PARTE I INTRODUÇÃO

1.1. HEURÍSTICAS

As principais características de uma heurística são:

- consiste em um método ou técnica aproximativa;
- desenvolvido para resolver um tipo de problema;
- tempo máximo polinomial (ideal);
- não garantem soluções ótimas.

1.2. METAHEURÍSTICAS

As principais características de uma metaheurística são:

- consiste em um método ou técnica aproximativa;
- resolve de forma genérica:
  - problemas de otimização;
  - classificação;
  - agrupamentos, etc
- heurísticas de uso geral ou heurística de heurísticas;
- boas soluções mas, não garantem soluções ótimas [Viana, 1998];
- podem ser definidas como **heurísticas "derivadas" da Natureza** – área limite entre a Pesquisa Operacional e a Inteligência Artificial [Colorni et al., 1996];
- inspiradas na Física, Biologia, Ciências Sociais;
- operam através de repetições de tentativas;
- utilizam um ou mais agentes (neurônios - RNA, partículas - *SA* ou *PSO*, cromossomos - AG, formigas – *ACO*);
- usam mecanismo de competição-cooperação;
- são geralmente aplicadas a problemas que não se conhece algoritmo eficiente;
- geram bons resultados em problemas de otimização combinatorial *NP-hard* (Não-Polinomial Árduo é um problema que é pelo menos tão difícil quanto qualquer problema em *NP*; a classe *NP*-Completo contém os problemas de maior dificuldade dentre todos em *NP*).

As metaheurísticas mais usadas são:

- *Genetic Algorithm* ou Algoritmo Genético (AG);
- *Artificial Neural Networks* ou Redes Neurais Artificiais (RNA ou RN);
- *Simulated Annealing* ou Têmpera Simulada (SA);
- *Tabu Search* ou Busca Tabu (BT);
- *Particle Swarm Optimization* ou Nuvem de partículas (PSO);

- *Ant Colony Optimization* ou Colônia de Formigas (ACO);
- *Greedy Randomized Adaptive Search Procedures* (GRASP);
- Algoritmos Meméticos (AM);
- Algoritmos Híbridos.

Os procedimentos comuns para melhoria de resultados são:

- emprego de mais agentes;
- procedimentos de auto-modificação dos parâmetros heurísticos ou da representação do problema.

Elementos da natureza

- Na natureza:
  - **seleção** premia indivíduos mais fortes e penaliza os mais fracos;
  - **mutação** introduz elementos aleatórios, permitindo o nascimento de novos indivíduos.
- Nas heurísticas:
  - **seleção** idéia básica para otimização (busca de melhores soluções);
  - **mutação** usada para procura não-determinística (melhoria).

Características das heurísticas da natureza:

- modelam um fenômeno da natureza;
- não-determinísticas;
- apresentam implicitamente estrutura paralela (múltiplos agentes);
- adaptativas: capacidade do sistema de usar informação anterior para modificar seus parâmetros e seu modelo interno.
  - exemplo: RNA, com a modificação dos pesos das ligações.

Classificação das Metaheurísticas

Metaheurísticas de relaxação:

- usam relaxação do modelo original;
- modificações do problema original para facilitar a solução;
- exemplo: relaxação Lagrangeana.

Metaheurísticas construtivas:

- estrutura que representa a solução: inicialmente vazia;
- incorporam iterativamente elementos à estrutura;
- baseadas em uma função gulosa;
- escolhe elementos que produzem melhores resultados imediatos;
- não possuem espaço de soluções estruturado;
- exemplos: Algoritmo Multistart, Colônia de Formigas, GRASP e Redes Neurais Artificiais. [Santana et al., 2004].

Metaheurísticas de busca:

- estratégias para percorrer o espaço de soluções do problema;
- transformam de forma iterativa a solução inicial;
- exemplo: busca local *k-opt*.

Metaheurísticas evolutivas:

- estratégias evolutivas no espaço de busca do conjunto de soluções;
- utiliza um conjunto de soluções simultaneamente;
- define a melhor solução do conjunto para gerar mais soluções;
- exemplos: Simulated Annealing, Busca Tabú, Algoritmos Genéticos e Algoritmos Meméticos.

Metaheurísticas de decomposição:

- determinam subproblemas a partir do problema original;
- intermediárias entre as de relaxação e as construtivas;
- objetivo principal: obter subproblemas mais fáceis de resolver que os originais.

	Espaço não-estruturado Solução única	Espaço não-estruturado Solução de população	Espaço estruturado Solução única	Espaço estruturado Solução de população
alg. construtivos	RNA	MS	GR	ACO
alg. de melhorias	GRASP	AG, ES	SA, TS	SC

MS: Algoritmo Multistart – Algoritmo que inicia de diferentes pontos iniciais
GR: Greedy Random – Método de procura randômica gulosa
ACO: Ant Colony Optimization – Colônia de Formigas
ES – Evolution Strategies
SC – Sampling and Clustering – Amostragem e Agrupamento.

Características importantes

- grau de *exploitation* (profundidade)
  - esforço da busca local em regiões do espaço de busca;
  - se a região é promissora, procure mais profundamente.
- grau de *exploration* (amplitude)
  - esforço da procura em regiões distantes;
  - algumas vezes se escolhe uma solução em uma região distante;
  - aceita-se uma solução pior para descobrir novas soluções;
  - probabilidade de encontrar melhores soluções.

São características conflitantes: uma boa troca entre elas é muito importante e devem ser cuidadosamente afinadas em cada algoritmo.

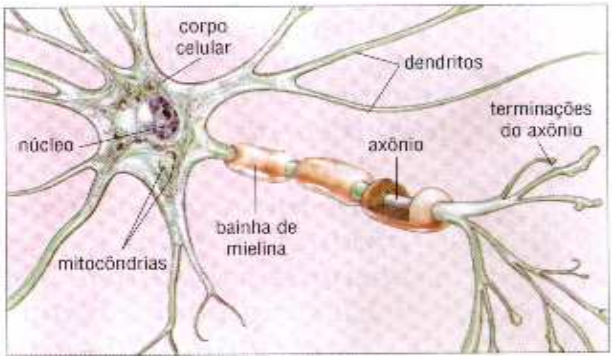
Outra troca que deve ser avaliada: esforço (número de iterações) x eficácia (solução final).

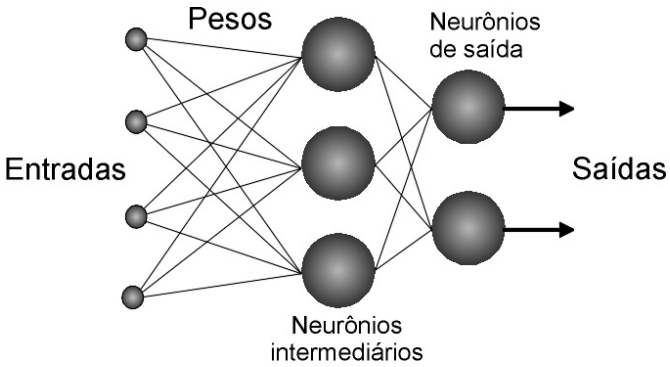
Alguns algoritmos usam um parâmetro de controle:

- também chamado de taxa de aprendizagem ou de equilíbrio;
- varia lentamente para evitar ótimos locais e permitir uma exploração maior do espaço de soluções;
- quanto mais lentamente variar, maior é a probabilidade de encontrar uma solução ótima global;
- existem metaheurísticas para procurar o valor ótimo deste parâmetro de controle.

PARTE II
REDES NEURAIS ARTIFICIAIS

Uma rede neural artificial consiste de um modelo inspirado no cérebro humano.
Tem habilidade de adquirir e armazenar conhecimento para realizar uma tarefa.
A motivação biológica tem os elementos básicos que são **neurônios** ou **nós**.





São compostas de elementos simples, inspirados pelo **sistema nervoso biológico.**
Operam em paralelo, onde a função é determinada pelas **conexões entre os neurônios.**
Pode-se treinar uma rede neural para executar uma função particular ajustando-se os valores das conexões entre os elementos.

Sinônimos:

- Sistemas Neurais Artificiais;
- Conexionismo;
- Sistemas Adaptativos;
- Neurocomputadores;
- Sistemas Paralelos Distribuídos.

Histórico:

- paradigmas básicos:
  - simbólico
  - conexionista
- perceptron: 1 camada de pesos ajustáveis-1957;
- descrédito a partir do final da década 60;
- em 1974, Werbos lança bases para o algoritmo Backpropagation;
- impulso a partir da década de 80:
  - 1985, com a rede de Hopfield, aplicada ao PCV;
  - 1986, com o Backpropagation;
- pouco formalismo matemático em muitas áreas.

Etapas de um projeto:

- definição do problema;
- escolha das informações:
  - obtenção dos dados; criação de arquivos da rede;
- treinamento da rede;
- testes da rede;
- adaptação para uso no problema definido.

Razões para usar uma RNA:

- paralelismo;
- capacidade de adaptação;
- memória distribuída;
- capacidade de generalização;
- facilidade de construção;
- desempenho depende da qualidade do pré-tratamento dos dados.

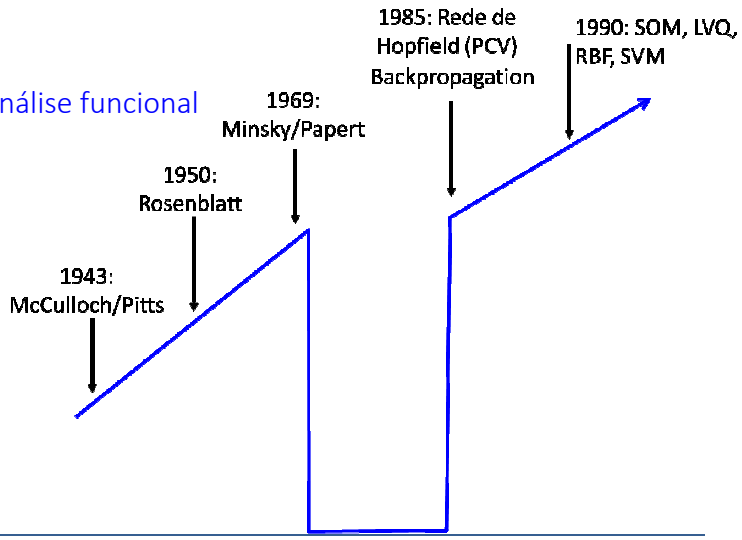
Alguns exemplos de aplicações das RNAs:

- regras desconhecidas de resolução de um problema ou difíceis de formalizar;
- problemas com conjuntos de exemplos e suas soluções;
- necessidade de grande rapidez na resolução do problema (respostas em tempo real);
- não existem soluções tecnológicas atuais;
- reconhecimento de formas;
- tratamento de sinal;
- visão, fala;
- previsão e modelagem;
- auxílio à decisão;
- problemas de otimização [Siqueira, Scheer, Steiner, 2005, 2006, 2007];
- robótica;
- diagnóstico médico [Bennett e Mangasarian, 1990], [Steiner, Carnieri, 1994];
- predição de falência bancária [Tam et al, 1992];
- aplicação à fabricação da pasta e papel industrial [Fadum, 1993];
- controle do processo de produção do papel industrial [Rudd, 1991], [Steiner et al, 1994];
- mundo financeiro [Business Week, 1993], [Cipra, 1992];
- controle de processos químicos [Nascimento et al, 1993];
- obtenção de um modelo organizacional [Almeida, 1995];
- detecção fraudes com cartões [Financial Times, 1993];
- problemas de administração de empresas [Almeida, 1995].

Relevância matemática:

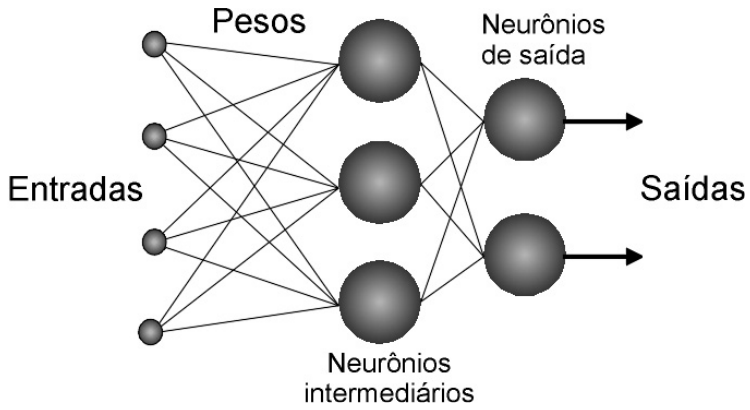
A matemática é utilizada em RNA para:

- desenvolver e propor algoritmos
  - álgebra linear, análise, estatística, otimização, física, geometria
- investigar aplicabilidade, avaliar algoritmos
  - estatística
- investigar propriedades teóricas
  - álgebra, análise, álgebra linear, análise funcional



2.1. ELEMENTOS DE UMA REDE NEURAL

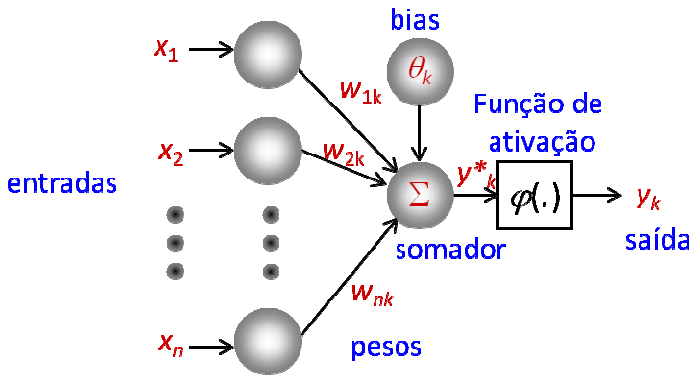
- conjunto de **pesos sinápticos** (ou **conexões sinápticas**);
- uma operação binária: combina entradas com as conexões;
- **regra de agregação:** combina entradas dos neurônios ponderados com as respectivas conexões sinápticas;
- **função de ativação:** introduz não-linearidade no modelo, ou pode confinar a saída do neurônio em um intervalo;
- **topologia organizada**: é o projeto da rede e determina o modo como os neurônios são conectados;
- **aprendizagem**: processo que modifica os pesos com a intenção de se atingir um objetivo.



Dendritos: entradas
Corpo celular:
- Soma ponderada
- Função não-linear

Axônio: distribuição aos neurônios

Elemento Processador (PE) de McCulloch-Pitts (1ª rede neural – 1943)



Equação entrada-saída: $y_k = f(y^*_k) = f\left(\sum_{i=1}^{n} w_{ik} x_i + \theta_k\right)$

Função de ativação f é chamada de sinal: $f(y^*_k) = \begin{cases} 1, & \text{se } y^*_k \geq 0 \\ -1, & \text{se } y^*_k < 0 \end{cases}$

Alguns modelos de neurônios também incluem um termo externo ou bias (tendência).

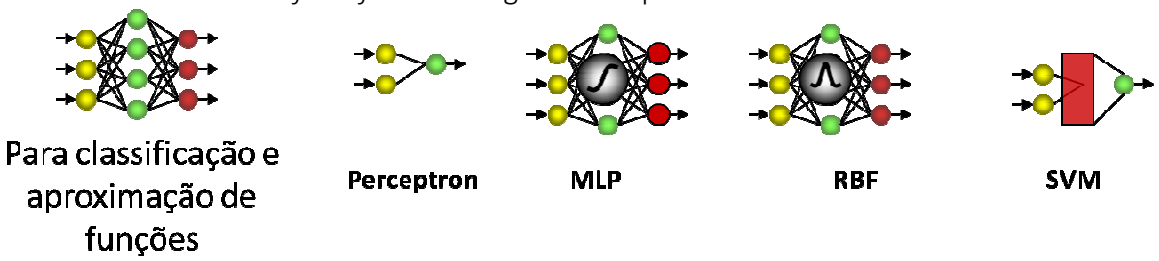
Entretanto, na maioria dos casos, o bias pode ser interpretado como um peso sináptico conectado a uma entrada constante.

Qual o papel do bias?

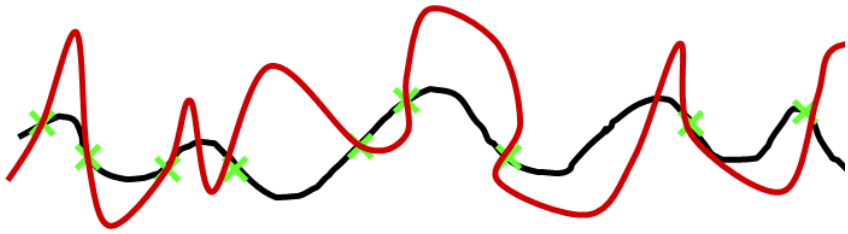
Notações para o bias: θ_k, b_k, w_0

REDES NEURAIS "PRÓ-ALIMENTADAS"

São chamadas de redes *feed-forward*. Alguns exemplos são:



O número de neurônios não garante uma boa convergência da rede. Exemplo:

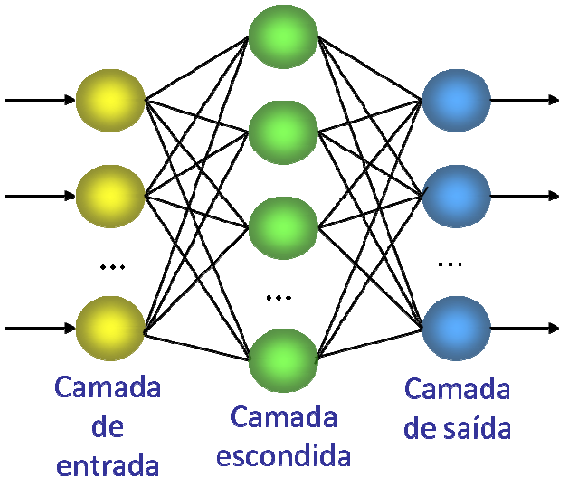


Nestas redes existem conjuntos de nós de entrada, de saída e os nós escondidos (intermediários entre os de entrada e saída).

Quando a RN está em operação, um valor de entrada será aplicado a cada nó de entrada

Cada nó passa seu dado valor para as conexões que saem dele.

Em cada conexão o valor é multiplicado por um peso associado.



Cada nó na camada seguinte recebe a soma dos valores produzidos pelas conexões que chegam até ele.

Cada nó realiza uma computação simples sobre esse valor: função sigmóide, limiar ou tanh.

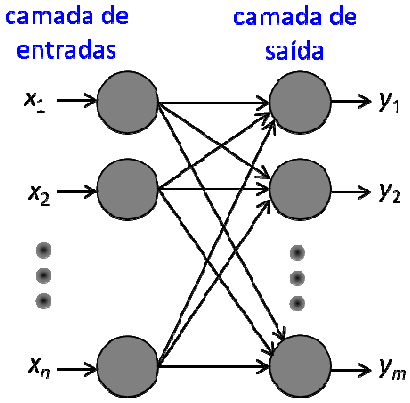
O processo é repetido com os resultados sendo passados através de camadas subsequentes de nós até que os nós de resultados sejam atingidos (camada de saída).

