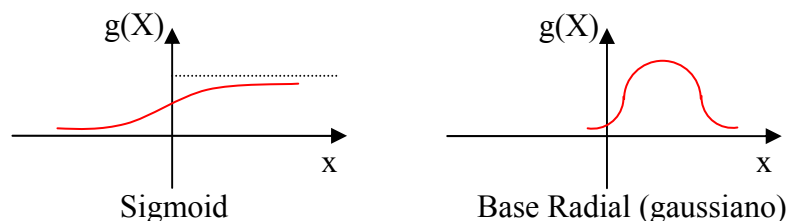
Esquemáticamente, tem-se:



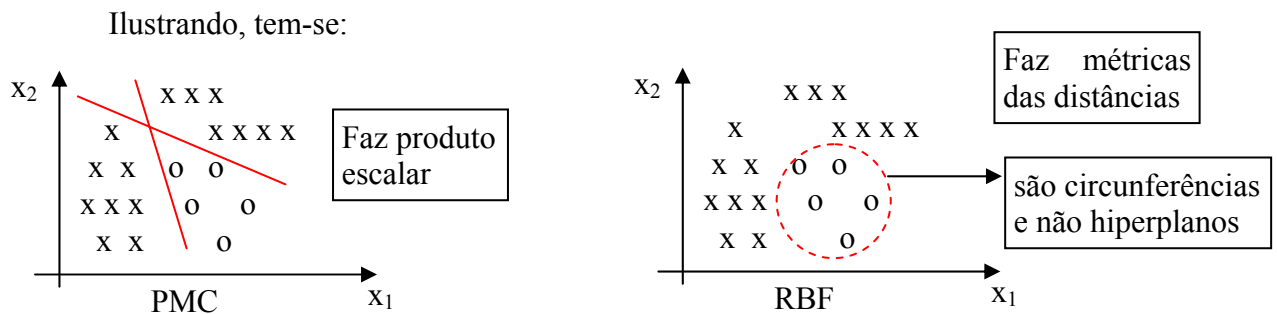
Assim, os neurônios de saída executam uma combinação linear das funções bases com entradas pelos neurônios da camada intermediária. Portanto, como no PMC, o teorema da aproximação universal é também satisfeito pela RBF.

No PMC, os neurônios escondidos formam funções Bases sigmóides que são diferentes de zero numa região infinitamente grande do espaço de entradas. Na RBF as respostas dos neurônios escondidos são diferentes de zero apenas dentro de uma região localizada “muito pequena”.

Ilustrando, têm-se:



Em relação aos problemas de classificação de padrões, as redes PMC conseguem separar os padrões referentes às diversas classes através de combinação de hiperplanos, enquanto que na RBF a curva de separação são funções radiais, onde critérios de distâncias métricas máximas em função de determinados pontos são especificadas usando a classificação dos dados de entrada.



## 8.2 Treinamento da RBF

O treinamento da RBF consiste geralmente de dois estágios, ou seja, treinamento da camada escondida seguido pelo treinamento da camada de saída.

### a-) Treinamento da camada escondida

Utiliza-se tipicamente um método não supervisionado, isto é, um algoritmo de agrupamento (clustering). O objetivo é ajustar o centro de cada gaussiana em regiões em que os vetores de entrada tenderão a se agrupar. Este procedimento pode ser feito através da regra dos “vizinhos mais próximos”.

#### Algoritmo “Clustering”

Nearest neighbour  
K-means

Inicializar os centros dos clusters  $w1_j \rightarrow j = 1 \dots N$

{ assume-se inicialmente igual aos N1 primeiros padrões }

Repita

{ agrupar todos os padrões com o centro mais próximos }

Para todo  $x(i)$  faça

Atribua  $x(i)$  ao conjunto  $\theta_j$  correspondendo à menor distância:  $\min_j \|x(i) - w1_j\|$

Fim\_para

{ computar as médias de cada conjunto }

Para todo  $w1_j$  faça

$$w1_j = \frac{1}{m_j} \sum_{x(i) \in \theta_j} x(i) \quad \{ m_j \text{ é o número de padrões em } \theta_j \}$$

Fim\_para

Até que haja mudança em atribuições de clusters de uma iteração para outra.

