

REDES NEURAIIS

ERI'98 – ENCONTRO REGIONAL DE INFORMÁTICA

SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

Nova Friburgo, RJ e Vitória, ES, Brasil, Março 1998

Thomas Walter Rauber

Departamento de Informática

Universidade Federal do Espírito Santo

Av. F. Ferrari, 29065-900 Vitória - ES, BRASIL

Tel.: (+55)(27) 3352654 — Fax: (+55)(27) 3352850

E-mail: thomas@inf.ufes.br — WWW-homepage: <http://www.inf.ufes.br/~thomas>

REDES NEURAIS

• INTRODUÇÃO

- A Inspiração da Neurociência
- História da Neurocomputação
- Referências para Aprofundamento na Matéria de Redes Neurais

• FUNDAMENTOS

- Modelo de Neurônio Artificial
- Topologia de Redes de Neurônios Artificiais
- Paradigmas de Aprendizagem
 - + Aprendizagem supervisionada
 - + Aprendizagem não-supervisionada
- Regras de Adaptação dos Pesos
 - + Aprendizagem pela Regra de Hebb
 - + Aprendizagem pela Regra de Delta
 - + Aprendizagem Competitiva
- Taxinomia de Redes

• REDES DE PROPAGAÇÃO PARA FRENTE

- Perceptron
- ADALINE

- + **O erro quadrático mínimo**
- + **Solução determinística**
- + **Solução iterativa: Descida de Gradiente**
- **Perceptron Multi-Camada e Retropropagação de Erro**
 - + **Arquitetura**
 - + **Adaptação dos pesos**

- **REDES COM REALIMENTAÇÃO**

- **Modelo de Hopfield**
- **Associatividade de Padrões na Rede de Hopfield**
- **Estabilidade e Aprendizagem de Pesos**
- **Relaxação, Minimização de Energia**
- **Aplicação: Recuperação de Imagens**

- **REDES COMPETITIVAS**

- **Determinação do Vencedor**
- **Adaptação dos Pesos**
- **O Mapa de Preservação Topológica de Kohonen**

- **CONCLUSÕES**

- **BIBLIOGRAFIA**

INTRODUÇÃO

- **Simulação de capacidades cognitivas do ser humano**
 - ⇒ **Simulação da inteligência biológica**
- **Local da inteligência humana = cérebro**
 - ⇒ **Simulação do funcionamento do cérebro**
- **Cérebro = Redes de neurônios interconectados (= Rede Neural Biológica)**
 - ⇒ **Simulação do funcionamento dos neurônios**
 - + **Topologia**
 - + **Processamento de informação**
 - + **Troca de informação**
 - + **Interfaces sensoriais e motóricas**
- **Rede Neural Artificial**
 - **Simulação das redes neurais biológicas**
 - **Estabelecimento de modelos**
 - + **Motivados por pesquisas de redes biológicas**
 - + **Empíricos**
 - **Estabelecimento de paradigmas de aprendizagem (= treinar a rede)**

A INSPIRAÇÃO DA NEUROCIÊNCIA

COMPARAÇÃO: HOMEM — MÁQUINA

• Qualidades do cérebro humano

- Robustez e tolerância a falhas:
 - + Eliminação de neurônios
 - ⇒ funcionalidade global inalterada
- Capacidade de aprendizagem:
 - + Novas tarefas aprendizadas com exemplos
- Processamento de informação incerta:
 - + Informação incompleta
 - + Ruído
 - + Informação contraditória
- Paralelismo:
 - + Neurônios ativos simultaneamente

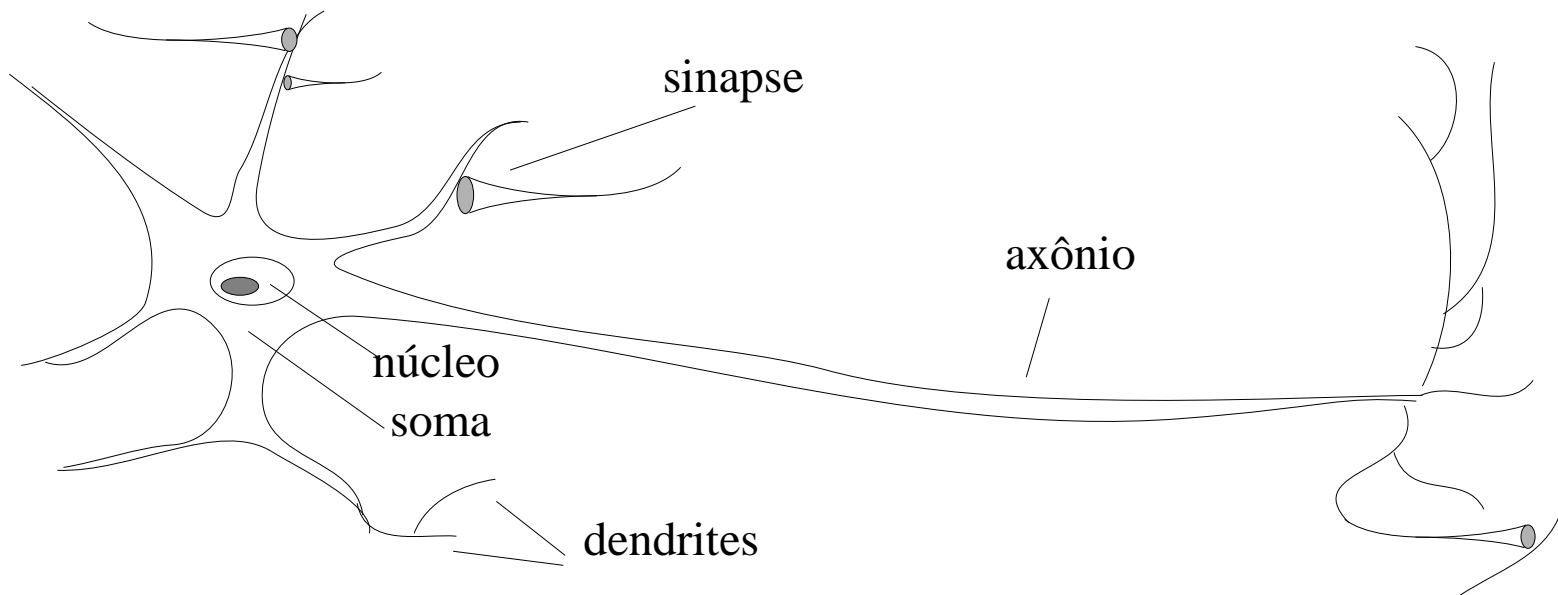
• Qualidades de uma máquina “von Neumann”

- Robustez e tolerância a falhas:
 - + Intolerante
- Capacidade de aprendizagem:
 - + Novas tarefas Têm que ser programadas
- Processamento de informação incerta:
 - + Tratamento explícito
- Paralelismo:
 - + Sequencialidade: Instrução após instrução pelo CPU

⇒ Tentativa de simulação do modelo biológico

MODELO DE NEURÔNIO BIOLÓGICO

- **Neurônio = célula com núcleo e corpo (soma)**
- **Reações químicas e elétricas = Processamento e transmissão de informação**
- **Ligação para dendrites de outros neurônios via sinapses**
- **Inteligência global por processamento local relativamente simples**



- **NATUREZA \neq MODELO TÉCNICO !!!**

HISTÓRIA DA NEUROCOMPUTAÇÃO

Tabela 1: Eventos Históricas da Neurocomputação

ANO	AUTOR	EVENTO
1943	McCulloch & Pitts	Modelo de neurônio artificial
1949	Hebb	Regra de Aprendizagem de um neurônio
1958	Rosenblatt	Perceptron
1960	Widrow & Hoff	Adaline
1969	Minsky & Papert	XOR em “Perceptrons”
1982	Hopfield	Rede realimentada com função de energia
1986	Rumelhart & al.	Retropropagação de erro

Tabela 2: Fases da Neurocomputação

INTERVALO	FASE
1943 — 1969	Entusiasmo
1969 — 1982	Depressão
1982 — presente	Renascença

REFERÊNCIAS PARA APROFUNDAMENTO EM REDES NEURAIS

- **FORUM DE DISCUSSÃO (“USENET NEWS”)**

- **comp.ai.neural-nets**

- “Questões mais perguntadas” (“FAQ — Frequently Asked Questions”)

- **LIVROS**

- **INICIANTES**

- + **Masters, T., Practical Neural Network Recipes in C++, 1994**

- + **Fausett, L., Fundamentals of Neural Networks: Architectures, Algorithms, and Applications, 1994**

- + **Anderson, J. A., An Introduction to Neural Networks, 1995**

- **INTERMEDIÁRIO E AVANÇADO**

- + **Bishop, C. M., Neural Networks for Pattern Recognition, 1995**

- + **Haykin, S., Neural Networks, a Comprehensive Foundation, 1994**

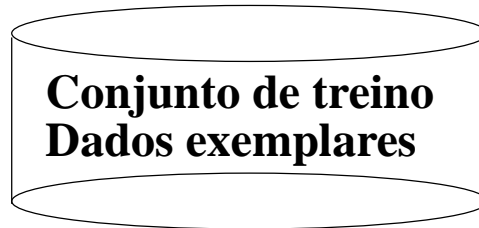
- + **Hertz, J., Krogh, A., and Palmer, R., Introduction to the Theory of Neural Computation, 1991**

- + **Ripley, B.D., Pattern Recognition and Neural Networks, 1996**

- **SIMULADORES**

- **“Stuttgart Neural Network Simulator” (SNNS)**

FUNDAMENTOS

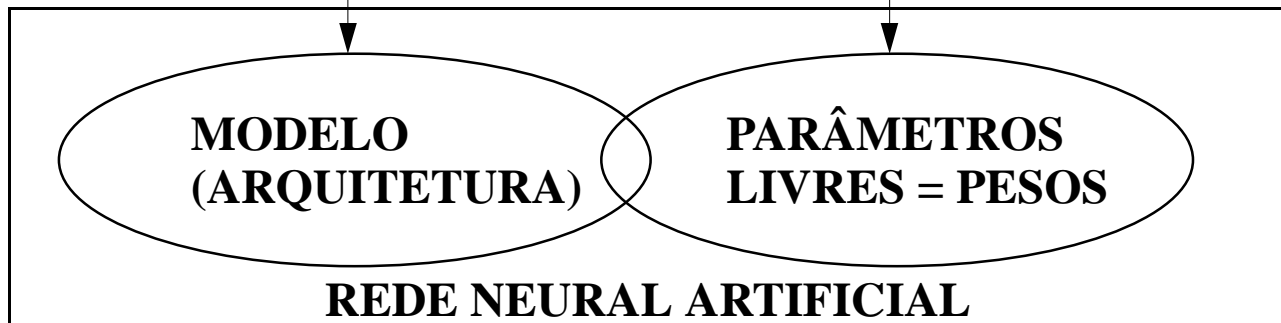


**PRINCÍPIOS DE DESENHO,
APLICAÇÃO**

ESCOLHA

ALGORITMO DE APRENDIZAGEM

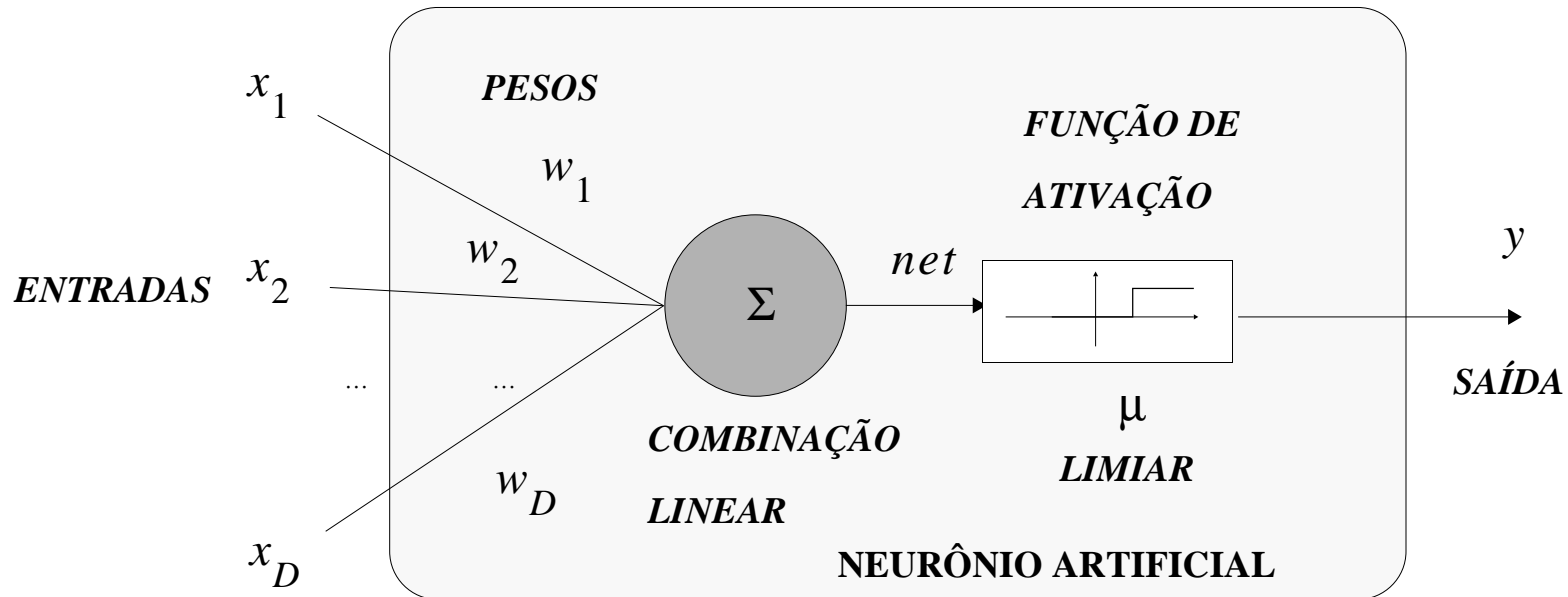
ADAPTAÇÃO



- **Aprendizagem por generalização dos exemplos de treino**
- **Conhecimento = Arquitetura + Pesos**