2.2. PERCEPTRON

É um tipo de rede neural usado para conjuntos de treinamento linearmente separáveis (Rosenblatt, 1950).

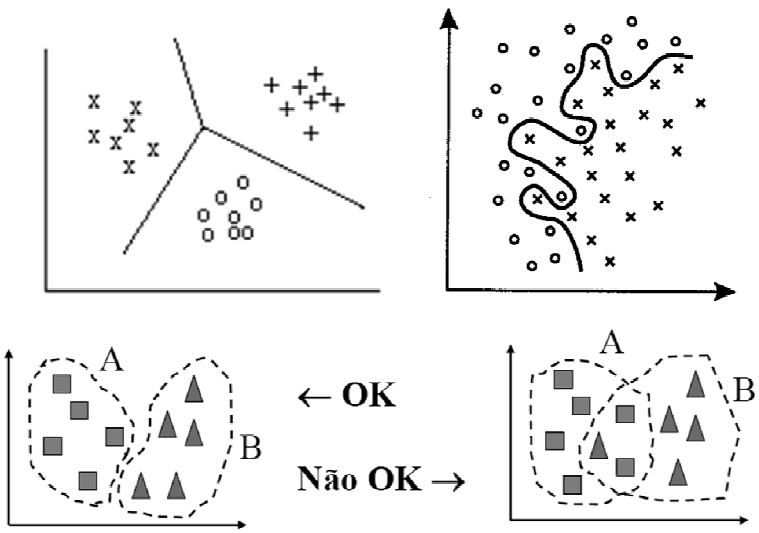
$$S = w_0 + \sum_{j=1}^{n} w_j x_j$$

Tem a inclusão de bias (tendência).

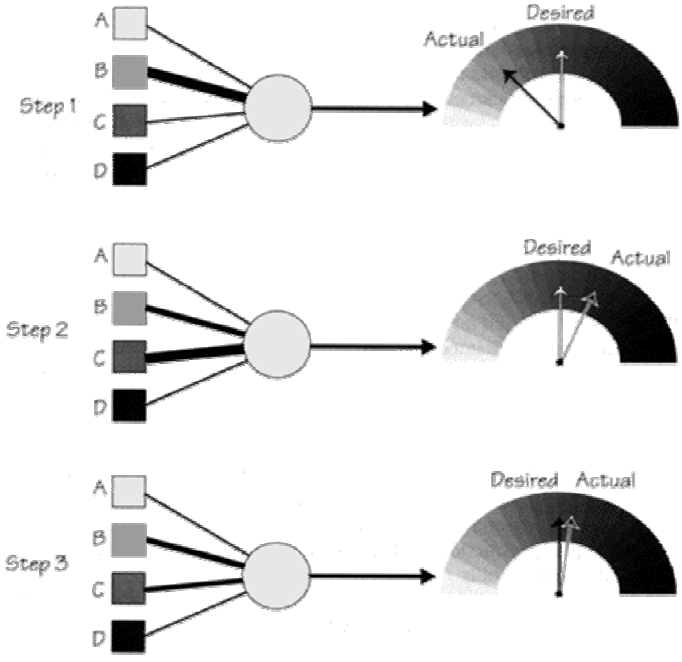


O algoritmo de aprendizagem do Perceptron procura um vetor ***w*** com projeção positiva (produto interno) com todos os exemplos positivos e projeção negativa com os exemplos negativos.

A aprendizagem do perceptron sempre tem sucesso em tempo finito para um conjunto de treinamento finito e separável de exemplos de treinamento.



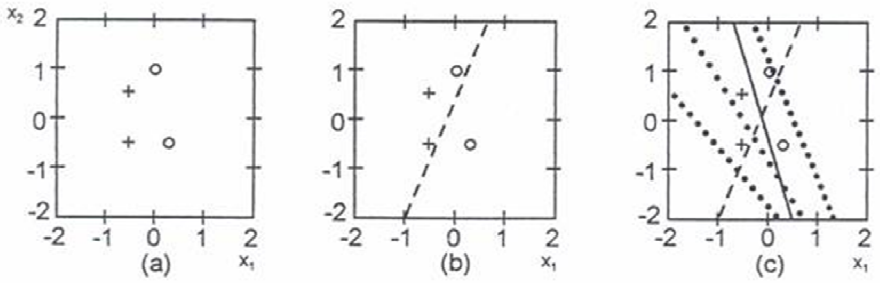
Um exemplo de funcionamento da aprendizagem de uma rede perceptron:



Algoritmo do Perceptron:

0. Inicializar os pesos, o bias e a taxa de aprendizado: $w = 0,\ \theta = 0,\ \alpha = 1$
   1. Enquanto o critério de parada não for satisfeito, execute os passos 2-6:
      2. Para cada par de dados de treinamento (x,d), execute os passos 3-5:
         3. Calcule $y^* = \theta + \Sigma_i x_i w_i$
         4. Se $y^* > \delta$, então $y = 1$
            Se $-\delta \leq y^* \leq \delta$, então $y = 0$
            Se $y^* < -\delta$, então $y = -1$
         5. Atualize os pesos e a tendência:
            Se $y \neq d$, faça
            $w_i^{atual} = w_i^{anterior} + \alpha d x_i$ e $\theta^{atual} = \theta^{anterior} + \alpha d$
            Caso contrário
            $w_i^{atual} = w_i^{anterior}$ e $\theta^{atual} = \theta^{anterior}$
6. Teste a condição de parada.

Funcionamento de um perceptron:



(a) As entradas (b) Separação inicial do espaço (c) Separação final [Demuth, 1994]

Os valores finais encontrados para os pesos e *bias* foram:

$w_1 = -2{,}1642;\ w_2 = -0{,}6922;\ \theta = -0{,}6433$

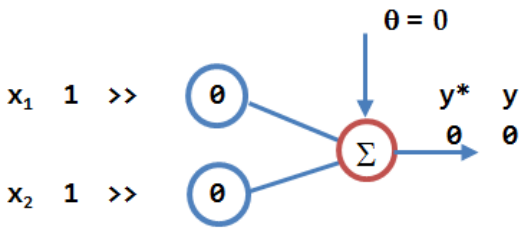
Perceptrons podem achar soluções diferentes se iniciarem o processo de aprendizado de diferentes condições iniciais. A rede anterior foi treinada novamente, e uma solução satisfatória, separando totalmente as entradas, foi encontrada, porém com diferentes valores:

$w_1 = -2{,}1642;\ w_2 = 0{,}0744;\ \theta = -0{,}6433$

Exercícios:

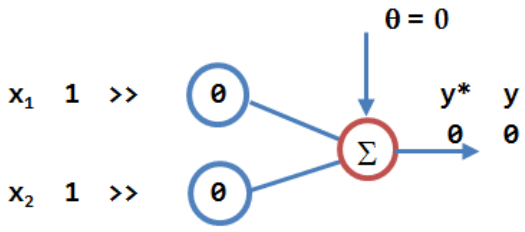
1. Classificação da função lógica "E" com entradas 0-1 e saídas bipolares:
   Valores iniciais: $\delta = 0{,}2 \quad \alpha = 1 \quad \theta = 0 \quad w = 0$

x_1	x_2	d
1	1	1
1	0	-1
0	1	-1
0	0	-1



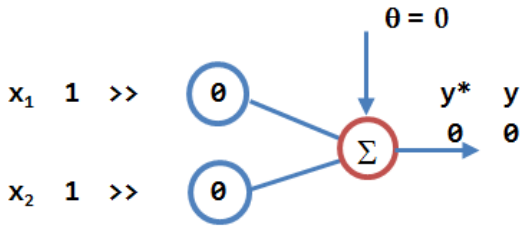
2. Classificação da função lógica “E” com entradas e saídas bipolares:

x_1	x_2	d
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1



3. Classificação da função lógica “OU” com entradas e saídas bipolares:

x_1	x_2	d
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

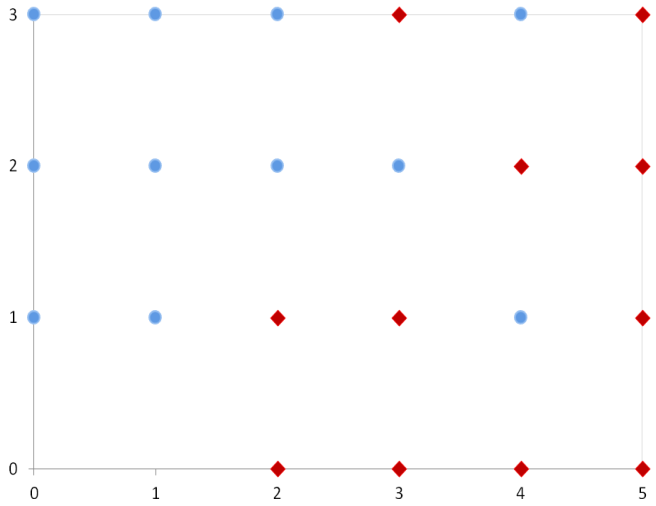


4. Classificação de padrões, conforme tabela abaixo:

x_1	x_2	classe
0,75	0,75	A
0,75	0,25	B
0,25	0,75	B
0,25	0,25	A

5. Classificação de padrões, conforme tabela abaixo, com o conjunto de treinamento dos 22 vetores. Após 3 iterações, apresente o conjunto de testes de 8 vetores dado abaixo:

n	x_1	x_2	d	n	x_1	x_2	d
1	0	1	1	12	2	0	-1
2	0	2	1	13	2	1	-1
3	1	1	1	14	3	0	-1
4	1	2	1	15	3	1	-1
5	1	3	1	16	3	3	-1
6	2	2	1	17	4	0	-1
7	2	3	1	18	4	2	-1
8	3	2	1	19	5	0	-1
9	4	1	1	20	5	1	-1
10	4	3	1	21	5	2	-1
11	0	3	1	22	5	3	-1



Conjunto de testes:

n	x_1	x_2	d	n	x_1	x_2	d
23	0	0	1	27	4	2,5	1
24	1	0	-1	28	1,5	1,5	1
25	4,5	0,5	-1	29	2	0,5	-1
26	3,5	1,5	1	30	2,5	2,5	1

Conclusões sobre o perceptron:

Para um conjunto finito de exemplos de treinamento ***d***:

- o algoritmo de aprendizagem do perceptron, em tempo finito:
  - produz um vetor peso que satisfaz todos os exemplos de treinamento (se e somente se ***d*** é separável); ou
  - abandona e reutilizará um vetor peso (se e somente se **d** é não-separável).
- quando ***d*** não é separável, então
  - não existe um vetor de pesos ***w*** que classifique corretamente todos os exemplos de treinamento em ***d*** utilizando o algoritmo do perceptron;
  - a alternativa é encontrar um vetor de pesos $\boldsymbol{w}^*$ que classifique tantos exemplos de treinamento quanto possível de ***d*** (conjunto ótimo de pesos).

Perceptron – Algoritmo do Bolso

O perceptron tem algoritmo de aprendizagem com comportamento pobre:

- mesmo após um grande número de iterações, não se tem garantia da qualidade dos pesos que são produzidos;
- o mesmo usa somente reforço negativo;
- ignora totalmente os exemplos que são corretamente classificados.

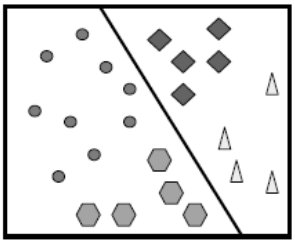
O algoritmo do bolso leva em conta as classificações corretas:

- mantém um conjunto de pesos em separado $\mathbf{W}^{\text{bolso}}$ "no bolso"
- armazena o número de iterações consecutivas que $\mathbf{W}^{\text{bolso}}$ classificou corretamente exemplos de treinamento escolhidos

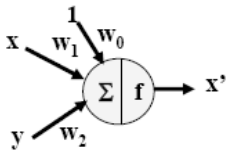
Algoritmo do Bolso:

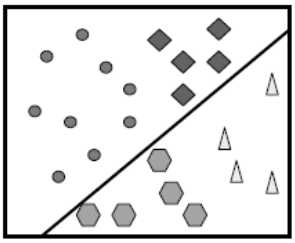
0. Inicializar os pesos, o bias e a taxa de aprendizado: $\boldsymbol{w} = \boldsymbol{0}, \boldsymbol{w}^{\text{bolso}} = \boldsymbol{0}, \boldsymbol{\theta} = \boldsymbol{0}, \boldsymbol{\alpha} = \boldsymbol{1}$.
   1. Enquanto o critério de parada não for satisfeito, execute os passos 2-7:
      2. Para cada par de dados de treinamento (x, d), execute os passos 3-5:
         3. Calcule $\boldsymbol{y}^* = \boldsymbol{\theta} + \Sigma_i \boldsymbol{x}_i \boldsymbol{w}_i$
         4. Se $\boldsymbol{y}^* > \boldsymbol{\delta}$, então $\boldsymbol{y} = \boldsymbol{1}$
            Se $-\boldsymbol{\delta} \leq \boldsymbol{y}^* \leq \boldsymbol{\delta}$, então $\boldsymbol{y} = \boldsymbol{0}$
            Se $\boldsymbol{y}^* < -\boldsymbol{\delta}$, então $\boldsymbol{y} = -\boldsymbol{1}$
         5. Atualize os pesos e a tendência:
            Se $\boldsymbol{y} \neq \boldsymbol{d}$, faça
            $\boldsymbol{w}_i^{\text{atual}} = \boldsymbol{w}_i^{\text{anterior}} + \boldsymbol{\alpha d x}_i$ e $\boldsymbol{\theta}^{\text{atual}} = \boldsymbol{\theta}^{\text{anterior}} + \boldsymbol{\alpha d}$
            Caso contrário
            $\boldsymbol{w}_i^{\text{atual}} = \boldsymbol{w}_i^{\text{anterior}}$ e $\boldsymbol{\theta}^{\text{atual}} = \boldsymbol{\theta}^{\text{anterior}}$
         6. Se w classifica corretamente mais exemplos do que w^{bolso}:
            $\boldsymbol{w}^{\text{bolso}} = \boldsymbol{w}$; grave o número de exemplos corretos
7. Teste a condição de parada.

Separação em 4 classes com Perceptron:

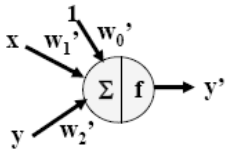


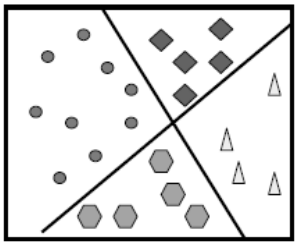
Neurônio 1: separa em classes C1 e C2

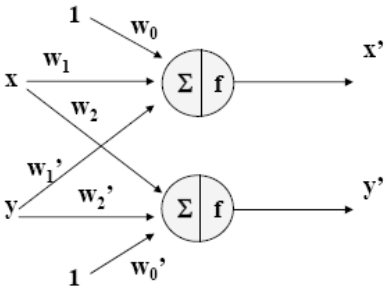




Neurônio 2: separa em classes C1' e C2'



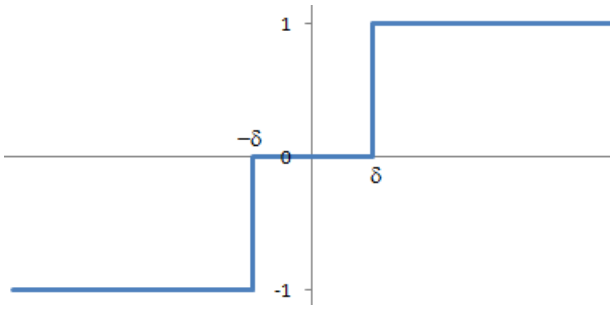




FUNÇÕES DE ATIVAÇÃO

Limiar: saídas em [0,1] ou [-1,1]

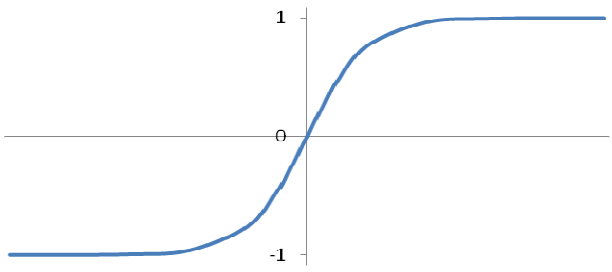
$$f(x_k)=\begin{cases}1, & \text{se } x_k>\delta\\ 0, & \text{se } -\delta\le x_k\le\delta\\ -1, & \text{se } x_k<-\delta\end{cases}$$

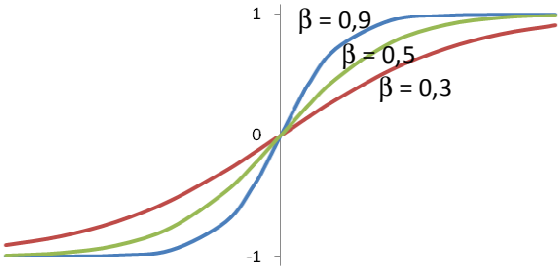


Tangente hiperbólica: saídas em [-1,1]

$$f(x_k)=tanh(\beta x_k)=\frac{e^{\beta x_k}-e^{-\beta x_k}}{e^{\beta x_k}+e^{-\beta x_k}}=\frac{1-e^{-2\beta x_k}}{1+e^{-2\beta x_k}}$$

$$\frac{\partial f}{\partial x_k}=\beta\left(1-f(x_k)^2\right)>0$$

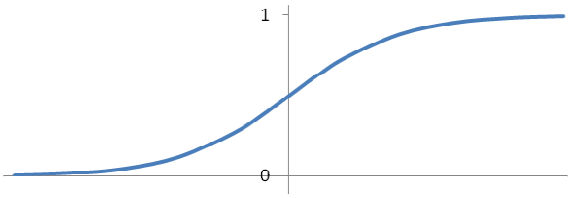


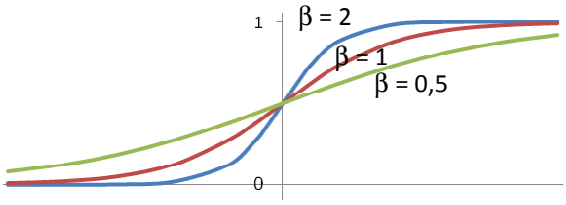


Sigmóide: saídas em [0,1]

$$f(x_k) = \frac{1}{1+e^{-\beta x_k}}$$

$$\frac{\partial f}{\partial x_k} = \frac{\beta e^{-\beta x_k}}{\left(1+e^{-\beta x_k}\right)^2}$$

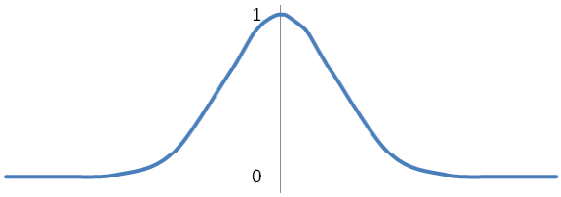


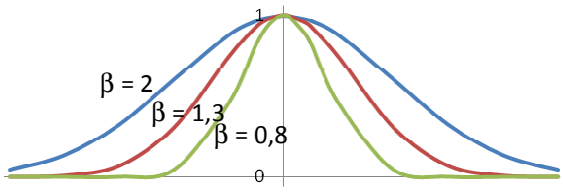


Gaussiana: saídas em [0,1]

$$f(x_k) = e^{-\frac{(x_k-c)^2}{2\beta^2}}$$

$$\frac{\partial f}{\partial x_k} = -(x_k - c)\frac{e^{-\frac{x^2-2cx+c^2}{2\beta^2}}}{\beta^2}$$ onde c é o centro.