{ calcular o vetor de variâncias }

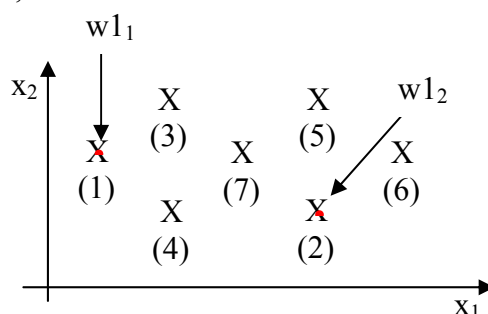
$$\sigma_j^2 = \frac{1}{m_j} \sum_{x(i) \in \theta_j} (x(i) - w1_j)^T * (x(i) - w1_j)$$

Fim

distâncias médias quadráticas

O centro é calculado de acordo com os elementos.

Ilustrando, tem-se:

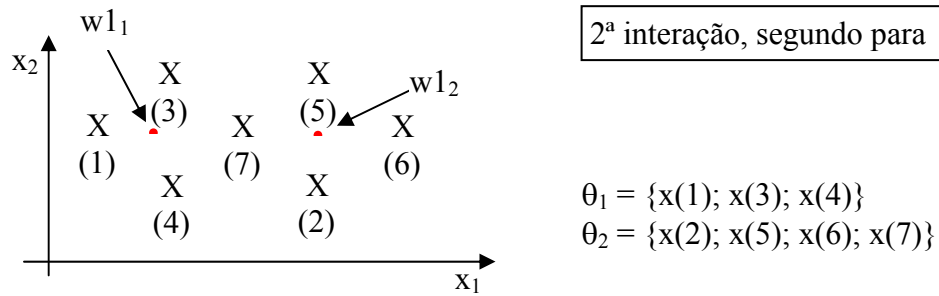


1ª interação, primeiro para

$$\theta_1 = \{x(1); x(3); x(4)\}$$

$$\theta_2 = \{x(2); x(5); x(6); x(7)\}$$





### b-) Treinamento da camada de saída

O treinamento da camada de saída é executado somente após a determinação dos parâmetros das funções bases que foi executada no treinamento da camada escondida.

A camada de saída da RBF é treinada utilizando o mesmo algoritmo ('Regra delta generalizada') usado na camada de saída do MLP, sendo que o conjunto de treinamento será formado por pares Entrada/Saída ( $\mu$ ,  $d$ ), onde os vetores  $\mu$  são especificados após o treino da primeira camada.

Normalmente a função de base gaussiana utilizada como função de ativação para os neurônios da camada escondida tem a forma dada por uma gaussiana, ou seja:

$$\mu_j = \exp\left(\frac{-(x - w1_j)^T * (x - w1_j)}{2\sigma_j^2}\right) \quad j = 1 \dots N1$$

Onde:

$\mu_j$  é a saída do  $j$ -ésimo neurônio da camada escondida.

$w1_j$  é o vetor de pesos (médias) para o  $j$ -ésimo neurônio da camada escondida, correspondendo ao centro da função gaussiana do neurônio  $j$ .

$\sigma_j^2$  é a variância associada à função gaussiana do neurônio  $j$ .

Então, a resposta máxima da cada neurônio deve ocorrer quando "x" estiver bem próximo de " $w1_j$ ". A resposta vai diminuindo à medida que " $x^T$ " afasta de " $w1_j$ ", sendo o cone gaussiano mais estreito quanto menor for a variância " $\sigma_j^2$ ". Estes cones produzem uma saída idêntica para entradas que estão a uma distância radial fixa do centro da gaussiana.

A saída " $y_j$ " dos neurônios da camada de saída calculadas da mesma forma que nas redes perceptrons, ou seja:

$$y_j = g\left(\sum_{i=0}^{N1} w2_{ji} * \mu_i\right) = \sum_{i=0}^{N1} w2_{ji} * \mu_i ; j = 1 \dots N2$$

Onde:

$Y_j$  é a saída do  $j$ -ésimo neurônio da camada de saída.

$w2_{ji}$  é a matriz sináptica da 2ª camada.

$g(.)$  é a função da ativação "rampa".

$\mu$  é o vetor de saída da primeira camada (adicionado pelo limiar  $\rightarrow m_0 = -1$ ).

Assim, a rede executa uma transformação não linear de  $|\mathbb{R}^N \rightarrow |\mathbb{R}^{N2}$  através de uma combinação linear das funções bases (gaussianas) não-lineares.