MODELO DE NEURÔNIO ARTIFICIAL

• Modelo de McCulloch & Pitts



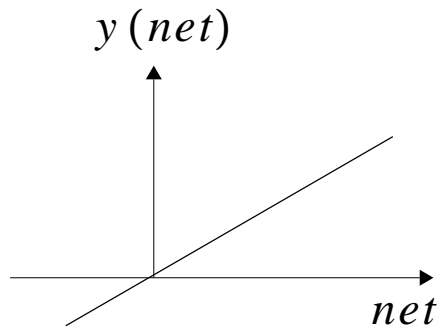
— **Combinação linear das entradas**

$$net = w_1 x_1 + w_2 x_2 + \dots + w_D x_D = \sum_{j=1}^D w_j x_j = \underline{w}^T \underline{x}$$

— **Pesos w_i , Vetor de pesos $\underline{w} = (w_1, \dots, w_j, \dots, w_D)^T$**

— **Função de ativação $F(net)$, Limiar μ : $y = \Theta(\sum_{j=1}^D w_j x_j - \mu)$**

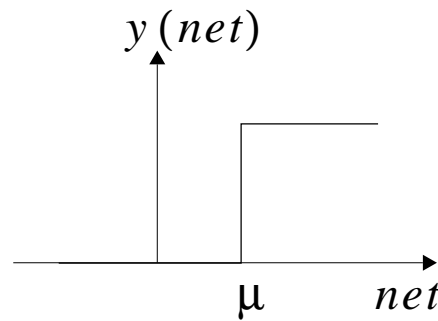
FUNÇÕES DE ATIVAÇÃO



LINEAR

$$y(net) = a \times net + b$$

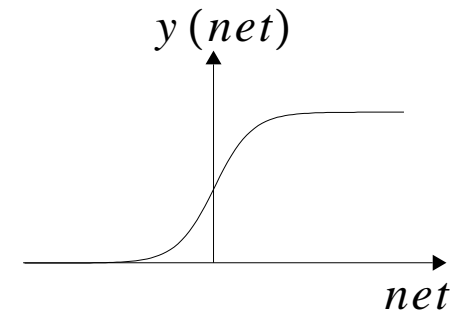
— **LINEAR**



ESCADA (HEAVESIDE)

$$y(net) = 1 \quad \text{se } net \geq \mu$$
$$y(net) = 0 \quad \text{se } net < \mu$$

— **NÃO - LINEAR**
— **DISCRETA (BINÁRIA)**



SIGMOIDAL

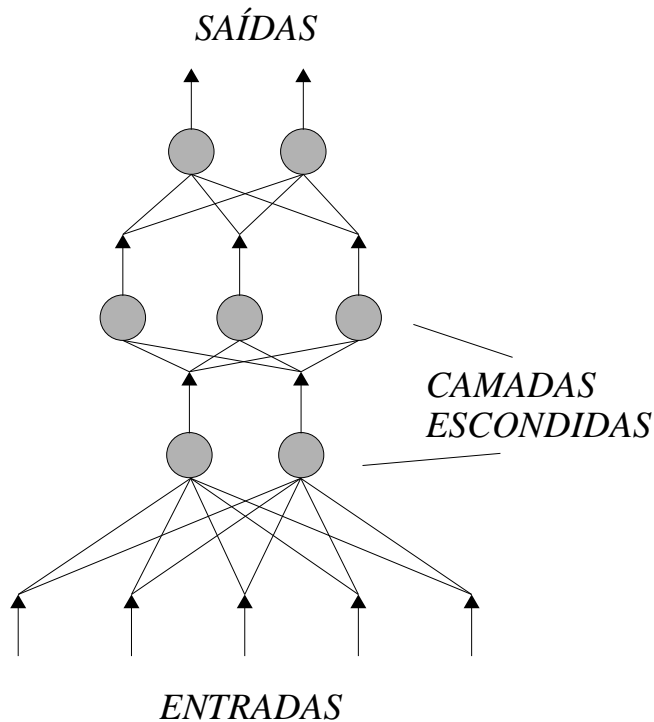
$$y(net) = \frac{1}{1 + e^{-net}}$$

— **NÃO - LINEAR**
— **CONTÍNUA**
— **MONÓTONA**
— **DERIVADA**
EXPRESSÁVEL PELA
PRÓPRIA FUNÇÃO

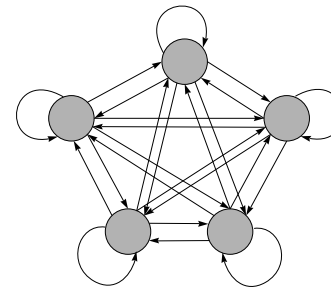
$$y'(net) = y(net) (1 - y(net))$$

TOPOLOGIA DE REDES DE NEURÔNIOS ARTIFICIAIS

- **INTERLIGAÇÃO DOS NEURÔNIOS, PARALELISMO**
⇒ **INTELIGÊNCIA GLOBAL**



Propagação para Frente

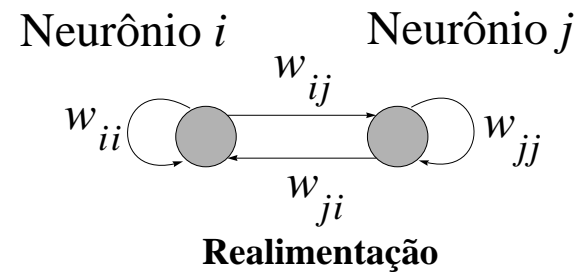
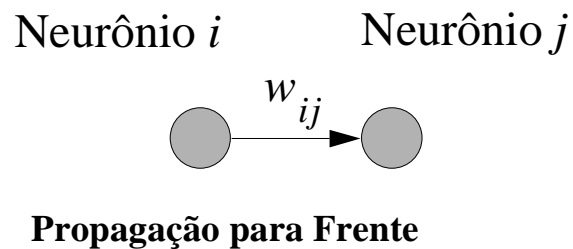


Realimentação

- **Método de propagação da informação recebida**
 - **Propagação para Frente: unidirecional por camadas, determinístico**
 - **Realimentação: dinâmico, recorrente**

PARADIGMAS DE APRENDIZAGEM

- **Aprendizagem = modificar os pesos w_{ij} entre o neurônio i e o neurônio j , segundo um algoritmo**



— **Aprendizagem supervisionada**

+ **Conjunto de treino = pares de entrada e saída desejada**

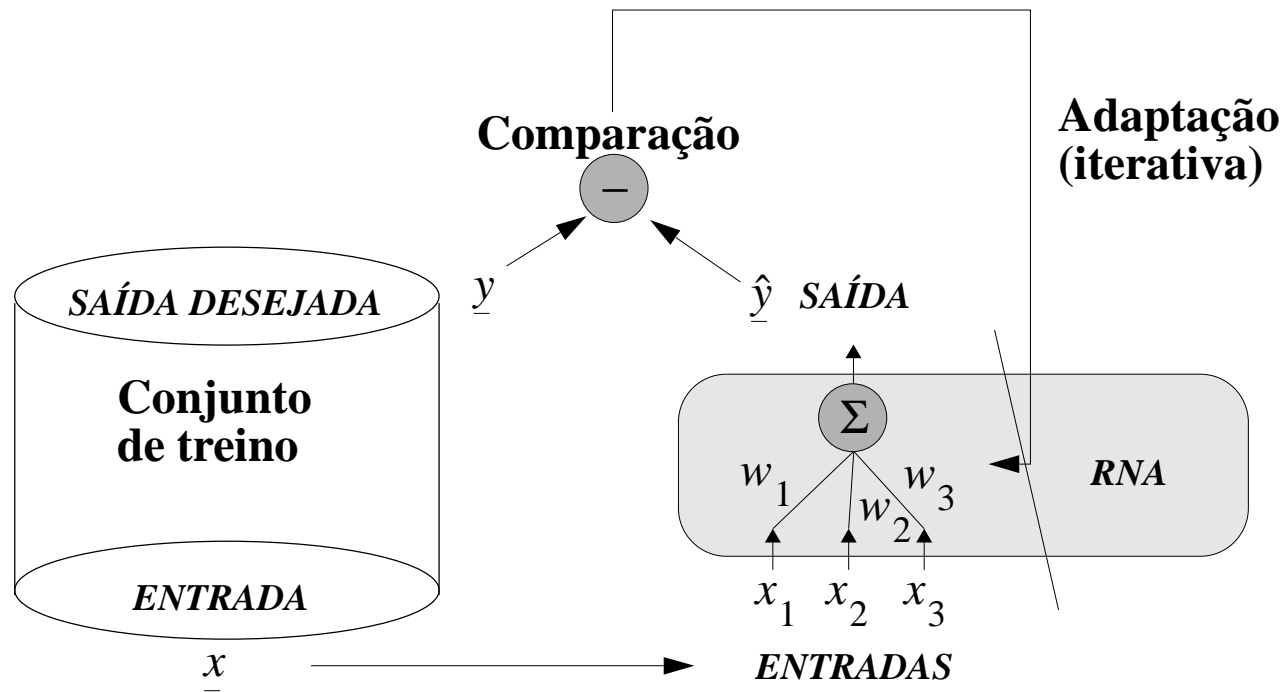
+ **Objetivo de aprendizagem: Aprender mapeamento entrada — saída**

— **Aprendizagem não-supervisionada**

+ **Conjunto de treino = dados (= dados de entrada) sem saída**

Aprendizagem supervisionada

- Conjunto de treino T composto por n pares de exemplos (x_{-p}, y_{-p}) $T = \{ (x_{-p}, y_{-p}) \}_{p=1}^n$
- Cada exemplo de treino acompanhado por valor desejado de saída

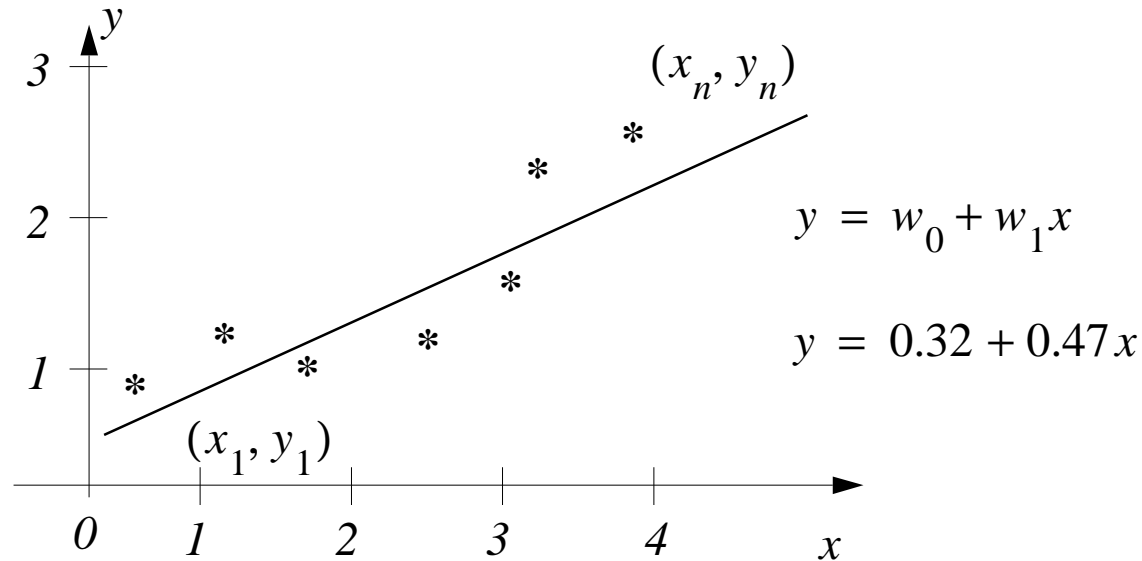


- Mapeamento desejado (x_{-p}, y_{-p}) , mapeamento calculado (x_{-p}, \hat{y}_{-p})

- **Adaptação dos pesos pela discrepância entre o desejado e o calculado**

Aprendizagem supervisionada (cont.)