As curvas sigmoide, tangente hiperbólica e gaussiana são suaves, ou seja, continuamente diferenciáveis.

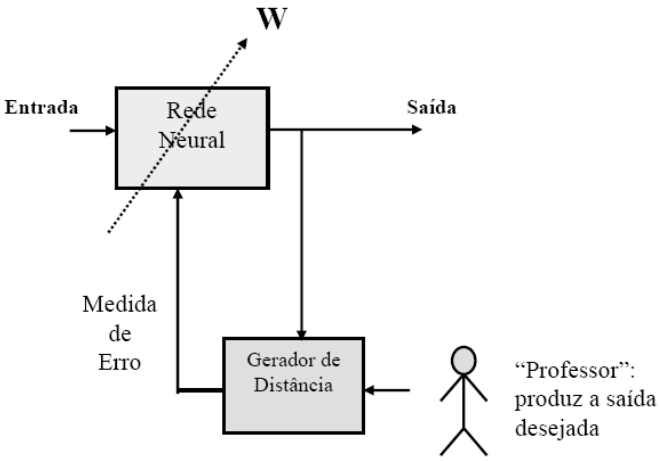
A função limiar também é chamada de degrau.

FORMAS DE APRENDIZAGEM

A característica principal das RNA é a aprendizagem através da experiência. Esta aprendizagem pode ser feita das seguintes formas:

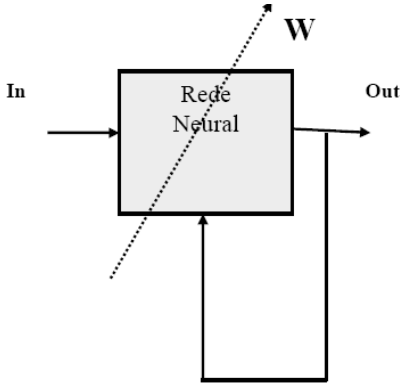
Supervisionada

- apresenta-se à rede um conjunto de treinamento com a saída desejada de cada padrão;
- a distância entre a saída desejada e a saída da rede é usada como erro e serve para corrigir os pesos da rede.



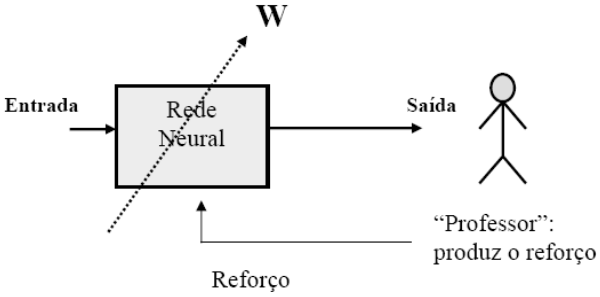
Não-supervisionada

- a rede atualiza os parâmetros e pesos sem o controle de saídas desejadas para os padrões de entrada;
- a rede descobre sozinha as propriedades das entradas, e constrói a saída baseando-se nestas "descobertas".

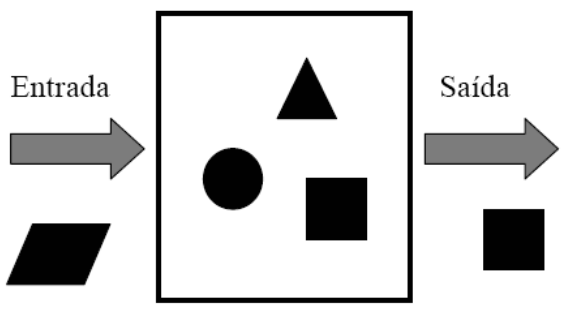


Com reforço

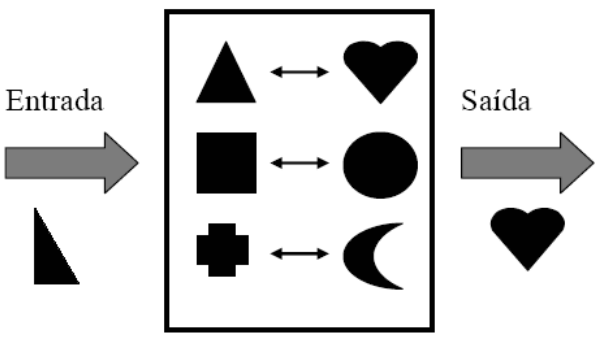
- cada entrada possui uma indicação (reforço) da saída desejada;
- o reforço constitui de penalidade para respostas "ruins" e estímulo para as respostas "boas";
- os pesos da rede ajustam-se para melhorar os estímulos dos reforços;
- não existe uma medida para as respostas desejadas.



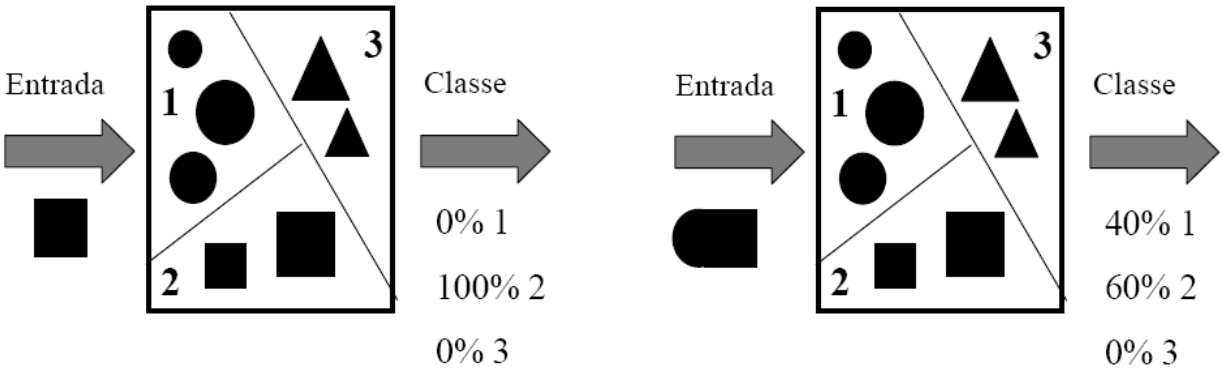
Autoassociação:



Heteroassociação:



Classificação:



Memória associativa:

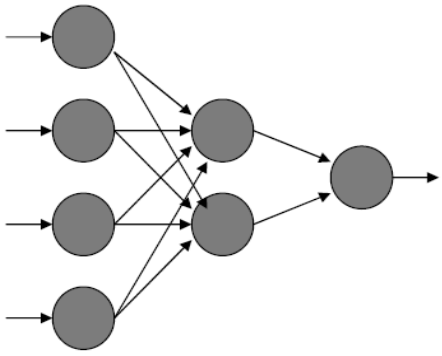
Um determinado conceito A "lembra" o conceito B.
Por exemplo:

Entrada		Saída
Hipotenusa	lembra ▸	Geometria
Derivada	lembra ▸	Cálculo
Seno	lembra ▸	Trigonometria

ARQUITETURAS DAS RNAs

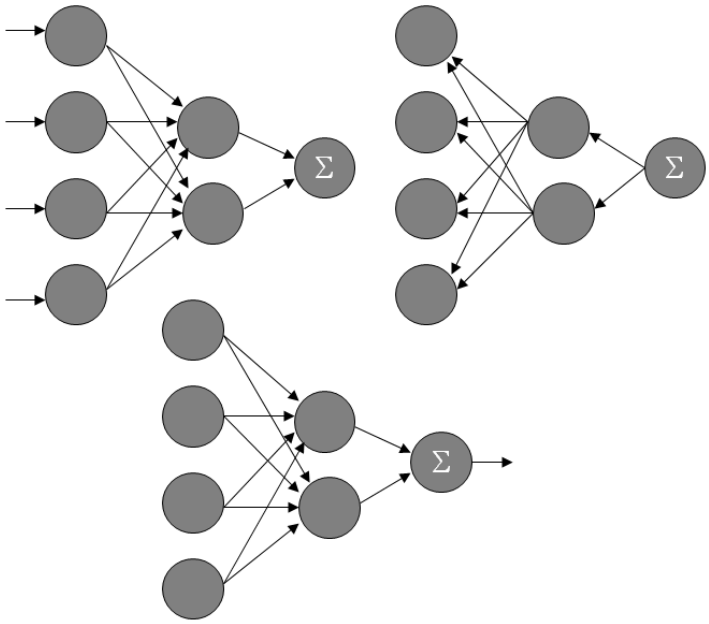
Sem realimentação:

Os sinais dos neurônios percorrem a rede em uma única direção: da entrada para a saída.
Os neurônios de uma mesma camada não são conectados.

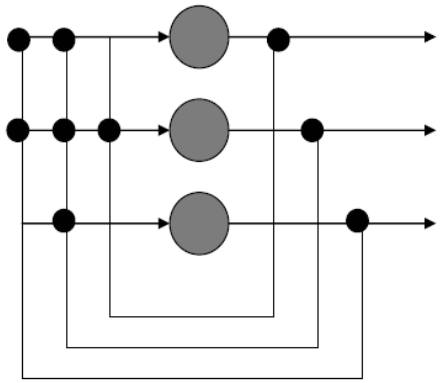


Com realimentação:

Os sinais dos neurônios percorrem a rede nas duas direções.



Recorrentes:

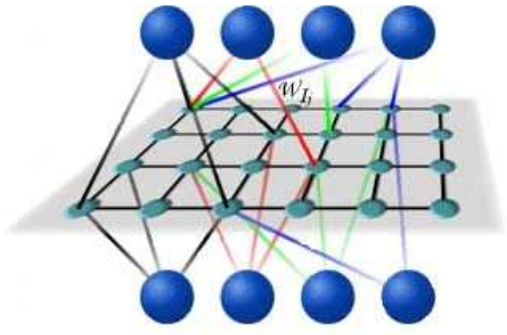


Os sinais dos neurônios percorrem a rede nas duas direções.

Os sinais de alguns neurônios alimentam neurônios da mesma camada, ou das camadas anteriores, além de camadas posteriores.

Em alguns casos os próprios neurônios são conectados a si mesmos.

Competitivas:

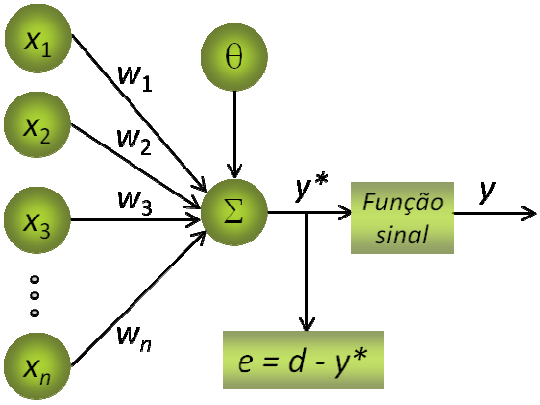


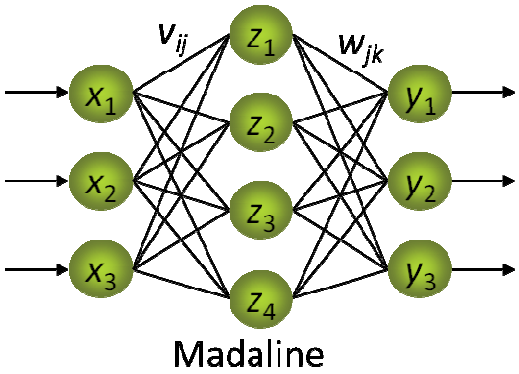
Os dados de entrada são designados a um mapa de neurônios, e escolhe-se o vencedor (que possui peso mais próximo da entrada).

Ocorrem os ajustes dos pesos dos neurônios, com o objetivo de minimizar as distâncias dos dados de entrada aos respectivos neurônios vencedores.

2.3. REDE ADALINE E A REGRA DELTA

A rede Adaline (**Adaptive Linear Neuron**) foi idealizada por Widrow e Hoff (1960) e também tem um funcionamento bem simples como o Perceptron. Pode ter camadas escondidas, assim como o MLP.





A saída é calculada da mesma forma que no Perceptron, com a função limiar ou limiar bipolar.

O erro desta rede é calculado como:

$$e = d - y^*$$

e pode ser usado para corrigir os pesos w. Este processo de ajuste de pesos é chamado de Regra Delta ou método do Gradiente descendente.

REGRA DELTA

A ideia básica é minimizar a distância entre as saídas desejadas e as respostas **y*** da RNA. Logo, existirá um vetor ótimo de pesos **w***, tal que:

$$E(\mathbf{w}^*) \leq E(\mathbf{w}),\ \forall \mathbf{w} \in \Re^{n+1}.$$

Considerando uma rede Adaline simples de uma saída y, a função de erro quadrático de uma saída k é definida como:

$$E_k = \frac{1}{2}(d_k - y^*)^2,$$

e para todas os **p** padrões de entrada teremos:

$$E(w) = \frac{1}{2}\sum_{k=1}^{p}(d_k - y^*)^2 .$$

Como $y^* = \sum_{i=1}^{n}(w_i x_i - \theta)$, temos que:

$$E(w) = \frac{1}{2}\sum_{k=1}^{p}\left(d_k - \sum_{i=1}^{n}(w_i x_i + \theta)\right)^2$$

$$\therefore E(w) = \frac{1}{2}\sum_{k=1}^{p}\left(d_k - \left(w^T x + \theta\right)\right)^2 .$$

Aplicando-se o gradiente em relação ao vetor **w** para encontrar o erro mínimo para o erro quadrático médio obtemos:

$$\nabla E(w) = \frac{\partial E(w)}{\partial w} = \sum_{k=1}^{p}\left(d_k - \left(w^T x + \theta\right)\right).(-x_k).$$

$$\therefore \nabla E(w) = -\sum_{k=1}^{p}(d_k - y^*).x_k .$$

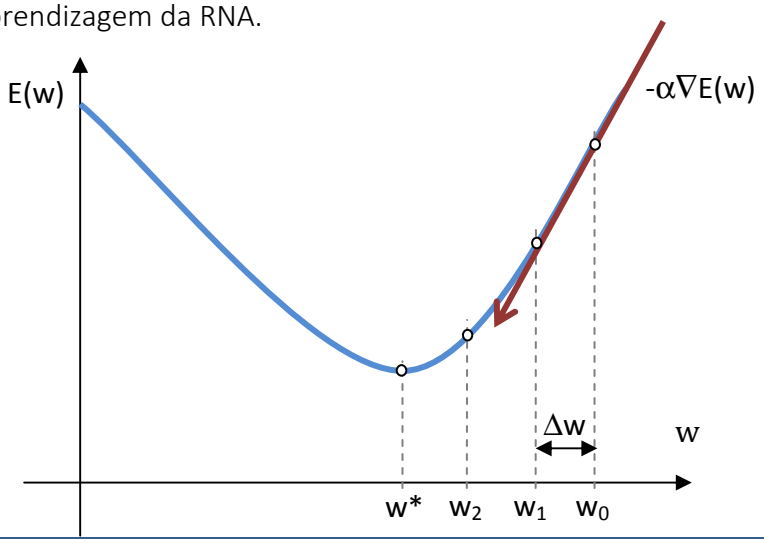
A adaptação da RNA deve ser feita na direção contrária à do gradiente, pois o objetivo é de minimizar o erro médio quadrático, ou seja,

$$\Delta w = -\alpha.\nabla E(w) = \alpha\sum_{k=1}^{p}(d_k - y^*).x_k .$$

Logo, a atualização dos pesos de uma RNA Adaline é feita através da expressão abaixo:

$$w^{atual} = w^{anterior} + \alpha\sum_{k=1}^{p}(d_k - y^*).x_k ,$$

onde α é a taxa de aprendizagem da RNA.



Algoritmo da rede Adaline:

0. Inicializar os pesos ($w = rnd$), a tendência ($\theta = 0$) e a taxa de aprendizagem $0 < \alpha < 1$ (convergência fica muito lenta quando a taxa é muito próxima de zero; e a convergência não é garantida para valores muito próximos de 1).
   1. Enquanto o critério de parada não for satisfeito, execute os passos 2-5:
      2. Para cada par de dados para treinamento (x,d), execute os passos 3-4:
         3. Faça $y^* = \theta + \Sigma_i x_i w_i$
         4. Atualize os pesos e a tendência:

            $w_i^{atual} = w_i^{anterior} + \alpha(d - y^*)x_i$

            $\theta^{atual} = \theta^{anterior} + \alpha(d - y^*)$

            se $y^* \geq 0$, $y = 1$; caso contrário, $y = 0$ (ou y = -1 para bipolar)
         5. Teste a condição de parada.
   6. Se a maior alteração de pesos não ultrapassa um limite mínimo de tolerância, pare; caso contrário, continue.