### 8.3 Comparação entre RBF e MLP

As principais diferenças entre a RBF e o MLP são:

**a-)** A RBF padrão possui apenas uma camada escondida, enquanto que o MLP pode ter uma ou mais camadas escondidas.

**b-)** Todos os neurônios do MLP compartilham de um mesmo modelo neural. Na RBF, os neurônios das camadas escondidas são completamente diferentes dos neurônios da camada de saída.

**c-)** Na RBF a função de ativação da camada escondida computa norma euclidiana entre o vetor de entrada e o centro da unidade. No MLP, a função de ativação executa o produto interno entre as entradas e os pesos sinápticos.

Na prática, as redes RBF tem fornecido resultados satisfatórios, quando comparada ao MLP, para problemas onde o número de variáveis de entrada é muito grande, tendo também um conjunto de treinamento muito grande.

## 9 Redes de Kohonen e Mapas Auto-organizáveis

### 9.1 Considerações Iniciais

Self Organization Maps (SOM), Teuvo Kohonen

Uma das características marcantes das redes apresentadas anteriormente é que as mesmas necessitam de pares de padrões de entrada/saída para serem treinadas, ou seja, o treinamento delas é supervisionado.

No entanto, há diversas aplicações em que a única informação presente é o conjunto de padrões de entrada, sendo que o mesmo está em certos casos representando o mapeamento entre as diversas situações que envolvem o comportamento do processo. Assim, nesses casos as características relevantes das informações devem ser extraídas a partir do próprio conjunto de entradas.

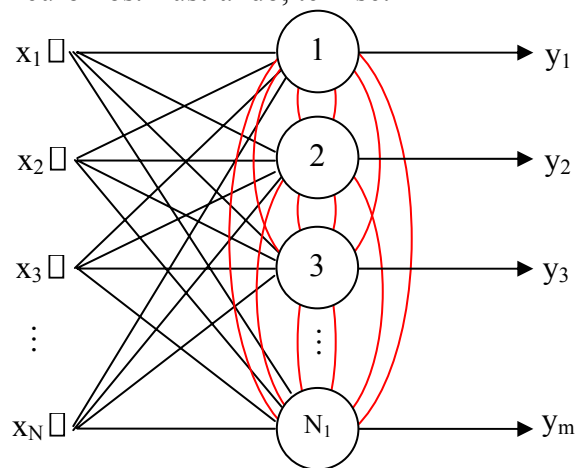
A maioria das redes utilizadas nesses tipos de problemas se auto-organizam através de métodos de treinamento competitivo, os quais tem a capacidade de detectar similaridades, regularidades e correlações entre os padrões do conjunto de entrada, agrupando-os em classes.

A plausibilidade biológica destas redes é encontrada em várias partes do córtex cerebral, tais como o córtex visual e auditivo, onde a localização espacial dos neurônios define as suas atividades funcionais.

Entre as arquiteturas de redes neurais que utilizam aprendizado competitivo destaca-se a rede Kohonen cuja simplicidade a transforma numa ferramenta progressiva para clusterização e reconhecimento de padrões.

### 9.2 A Arquitetura de Kohonen

A arquitetura de Kohonen se configura como uma rede auto-organizável de uma única camada, a qual é treinada por um algoritmo competitivo, tendo conexões laterais entre os neurônios. Ilustrando, tem-se:



Onde:

$$\mathbf{x} = [x_1, x_2, \dots, x_N]^T$$

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \dots, w_{jN}]^T$$

- Difícil ter mais de uma camada.
- As conexões laterais somente indicam que os neurônios se competem.
- Não só os valores do vencedor que vão ser ajustados, os seus vizinhos também.
- O vetor de pesos mais próximo da entrada é que vai ganhar a competição

As conexões laterais modelam a competição entre os neurônios, onde apenas um será o vencedor.

O objetivo principal da rede de Kohonen consiste de mapear um conjunto de padrões de entrada, sendo cada um deles n-dimensional, para um mapeamento que pode ser topologicamente representado em uma ou duas dimensões. A forma e a dimensão da saída é dada pelo mapa topológico que indica a organização espacial dos neurônios.