• Exemplo: Regressão linear



— **Dados de treino:** pares de números reais (x_p, y_p) (veja dados na Tabela 3)

— **Sistema (Rede Neural Artificial Linear):** $y = f(x) = w_0 + w_1 x$

— **Objetivo:** Adaptar w_0 e w_1 para minimizar a diferença (quadrática) entre

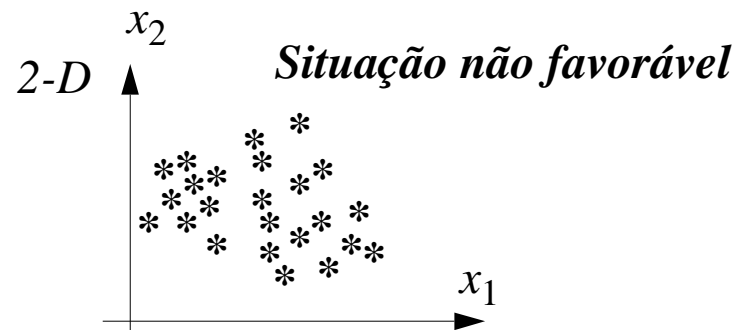
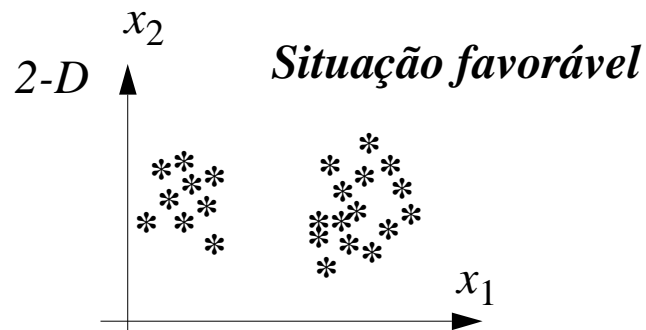
+ A resposta desejada y_p e

+ A resposta dada pela rede $\hat{y}_p = f(x_p)$

+ ... em média sobre todos os exemplos $\frac{1}{N} \sum_{p=1}^n (y_p - \hat{y}_p)^2$

Aprendizagem não-supervisionada

- **Conjunto de treino T composto por n exemplos** $(x_{-p}) T = \{ (x_{-p}) \}_{p=1}^n$
- **Número de categorias ou classes não definido a priori**
 - **Existe somente ESTRUTURA entre os dados**
 - + **Em algumas áreas existe densidade maior dos dados, existe aglomeração (*clustering*)**
 - **Número desconhecido de aglomerações**
 - **Associação Padrão-Aglomeração desconhecida (a qual cluster pertence x ?)**

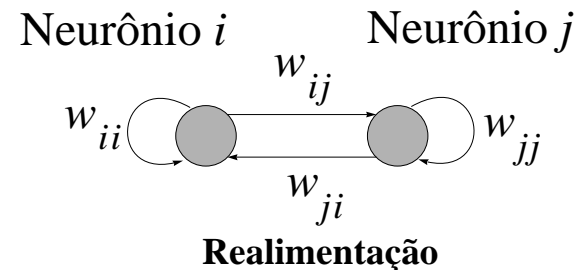
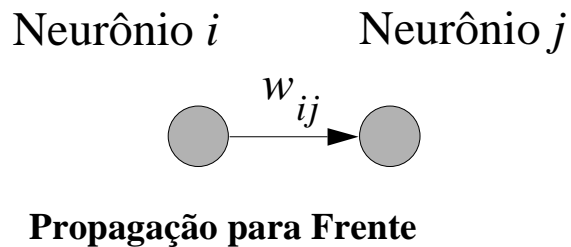


- **Objetivo: Descobrir a estrutura entre os dados (x_{-p}) , descobrir aglomerações**

REGRAS DE ADAPTAÇÃO DOS PESOS

- **APRENDIZAGEM = ADAPTAÇÃO ITERATIVA DOS PESOS**

- **Peso w_{ij} entre neurônio i e neurônio j**



- **Iteração:** $w_{ij}^{(0)} \rightarrow w_{ij}^{(1)} \rightarrow \dots \rightarrow w_{ij}^{(l)} \rightarrow w_{ij}^{(l+1)} \rightarrow \dots \rightarrow w_{ij}^{(n)}$
- **Algoritmo de aprendizagem modifica peso por $\Delta w_{ij}^{(l)}$ na iteração $l \Rightarrow$**
- **Regra de adaptação de peso:** $w_{ij}^{(l+1)} = w_{ij}^{(l)} + \Delta w_{ij}^{(l)}$
- **Condições de parada do algoritmo (= número n de iterações)**
 - n fixo
 - Pesos parecidos em duas iterações consecutivas: $\Delta w_{ij}^{(l)} \approx 0$
 - Outros critérios de qualidade da funcionalidade da rede

Aprendizagem pela Regra de Hebb

- Trabalho pioneiro de HEBB, 1949
- Baseado em estudos biológicos do cérebro (empiricamente)
- Peso de ligação entre dois neurônios que estão ativos aos mesmo tempo deve ser reforçado
- **REGRA DE APRENDIZAGEM DE HEBB:** $\Delta w_{ij} = \eta y_i y_j$
 - Velocidade da aprendizagem: η
- Aplicação da regra de aprendizagem de Hebb na rede de Hopfield (veja mais tarde)

Aprendizagem pela Regra de Delta

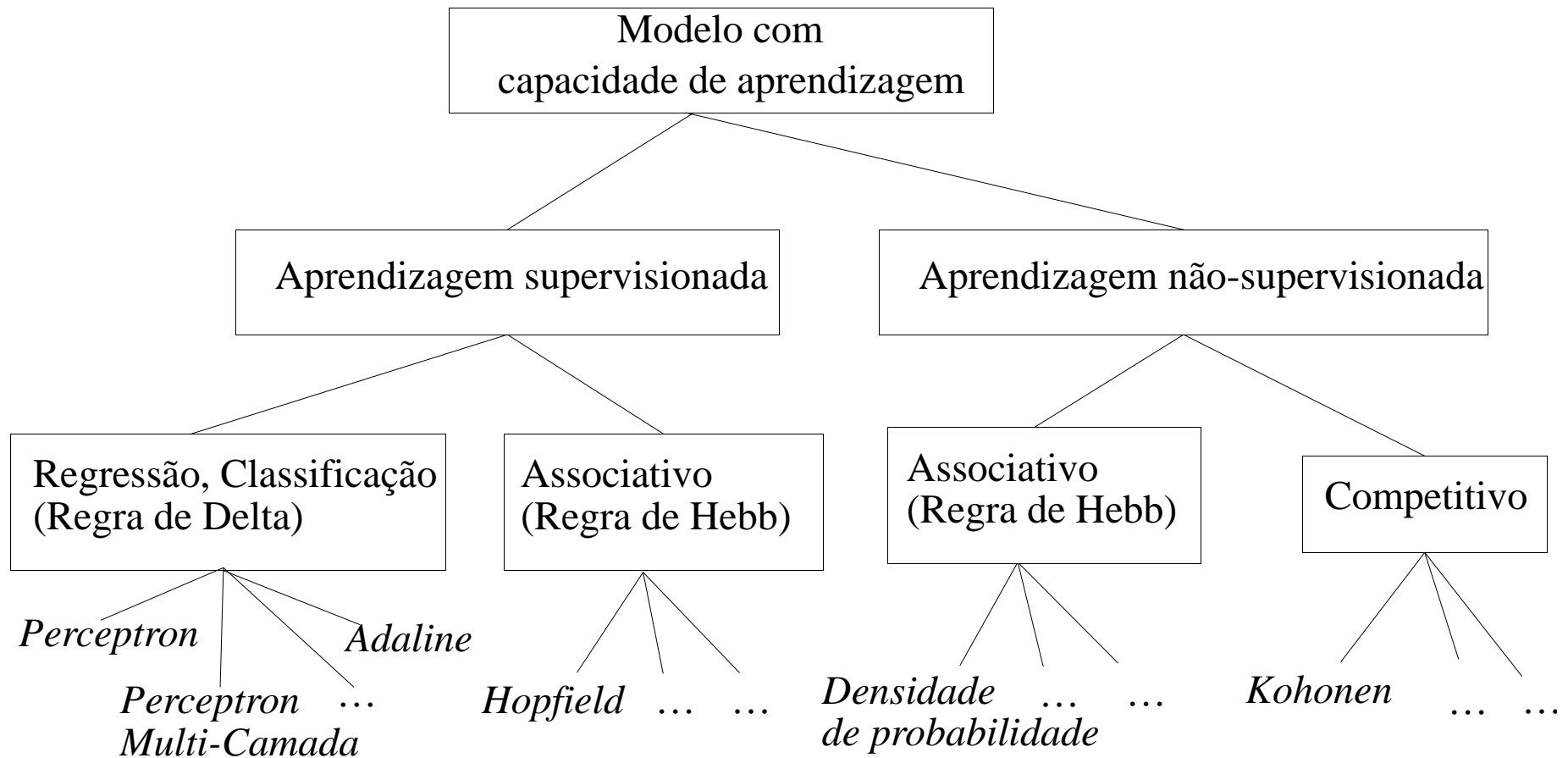
- **Regra de delta ou Regra de Widrow-Hoff, 1960**
- **Diferença entre valor desejado e calculado pela rede na aprendizagem supervisionada:**
 - Rede calcula na saída (no neurônio i) função y'_i .
 - Na aprendizagem supervisionada conhece-se o valor desejado y_i que a rede deve calcular
 - Erro $e_i = y_i - y'_i$ entre o calculado e o desejado
 - Peso entre no neurônio i e o no neurônio j que é responsável por esse erro então deve ser modificado proporcional à ativação e ao erro, escalado por uma taxa de aprendizagem η
- **REGRA DE DELTA:** $\Delta w_{ij} = \eta e_i y_j = \eta (y_i - y'_i) y_j$
- **Objetivo do algoritmo de aprendizagem:** minimizar o erro entre os valores calculados pela rede e desejados pelos exemplos fornecidos num problema de aprendizagem supervisionada
- **Aplicação da Regra de Delta no ADALINE (veja mais tarde)**

Aprendizagem Competitiva

- Rede com vários neurônios
- Somente um único neurônio pode ser ativo ao mesmo tempo = **VENCEDOR** i^*
 - Vencedor i^* emite um sinal de ativação $y_{i^*} = 1$
 - Restantes neurônios $i \neq i^*$ inativos: $y_i = 0$
- **APRENDIZAGEM COMPETITIVA:** $\Delta w_{ij} = \eta y_i (x_j - w_{ij})$
- **Efeito:** pesos \underline{w}_i se deslocam em direção do estímulo (entrada) da rede \underline{x}
- Rede de Kohonen como exemplo de uma rede neural artificial que usa aprendizagem competitiva para adaptar os pesos
- Aplicação da Aprendizagem Competitiva Rede de Kohonen (veja mais tarde)

TAXINOMIA DE REDES

- Divisão hierárquica para os modelos de redes neurais apresentados nesse texto
 - Pelo paradigma de aprendizagem
 - Pelas regras de adaptação dos pesos