Exercícios:

1. Classificação da função lógica "OU" com entradas e saídas bipolares:
   Valores iniciais: $\alpha = 0,5$ $w_1 = 0,3$ $w_2 = 0,5$

x_1	x_2	d
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

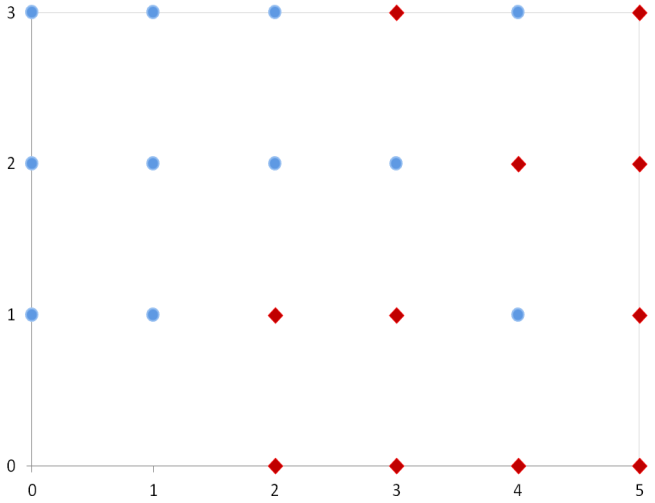
2. Classificação da função lógica "E" com entradas e saídas bipolares:
   Valores iniciais: $\alpha = 0,4$ $w_1 = 0,9$ $w_2 = -0,7$

x_1	x_2	d
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

3. Utilizando uma rede do tipo Adaline, calcule uma matriz de pesos que seja capaz de fazer a classificação dos vetores dados abaixo (parâmetros iniciais: $\alpha = 0,2$ e $w=0$): $p_1 = (1,1,1,1)$, $d_1 = 0,9$; $p_2 = (-1,-1,-1,-1)$, $d_2 = 0,8$; $p_3 = (1,1,-1,-1)$, $d_3 = -0,7$; $p_4 = (1,-1,-1,1)$, $d_4 = -0,8$; $p_5 = (-1,-1, 1,1)$, $d_5 = -0,7$.

4. Classificação de padrões, conforme tabela abaixo, com o conjunto de treinamento dos 22 vetores. Após 3 iterações, apresente o conjunto de testes de 8 vetores dado abaixo:

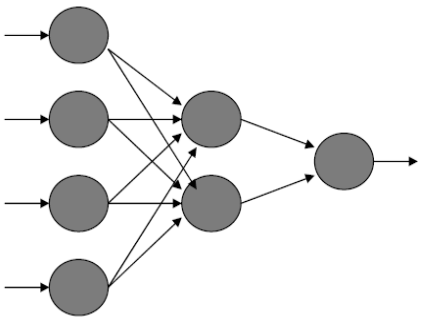
n	x_1	x_2	d	n	x_1	x_2	d
1	0	1	1	12	2	0	-1
2	0	2	1	13	2	1	-1
3	1	1	1	14	3	0	-1
4	1	2	1	15	3	1	-1
5	1	3	1	16	3	3	-1
6	2	2	1	17	4	0	-1
7	2	3	1	18	4	2	-1
8	3	2	1	19	5	0	-1
9	4	1	1	20	5	1	-1
10	4	3	1	21	5	2	-1
11	0	3	1	22	5	3	-1



Conjunto de testes:

n	x_1	x_2	d	n	x_1	x_2	d
23	0	0	1	27	4	2,5	1
24	1	0	-1	28	1,5	1,5	1
25	4,5	0,5	-1	29	2	0,5	-1
26	3,5	1,5	1	30	2,5	2,5	1

2.4. PERCEPTRON MULTI CAMADAS (Multi layer perceptron – MLP)



Consiste de um conjunto de Perceptrons arranjados em diversas camadas.

Possuem pelo menos uma camada escondida.

Solucionam problemas não-lineares de classificação.

Uma rede MLP com pelo menos uma camada escondida supera as limitações de classificação do perceptron [Minsky e Papert, 1969].

O ajuste dos pesos foi proposto por Rumelhart, Hinton, Williams, Parker, Le Cun e Werbos .

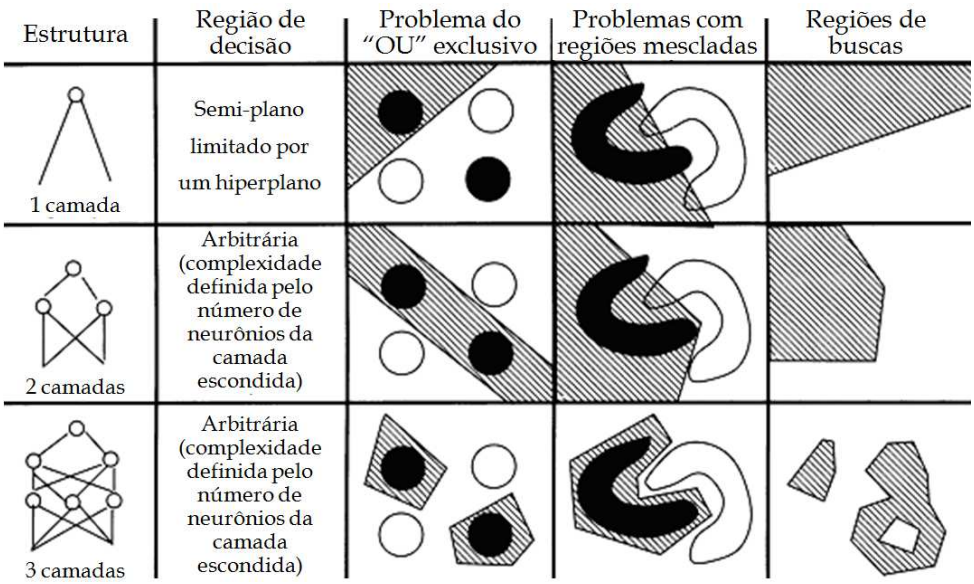
Cada camada recebe dados da camada imediatamente inferior e envia para a camada seguinte.

Não existem conexões entre elementos da mesma camada.

Uma MLP pode ter qualquer número de camadas e de neurônios.

Uma MLP com uma camada escondida é suficiente para aproximar com precisão arbitrária qualquer função com um número finito de descontinuidades, desde que a função de ativação dos neurônios escondidos seja não-linear.

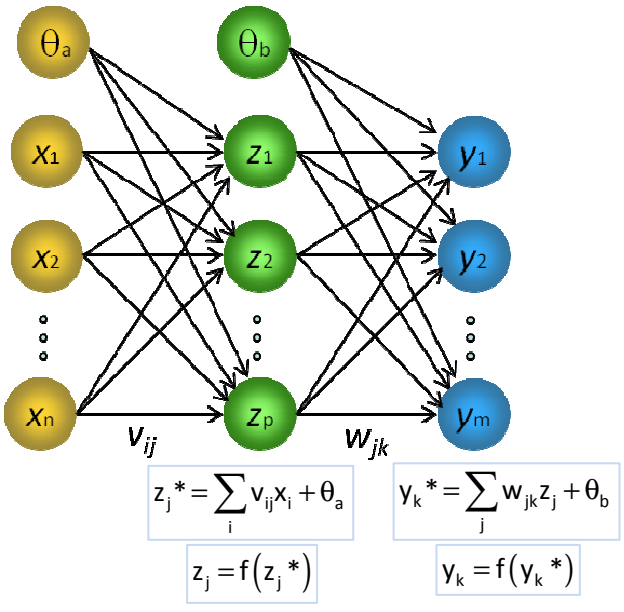
Superfícies de separação



REGRA DELTA GENERALIZADA

Usando o mesmo raciocínio da Regra Delta para a Rede Adaline, podemos fazer a denominada Regra Delta Generalizada, válida para rede MLP.

Considere o caso de uma MLP com uma camada escondida, conforme figura abaixo.



A ativação é a função diferenciável da saída da rede:

$$y_k^* = \sum_j w_{jk} z_j + \theta_b \text{ , onde } y_k = f(y_k^*).$$

A medida do erro E para um determinado padrão é:

$$E_k = \frac{1}{2}(d_k - y_k)^2.$$

Para obter o decréscimo mais rápido do erro, devemos ter o erro na direção $-\frac{\partial E_k}{\partial w_{jk}}$.

Usando a Regra da Cadeia, temos:

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k *}\frac{\partial y_k *}{\partial w_{jk}}$$

Mas

$$\frac{\partial y_k *}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}\left(\sum_j z_j w_{jk} + \theta_b\right) = z_j \Rightarrow \frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k *} z_j$$

Aplicando novamente a Regra da Cadeia, obtemos:

$$\frac{\partial E_k}{\partial y_k *} = \frac{\partial E_k}{\partial y_k}\frac{\partial y_k}{\partial y_k *} \Rightarrow \frac{\partial y_k}{\partial y_k *} = f'(y_k *) \Rightarrow \frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k} f'(y_k *) z_j .$$

Como $\frac{\partial E_k}{\partial y_k} = -(d_k - y_k)$ podemos concluir que o erro descresce mais rapidamente quando o ajuste dos pesos é feito da forma:

$$\Delta w_{jk} = \alpha (d_k - y_k) f'(y_k *) z_j \qquad (1)$$

No caso da utilização da função de ativação sigmoidal, temos:

$$f(y_k *) = \frac{1}{1 + e^{-y_k *}} \text{ e } f'(y_k *) = \frac{e^{-y_k *}}{\left(1 + e^{-y_k *}\right)^2} \qquad (2)$$

Substituindo-se (2) em (1) obtemos:

$$\frac{\partial E_k}{\partial w_{jk}} = -(d_k - y_k)\frac{e^{-y_k *}}{\left(1 + e^{-y_k *}\right)^2} z_j = -\frac{1}{1 + e^{-y_k *}}\frac{e^{-y_k *}}{1 + e^{-y_k *}}(d_k - y_k) z_j$$

Como $\frac{1}{1 + e^{-y_k *}} = y_k$ e $\frac{e^{-y_k *}}{1 + e^{-y_k *}} = \frac{1 + e^{-y_k *}}{1 + e^{-y_k *}} - \frac{1}{1 + e^{-y_k *}} = 1 - \frac{1}{1 + e^{-y_k *}}$ temos:

$$\frac{\partial E_k}{\partial w_{jk}} = -y_k (1 - y_k)(d_k - y_k) z_j ,$$

$$\therefore \Delta w_{jk} = -y_k (1 - y_k)(d_k - y_k) z_j .$$

Exercício:

Qual parâmetro foi usado na função sigmoidal da dedução acima? Calcule a atualização de Δw_{jk} quando a função de ativação for tangente hiperbólica.

Agora vamos calcular a atualização para a camada escondida. A ativação é a função diferenciável da camada da rede:

$$z_j * = \sum_i v_{ij} x_i + \theta_a \text{ , onde } z_j = f(z_j *)$$

A medida do erro E para um determinado padrão na camada escondida é:

$$\frac{\partial E_j}{\partial v_{ij}} = \frac{\partial E_j}{\partial y}\frac{\partial y}{\partial v_{ij}}. \tag{3}$$

Considere $\frac{\partial E_j}{\partial y} = -\sum_k (d_k - y_k)$. Usando a Regra da Cadeia, temos:

$$\frac{\partial y}{\partial v_{ij}} = \frac{\partial y}{\partial y^*}\frac{\partial y^*}{\partial v_{ij}}. \tag{4}$$

Mas

$$\frac{\partial y^*}{\partial v_{ij}} = \frac{\partial y^*}{\partial z_j}\frac{\partial z_j}{\partial v_{ij}} = \sum_k w_{jk}\frac{\partial z_j}{\partial v_{ij}}. \tag{5}$$

Temos também que:

$$\frac{\partial z_j}{\partial v_{ij}} = \frac{\partial z_j}{\partial z_j^*}\frac{\partial z_j^*}{\partial v_{ij}} = f'(z_j^*)x_i$$

Substituindo-se (5) em (4), e (4) em (3) obtemos:

$$\frac{\partial E_j}{\partial v_{ij}} = -\sum_k \left[(d_k - y_k) f'(y_k^*) w_{jk}\right] f'(z_j^*) x_i,$$

$$\therefore \boxed{\Delta v_{ij} = \alpha \sum_k (d_k - y_k) f'(y^*_k) w_{jk} f'(z^*_j) x_i}. \tag{6}$$

No caso da utilização da função de ativação sigmoidal, temos:

$$f'(y_k^*) = \frac{e^{-y_k^*}}{\left(1+e^{-y_k^*}\right)^2} \text{ e } f'(z_j^*) = \frac{e^{-z_j^*}}{\left(1+e^{-z_j^*}\right)^2}. \tag{7}$$

Substituindo (7) em (6) obtemos:

$$\frac{\partial E_j}{\partial v_{ij}} = -\sum_k \left[(d_k - y_k)\frac{e^{-y_k^*}}{\left(1+e^{-y_k^*}\right)^2} w_{jk}\right] \frac{e^{-z_j^*}}{\left(1+e^{-z_j^*}\right)^2} x_i,$$

ou seja,

$$\frac{\partial E_j}{\partial v_{ij}} = -\sum_k \left[(d_k - y_k) y_k (1-y_k) w_{jk}\right] z_j (1-z_j) x_i.$$

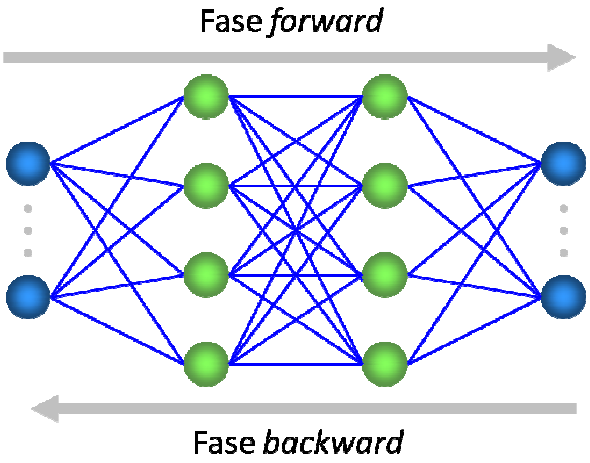
$$\therefore \boxed{\Delta v_{ij} = \alpha \sum_k (d_k - y_k) y_k (1-y_k) w_{jk} z_j (1-z_j) x_i}.$$

Exercício:

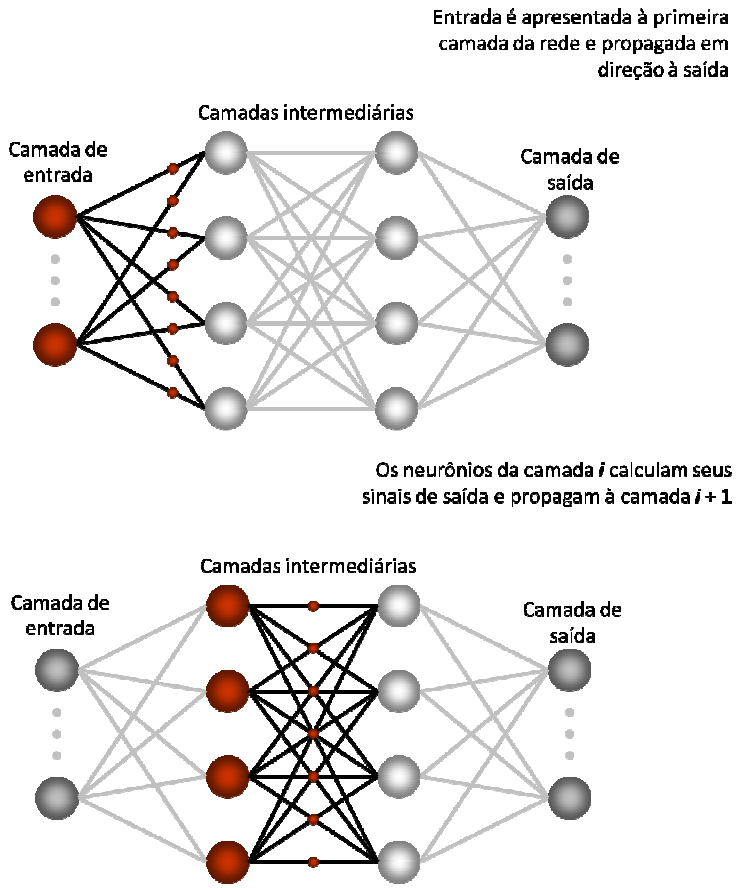
Calcule a atualização de Δv_{ij} quando a função de ativação for a tangente hiperbólica.

ALGORITMO BACKPROPAGATION

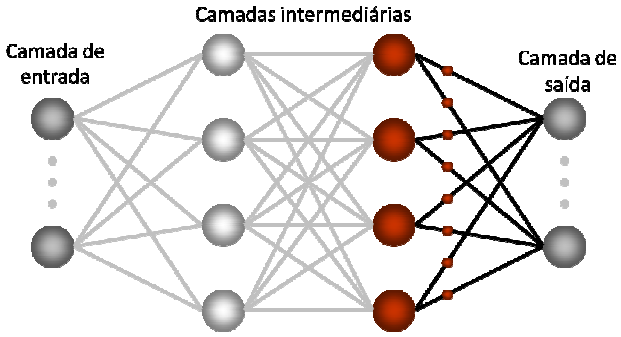
Neste algoritmo, o treinamento da RNA é feito em duas fases:



Fase forward:

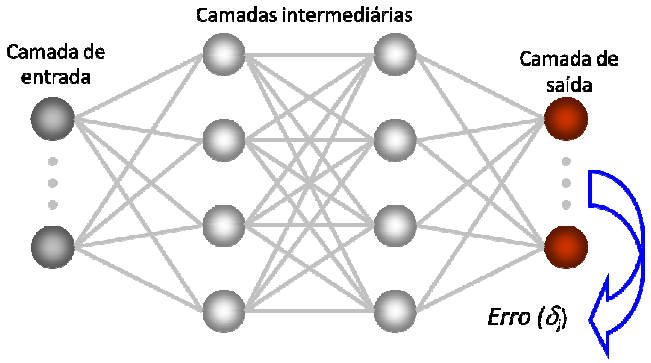


A última camada oculta calcula seus sinais de saída e os envia à camada de saída

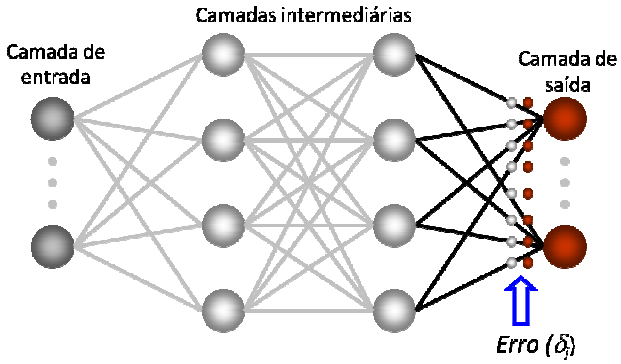


Fase backward:

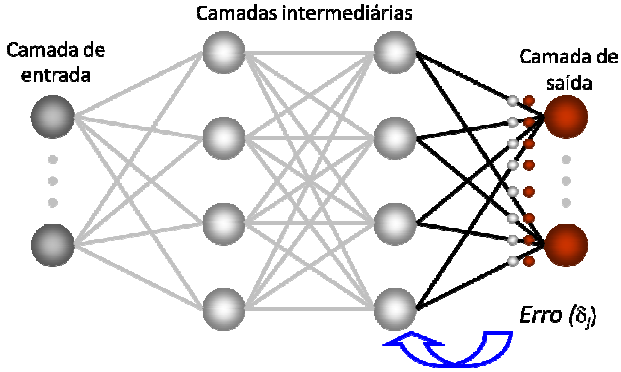
A camada de saída calcula o erro da rede: δ_j