Em suma, o funcionamento básico de uma rede de Kohonen pode ser resumido através dos seguintes passos:

- I-) Cada neurônio da rede computa o nível de proximidade de seu vetor de pesos em relação a cada padrão de entrada.
- II-) Aplica-se um mecanismo de competição entre os neurônios com o objetivo de escolher o vencedor.
- III-) A partir da definição do neurônio vencedor, resgata-se através do mapa topológico aqueles neurônios que são vizinhos do vencedor.
- IV-) Os pesos do neurônio vencedor e de seus vizinhos são incrementados com o objetivo de aumentar o nível de proximidade com a respectiva entrada.
- V-) A rede estará treinada quando a mesma estiver estabilizada.

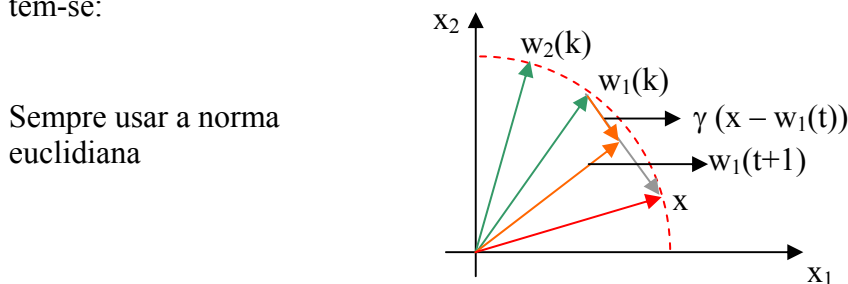
### 9.3 O processo de aprendizado competitivo

Para a rede de Kohonen ser auto-organizável, deve-se estabelecer o mecanismo de competição que indicará qual neurônio será o vencedor.

Este critério de competição é normalmente baseado em medidas da proximidade, ou seja, aquele neurônio (vetor de pesos) que estiver mais próximo de determinado padrão “x” vencerá a competição. Em seguida, ajusta-se o vetor de pesos “w<sub>j</sub>” do neurônio vencedor “j” em relação ao vetor de entrada “x” através do seguinte processo adaptativo:

$$w_j(t+1) = w_j(t) + \gamma * (x - w_j(t)) \quad (1)$$

Onde “j” é o índice do neurônio vencedor e “γ” é a taxa de aprendizagem. Para visualização da aplicação da equação acima, considerando os vetores “w<sub>j</sub>” e “x” normalizado, tem-se:



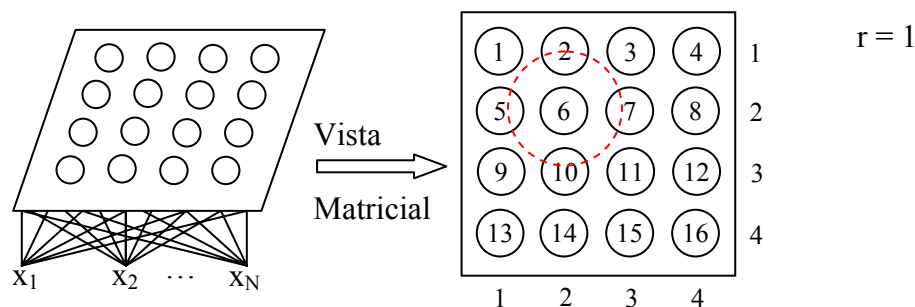
Assim, a equação acima simplesmente rotaciona o vetor vencedor “w<sub>j</sub>” em direção ao vetor “x”.

Entretanto, o processo de ajuste definido em (1) é aplicado tanto ao neurônio vencedor “j” como também para todos os seus vizinhos que são definidos pelo mapa topológico.

### 9.4 Mapa Topológico Auto-organizável

O mapa topológico da rede de Kohonen, também conhecido como SOM (Self-Organization Map) informa como cada neurônio está arranjado espacialmente, indicando ainda o critério de vizinhança adotado para definir os neurônios vizinhos uns aos outros.