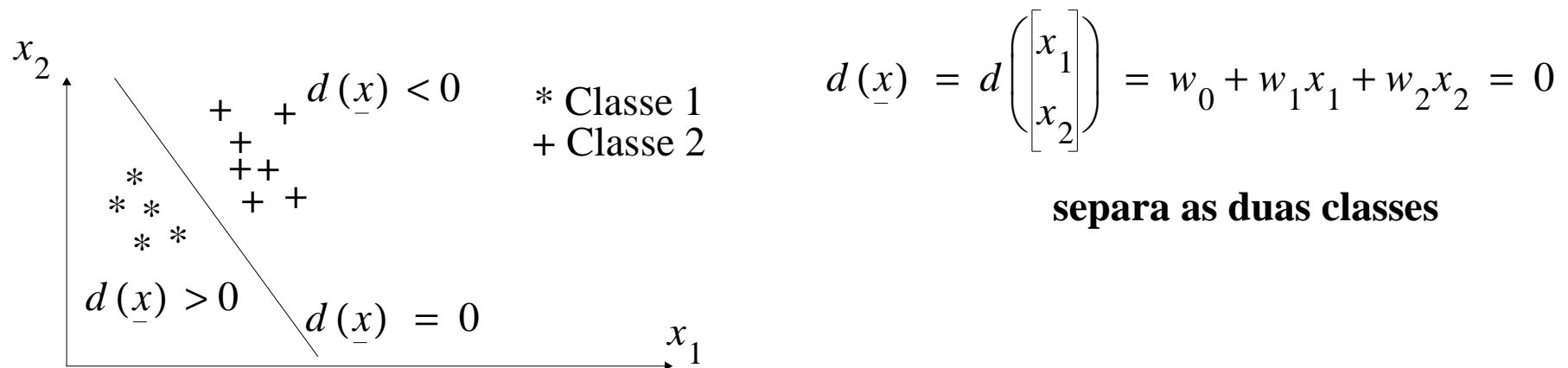


REDES DE PROPAGAÇÃO PARA FRENTE

- **Aprendizagem supervisionada**
- **Redes de propagação para frente organizadas em camadas**
- **Inicialmente chamadas PERCEPTRONS**
- **Perceptron com uma única camada**
- **Perceptron multi-camadas \Rightarrow Existem camadas *escondidas***
- **Fluxo de informação sempre unidirecional**
- **Pesos (assimétricos) unidirecionais entre dois neurônios**
 - **Neurônios necessariamente em camadas diferentes**

PERCEPTRON

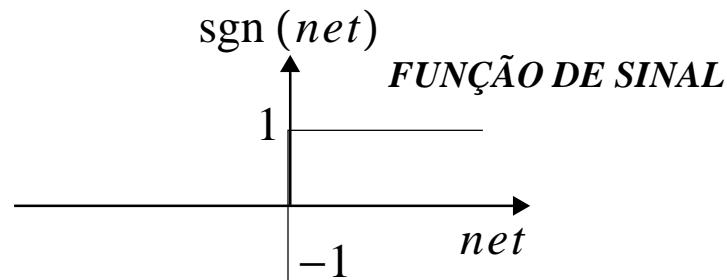
- **Aplicação principal de redes neurais artificiais: CLASSIFICAÇÃO**
 - **Reconhecimento automático de caracteres**
 - **Detecção de falhas em processos**
 - **Identificação de pessoas por impressões digitais, voz, iris do olho**
 - **Diagnóstico médico**



- **PERCEPTRON (Rosenblatt 1958)**
 - **Classificação entre classes linearmente separáveis**
 - **Modelo do neurônio de McCulloch & Pitts**
 - **Método de adaptação dos pesos = Regra de Aprendizagem do Perceptron**
 - + **Prova formal da convergência da regra em número finito de iterações em caso da separabilidade linear**

PERCEPTRON (cont.)

- **McCulloch & Pitts:** $y = \Theta \left(\sum_{j=1}^D w_j x_j - \mu \right)$
- **Perceptron: Regra de Classificação:** $d(\underline{x}) = \text{sgn} \left(\sum_{j=0}^D w_j x_j \right)$



— **Função de ativação: função de sinal**

— **Absorver o limiar μ no cálculo como $\mu = -w_0$, com $x_0 = 1$**

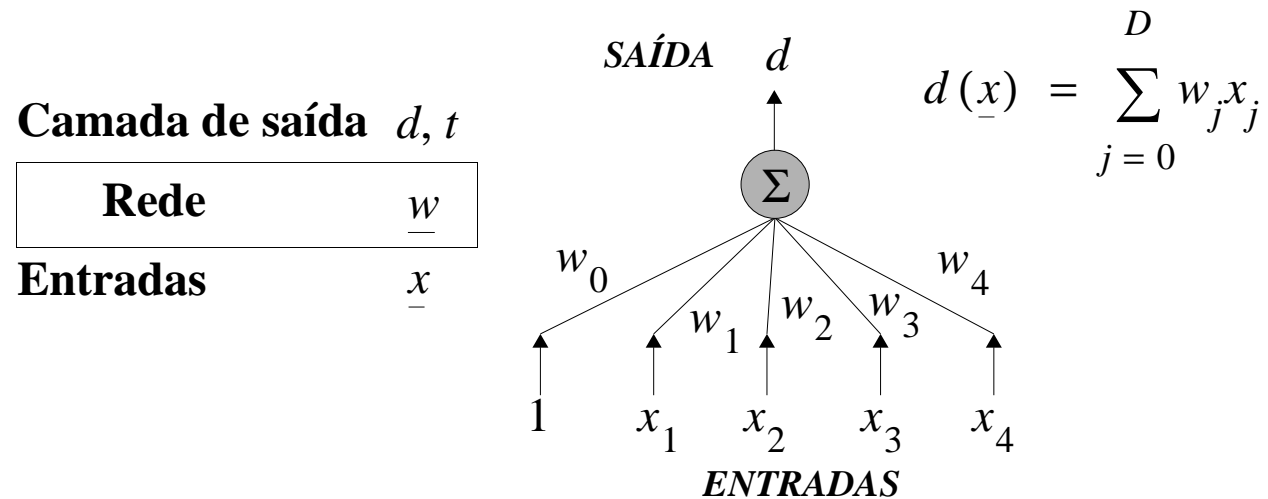
- **Função $d(\underline{x})$ fornece resposta binária do “lado” do objeto $\underline{x} = (x_1, x_2)^T$**
 \Rightarrow **classificação linear entre duas classes**
- **Perceptron não apropriado para classificação não-linear**
- **Objetivo do algoritmo de aprendizagem: Cálculo dos pesos $\underline{w} = (w_0, \dots, w_j, \dots, w_D)^T$**

PERCEPTRON (cont.)

- **Algoritmo de aprendizagem: Iterativo** $w_j^{(l+1)} = w_j^{(l)} + \Delta w_j^{(l)}$
- **Conjunto de treino:** $T = \{ (x_{-p}, t_p) \}_{p=1}^n$
 - **Valores de entrada** x_{-p} , com $x_{-p} = (x_{1p}, x_{2p})^T$ para duas dimensões
 - **Valor de alvo = valor de saída desejada** $t_p \in \{-1, 1\}$
- **Classificação do exemplo de treino** x_{-p} **tem dois resultados possíveis**
 - 1.) **Classificação correta** $d(x_{-p}) = t_p \Rightarrow$ **Nenhuma alteração dos pesos**
 - 2.) **Classificação errada** $d(x_{-p}) \neq t_p \Rightarrow$ **Modificação dos pesos: $\Delta w_j^{(l)} \neq 0$**
- **REGRA DE APRENDIZAGEM DE PERCEPTRON (versão simples):**
 - $\Delta w_j = \eta t_p x_j$ se $d(x_{-p}) \neq t_p$ e
 - $\Delta w_j = 0$ se $d(x_{-p}) = t_p$

ADALINE (Widrow e Hoff, 1960)

- **Perceptron:** Saída binária $t_p \in \{-1, 1\}$ — **Adaline:** Saída contínua $t_p \in \mathbb{R}$
- **Função do ADALINE:** $d(\underline{x}) = \sum_{j=0}^D w_j x_j = \underline{w}^T \underline{x}$



• Aproximação de função

- Exemplo: Controle da temperatura de uma sala, usando d como variável do dispositivo de ação (sistema de aquecimento)
- Exemplo: Regressão linear

• **Dados de treino** $T = \{ (x_{-p}, t_p) \}_{p=1}^n$, com $x_{-p} \in \mathbf{R}^D$ e $t_p \in \mathbf{R}$

ADALINE (cont.)

ERRO QUADRÁTICO MÍNIMO

- Para um exemplo de treino x_{-p} : $e(x_{-p}) = \text{desejado}(x_{-p}) - \text{calculado}(x_{-p})$

— Valor de alvo $t_p = \text{desejado}(x_{-p})$

— Resposta do ADALINE = $\text{calculado}(x_{-p}) = d(x_{-p}) = w_{-p}^T x_{-p}$

— Erro quadrático para um exemplo x_{-p} (erro pode ser negativo ou positivo):

$$e^2(x_{-p}) = (t_p - d(x_{-p}))^2 = \left(t_p - w_{-p}^T x_{-p}\right)^2$$

- Objetivo do algoritmo: minimização do erro, considerando a média de todos os exemplos (valor esperado $E\{e^2(x_{-p})\}$)

- Erro quadrático médio: $EQM = \frac{1}{n} \sum_{p=1}^n e^2(x_{-p}) = \frac{1}{n} \sum_{p=1}^n \left(t_p - w_{-p}^T x_{-p}\right)^2$
= Critério a ser minimizado pelo algoritmo de aprendizagem

- Solução determinística vs. Solução iterativa

ADALINE (cont.)

ERRO QUADRÁTICO MÍNIMO: SOLUÇÃO DETERMINÍSTICA: PSEUDOINVERSA

- **ADALINE: Input: exemplo \underline{x}_{-p} ; Output: Valor de alvo t_p (valor desejado);**

Cálculo: $\underline{w}_{-p}^T \underline{x}_{-p} = t_p, p = 1, \dots, n$, i.e. $\underline{w}_{-1}^T \underline{x}_{-1} = t_1, \dots, \underline{w}_{-n}^T \underline{x}_{-n} = t_n$

— **Dados de treino: Representado na matriz $n \times (D + 1)$ como** $X = \begin{bmatrix} \dots \\ \underline{x}_{-1}^T \\ \dots \\ \underline{x}_{-n}^T \\ \dots \end{bmatrix}$, $\underline{x}_{-p} = \begin{bmatrix} x_{p0} \\ \dots \\ x_{pD} \end{bmatrix}$

— \Rightarrow **Vetor de alvo de todas as amostras:** $\underline{t} = \begin{bmatrix} t_1 \\ \dots \\ t_n \end{bmatrix}$

— **Formulação do mapeamento desejado como sistema de equações lineares**

$$\underline{X}\underline{w} = \underline{t}, \text{ com dimensões } (n \times (D + 1)) ((D + 1) \times 1) = (n \times 1)$$

$$\Rightarrow \underline{X}^T \underline{X}\underline{w} = \underline{X}^T \underline{t}$$

— **Solução determinística: PSEUDOINVERSA X^\dagger**

$$\underline{w} = (\underline{X}^T \underline{X})^{-1} \underline{X}^T \underline{t} \text{ com } X^\dagger = (\underline{X}^T \underline{X})^{-1} \underline{X}^T$$

ADALINE (cont.)

**ERRO QUADRÁTICO MÍNIMO: SOLUÇÃO DETERMINÍSTICA:
PSEUDOINVERSA, EXEMPLO: REGRESSÃO LINEAR**

Tabela 3: Exemplo: Regressão Linear

Exemplo # p	x_p	y_p
1	0.4	0.7
2	0.9	1.0
3	1.5	0.8
4	2.3	0.9
5	2.9	1.4
6	3.1	2.1
7	3.7	2.4

$$X = \begin{bmatrix} 1 & 0.4 \\ 1 & 0.9 \\ 1 & 1.5 \\ 1 & 2.3 \\ 1 & 2.9 \\ 1 & 3.1 \\ 1 & 3.7 \end{bmatrix}, \quad t = \begin{bmatrix} 0.7 \\ 1.0 \\ 0.8 \\ 0.9 \\ 1.4 \\ 2.1 \\ 2.4 \end{bmatrix} \Rightarrow X^T X = \begin{bmatrix} 7.0 & 14.8 \\ 14.8 & 40.2 \end{bmatrix}, \quad \underline{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = (X^T X)^{-1} X^T t = \begin{bmatrix} 0.32 \\ 0.47 \end{bmatrix}$$

Valor inicial:

ADALINE (cont.)