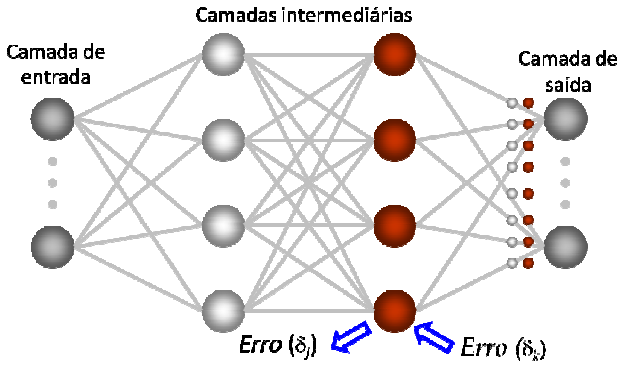
Calcula o termo de correção dos pesos $\Delta w_{ji} = \alpha \delta_j x_i$



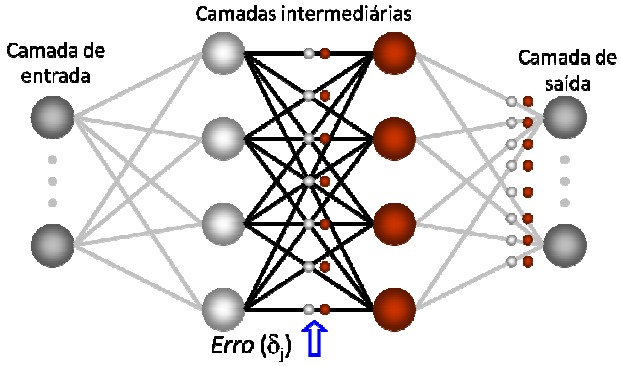
Envia o erro para a última camada oculta



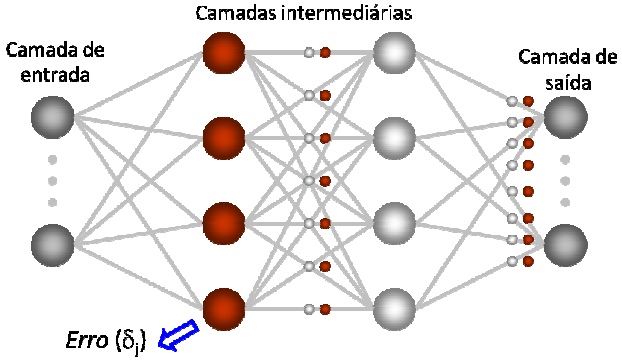
A camada oculta calcula o seu erro $\delta_j = f'(u_j).\sum \delta_k w_{jk}$



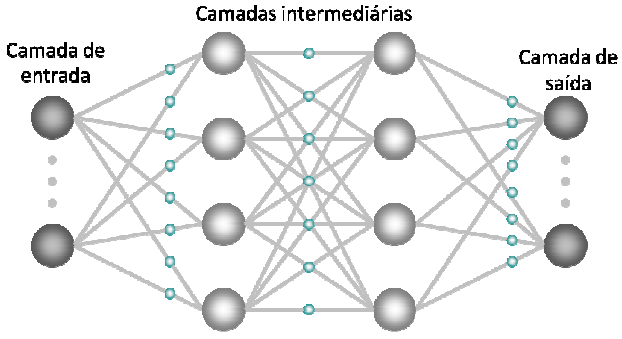
Calcula o termo de correção dos pesos $\Delta w_{ij} = \alpha \delta_j x_i$



A camada oculta calcula o seu erro $\delta_j = f'(u_j).\sum \delta_k w_{jk}$

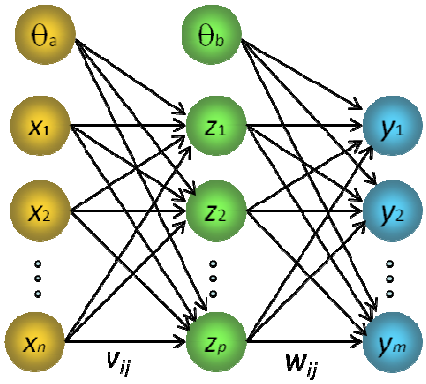


Cada unidade atualiza seus pesos $w_{ij}(\text{novo}) = w_{ij}(\text{atual}) + \Delta w_{jk}$



Repete-se o processo enquanto a rede não aprender o padrão de entrada

Resumindo:



$z^* = v^T x + \theta_a$, $z = f(z^*)$

$y^* = w^T z + \theta_b$, $y = f(y^*)$

erro do vetor z: $\delta^* = \delta w$, $\delta = \delta^* f'(z^*)$, $\Delta v = \alpha \delta x$, $\Delta \theta_a = \alpha \delta$

erro do vetor y: $\delta = (d - y) f'(y^*)$, $\Delta w = \alpha \delta z$, $\Delta \theta_b = \alpha \delta$

Algoritmo 1:

1 camada escondida, funções de ativação sigmoidais, *m* saídas

0. Inicialize os pesos das conexões e do bias com valores aleatórios; inicialize a taxa de aprendizagem α. Para cada padrão de entrada, execute os passos de 1 a 3:
   1. Calcule as entradas na camada escondida, e a saída da rede:
      $z_j^* = \Sigma_i v_{ij} x_i + \theta a_j$ ➡ $z_j = 1/(1 + e^{(-z_j^*)})$, onde $j = 1, \ldots, p$, $i = 1, \ldots, n$
      $y_k^* = \Sigma_j w_{jk} z_j + \theta b_k$ ➡ $y_k = 1/(1 + e^{(-y_k^*)})$, onde $k = 1, 2, \ldots, m$
   2. Calcule as correções das conexões da camada de saída:
      $\Delta w_{jk} = \alpha y_k (1 - y_k)(d_k - y_k) z_j$ ➡ $w_{jk} = w_{jk} + \Delta w_{jk}$
      $\Delta \theta b_k = \alpha y_k (1 - y_k)(d_k - y_k)$ ➡ $\theta b_k = \theta b_k + \Delta \theta b_k$
   3. Calcule as correções das conexões da camada escondida:
      $\Delta v_{ij} = \alpha \Sigma_k [(d_k - y_k) y_k (1 - y_k) w_{jk}] z_j (1 - z_j) x_i$ ➡ $v_{ij} = v_{ij} + \Delta v_{ij}$
      $\Delta \theta a_j = \alpha \Sigma_k [(d_k - y_k) y_k (1 - y_k) w_{jk}] z_j (1 - z_j)$ ➡ $\theta a_j = \theta a_j + \Delta \theta a_j$
   4. Atualize a taxa de aprendizagem, verifique os erros para todos os padrões de entrada, e teste o critério de parada.

Algoritmo 2:

1 camada escondida, funções de ativação sigmoidais, 1 saída

0. Inicialize os pesos das conexões e dos bias com valores aleatórios; inicialize a taxa de aprendizagem α. Para cada padrão de entrada, execute os passos de 1 a 3:
   1. Calcule as entradas na camada escondida, e a saída da rede:
      $z_j^* = \Sigma_i v_{ij} x_i + \theta a_j$ ➡ $z_j = 1/(1 + e^{(-z_j^*)})$, onde $j = 1, \ldots, p$, $i = 1, \ldots, n$
      $y^* = \Sigma_j w_j z_j + \theta b$ ➡ $y = 1/(1 + e^{(-y^*)})$
   2. Calcule as correções das conexões da camada de saída:
      $\Delta w_j = \alpha y (1 - y)(d - y) z_j$ ➡ $w_j = w_j + \Delta w_j$
      $\Delta \theta b = \alpha y (1 - y)(d - y)$ ➡ $\theta b = \theta b + \Delta \theta b$

3. Calcule as correções das conexões da camada escondida:

$\Delta v_{ij} = \alpha(d - y)y(1 - y)w_j z_j(1 - z_j)x_i$ ➡ $v_{ij} = v_{ij} + \Delta v_{ij}$

$\Delta\theta a_j = \alpha(d - y)y(1 - y)w_j z_j(1 - z_j)$ ➡ $\theta a_j = \theta a_j + \Delta\theta a_j$

4. Atualize a taxa de aprendizagem, verifique os erros para todos os padrões de entrada, e teste o critério de parada.

Algoritmo 3:

1 camada escondida, funções de ativação tanh, *m* saídas

0. Inicialize os pesos das conexões e dos bias com valores aleatórios; inicialize a taxa de aprendizagem α. Para cada padrão de entrada, execute os passos de 1 a 3:

1. Calcule as entradas na camada escondida, e a saída da rede:

$z_j^* = \Sigma_i v_{ij}x_i + \theta a_j$ ➡ $z_j = tanh(z_j^*)$, onde $j = 1, \ldots, p$, $i = 1, \ldots, n$

$y_k^* = \Sigma_j w_{jk}z_j + \theta b_k$ ➡ $y_k = tanh(y_k^*)$, onde $k = 1, 2, \ldots, m$

2. Calcule as correções das conexões da camada de saída:

$\Delta w_{jk} = \alpha(1 - y_k^2)(d_k - y_k)z_j$ ➡ $w_{jk} = w_{jk} + \Delta w_{jk}$

$\Delta\theta b_k = \alpha(1 - y_k^2)(d_k - y_j)$ ➡ $\theta b_k = \theta b_k + \Delta\theta b_k$

3. Calcule as correções das conexões da camada escondida:

$\Delta v_{ij} = \alpha\Sigma_k[(d_k - y_k)(1 - y_k^2)w_{jk}](1 - z_j^2)x_i$ ➡ $v_{ij} = v_{ij} + \Delta v_{ij}$

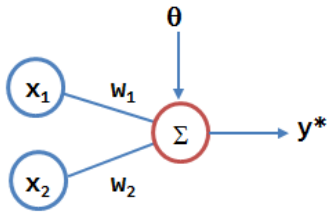
$\Delta\theta a_j = \alpha\Sigma_k[(d_k - y_k)(1 - y_j^2)w_{jk}](1 - z_j^2)$ ➡ $\theta a_j = \theta a_j + \Delta\theta a_j$

4. Atualize a taxa de aprendizagem, verifique os erros para todos os padrões de entrada, e teste o critério de parada.

Exercícios:

1. Utilizando a Rede Neural MLP com aprendizagem Backpropagation em 2 camadas, resolva o problema de classificação dos pontos A_i e B_j dados abaixo:

	x_1	x_2	d
A_1	0	2	1
A_2	1	2	1
A_3	1	3	1
B_1	1	0	0
B_2	2	1	0



$y^* = \Sigma w_j x_j + \theta_j$

$y = 1/(1+e^{(-y^*)})$

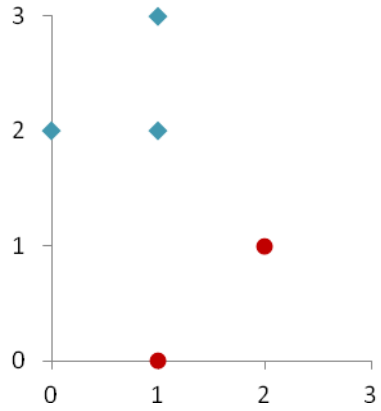
$\Delta w_j = \alpha(d-y)x_j y(1-y)$

$w_j = w_j + \Delta w_j$

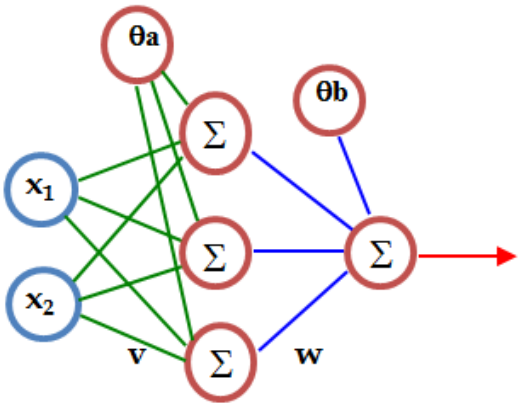
$\Delta\theta = \alpha(d-y)y(1-y)$

Pesos iniciais:

w_1=0,9; w_2=-0,9; θ=0,9



2. Com os dados do problema anterior, utilize a Rede Neural Perceptron com aprendizagem Backpropagation em 3 camadas para resolver o problema de classificação dos pontos.

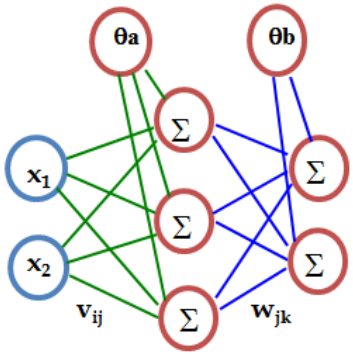


$z_j^* = \Sigma v_{ij} x_i + \theta a_j$
$z_j = 1/(1+e^{(-z_j^*)})$
$y^* = \Sigma w_j z_j + \theta b$
$y = 1/(1+e^{(-y^*)})$
$\Delta w_j = \alpha(d-y) z_j y(1-y)$
$w_j = w_j + \Delta w_j$
$\Delta \theta b = \alpha(d-y)y(1-y)$
$\Delta v_{ij} = \alpha y(1-y)(d-y) w_j z_j (1-z_j) x_i$
$v_{ij} = v_{ij} + \Delta v_{ij}$
$\Delta \theta a_j = \alpha y(1-y)(d-y) w_j z_j (1-z_j)$

Pesos iniciais:

θa	1	2	3	v	1	2	3	w	1
1	0,9	0,9	-0,9	1	-0,9	0,9	-0,9	1	0,9
θb	1			2	0,9	-0,9	0,9	2	-0,9
1	-0,9							3	0,9

3. Utilize a MLP com aprendizagem Backpropagation em 3 camadas para resolver o problema de classificação dos pontos abaixo, com 2 entradas e 2 saídas.



x_1	x_2	d_1	d_2
-1	-1	0	0
-1	1	0	1
-1	-1	0	0
1	1	1	1

$z_j^* = \Sigma v_{ij} x_i + \theta a_j$: $z_j = 1/(1+e^{(-z_j^*)})$
$y_k^* = \Sigma w_{jk} z_j + \theta b_k$: $y_k = 1/(1+e^{(-y_k^*)})$
$\Delta w_{jk} = \alpha y_k (1-y_k)(d_k - y_k) z_j$
$w_{jk} = w_{jk} + \Delta w_{jk}$
$\Delta \theta b_k = \alpha (d_k - y_k) y_k (1-y_k)$
$\Delta v_{ij} = \alpha \Sigma [y_k (1-y_k)(d_k - y_k) w_{jk}] z_j (1-z_j) x_i$
$v_{ij} = v_{ij} + \Delta v_{ij}$
$\Delta \theta a_j = \alpha \Sigma [y_k (1-y_k)(d_k - y_k) w_{jk}] z_j (1-z_j)$

Pesos iniciais:

θa	1	2	3	v	1	2	3	w	1	2
1	0,30	0,25	0,30	1	0,75	0,90	0,85	1	0,65	0,70
θb	1	2		2	0,85	0,90	0,75	2	0,60	0,75
1	-0,80	-0,40						3	0,65	0,70

4. Utilize uma rede neural MLP com aprendizagem Backpropagation com pelo menos uma camada escondida para resolver o problema de classificação abaixo. Faça 4 iterações completas e interprete graficamente a solução no final de cada iteração.

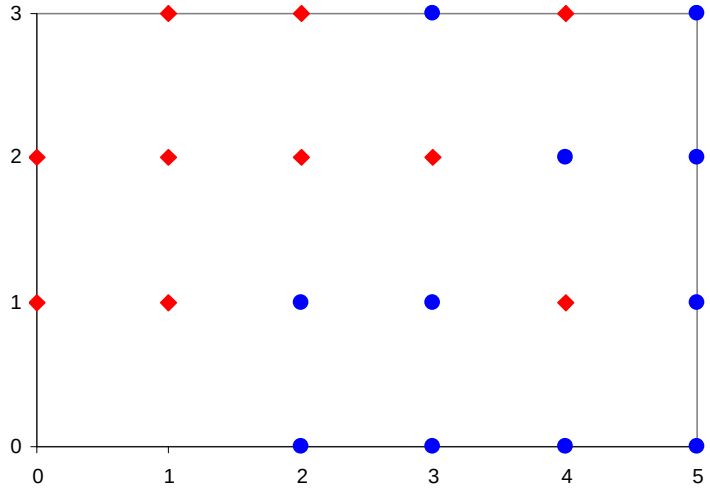
$$X = \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{matrix}\begin{pmatrix} -0{,}92 & -0{,}29 \\ -0{,}28 & -0{,}32 \\ 0{,}21 & 0{,}07 \\ 0{,}81 & -1 \\ -0{,}5 & 1 \\ 0{,}62 & 0{,}87 \\ -0{,}71 & -0{,}87 \\ 0{,}42 & 0{,}09 \end{pmatrix}, \text{ onde } d = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

5. Utilize uma rede neural MLP com aprendizagem Backpropagation com 1 camada escondida para resolver o problema de classificação de 22 pontos da página 19. Utilize pesos aleatórios.