Um dos principais critérios de vizinhança consiste de especificar um raio “r” de abrangência que será utilizado pelos neurônios da rede para definir seus vizinhos senão todos aqueles neurônios que estarão a uma distância máxima dele que seja menor ou igual a “r”. Ilustrando-se para um mapa bidimensional 4x4 (16 neurônios), temos:



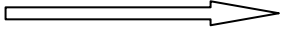
Para o mapa acima. Adotam-se  $r=1$ , temos que os vizinhos para o neurônio 6 são delimitados pela circunferência tracejada. Assim, tem-se os seguintes conjuntos de vizinhança  $\Omega_j^{(r)}$  em relação a cada neurônio “j”:

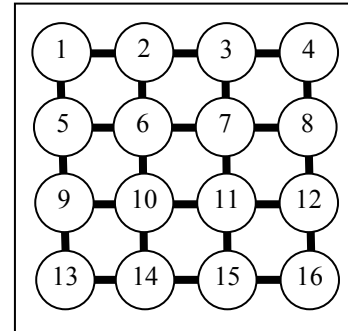
$$\Omega_1^{(1)} = \{2; 5\}$$

$$\Omega_2^{(1)} = \{1; 3; 6\}$$

(...)

$$\Omega_6^{(1)} = \{2; 5; 7; 10\}$$

Grid  
  
 Após constituição dos  $\Omega_j^{(1)}$



A sistemática anterior pode ser expandida para outras raia de vizinhança que compreende o mapa topológico.

A grande importância dos mapas auto-organizáveis está na capacidade de poder classificar qualquer agrupamento em função apenas de seu espaço de entrada, mais especificamente, os mapas conseguem extrair as características comuns dos padrões de entradas definidas num espaço n-dimensional para uma representação normalmente bidimensional, onde é possível visualizar agrupamentos de padrões com características regulares.

## 9.5 Algoritmo de Treinamento

O algoritmo de treinamento de mapas auto-organizáveis de Kohonen podem ser sintetizados nas seguintes instruções:

- 1-) Inicializar  $w_j$  aleatoriamente
- 2-) Definir o mapa topológico
- 3-) Montar os conjuntos de vizinhanças
- 4-) Repita
  - Para todo  $x(i)$  faça
    - Calcular a norma euclidiana entre  $x(i)$  e  $w_j$ ;  $j = 1 \dots N1$
    - Declarar vencedor o neurônio  $j$  com menor norma
    - ARG  $\min_j \|x(i) - w_j\|$
    - Ajustar os pesos do neurônio vencedor  $j$  e de todos seus vizinhos em  $\Omega_j$
    - $w_j(t+1) = w_j(t) + \gamma * (x(i) - w_j(t))$
  - Fim\_para
  - Até que não haja mudanças nos vetores  $w_j$
- 5-) Realizar análises dos resultados para definir as classes.

Para executar a classificação de padrões, após o treinamento, deve-se executar os seguintes procedimentos:

- Para todo  $x(i)$  faça
  - Obter o neurônio vencedor  $j$
  - Atribuir o padrão  $i$  à classe que está inserida no neurônio  $j$
- Fim\_para

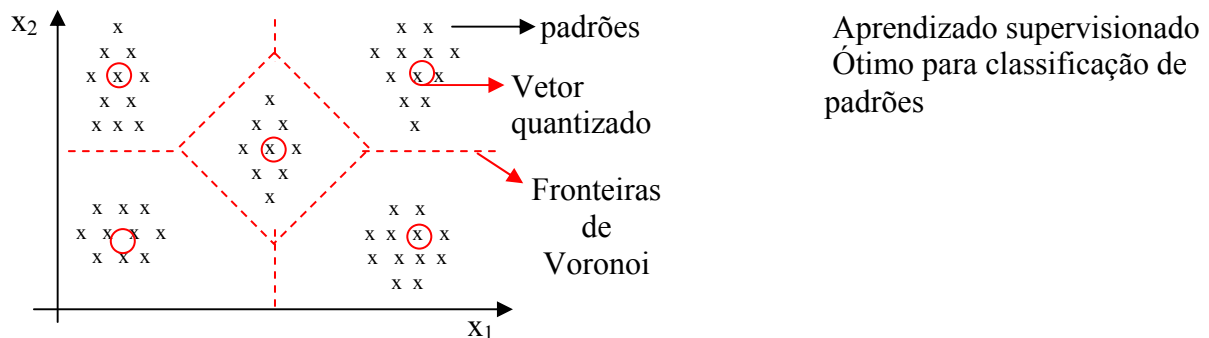
Assim, todos os padrões com similaridade em comum serão agrupados na mesma classe.

## 10 Rede LVQ (Learning Vector Quantization) e Counter-propagation

### 10.1 Considerações Iniciais

O objetivo principal envolvido com um processo de quantização vetorial através de redes neurais consiste em dividir o espaço amostral de entrada em diversos subespaços disjuntos, sendo que cada um dos vetores de entrada (padrões) deve pertencer somente a um desses subespaços, os quais estarão representando a classe associada ao problema considerado.