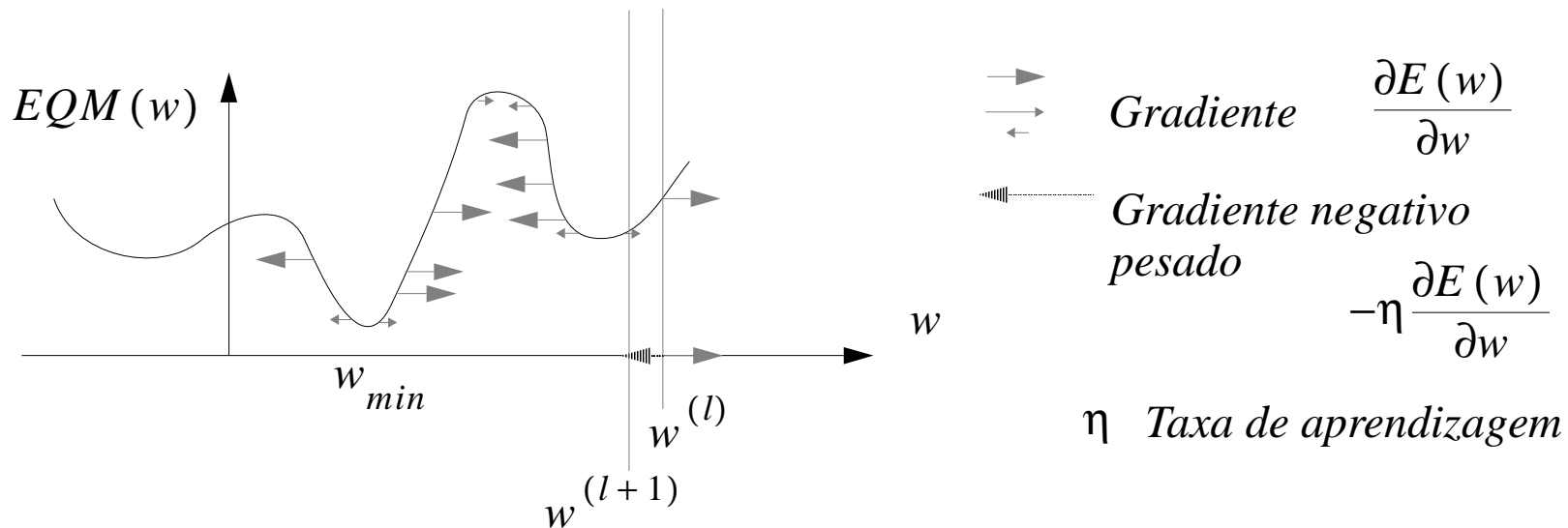
ERRO QUADRÁTICO MÍNIMO: SOLUÇÃO ITERATIVA: DESCIDA DE GRADIENTE

— **Função de custos:** erro quadrático médio EQM = função dos pesos \underline{w} da rede;

$$Erro(\underline{w}) = E(\underline{w}) = EQM(\underline{w}) = \frac{1}{n} \sum_{p=1}^n e^2(x_p) = \frac{1}{n} \sum_{p=1}^n \left(t_p - \underline{w}^T x_p \right)^2$$

— **Objetivo:** Iterativamente minimize EQM por adaptação dos pesos \underline{w} da rede

— **Caso ilustrativo:** único peso w :



— **Método:** Descida de gradiente

+ Movimenta \underline{w} na direção de descida de gradiente, procurando w_{min}

ADALINE (cont.)

ERRO QUADRÁTICO MÍNIMO: SOLUÇÃO ITERATIVA: DESCIDA DE GRADIENTE

- **Conhecido na iteração l**

- **Valor da função no ponto $w^{(l)}$: $EQM(w^{(l)})$**

- **Derivada da função no ponto $w^{(l)}$: $\nabla E = dE(\underline{w}) / d\underline{w}$ (1-D: $E'(w) = dE / dw$)**

- **Modificação de $w^{(l)}$ na direção do gradiente negativo ponderado \Rightarrow**

- **Regra de adaptação de peso por descida de gradiente:**

$$\underline{w}^{(l+1)} = \underline{w}^{(l)} + \Delta \underline{w}^{(l)} = \underline{w}^{(l)} - \eta \nabla E(\underline{w})^{(l)}$$

- **Parada da iteração: 1.) n fixo, 2.) gradiente muito pequeno**

- **Problemas**

- **Mínimo local**

- **Escolha da taxa de aprendizagem η**

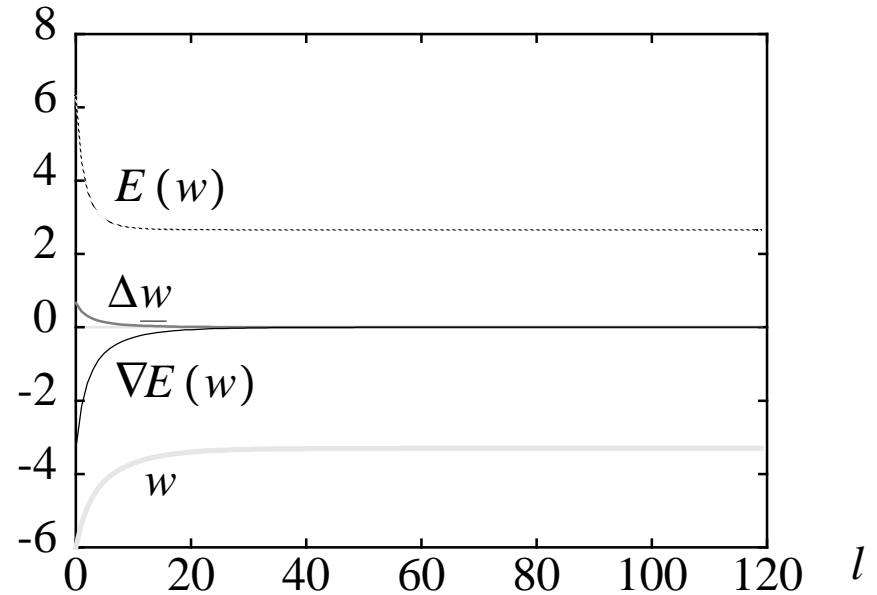
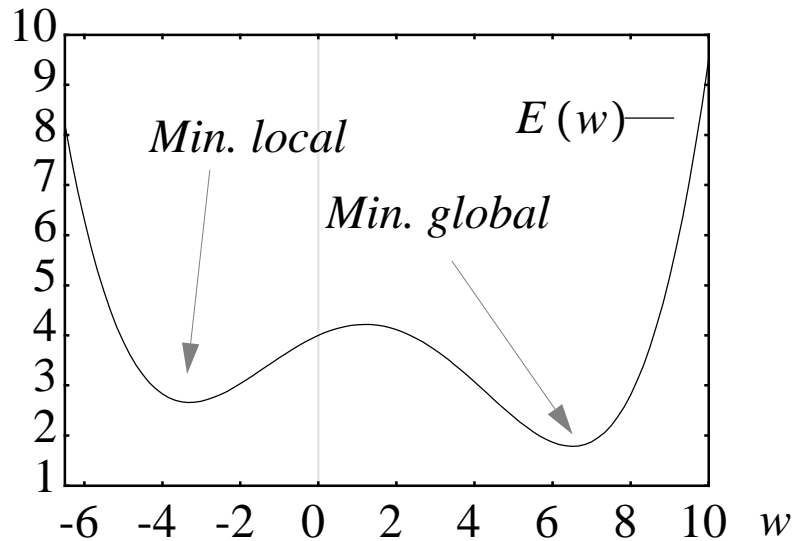
- + **Muito pequena \Rightarrow Aprendizagem lenta**

- + **Muito grande \Rightarrow Oscilações do erro**

ADALINE (cont.)

ERRO QUADRÁTICO MÍNIMO: SOLUÇÃO ITERATIVA: DESCIDA DE GRADIENTE

• **EXEMPLO** $\underline{w}^{(l+1)} = \underline{w}^{(l)} + \Delta \underline{w}^{(l)} = \underline{w}^{(l)} - \eta \nabla E(\underline{w})^{(l)}$



$$E(w) = 4 + .35w - .12w^2 - .02w^3 + .0034w^4$$

$$\nabla E(\underline{w}) = \frac{d}{dw}E(w) = .35 - .24w - .06w^2 + .0136w^3$$

$$w^{(0)} = -6$$

Valor do mínimo local:

$$w^{(120)} = -3.299762$$

ADALINE (cont.)

ERRO QUADRÁTICO MÍNIMO: SOLUÇÃO ITERATIVA: DESCIDA DE GRADIENTE

• **DEFINIÇÃO: FUNÇÃO DE CUSTOS** $Erro(\underline{w}) = E(\underline{w}) = EQM(\underline{w})$

• **ERRO:** $e_{-p} = t(x_{-p}) - d(x_{-p}) = t_p - \underline{w}^T x_{-p}$

Erro de aproximação do cálculo da entrada x_{-p} entre o desejado (valor de alvo) e o calculado (função de decisão)

• **Dois métodos de minimizar EQM**

— **1.) Aprendizagem estocástica**

+ **Atualização dos pesos depois da apresentação de cada entrada x_{-p} ao algoritmo de adaptação, ordem aleatória**

— **2.) Aprendizagem “batch”**

+ **Acumulação do erro para todas as entradas e depois atualização dos pesos**

ADALINE (cont.)

ERRO QUADRÁTICO MÍNIMO: DESCIDA DE GRADIENTE: Minimizar EQM

- **REGRA DE DELTA, REGRA DE ADALINE, REGRA DE WIDROW-HOFF, REGRA DAS MÉDIAS DO ERRO QUADRÁTICO MÍNIMO**

- **1.) Minimizar baseado numa única entrada x_{-p} : \Rightarrow aprendizagem estocástica**

— **Definição de erro:** $E = E(x_{-p}) = e^2(x_{-p}) = \left(t_p - \underline{w}^T x_{-p}\right)^2$

— **Cálculo do gradiente:** $\nabla E_j = \partial E / \partial w_j = \partial \left(e^2(x_{-p}) \right) / \partial w_j = 2e(x_{-p}) (\partial e(x_{-p}) / \partial w_j)$
 $= 2e(x_{-p}) (\partial (t_p - \sum_{j=0}^D w_j x_{pj}) / \partial w_j) = -2e(x_{-p}) x_{pj}$

— **Regra de adaptação** $w_j^{(l+1)} = w_j^{(l)} + \eta e(x_{-p}) x_{pj}$

- **2.) Minimizar sobre a média de todas as entradas x_{-p} : \Rightarrow aprendizagem “batch”:**

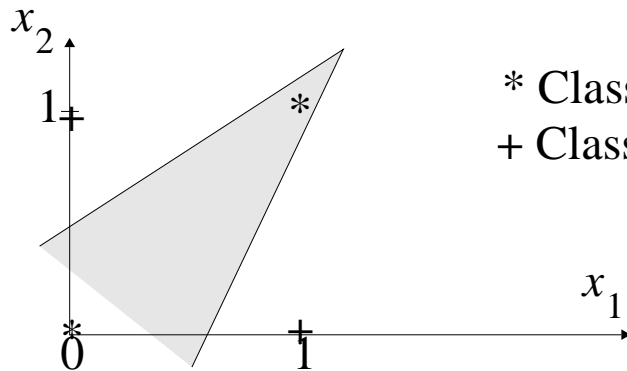
— **Definição de erro:** $E = \sum_{p=1}^n e^2(x_{-p})$

— **Cálculo do gradiente:** $\nabla E_j = \partial E / \partial w_j = -2 \sum_{p=1}^n e(x_{-p}) x_{pj}$

— **Regra de adaptação** $w_j^{(l+1)} = w_j^{(l)} + \eta \sum_{p=1}^n e(x_p) x_{pj}$

PERCEPTRON MULTI-CAMADA E RETROPROPAGAÇÃO DE ERRO

- Modelos lineares (Perceptron, ADALINE) incapazes de calcular funções não-lineares, e.g. problema XOR



* Classe 1 $f(0, 0) = f(1, 1) = 0$
+ Classe 2 $f(0, 1) = f(1, 0) = 1$

*Problema de classificação XOR:
Não existe função linear que
separa as duas classes*

- Minsky & Papert, 1969 “Perceptrons”
 - Introdução de mais camadas no perceptron resolve problema XOR (camadas escondidas)
 - Não se conhecia método para adaptar os pesos
 - Popularização do método de Retropropagação de Erro, ca. 1986

PERCEPTRON MULTI-CAMADA E RETROPROPAGAÇÃO DE ERRO (cont.)