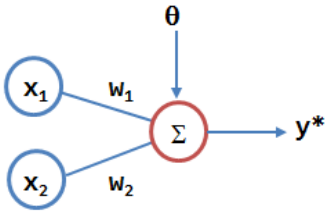
Exemplo:

Com os dados abaixo, utilize a Rede Neural Perceptron com aprendizagem Backpropagation em 2 camadas para resolver o problema de classificação dos pontos.

x_1	x_2	d
0	1	1
0	2	1
1	1	1
1	2	1
1	3	1
2	2	1
2	3	1
3	2	1
4	1	1
4	3	1
2	0	0
2	1	0
3	0	0
3	1	0
3	3	0
4	0	0
4	2	0
5	0	0
5	1	0
5	2	0
5	3	0



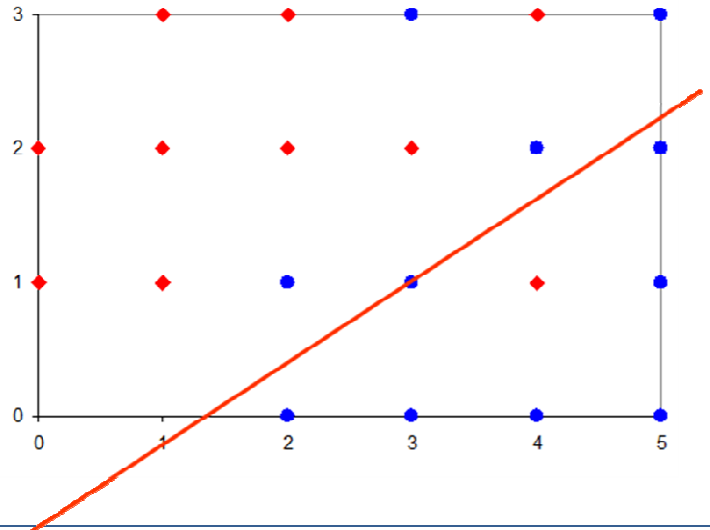
Usando o Perceptron com 2 camadas, e aprendizagem Backpropagation, temos os seguintes resultados:



Na 10ª iteração, temos:

w_1=2,95 w_2=-2,79 θ=-1,94

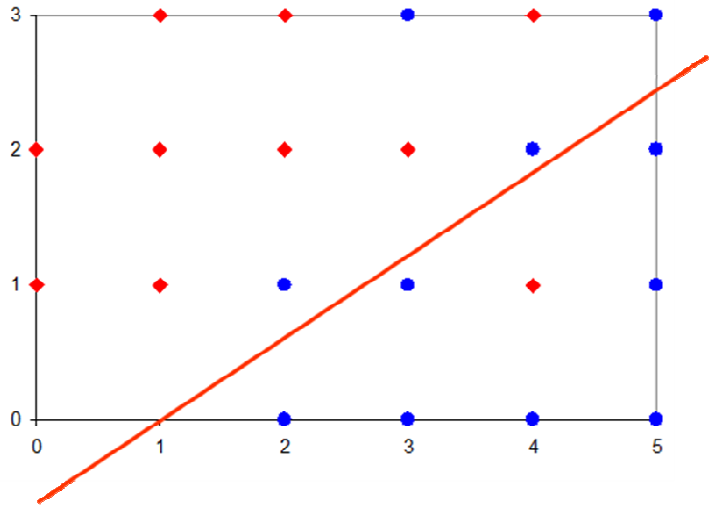
erro=1,52



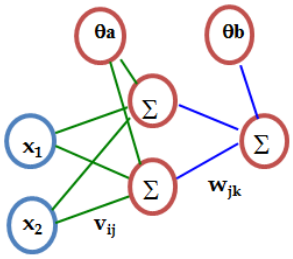
Na 100ª iteração, temos:

w_1=4,19 w_2=-4,08 θ=-4,39

erro=1,38



Adicionando-se uma camada escondida, e utilizando os pesos abaixo com saídas lineares em todos os neurônios, temos:



θa	1	2	v	1	2	w	1
1	0,0	0,1	1	0,0	0,0	1	0,2
θb	1		2	0,1	-0,1	2	-0,2
1	0,2						

Na 100ª iteração, temos: erro=1,07

θa	1	2	v	1	2	w	1
1	-1,37	1,39	1	2,83	-3,63	1	2,87
θb	1		2	-2,86	3,71	2	-2,74
1	-0,19						

Equações discriminantes encontradas:

$w_1(x_1v_{11}+x_2v_{21}+\theta a_1)+\theta b=0$

$2,87(2,83x_1-3,63x_2-1,37)-0,19=0$

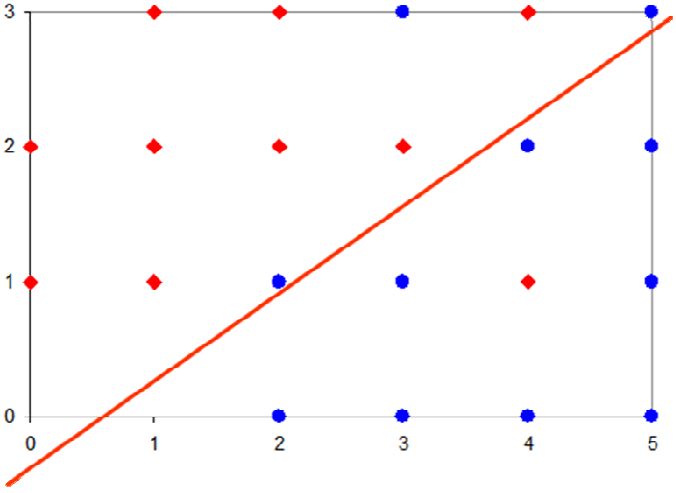
$8,12x_1-10,4x_2=4,12$

$w_2(x_1v_{12}+x_2v_{22}+\theta a_2)+\theta b=0$

$-2,74(-2,86x_1+3,71x_2+1,39)-0,19=0$

$7,84x_1-10,17x_2=4$

$15,96x_1-20,57x_2=8,12$

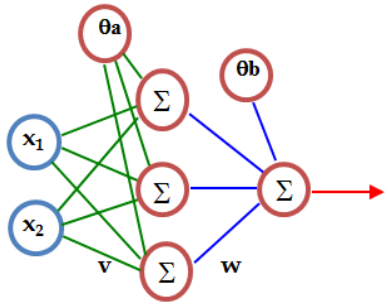


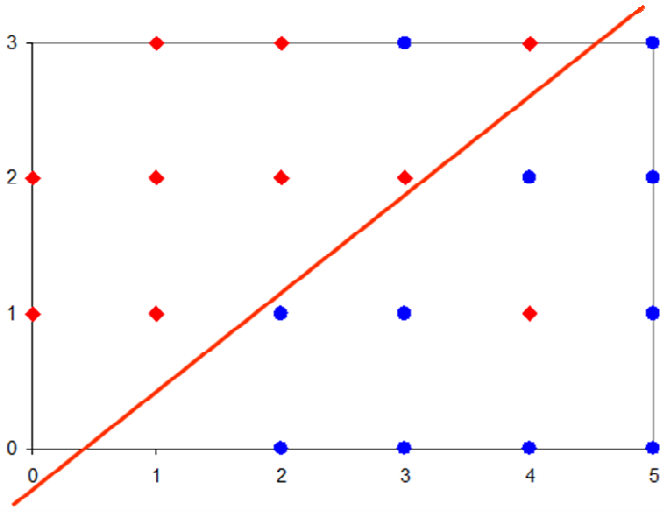
...

Na 500ª iteração, temos:

erro=0,966

Adicionando-se mais um neurônio na camada escondida, temos:





Na 100ª iteração, temos:

erro=1,05

Na 500ª iteração, temos:

erro=0,863

Observações:

Sobre a parada da rede:

- se os pesos forem ajustados em valores muito grandes a ativação se torna zero ou um e os ajustes passam a ser nulos, parando a rede

Sobre mínimos locais:

- a superfície de erro de uma rede complexa é cheia de montanhas e vales
- a RNA pode ficar presa em um ponto de mínimo local

Uma MLP pode conduzir a erro mínimo local ao invés de global

- este erro mínimo local pode ser satisfatório, mas e se não for?
  - uma rede com mais neurônios poderá fazer um trabalho melhor

O número de neurônios ou camadas adequados não é de determinação simples:

- pode-se usar diferentes conjuntos de soluções iniciais para melhorar a solução do problema

Sobre a escolha do número de neurônios das camadas:

- Para a camada de entrada, o número de variáveis fornecidas
- Para a camada de saída, um neurônio para cada item de classificação
- Para camadas escondidas com i unidades, onde $0 \leq i \leq k$:
  - começa-se com $i = 0$ (sem camada oculta) e verifica-se o número de padrões classificados corretamente
  - prossegue-se com $i = 1$ (um neurônio na camada escondida) e verifica-se o número de padrões classificados corretamente
  - ... e assim continua-se até $i = k$. Destas k tentativas escolhe-se para i, aquela que classificou o maior número de padrões corretamente

Sobre a taxa de aprendizagem inicial:

- α deve ser alta no início do treinamento e decline gradativamente à medida que ele evolui [Gorni, 1993].

- Isto deve proporcionar rapidez na convergência do treinamento, estabilidade e resistência ao aparecimento de mínimos locais.
- Escalonamento: $\alpha(n)=\frac{\alpha_0}{1+\frac{n}{n_0}}$, onde $\begin{cases}\alpha_0 \text{ é o tamanho inicial do passo}\\ n_0 \text{ é um contador iterativo}\end{cases}$, tal que:
  - α_0 é muito grande conduz à aprendizagem instável
  - quando n_0 for muito pequeno não existe aprendizagem
  - quando n_0 for muito grande, então a aprendizagem é lenta

Diferentes pesos conduzem a resultados diferentes
- conjuntos de pesos diferentes podem conduzir a mínimos locais diferentes ou ao mínimo global
- os pesos, *W*, para cada uma das topologias $0 \le i \le k$, são arbitrários, variando no intervalo (-1, 1)
- todo o processo é repetido para *r* conjuntos de pesos diferentes.
- registra-se a situação que para a topologia com *i** unidades e com um conjunto de pesos *W* apresentar melhor desempenho

Sobre o ajuste dos pesos
- é mais recomendada a regra de aprendizado aplicada a cada padrão separadamente, isto é, o padrão *p* é aplicado, calculado, e os pesos são atualizados
- Existe uma indicação empírica que isto resulta em convergência mais rápida [Krose et al, 1993]
- Vários autores fazem a atualização dos pesos após a apresentação completa do conjunto de padrões

Sobre os possíveis critérios de parada: considere a rede com número de neurônios *i* na camada escondida, $0 \le i \le k$, e conjunto de pesos *W*:
- o desempenho da rede apresenta 0% de erro
  - *W* é registrado e o processo é finalizado
- a rede com *i* neurônios na camada escondida apresenta uma variação de erro entre 2 iterações consecutivas menor do que um valor ξ
  - se todas as topologias, $0 \le i \le k$, já foram analisadas, então toma-se *W* (da topologia com *i* neurônios) que tiver maior percentagem de acerto *i**
  - repete-se este processo para os *r* conjuntos de pesos, obtendo-se *r* valores para *i**
  - entre os *r* conjuntos de pesos, escolhe-se o conjunto *W* correspondente a *i**, que classifique o maior número de padrões corretamente

A apresentação dos padrões pode ser feita em ordem aleatória para a rede durante o treinamento
- a cada iteração na aplicação do algoritmo *backpropagation* é recomendável fazer um sorteio na sequência de apresentação dos padrões de entrada

O dimensionamento da rede deve ser feito com critério, fazendo testes com várias topologias ou arquiteturas
- deve-se evitar uma rede super-dimensionada (com muitos neurônios)
- deve-se evitar também uma rede sub-dimensionada (poucos neurônios)

Prováveis motivos para a não convergência de uma rede [Hadjiprocopis, 1993]
- coeficiente de aprendizado inicial muito alto
- dados com magnitude incompatível com a função de ativação escolhida

- rede mal dimensionada

Critérios de parada de uma RNA:

- Número de iterações
- Determinado valor do erro total mínimo:
  - variável indireta da classificação
  - porém, não há garantia de que o sistema possa atingir um erro mínimo especificado
- Taxa de decréscimo de erro:
  - indica quando não há mais extração de informação da rede
  - porém, em alguns casos pode haver parada prematura da rede
- Parada com validação cruzada:
  - dois conjuntos: treinamento e validação
  - conjunto de validação: geralmente entre 10 e 25% do total
  - com certa freqüência (em geral, de 5 a 10 iterações) verifica-se o desempenho no conjunto de validação
  - quando o erro aumenta, pára-se o treinamento

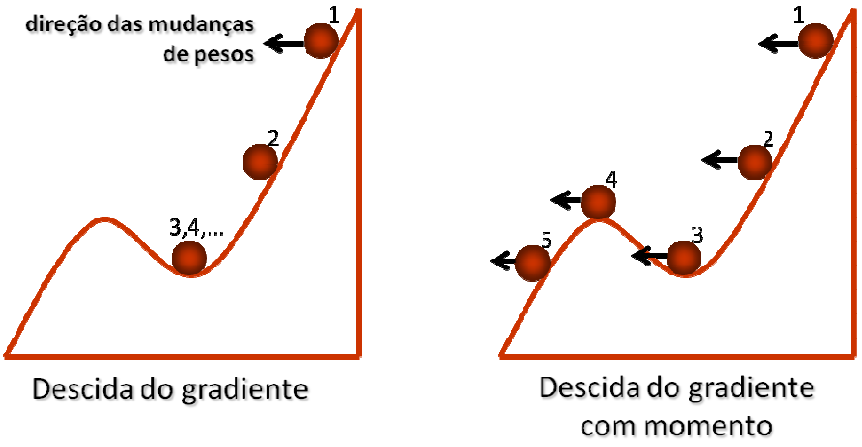
Algumas heurísticas para melhorar a convergência de uma RNA:

- Normalizar os dados em relação à faixa de ativação da rede
- Usar não-linearidade do tipo tanh ou sigmóide
- Normalizar o sinal desejado ligeiramente acima/abaixo do limite (por exemplo ±0,9 e não ± 1)
- Sempre ter mais padrões de treinamento que pesos
- Usar validação cruzada para parar o treinamento
- Rodar a rede várias vezes para medir o desempenho

APRENDIZAGEM COM TAXA DE MOMENTO

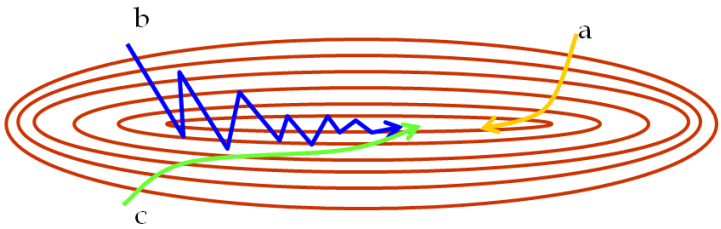
Aprendizagem com momento usa uma memória (incremento anterior) para aumentar a velocidade e estabilizar a convergência. É uma variação simples do Algoritmo Backpropagation para acelerar a convergência da RNA.

$$w_{ij}(n+1) = w_{ij}(n) + \alpha\,\delta_i(n)\,x_j(n) + \gamma\left[w_{ij}(n) - w_{ij}(n-1)\right]$$



Descida do gradiente — Descida do gradiente com momento

Normalmente, γ é ajustada entre 0,5 e 0,9.



Quando o termo momento é acrescentado e a taxa de aprendizagem é pequena, leva bastante tempo para o mínimo ser alcançado **(trajetória "a")**;

Quando o termo momento não é considerado e a taxa de aprendizagem é alta, o mínimo nunca é alcançado porque ocorrem oscilações **(trajetória "b")**;

Quando a taxa de aprendizagem é alta, mas o termo momento é considerado, o mínimo é alcançado rapidamente **(trajetória "c")** [Krose et al, 1993].

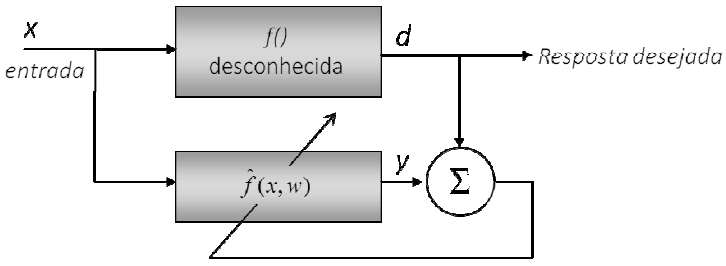
Exercício:

Resolva o exercício 2 da página 28 aplicando a taxa de momento com coeficiente $\gamma = 0{,}6$.

APROXIMAÇÃO DE FUNÇÕES

O objetivo da aprendizagem é descobrir a função *f* dado um número finito (de preferência pequeno) de pares entrada-saída (*x*,*d*).

As RNAs são úteis para aproximação de funções pois são aproximadores universais, eficientes e podem ser implementadas como máquinas de aprendizagem



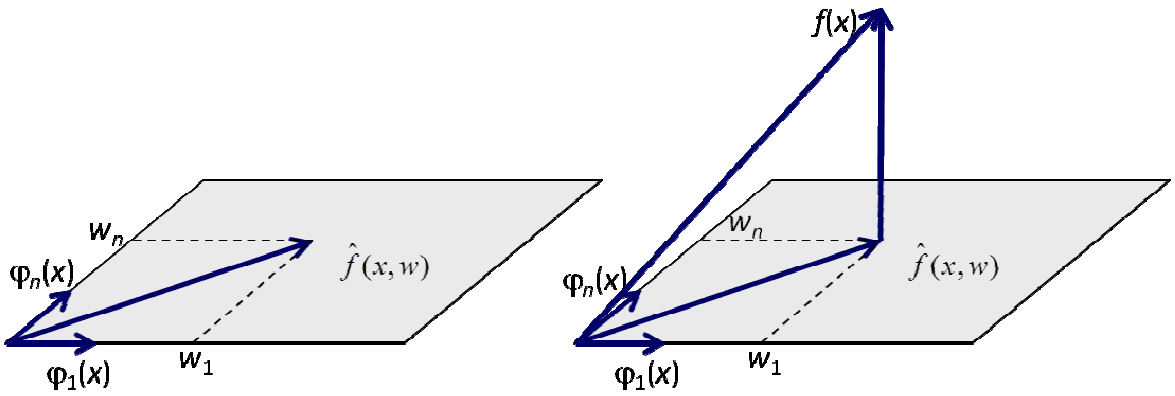
Teorema da projeção linear:

Em uma área compacta *S* do espaço de entrada descrever uma função $f(\boldsymbol{x})$, pela combinação de funções $\varphi_i(\boldsymbol{x})$ mais simples:

$$\hat{f}(x,w)=\sum_{i=1}^{n} w_i \varphi_i(x)$$

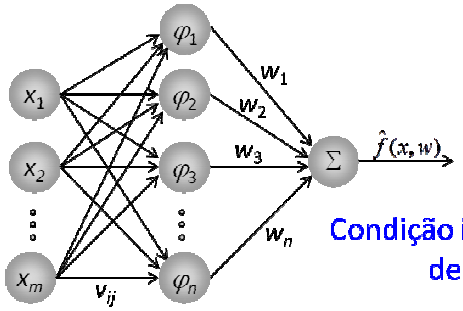
onde $\boldsymbol{w_i}$ são elementos reais do vetor $\boldsymbol{w} = [w_1,\ldots,w_n]$ tais que $\left|f(x)-\hat{f}(x,w)\right|<\varepsilon$ e ε pode ser arbitrariamente pequeno. A função é chamada de *aproximante* e as funções $\{\varphi_i(\boldsymbol{x})\}$ são chamadas de *funções elementares* $\hat{f}(x,w)$.

Quando $f(x)$ é externo ao espaço de projeção, o erro diminui fazendo $\hat{f}(x,w)$ mais próximo de $f(x)$. Para diminuir o erro da aproximação, basta aumentar o número de "bases" (funções elementares).



O requisito para o método funcionar é que $\varphi^{-1}(x)$ deve existir. Se as funções elementares constituírem uma base, isto é, elas forem linearmente independentes teremos:

$$w_1\varphi_1(x)+\cdots+w_n\varphi_n(x)=0 \quad \text{se e somente se} \quad (w_1,\cdots,w_n)=0$$



Condição importante: a inversa de φ deve existir

As decisões básicas na aproximação de funções são:

- escolha das funções elementares $\{\varphi_i(x)\}$
- como calcular os pesos w_i
- seleção do número de funções elementares

Se o número de vetores de entrada x_i é igual ao número de funções elementares $\{\varphi_i(x)\}$ a solução torna-se $w=\varphi^{-1}\hat{f}$.