A fim de ilustrar este processo, a figura seguinte ilustra um conjunto de padrões de entrada constituído por pares  $(x_1, x_2)$  de valores.



A partir da figura anterior observa-se que o conjunto amostral pode ser então dividido em cinco subespaços, os quais estarão representando as eventuais classes do problema. Assim, em cada um desses subespaços é associado um vetor quantizador que estará representando todos os padrões vinculados ao respectivo subespaço.

Portanto, quando um novo padrão de teste for apresentado ao sistema o mesmo ativará o vetor quantizador representando o qual subespaço (conjuntos) em que o padrão estará associado.

É importante ressaltar que este processo trabalha de forma diferente da rede PMC aplicado em classificação de padrões, pois na quantização vetorial a região de separação é identificada somente por um vetor, enquanto que no PMC esta região é definida a partir de uma combinação de hiperplanos.

### 10.2 As redes LVQ

As redes LVQ (Learning Vector Quantization) são redes cujo treinamento é supervisionado, sendo que a mesma utiliza um processo competitivo para o ajuste de seus pesos, os quais estarão representando os respectivos vetores quantizadores.

Nesta abordagem, conhece-se todas as  $n$ -classes associadas aos padrões de entrada do sistema. O objetivo da rede é a quantização ótima do espaço de entrada em  $n$ -espaços.

Após o treinamento, a rede pode ser utilizada para classificar outros padrões de entrada entre as várias classes do sistema.

#### A-) Rede LVQ-1

Nesta caso, considera-se que cada vetor de entrada  $x(i)$ , utilizado no treinamento, pertença a uma classe "j". O algoritmo de aprendizagem é idêntico ao de Kohonen, modificando apenas o processo de ajuste dos pesos, o qual é aplicado ao vencedor.

**Algoritmo LVQ-1**

1-) Iniciar  $w_j$  aleatoriamente

2-) Repita

Para todo  $x(i)$  faça

Calcular a norma entre  $x(i)$  e  $w_j$

Declarar vencedor aquele com menor norma

Ajustar os pesos do vencedor:

Se  $x(i) \in C_j$

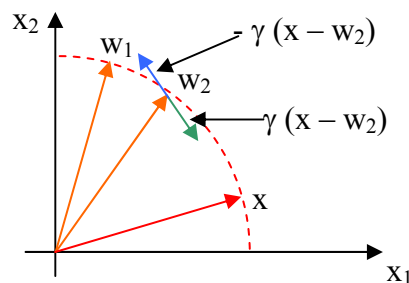
Então  $w_j(t+1) = w_j(t) + \gamma (x(i) - w_j(t))$

Senão  $w_j(t+1) = w_j(t) - \gamma (x(i) - w_j(t))$

Fim\_para

Até que não haja mudanças nos vetores  $w_j$ .

Ilustrando o processo de ajuste do neurônio vencedor, tem-se:



Portanto, conclui-se que se o neurônio vencedor “ $w_2$ ” estiver representando a classe a qual o vetor “ $x$ ” pertence, então ele será atraído em direção a “ $x$ ”; caso contrário, o vetor “ $w_2$ ” será repelido dando chance para que o neurônio que esteja representando a respectiva class possa vencer em etapas posteriores.

**B-) Rede LVQ-2**

Na rede LVQ-2, diferentemente da LVQ-1, a atualização dos pesos poderá ser efetuado para o neurônio vencedor (l) e, também, para o vice (m).

**Algoritmo LVQ-2**

1-) (...)

2-) Repita

Para (...)

Calcular (...)

Declarar e obter vencedor (l) e o vice (m)

Ajustar os pesos do vencedor e vice:

Se  $x(i) \in C_2$  and  $x(i) \notin C_m$

Então  $w_2(t+1) = w_2(t) + \gamma (x(i) - w_l(t))$

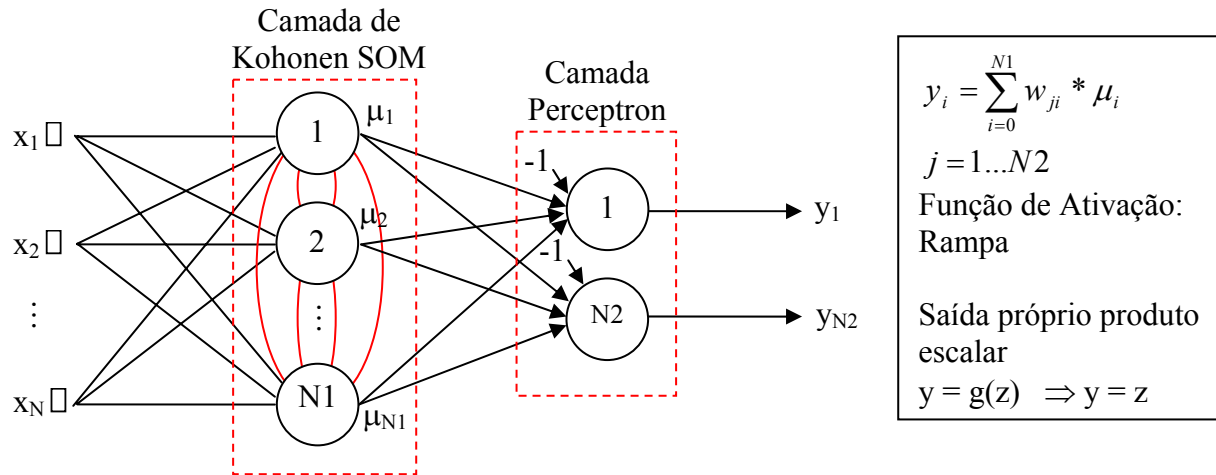
$w_m(t+1) = w_m(t) - \gamma (x(i) - w_l(t))$

Fim\_para

Até (...)

### 10.3A rede counter-propagation (Rede supervisionada, proposta por Hetch-Nielsen)

A counter-propagation é uma rede que permite executar tanto a classificação de padrões como a aproximação funcional. A sua arquitetura é constituída de uma camada de Kohonen e uma camada de perceptron conforme ilustrado a seguir:



Assim, tem-se para a rede counter-propagation uma camada intermediária competitiva (Kohonen) seguido por uma camada da saída supervisionada (Perceptron). Os ajustes das matrizes de pesos são realizados de forma sequencial, ajustando em primeira instância os pesos da camada intermediária, acompanhado do ajuste da camada de saída.

#### Algoritmo counter-propagation

- 1-) Inicializar as matrizes de pesos das duas camadas aleatórias.
- 2-) Ajustar os pesos da camada intermediária através do algoritmo de Kohonen (SOM).
- 3-) Ajustar os pesos da camada de saída através da “Regra Delta Generalizada”.



## 11 Redes Art (Adaptive Resonance Theory) (Carpenter & Grossberg, 1987)

### 11.1 Considerações Iniciais

Rede com aprendizado não supervisionado com característica recorrente. Possui a propriedade de aprender novos padrões sem destruir as informações já aprendidas anteriormente.

A arquitetura Art pode ser dividida em dois paradigmas que são definidos pelo tipo de entrada, ou seja:

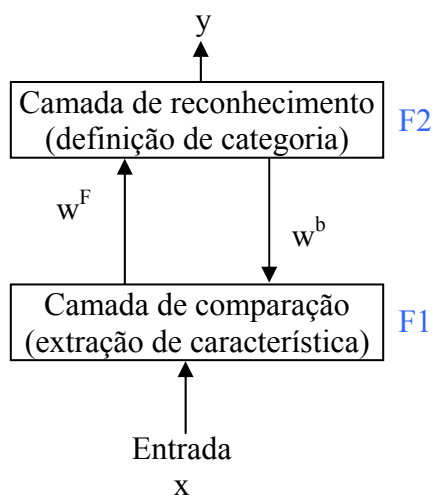
Art-1 → Entradas Binárias

Art-2 → Entradas Contínuas

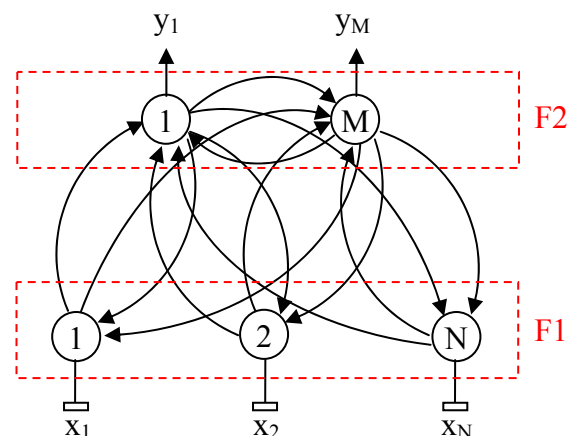
A rede Art é um classificador vetorial que recebe uma entrada e tenta classificá-la para uma categoria já definida. Se a classificação não for possível (casada), uma nova categorização é criada para a respectiva entrada.