- **Capacidade de regressão não-linear:**

— **Dado um conjunto de pares de treino** $\{ (x_{-p}, t_{-p}) \}_{p=1}^n$, **gera um conjunto de funções** $d(\underline{x}) = [d_1(\underline{x}) \dots d_c(\underline{x})]^T$, **com** $\underline{x} = [x_0 \dots x_D]^T$ **que minimiza o erro de aproximação entre o valor desejado** $t_{-p} = [t_{1p} \dots t_{cp}]^T$ **(vetor de alvo) e a resposta da função** $d(\underline{x})$

— \Rightarrow **Aproximador universal de funções**

- **Dois aspectos do perceptron multi-camadas**

- **Arquitetura (Modelo)**

- **Algoritmo de adaptação dos pesos**

- **Grande sucesso em várias áreas de aplicação**

- **Capacidade de modelagem de distribuições probabilísticas complexas +**

- **ausência de necessidade de conhecimento profundo do modelo (praticamente único grau de liberdade é número dos neurônios da camada escondida)**

- **Aplicações em regressão, classificação**

PERCEPTRON MULTI-CAMADAS: ARQUITETURA

- **Restrição: Modelo de duas camadas: Camada escondida e camada de saída (camada de entrada somente replica os valores de entrada)**
 - **Motivação: Teorema de Kolmogorov: PMC com duas camadas tem capacidade de aproximação universal de funções**

- **Processamento de um neurônio:**

- **1.) Combinação linear das entradas** $z = net = \sum_{j=0}^D w_j x_j$

- **2.) Função de ativação não-linear do resultado: e.g. função sigmoideal**

$$g(z) = \frac{1}{1 + e^{-z}}, \text{ com propriedade desejável } dg(z) / dz = g(z) (1 - g(z)), \text{ i.e.}$$

+ não-linear

+ derivável para qualquer z

+ derivada com forma simples

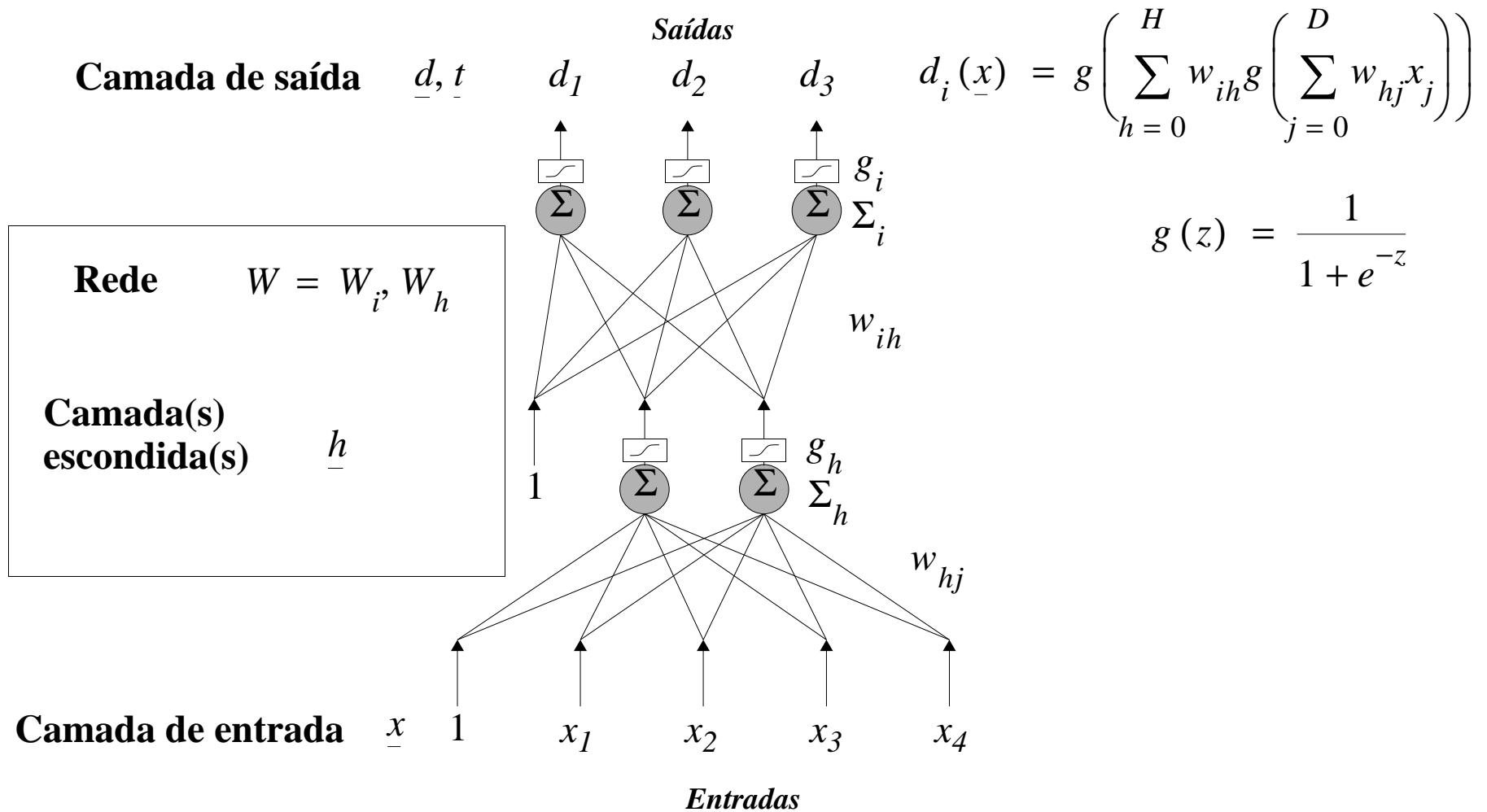
- **Modelo de duas camadas: Mapeamento:** $d_i(x) = g \left(\sum_{h=0}^H w_{ih} g \left(\sum_{j=0}^D w_{hj} x_j \right) \right)$

H : número dos neurônios da camada escondida

w_{hj} : Pesos entrada-escondida, w_{ih} : Pesos escondida-saída

PERCEPTRON MULTI-CAMADAS: REPRESENTAÇÃO GRÁFICA

• Rede Neural Artificial



PERCEPTRON MULTI-CAMADAS: ADAPTAÇÃO:

REGRA DE DELTA GENERALIZADA

- **Generalização dos conceitos do ADALINE**
- **Solução iterativa: Erro quadrático mínimo e descida de gradiente**

— **Função de custos: Erro quadrático mínimo**

(usando conjunto de treino finito $\{ (x_{-p}, t_{-p}) \}_{p=1}^n$):

$$EQM = \frac{1}{n} \sum_{p=1}^n (\text{desejado}(x_{-p}) - \text{calculado}(x_{-p}))^2$$

— **Objetivo: Minimize EQM por adaptação dos parâmetros livres da função calculada $d(\underline{x}) = \text{calculado}(\underline{x})$**

— **PMC: $EQM = \frac{1}{n} \sum_{p=1}^n \left\| t_{-p} - (g_1(x_{-p}), \dots, g_c(x_{-p}))^T \right\|^2$, g_i na camada de saída**

PERCEPTRON MULTI-CAMADAS: ADAPTAÇÃO (cont.)

— Matrizes de pesos

+ Camada de entrada — camada escondida $W_h = [w_{hj}]$

+ Camada escondida — camada de saída $W_i = [w_{ih}]$

— Objetivo: Minimize EQM por adaptação dos pesos $[w_{hj}]$ e $[w_{ih}]$

+ Camada de entrada — camada escondida:
REGRA DE DELTA

+ Camada escondida — camada de saída:
REGRA DE DELTA GENERALIZADA

• Adaptação de todos os pesos: Descida de gradiente

$$w^{(l+1)} = w^{(l)} + \Delta w^{(l)} = w^{(l)} - \eta \nabla E^{(l)}$$

• Dois métodos de minimizar EQM

— 1.) Aprendizagem estocástica

+ Atualização dos pesos depois da apresentação de cada entrada x ao
algoritmo de adaptação, ordem aleatória

— 2.) Aprendizagem “batch”

+ Acumulação do erro para todas as entradas e depois atualização dos pesos

PERCEPTRON MULTI-CAMADAS: ADAPTAÇÃO (cont.)

SAÍDA—ESCONDIDA: Minimizar EQM

• **REGRA DE DELTA, REGRA DE ADALINE, REGRA DE WIDROW-HOFF, REGRA DAS MÉDIAS DO ERRO QUADRÁTICO MÍNIMO**

• **1.) Minimizar baseado na entrada x_{-p} : \Rightarrow aprendizagem estocástica**

— **Definição do erro:** $E = e^2(x_{-p}) = \left\| t_{-p} - d(x_{-p}) \right\|^2 = \sum_{i=0}^c (t_{pi} - d_i(x_{-p}))^2$,

— **Notação:**

$$d_i(x_{-p}) = g_i = g(\Sigma_i) = g\left(\sum_{h=0}^H w_{ih}g_h\right) = g\left(\sum_{h=0}^H w_{ih}g(\Sigma_h)\right) = g\left(\sum_{h=0}^H w_{ih}g\left(\sum_{j=0}^D w_{hj}x_j\right)\right)$$

— **Cálculo do gradiente ∇E_{ih} :**

$$\begin{aligned}\nabla E_{ih} &= \partial E / \partial w_{ih} = \partial e^2 / \partial w_{ih} = \partial \left(\sum_{i=1}^c e_i^2 \right) / \partial w_{ih} = \partial \left(\sum_{i=1}^c (t_i - g_i)^2 \right) / \partial w_{ih} \\ &= \sum_{i=1}^c \partial (t_i - g_i)^2 / \partial w_{ih} = (-2 \sum_{i=1}^c (t_i - g_i) (\partial g_i / \partial w_{ih})) = -2 (t_i - g_i) (\partial g_i / \partial w_{ih}) \\ &= -2e_i (\partial g_i / \partial w_{ih}) = -2e_i [g_i (1 - g_i) (\partial \Sigma_i / \partial w_{ih})] = -2e_i [g_i (1 - g_i) g_h] = -2\delta_i g_h\end{aligned}$$

PERCEPTRON MULTI-CAMADAS: ADAPTAÇÃO (cont.)

ESCONDIDA—ENTRADA: Minimizar EQM

- **REGRA DE DELTA GENERALIZADA**

- **1.) Minimizar baseado na entrada x_{-p} : \Rightarrow aprendizagem estocástica**

- **Definição de erro:** $E = \delta_{pi}^2$, $\delta_i = t_i(x_{-p}) - d_i(x_{-p})$

- **Cálculo do gradiente:**

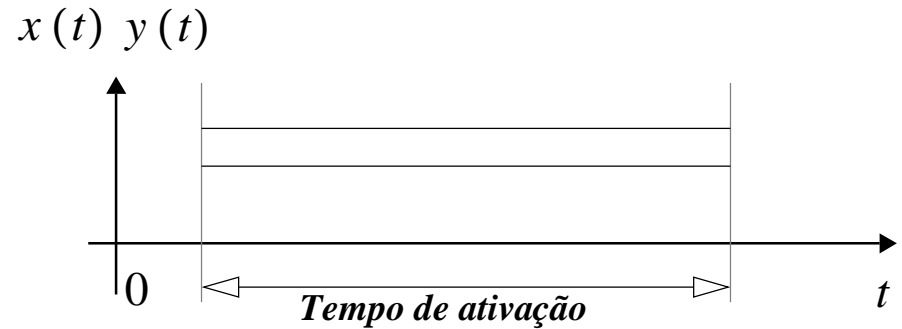
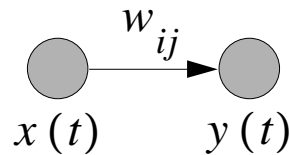
$$\begin{aligned}\nabla E_{hj} &= \partial E / \partial w_{hj} = \partial e^2 / \partial w_{hj} = \partial \left(\sum_{i=1}^c e_i^2 \right) / \partial w_{hj} = \partial \left(\sum_{i=1}^c (t_i - g_i)^2 \right) / \partial w_{hj} \\ &= \sum_{i=1}^c \partial (t_i - g_i)^2 / \partial w_{hj} = (-2 \sum_{i=1}^c (t_i - g_i) \partial g_i / \partial w_{hj}) = -2 \sum_{i=1}^c e_i (\partial g(\Sigma_i) / \partial w_{hj}) \\ &= -2 \sum_{i=1}^c e_i g_i (1 - g_i) (\partial \Sigma_i / \partial w_{hj}) = -2 \sum_{i=1}^c e_i g_i (1 - g_i) \left[\partial \left(\sum_{h=0}^H w_{ih} g_h \right) / \partial w_{hj} \right] \\ &= -2 \sum_{i=1}^c e_i g_i (1 - g_i) \left[\left(\sum_{h=0}^H w_{ih} \partial g_h \right) / \partial w_{hj} \right] = -2 \sum_{i=1}^c e_i g_i (1 - g_i) \left[\left(\sum_{h=0}^H w_{ih} \partial g_h \right) / \partial w_{hj} \right] \\ &= -2 \sum_{i=1}^c e_i g_i (1 - g_i) \left[g_h (1 - g_h) (\partial \Sigma_h / \partial w_{hj}) \right] = -2 \sum_{i=1}^c e_i g_i (1 - g_i) \left[w_{ih} g_h (1 - g_h) x_j \right] \\ &= -2 g_h (1 - g_h) \sum_{i=1}^c e_i g_i (1 - g_i) w_{ih} x_j = -2 \delta_h x_j\end{aligned}$$