As funções elementares podem ser *globais* (abrangem todo o espaço de entrada) ou *locais*, (abrangem uma área limitada do espaço de entrada).

Uma rede MLP com uma camada escondida com um neurônio de saída linear pode ser considerada uma implementação de um sistema para aproximação de funções, onde as bases são os neurônios escondidos.

A composição de neurônios sigmoidais corresponde a todo o espaço de entrada. A MLP implementa uma aproximação com funções elementares globais.

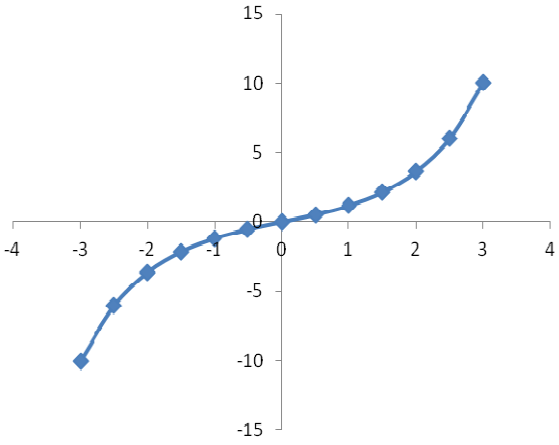
MLPs realizam aproximação de funções com um *conjunto adaptativo de bases*, determinado a partir dos dados *entrada-saída.* As bases são alteradas em função dos dados: o espaço de projeção é dependente dos dados.

O treinamento é mais difícil, pois não somente a projeção como também a base está sendo alterada. Devido à alta conectividade e natureza global das funções elementares, um bom ajuste é obtido com poucas bases (poucos neurônios escondidos).

Exercícios:

1. Dada a matriz de valores de entrada x e os desejados d, elabore uma arquitetura de MLP, com pelo menos uma camada escondida, que seja capaz de aproximar a função dada abaixo:

x	d
-3	-10
-2,5	-6
-2	-3,6
-1,5	-2,1
-1	-1,2
-0,5	-0,5
0	0
0,5	0,52
1	1,18
1,5	2
2	3,6
2,5	6,05
3	10,02



2. Utilizando o algoritmo do Perceptron simples, encontre os pesos e bias para classificar os seguintes conjuntos de treinamento:

a)

x_1	x_2	x_3	d
1	1	1	1
1	1	0	0
1	0	1	0
0	1	1	0

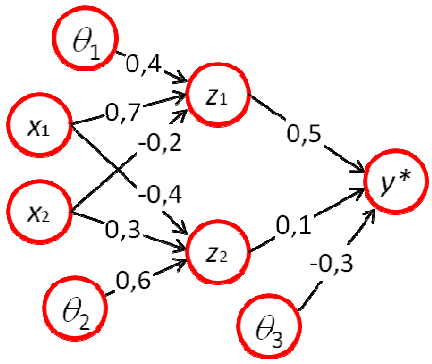
b)

x_1	x_2	x_3	x_4	d
1	1	1	1	1
-1	1	-1	-1	1
1	1	1	-1	-1
1	-1	-1	1	-1

c)

x_1	x_2	x_3	d
1	1	1	1
1	1	-1	-1
1	-1	1	-1
-1	1	1	-1

3. Encontre os novos pesos e bias da rede MLP ao lado quando o padrão x=(0,1) é apresentado à rede, com d=1 (use α=0,25 e função de ativação sigmóide, aprendizagem backpropagation)



4. Encontre os novos pesos e bias da rede MLP ao lado quando o padrão x=(-1,1) é apresentado à rede, com d=1 (use α=0,25, função de ativação tanh e aprendizagem backpropagation)

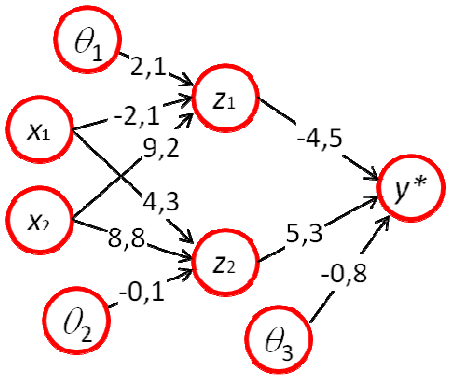
5. Dada a rede MLP ao lado, calcule os novos pesos e bias na seguinte sequência de apresentação de dados x_i de entrada (use α=0,4, função de ativação sigmóide, e calcule o erro de classificação em cada entrada):

   5.1. x_1=(-1,1), com d=0,9;

   5.2. x_2=(1,-1), com d=0,8;

   5.3. x_3=(1,1), com d= -0,7.

   Depois calcule a saída (com os pesos e bias atualizados) para o vetor de testes x=(-1,-1), com d= -0,8, e calcule o erro de classificação.



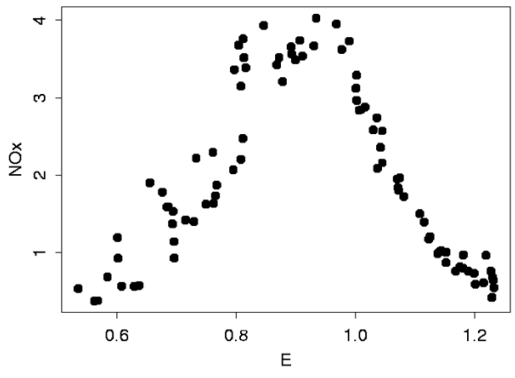
Exemplo de aplicação de MLP para aproximação de funções

No exemplo a seguir, o nó de saída faz uma combinação de duas funções:

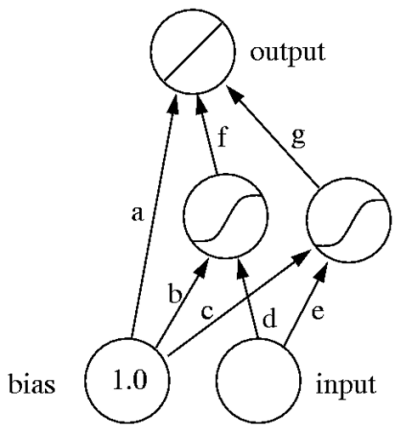
$$y = f\tanh_1(x) + g\tanh_2(x) + a, \text{ onde}$$
$$\tanh_1(x) = \tanh(dx + b)$$
$$\tanh_2(x) = \tanh(ex + c)$$

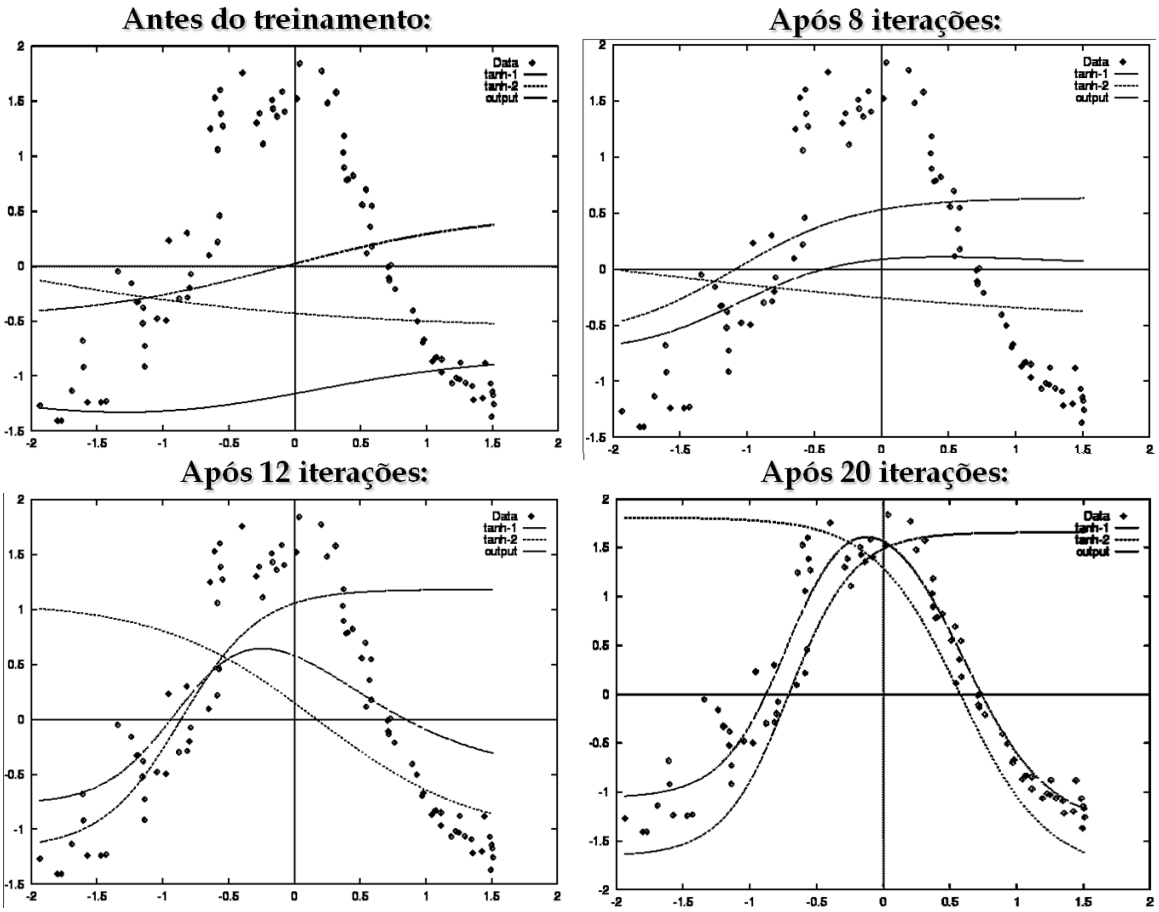
Os pesos foram inicializados com valores aleatórios e cada nó escondido computa uma função tanh aleatória.

Dados:



Arquitetura da rede:





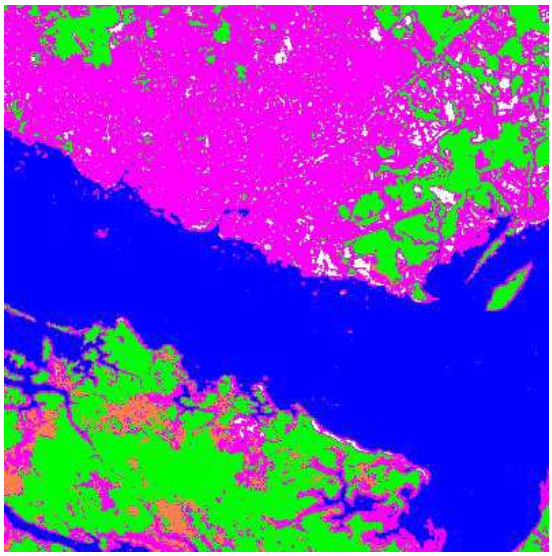
Exemplo de aplicação de MLP para classificação de imagens

Um dos trabalhos pioneiros neste estudo compara o desempenho de métodos estatísticos de classificação de dados multiespectrais, com o desempenho de classificadores por redes neurais [Benediktsson, Swain & Ersoy, 1990].

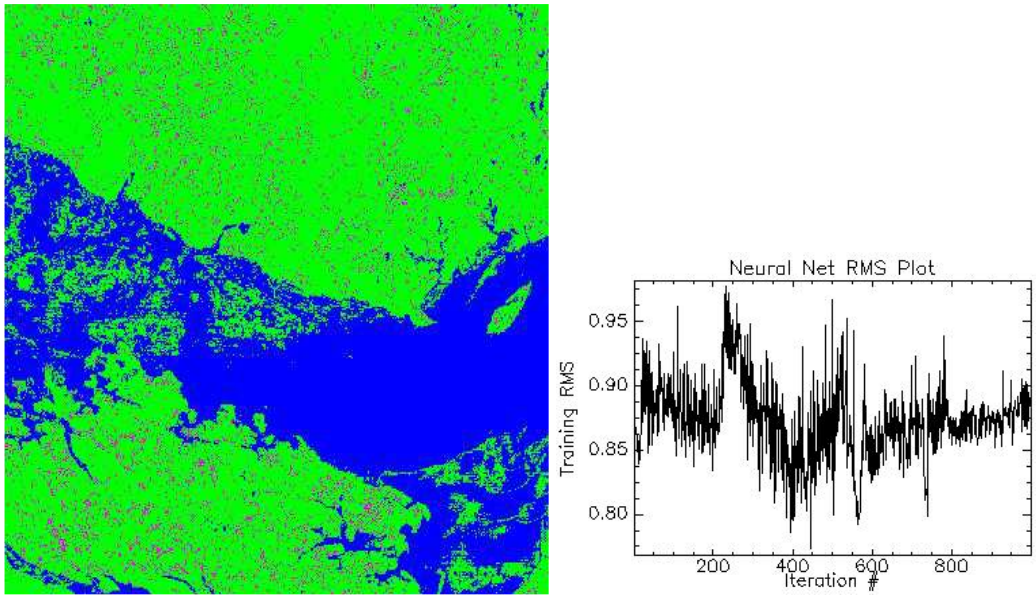
As redes neurais com conjunto de treinamento mínimo, fornecem resultados superiores aos obtidos por classificadores supervisionados tradicionais [Hepner *et all,* 1990].

Seguem abaixo as imagens e respectivas classificações dos autores:

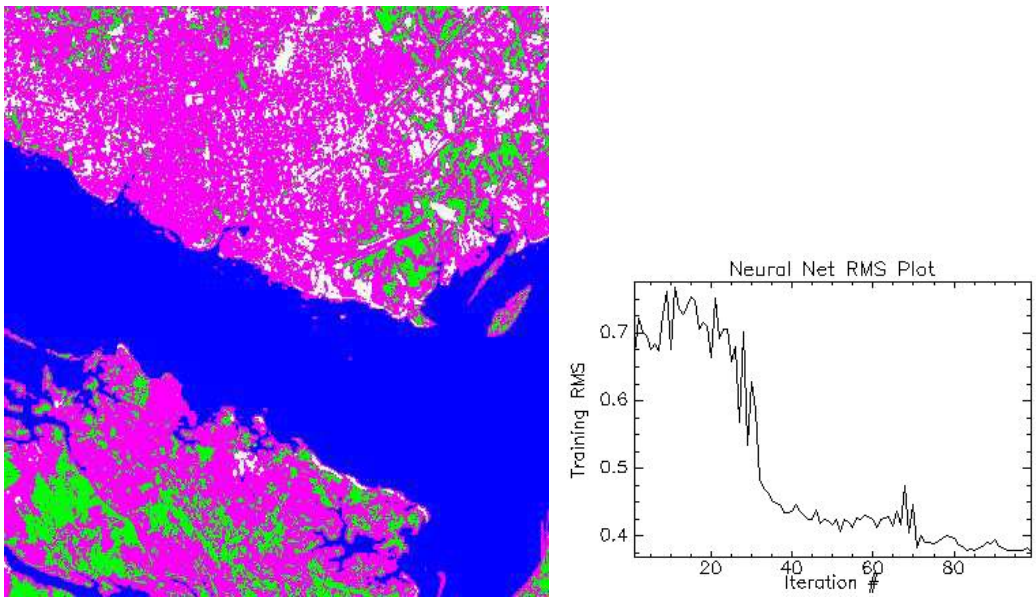
Foto aérea digitalizada, com os dados que devem ser classificados



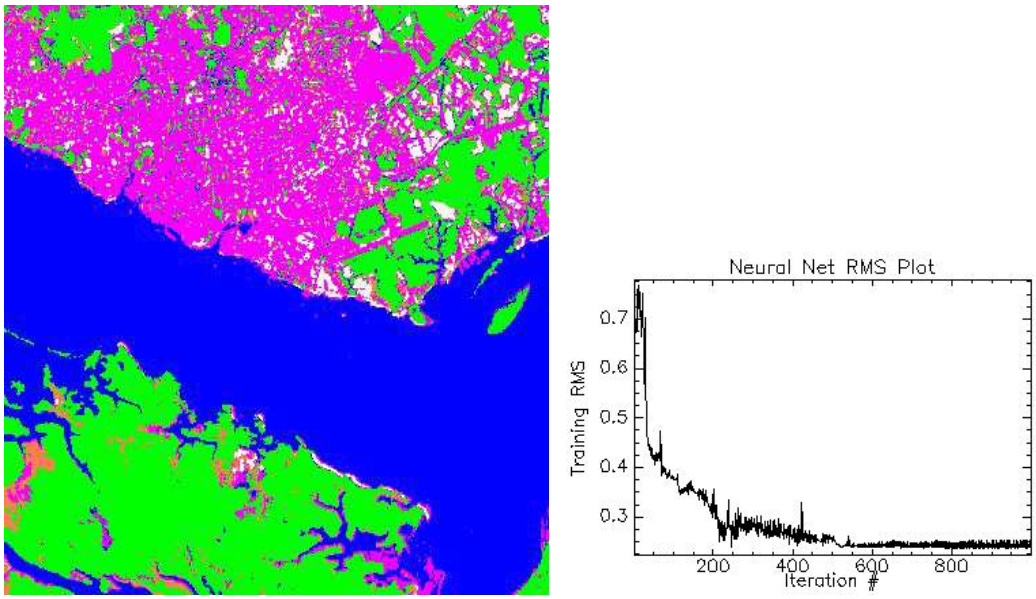
Aplicação do método de Máxima Verossimilhança (MAXVER). Dados classificados corretamente: 71,19%



Perceptron com Backpropagation (usando 20 dados de entrada). Classificação correta = 18,82%



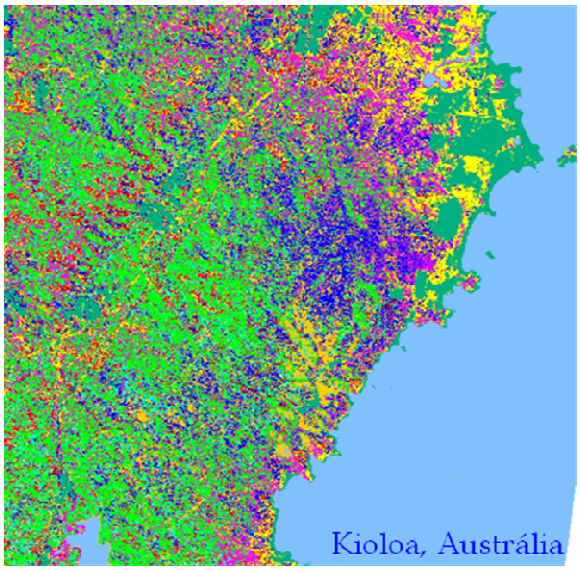
Perceptron com Backpropagation (usando 1 amostra de cada classe que se deseja classificar). Classificação correta = 58,52%



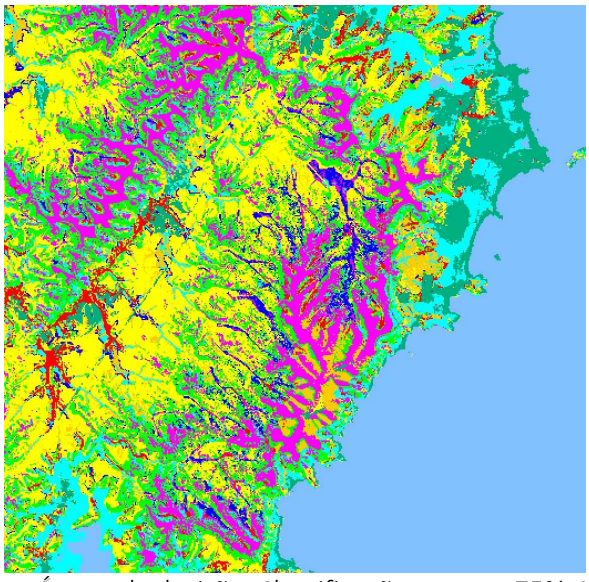
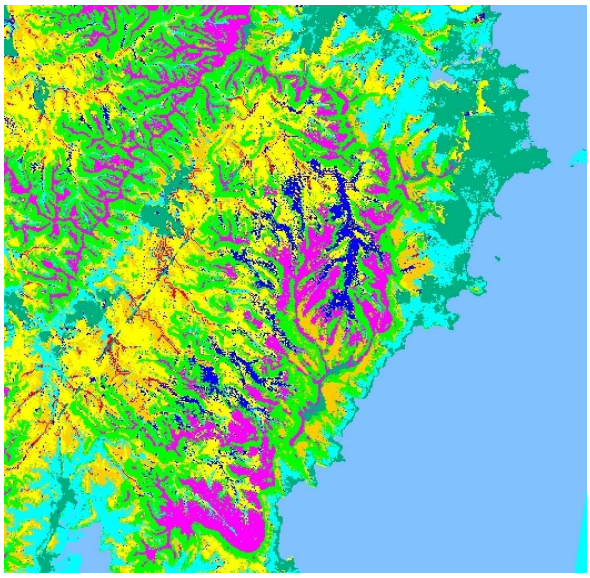
Perceptron com Backpropagation (usando mais amostras de cada classe que se deseja classificar). Classificação correta = 95,19%

No trabalho de [Mark Gahegan, Geoff West, 1998] os autores utilizaram 3 técnicas para classificações de imagens.