### 11.2A arquitetura Art-1

A configuração da Art-1 é constituída de duas camadas neurais conforme ilustrado na figura seguinte:



Expandindo os blocos, tem-se:



Na camada F2 tem competição

A primeira entrada “x” cria o primeiro cluster. A próxima entrada é comparada com o primeiro cluster, se ela for “parecida” vai ser agrupada com o primeiro; caso contrário, ela formará um novo cluster, portanto o número de cluster cresce com o tempo e depende apenas da “distância” utilizada para comparar as entradas aos clusters já definidos.

Se o padrão de entrada e o padrão realimentado coincidirem, então ocorre um estado de ressonância adaptativa (amplificação e prolongamento da atividade neural).

### 11.3A operação da Art-1

O funcionamento da Art-1 pode ser resumido da seguinte forma:

**Passo 1** – Inicialização

$$\left. \begin{aligned} w_{ji}^F(0) &= \frac{1}{1+N} \\ w_{ij}(0) &= 1 \end{aligned} \right\} \begin{aligned} i &= 1..N \\ j &= 1..M \end{aligned}$$

**Passo 2** - Apresentar um novo padrão de entrada “x”

**Passo 3** - Computar as ativações usando as conexões da matriz  $w^F$

$$\mu_j = \sum_{i=1}^N w_{ji}^F(t) * x_i \quad j = 1..M \quad \boxed{= w^F * x \text{ (produto escalar)}}$$

**Passo 4** - Selecionar o neurônio vencedor  $k \rightarrow \max \{\mu_k\} \quad j < k \leq M$

Leva em consideração apenas os com atividade neural

**Passo 5** – Executar o teste de vigilância

$$\boxed{\begin{aligned} \text{se } \frac{\langle w_k^b(t), x \rangle}{\langle x, x \rangle} > \rho & \quad \text{Então } \rightarrow \text{ vá para o passo 7} \\ & \quad \text{Senão } \rightarrow \text{ vá para o passo 6} \end{aligned}} \quad \boxed{\text{verifica o nível de similaridade}}$$

→ Produto interno, quantidade de uns em comum

**Passo 6** – Desabilitar o neurônio k, temporariamente  $\{\mu_k = 0\}$   
vá para o passo 4

**Passo 7** – Adaptar pesos  $w^F$  e  $w^b$

$$\left. \begin{aligned} w_{kL}^F(t-1) &= \frac{w_{Lk}^b(t) * x_L}{\frac{1}{2} + \sum_{i=1}^N w_{ik}^b(t) * x_i}; L = 1..N \\ w_{Lk}^b(t+1) &= w_{Lk}^b(t) * x_L; L = 1..N \end{aligned} \right\} \text{Equação de Carpenter \& Grossberg}$$

**Passo 8** – Habilitar todos os neurônios em F2 e voltar ao passo 2.

### 11.4 Algoritmo Computacional

1-) Definir  $\rho$  e  $\nu$

2-) Computar  $w^b$  e  $w^F$  iniciais

3-) Para todo  $x(i)$  faça

Computar o vetor  $\mu \rightarrow w^F * x$

Flag  $\leftarrow 0$

Enquanto Flag = 0 faça

Computar o neurônio vencedor k

$$\text{se } \frac{\langle w_k^b, x \rangle}{\langle x, x \rangle} > \rho$$

Então  $\left\{ \begin{aligned} &\text{Atualizar } w^F \text{ e } w^b \\ &\text{Atribuir padrão I ao conjunto } \theta_k \\ &\text{Flag } \leftarrow 1 \end{aligned} \right.$

Senão  $\mu_k \leftarrow -1$

Fim\_se

Fim\_enquanto  
Criar um novo neurônio  
Fim\_para

### **11.5 Considerações Finais**

As principais particularidades da arquitetura Art são:

- A-) Novo conhecimento não destrói informações já aprendidas.
- B-) Habilidade para criar novas categorias.
- C-) Funciona como memória associativa autônoma.