- \Rightarrow **RETROPROPAGAÇÃO DE ERRO: Deltas retropropagadas pelas camadas**

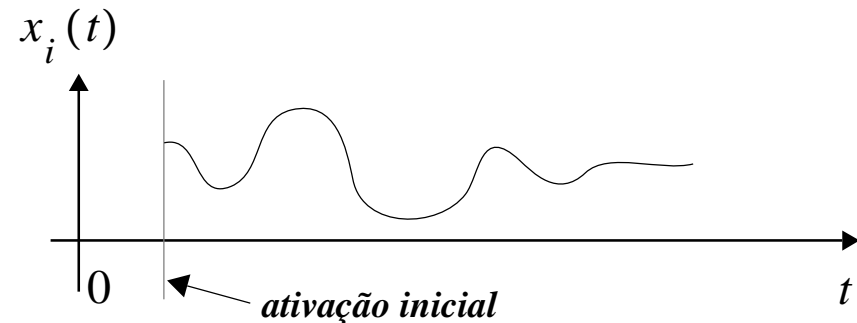
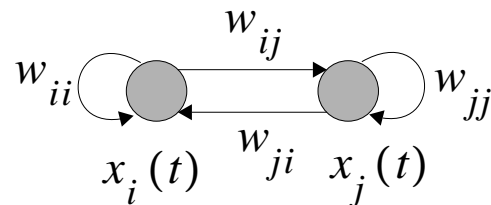
REDES COM REALIMENTAÇÃO

• COMPORTAMENTO ESTÁTICO VS. COMPORTAMENTO DINÂMICO

Propagação para Frente



Realimentação



— Redes com Propagação para Frente

+ Fluxo de informação unidirecional

+ Comportamento determinístico \Rightarrow Modelagem de MAPEAMENTOS

— Redes com Realimentação

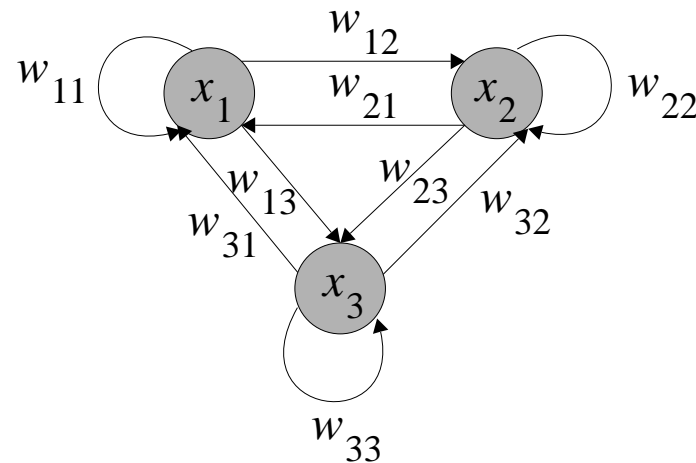
+ Fluxo de informação multidirecional

+ Comportamento estocástico, estados internos da rede

\Rightarrow Modelagem de PROCESSOS DINÂMICOS

MODELO DE HOPEFIELD, 1982

- Neurônio $x_i(t)$ representa **ENTRADA, SAÍDA E ESTADO** no momento t



- **Valor BINÁRIO** $x_i(t) \in \{-1, 1\}$
- **REDE:** H neurônios \Rightarrow Rede em **ESTÁDO** $\underline{x}(t) \in \{-1, 1\}^H$ no momento t
 - Exemplo com 6 neurônios: $Estado = (-1, 1, 1, -1, 1, -1)^T = \underline{x}(t)$
- **MODELO** de neurônio de McCulloch-Pitts, com $\text{sgn}(\cdot)$ como função de ativação
- **Regra de ADAPTAÇÃO DINÂMICA** da saída de neurônio i : $x_i = \text{sgn}(\sum_{j=0}^D w_{ij} x_j)$

MODELO DE HOPEFIELD (cont.)

- **Comportamento dinâmico:** mudança do estado $\underline{x}(t)$ no momento t para o estado $\underline{x}(t + \Delta t)$ no momento $t + \Delta t$
- **Estabilidade:** $\underline{x}(t + \Delta t) = \underline{x}(t)$ para todos $\Delta t \geq 0$
- **Simulação do comportamento dinâmico:** Atualização sequencial aleatória de $x_i(t)$ para $x_i(t + 1)$

• ASSOCIATIVIDADE DE PADRÕES

— **Objetivo da rede:** memorizar n padrões x_{-p} de um conjunto de padrões

$$T = \{x_{-p}\}_{p=1}^n, \text{ e.g. imagens binárias}$$

— **“RECALL”:** Estímulo inicial $\underline{x}(0) = x_{-p} \Rightarrow$ Resposta final $\underline{x}(\infty) = x_{-p}$

— **“RECALL” ASSOCIATIVO:** Estímulo inicial $\underline{x}(0) \approx x_{-p} \Rightarrow$ Resposta final

$$\underline{x}(\infty) = x_{-p}$$

+ \Rightarrow Capacidade de recuperar informação memorizada com informação inicial incompleta, corrompida ou parcialmente errada

— MEMÓRIA ASSOCIATIVA

MODELO DE HOPEFIELD (cont.)

ESTABILIDADE DA REDE E APRENDIZAGEM DE PESOS

• **Único Padrão:** $\underline{x} = (x_1, \dots, x_H)^T$

— **Regra de Adaptação Dinâmica do neurônio i :** $x_i(t + \Delta t) = \text{sgn}(\sum_{j=0}^D w_{ij} x_j(t))$
não altera estado da rede, i.e. $x_i(t + \Delta t) = x_i(t)$ para todos os neurônios

$$i = 1, \dots, H$$

— **Substituindo o peso w_{ij} pela expressão $x_i(t) \cdot x_j(t)$ na regra de Adaptação Dinâmica** \Rightarrow

$$x_i(t + \Delta t) = \text{sgn}\left(\sum_{j=0}^D x_i(t) \cdot x_j(t) \cdot x_j(t)\right) = \text{sgn}\left(\sum_{j=0}^D x_i(t) \cdot 1\right) = \text{sgn}(D \cdot x_i(t)) = x_i(t)$$

— **Normalização por H :** \Rightarrow

— **REGRA DE APRENDIZAGEM para o peso w_{ij} para um padrão:** $w_{ij} = \frac{1}{H} x_i x_j$,
usando paradigma da **REGRA DE HEBB**

— **RELAXAÇÃO** do estado da rede para \underline{x} (mesmo com a metade (menos 1) das entradas diferente do padrão \underline{x})

MODELO DE HOPEFIELD (cont.)

ESTABILIDADE DA REDE E APRENDIZAGEM DE PESOS (cont.)

- n padrões $T = \{x_p\}_{p=1}^n$: $x_p = (x_{p1}, \dots, x_{pH})^T$

- Superposição do peso w_{ij} para todos os padrões

- REGRA DE APRENDIZAGEM para o peso w_{ij} para n padrões:

REGRA DE HEBB GENERALIZADA $w_{ij} = \frac{1}{H} \sum_{p=0}^n x_{pi} x_{pj}$

x_{pi} : estado do neurônio i para padrão número p

- \Rightarrow Método direto de calcular os pesos w_{ij} entre os neurônios, dado um conjunto de padrões T

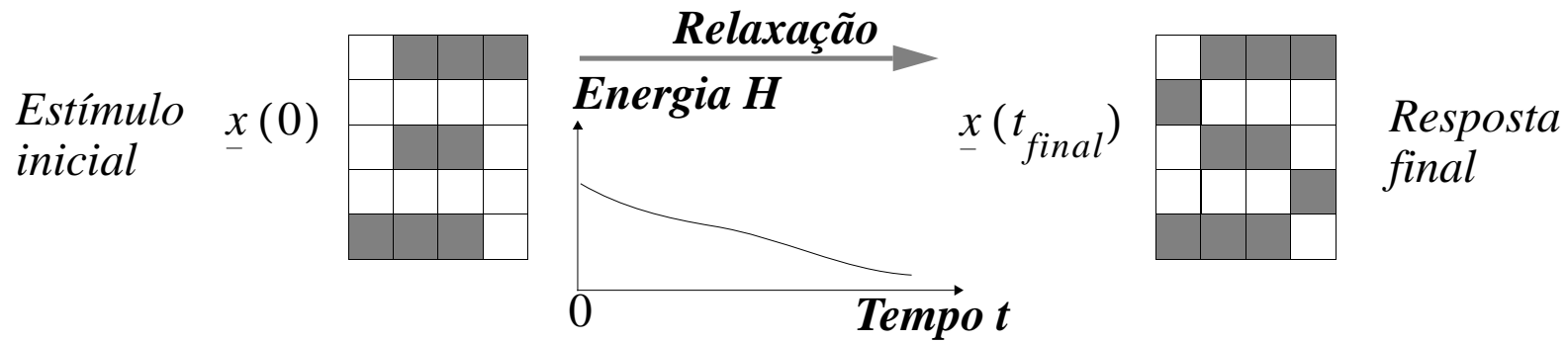
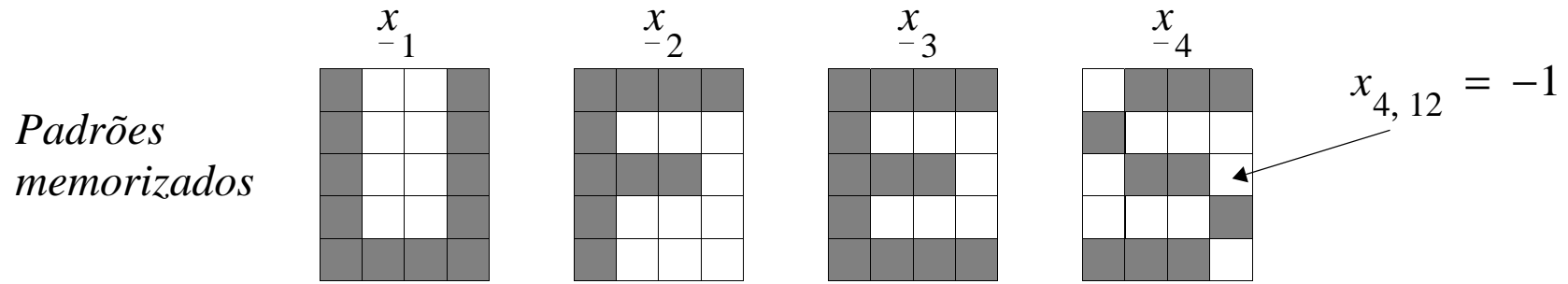
MODELO DE HOPEFIELD (cont.)

RELAXAÇÃO, MINIMIZAÇÃO DE ENERGIA

- Pesos entre neurônios são simétricos, i.e. $w_{ij} = w_{ji}$
- Peso nulo da realimentação do próprio neurônio i.e. $w_{ii} = 0$
- \Rightarrow Introdução de FUNÇÃO DE ENERGIA:
$$H = -\frac{1}{2} \sum_{i=0}^H \sum_{j=0}^H w_{ij} x_i x_j$$
- \Rightarrow Garantia de relaxação para estado estável (pesos simétricos obrigam diminuição da função de energia)
- Existência de estados $\tilde{x} \neq x$
 - + Estados reversos
 - + Estados espúrios
 - + não correspondem aos padrões memorizados

MODELO DE HOPEFIELD (cont.)

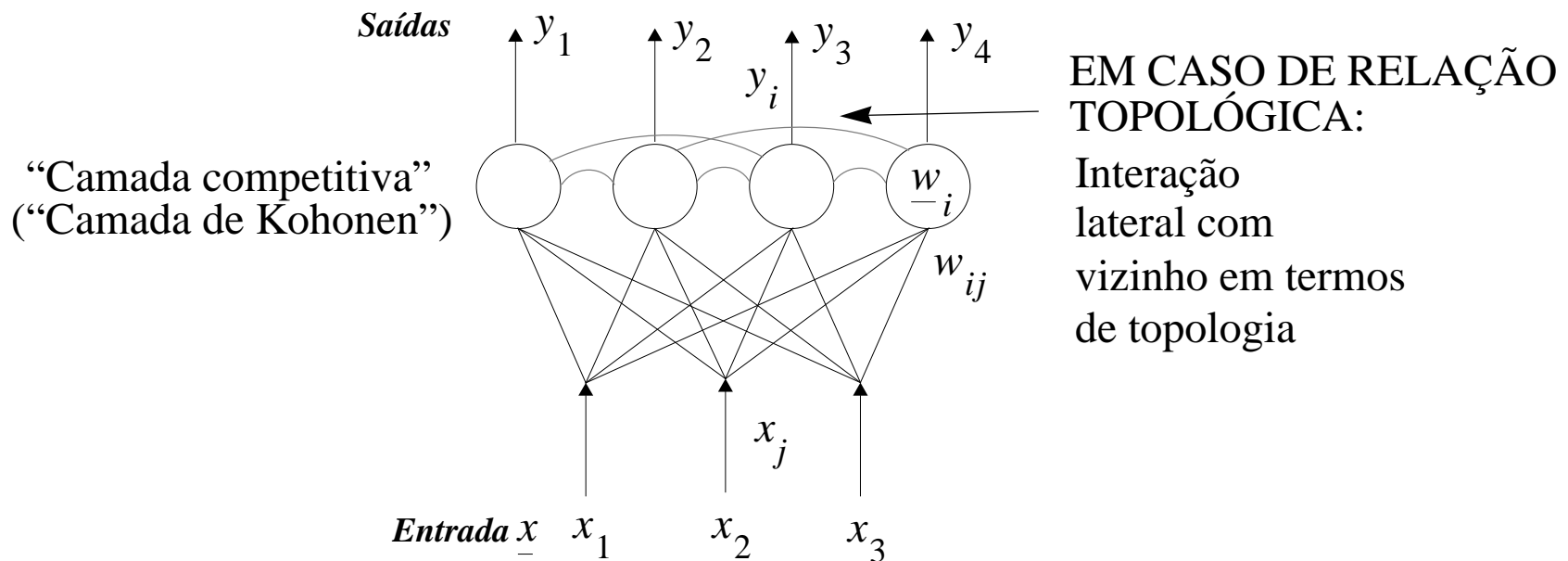
APLICAÇÃO: RECUPERAÇÃO DE IMAGENS



- Imagem binária com dimensão $A \cdot B \Rightarrow$ Rede de Hopfield com $H = A \cdot B$ neurônios
- Uma imagem = um padrão $\underline{x}_{-p} = (x_{p1}, \dots, x_{pH})^T$ (linearização da imagem)
- Memória associativa de um conjunto de imagens $\{\underline{x}_{-p}\}$

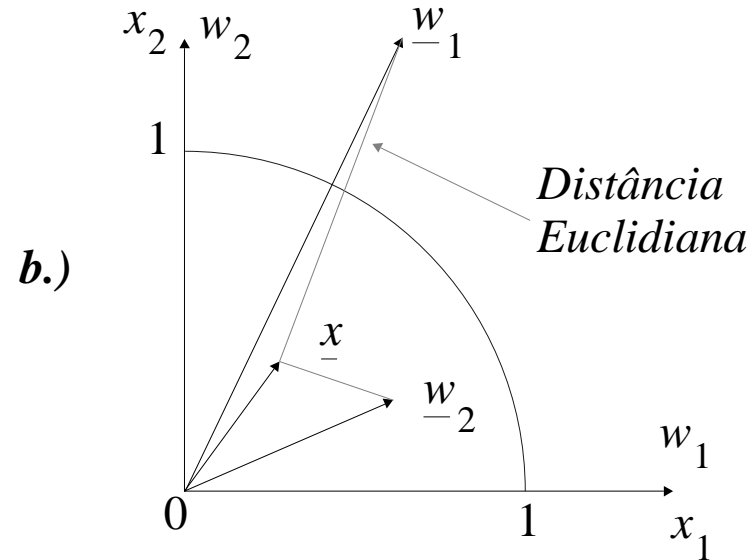
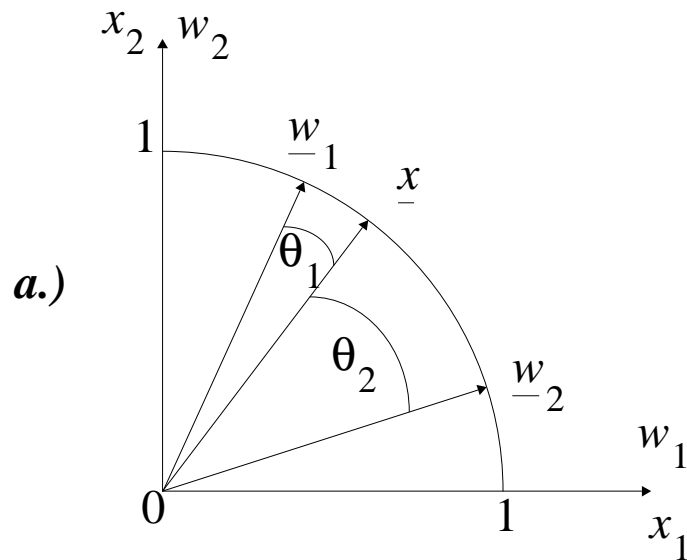
REDES COMPETITIVAS

- **Aprendizagem não-supervisionada: Conjunto de exemplos** $T = \{ (x_{-p}) \}_{p=1}^n$
- **Todos neurônios i recebem mesmo estímulo \underline{x}**
 - **Entre entrada x_j e neurônio y_i existe peso w_{ij} (vetor \underline{w}_{-i} liga entrada \underline{x} a saída y_i)**
- **Todos neurônios formam camada competitiva**
 - **Vencedor dentro da camada competitiva \Rightarrow Emissão de sinal $y_{i^*} = 1$**
(“winner takes all”), restantes neurônios inativos.



DETERMINAÇÃO DO VENCEDOR

- Determinação de semelhança entre estímulo \underline{x} e pesos \underline{w}_i
- Duas possibilidades de determinar vencedor dentro da camada competitiva:



— a.) **MAIOR PRODUTO INTERNO (= MENOR ÂNGULO θ_i ENTRE \underline{x} E \underline{w}_i)**

+ Normalização de \underline{x} e \underline{w}_i para comprimento 1: $\underline{x} \leftarrow \underline{x} / \|\underline{x}\|$, $\underline{w}_i \leftarrow \underline{w}_i / \|\underline{w}_i\|$

+ Vencedor: $w_{i^*}^T \underline{x} > w_i^T \underline{x}$

— b.) **MENOR DISTÂNCIA (EUCLIDIANA)**

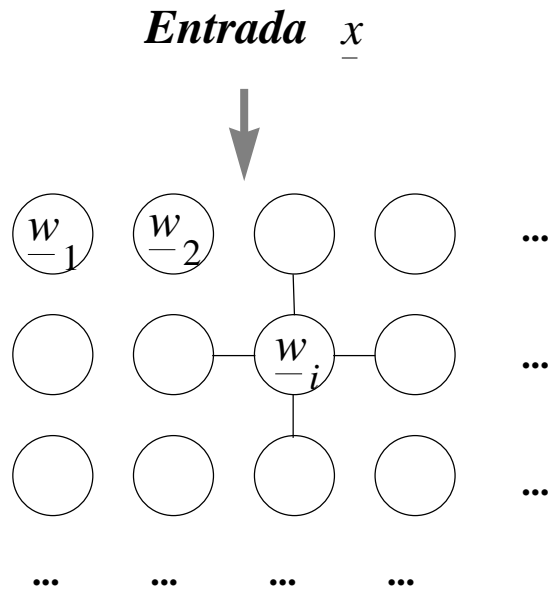
$$+ \left\| \underline{x} - \underline{w}_i \right\| = \sqrt{\sum_{j=0}^D (x_j - w_{ij})^2}$$

ADAPTAÇÃO DOS PESOS

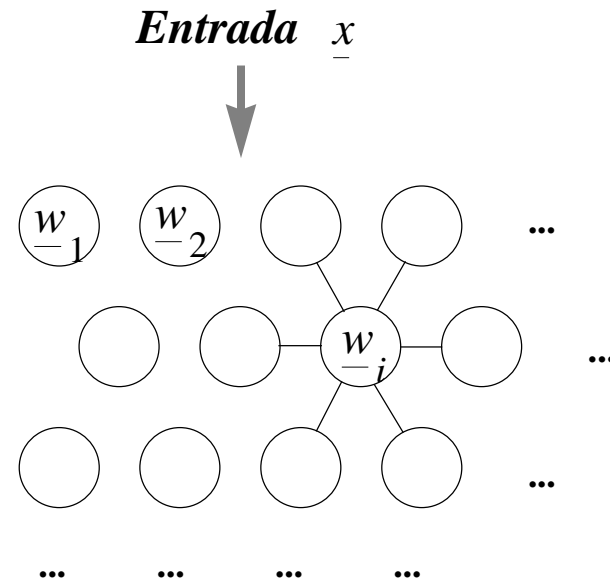
- **Método iterativo de adaptação de pesos** $w_{ij}^{(l+1)} = w_{ij}^{(l)} + \Delta w_{ij}^{(l)}$
- **Modificação dos pesos exclusivamente baseado na estrutura dos dados de treino T**
 - **Estímulos x_{-p} e x_{-q} parecidos provocam a resposta do mesmo vencedor**
 - **Movimentação dos pesos em direção aos estímulos (em direção ao peso do neurônio vencedor w_{i^*})** $\Rightarrow \Delta w_{i^*} = x_{-p} - w_{i^*}$
 - **Ponderação da velocidade de aproximação pela taxa de aprendizagem η**
 - **Saídas dos neurônios não-vencedoras nulos para $i \neq i^*$** \Rightarrow
- **REGRA DE APRENDIZAGEM COMPETITIVA:** $w_{ij}^{(l+1)} = w_{ij}^{(l)} + \eta y_{pi} (x_{pj} - w_{ij}^{(l)})$
 - y_{pi} : saída de neurônio i para entrada x_{-p}
 - x_{pj} : j -ésima componente da entrada x_{-p}

MAPA DE PRESERVAÇÃO TOPOLÓGICA DE KOHONEN

- **Relação topológica entre neurônios da mesma camada (camada competitiva)**
 - Na fase da adaptação dos pesos: efeito colateral sobre vizinhos
 - **Motivação biológica: no cérebro estímulos sensoriais externos parecidos excitam áreas vizinhas do cérebro**



*Mapa auto-organizável de Kohonen
Topologia retangular*



*Mapa auto-organizável de Kohonen
Topologia hexagonal*

- **Relação topológica 2-D**
 - **Topologia retangular: Cada neurônio tem 4 vizinhos imediatos**
 - **Topologia hexagonal: Cada neurônio tem 6 vizinhos imediatos**

MAPA DE PRESERVAÇÃO TOPOLÓGICA DE KOHONEN (cont.)

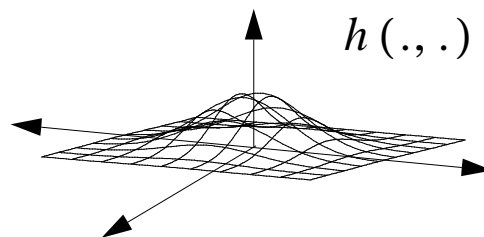
- Na fase de aprendizagem: Modificação de todos os pesos, dependendo da função de relação topológica (função de vizinhança) $h(i^*, i)$ entre vencedor i^* e seus vizinhos i , com $h(i^*, i^*) = 1$

- **REGRA DE APRENDIZAGEM COMPETITIVA NA CAMADA DE KOHONEN:**

$$w_{ij}^{(l+1)} = w_{ij}^{(l)} + \eta h(i^*, i) (x_{pj} - w_{ij}^{(l)})$$

- **FUNÇÃO DE RELAÇÃO TOPOLÓGICA (FUNÇÃO DE VIZINHANÇA) $h(i^*, i)$:**

- Valor máximo no centro (vencedor)
- Valor decrescente com distância crescente do centro
- Exemplo: função Gaussiana



Gaussiana em 2-D

- Atração dos pesos dos vizinhos do vencedor em direção do estímulo, mas em quantidade menor

CONCLUSÕES

REDES NEURAIS ARTIFICIAIS

- **Conceitos básicos da neurocomputação**
 - **Neurônio artificial**
 - **Topologia da rede (Propagação para Frente, Realimentados)**
 - **Paradigmas de aprendizagem (supervisionado, não-supervisionado)**
 - **Regras para adaptação dos pesos (Hebb, Delta, Competitiva)**
 - **Limitações da classe de problemas resolúveis**
- **Rede neural artificial como aproximador universal de funções**
 - **Problema mapeável para função?**
 - **Aprendizagem por exemplos**
 - **Poucos graus de liberdade \Rightarrow Caráter universal**
- **Sistemas inteligentes cognitivos no futuro**
 - **Baseados em princípios de redes neurais artificiais???**