As imagens a seguir mostram os resultados encontrados:



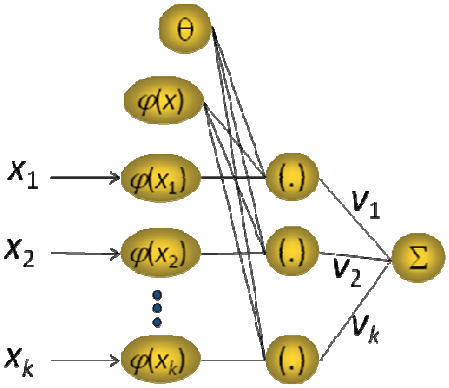
Máxima Verossimilhança. Classificação correta: 40%

Árvore de decisão. Classificação correta: 75%; MLP + BackPropagation. Classificação correta: 85%

2.5. SUPPORT VECTOR MACHINES (SVM)

A idéia básica é a de separação de dados linear, a mesma do Perceptron. A fundamentação teórica surgiu com separações não lineares [Boser, Guyon e Vapnik, 1992].

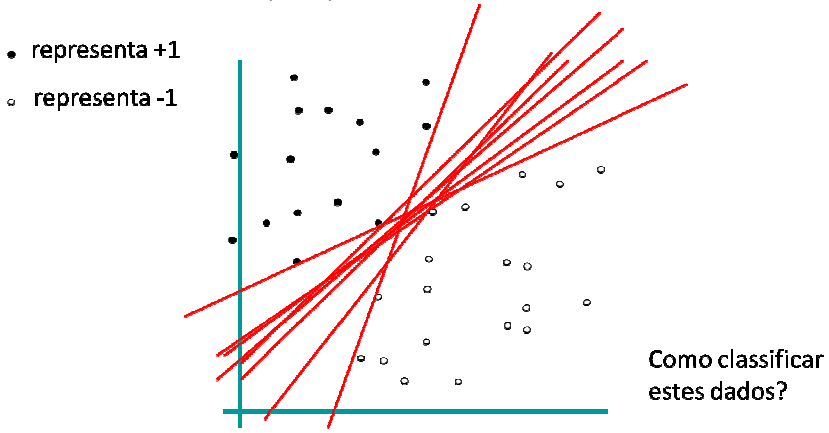


Funções não-lineares são apresentadas para separação não-linear de dados [Chervonenkis, 1992], usando hiperplanos de margens fixas e flexíveis [Cortes e Vapnik, 1995]. Mais detalhes podem ser encontrados na página de Andrew W. Moore: www.cs.cmu.edu/~awm.

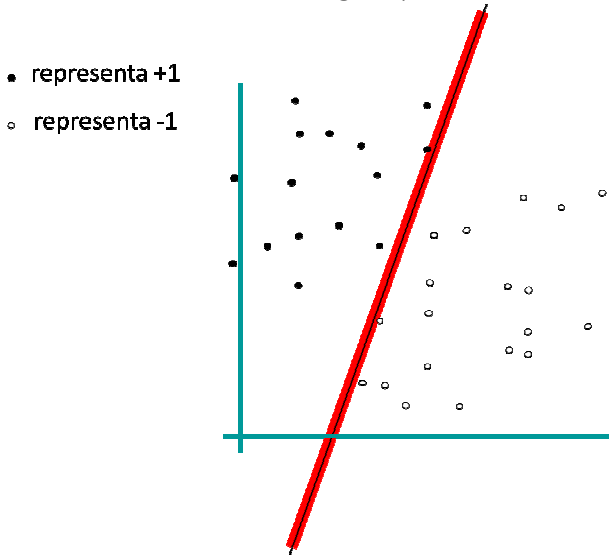
Esta rede neural também pode ser usada na resolução dos seguintes problemas:

- → Classificação de textos
- → Classificação de imagens
- → Bioinformática (classificação de proteínas, diagnóstico médico)
- → Reconhecimento de caracteres escritos à mão livre

Um classificador simples pode obter diversos resultados, como mostra a figura abaixo:

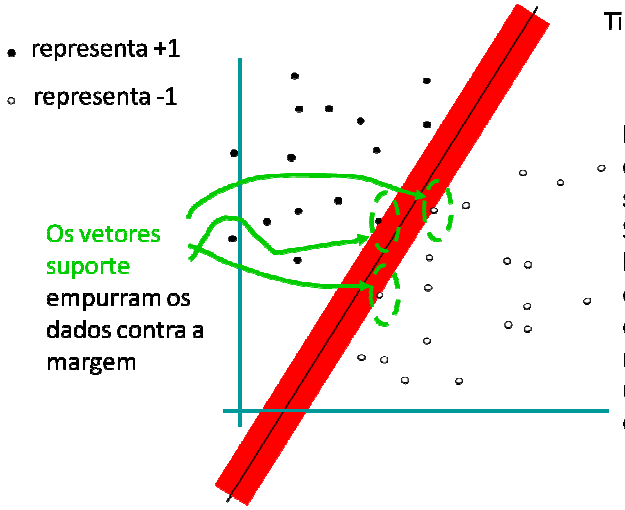


Um classificador com margem pode obtem a seguinte configuração:



Define a margem de um classificador linear como o comprimento de uma vizinhança da região de classificação.

Este tipo de classificador define uma margem máxima:

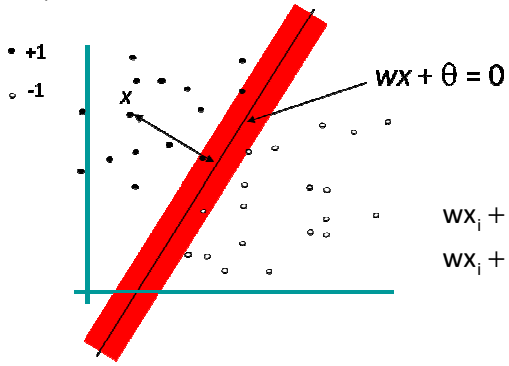


Tipo mais simples de SVM: linear

Os vetores suporte empurram os dados contra a margem

Intuitivamente, a classificação é mais segura.
Se um pequeno erro na localização da fronteira existir (se for alterado na direção perpendicular à margem) isto pode causar um erro de classificação de fácil detecção

A largura ideal desta margem pode ser encontrada da maneira descrita abaixo:



$$wx_i + \theta \geq +1 \quad \text{para} \quad y_i = +1$$
$$wx_i + \theta \leq -1 \quad \text{para} \quad y_i = -1$$

Primeiro vamos determinar qual é a distância de um ponto x à reta suporte $\boldsymbol{wx}+\theta = 0$:

$$d(x) = \frac{|wx + \theta|}{\sqrt{\|w\|_2^2}} = \frac{|wx + \theta|}{\sqrt{\sum_{i=1}^{k} w_i^2}}$$

O ponto mais próximo da margem pode ser encontrado da seguinte forma:

$$m = \min_{x \in D} d(x) = \min_{x \in D} \frac{|wx + \theta|}{\sqrt{\sum_{i=1}^{k} w_i^2}}$$

A ideia de uma SVM é de maximizar a margem de tal forma que respeite a menor distância m, ou seja:

$$M = \max_{w,\theta}\left(\min_{x_i \in D} d(x_i)\right) = \max_{w,\theta}\left(\min_{x_i \in D} \frac{|wx_i + \theta|}{\sqrt{\sum_{i=1}^{k} w_i^2}}\right)$$

Para encontrar a margem ideal recaímos na resolução do seguinte problema de programação quadrática:

$$\max_{w,\theta}\left(\min_{x_i \in D} \frac{|wx_i + \theta|}{\sqrt{\sum_{i=1}^{k} w_i^2}}\right)$$

sujeito a $y_i(wx_i + \theta) \geq 1,\ \forall x_i \in D$

Mas $|wx_i + \theta| \geq 1,\ \forall x_i \in D$, logo o problema fica simplificado como:

$$\min_{w} \sum_{i=1}^{k} w_i^2$$

sujeito a $y_i(wx_i + \theta) \geq 1,\ \forall x_i \in D$

Utilizando os multiplicadores de Lagrange, obtemos:

$$L(w,\theta,\alpha) = \tfrac{1}{2}\|w\|_2^2 - \sum_{i=1}^{k} \alpha_i \left(y_i\left(wx_i + \theta\right) - 1\right)$$

$$\frac{\partial}{\partial \theta} L(w,\theta,\alpha) = 0,\ \frac{\partial}{\partial w} L(w,\theta,\alpha) = 0$$

Logo, o hiperplano classificador fica com as seguintes características:

$$\sum_{i=1}^{k} \alpha_i y_i = 0 \qquad w = \sum_{i=1}^{k} \alpha_i y_i x_i$$

Usando as condições KKT (Karush–Kuhn–Tucker) temos:

$$\alpha_i \left[y_i\left(wx_i + \theta\right) - 1\right] = 0,\ \ i = 1,\ldots,k\,.$$

Os vetores suporte são os quais possuem $\alpha_i > 0$.

Encontrando o Problema de otimização dual do Lagrangeano temos:

$$\max\ W(\alpha) = \sum_{i=1}^{k} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{k} \alpha_i \alpha_j y_i y_j x_i x_j$$

sujeito a $\sum_{i=1}^{k} \alpha_i y_i = 0$ e $\alpha_i \geq 0,\ i = 1,\ldots,k,$

onde a função de decisão é $f(x) = \operatorname{sgn}\left(\sum_{i=1}^{k} y_i \alpha_i x_i x + \theta\right) = \operatorname{sgn}\left(wx + \theta\right)$.

As funções básicas de uma SVM podem ser:

- função polinomial de x_k
- radial basis function (RBF) de x_k : $\phi_j(x_k) = \exp\left(-\frac{|x_k - c_j|^2}{\sigma^2}\right)$
- sigmoidal de x_k

→ funções Kernel (núcleo): $K(x_i,x_j)=\phi(x_i)\phi(x_j)$

Para utilizar outras funções na SVM basta substituir x pela função ϕ(x), resultando em:

max $W(\alpha)=\sum_{i=1}^{k}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{k}\alpha_i\alpha_j y_i y_j \phi(x_i)\phi(x_j)$

sujeito a $\sum_{i=1}^{k}\alpha_i y_i=0$ e $\alpha_i\geq 0,\ i=1,\ldots,k,$

onde a função de decisão é $f(x)=\text{sgn}\left(\sum_{i=1}^{k} y_i\alpha_i\phi(x_i)\phi(x)+\theta\right)$.

Na função w, como $w=\sum_{i=1}^{k}\alpha_i y_i x_i$, então $w=\sum_{i=1}^{k}\alpha_i y_i\phi(x_i)$.

As funções *Kernel* precisam ser decompostas: $K(a,b)=\phi(a)\phi(b)$.

Condição de Mercer

Para decompor uma função *Kernel* através de um produto interno $K(x,y)=\varphi(x)\varphi(y)$, $K(x,y)$ deve ser semi-definida positiva, ou seja, para qualquer função $f(x)$ que $\int f^2(x)dx$ seja finito:

$$\int f(x)K(x,y)f(y)dxdy\geq 0$$

$K(a,b)=(ab+1)^d$ é um exemplo de função *Kernel* SVM

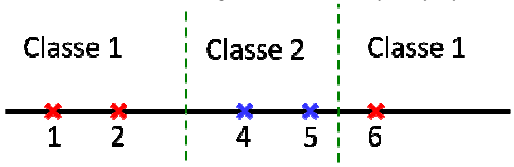
Outras funções não lineares *Kernel* SVM:

Kernel em formato RBF: $K(a,b)=\exp\left(-\frac{(a-b)^2}{2\sigma^2}\right)$.

Kernel do tipo Rede Neural: $K(a,b)=\tanh(\kappa a.b-\delta)$.

Exemplo:

Seja o conjunto de 5 pontos x_1=1, x_2=2, x_3=4, x_4=5, x_5=6, que devem ser separados em 2 classes: y_1=1, y_2=1, y_3=-1, y_4=-1 e y_5=1. Usando a função Kernel $K(a,b)=(1+a^Tb)$, temos:



Resolvendo-se o PPQ abaixo:

Maximizar $W(\alpha)=\sum_{i=1}^{5}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{5}\alpha_i\alpha_j y_i y_j\left(1+x_i x_j\right)^2$

sujeito a $\sum_{i=1}^{5}\alpha_i y_i=0$ e $\alpha_i\geq 0,\ i=1,\ldots,5$,

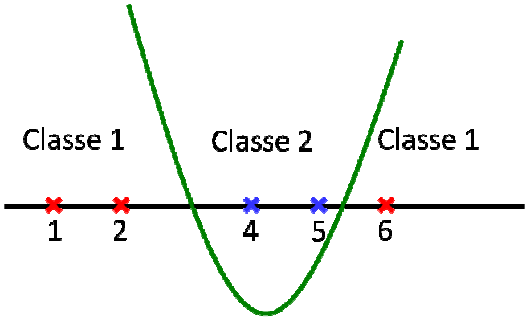
Obtemos: α_1=0, α_2=2,5, α_3=0, α_4=7,333 e α_5=4,833.

Logo os vetores suporte serão x_2=2, x_4=5 e x_5=6.

A função de decisão fica:

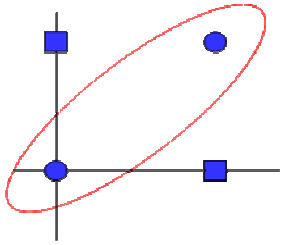
$$f(x)=\sum_{i=1}^{5} y_i\alpha_i\phi(x_i)\phi(x)+\theta = 1.2,5(1+2x)^2+(-1).7,333(1+5x)^2+1.4,833(1+6x)^2+\theta =$$
$$= 0,6667x^2-5,333x+\theta.$$

Fazendo-se $f(2)=1$ ou $f(5)=-1$ ou $f(6)=1$, temos $\theta=9$. Logo, $f(x)=0,6667x^2-5,333x+9$.



Exemplo:

Considere o exemplo de classificação da função lógica "ou" exclusiva.



$$\Phi(w)=\tfrac{1}{2}w^Tw$$

$$L(w,\theta,\alpha)=\tfrac{1}{2}w^Tw-\sum_{i=1}^{n}\alpha_i\left(d_i\left(w^Tx_i+\theta\right)-1\right)$$

$$W(\alpha)=\sum_{i=1}^{n}\alpha_i-\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_jd_id_jx_i^Tx_j$$

Usando a função Kernel $K(x,x_i)=\left(1+x^Tx_i\right)^2=1+x_1^2x_{i1}^2+2x_1x_{i1}x_2x_{i2}+x_2^2x_{i2}^2+2x_1x_{i1}+2x_2x_{i2}$

temos $\phi(x)=\left[1,x_1^2,\sqrt{2}x_1x_2,x_2^2,\sqrt{2}x_1,\sqrt{2}x_2\right]^T$. Logo,

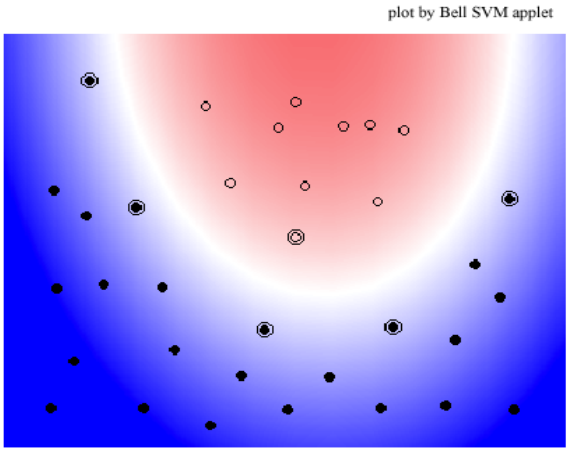
$$W(\alpha)=\alpha_1+\alpha_2+\alpha_3+\alpha_4-\tfrac{1}{2}\left(9\alpha_1^2-2\alpha_1\alpha_2-2\alpha_1\alpha_3+2\alpha_1+9\alpha_2^2+2\alpha_2\alpha_3-2\alpha_2\alpha_4+9\alpha_3^2-2\alpha_3\alpha_4+9\alpha_4^2\right)$$

...

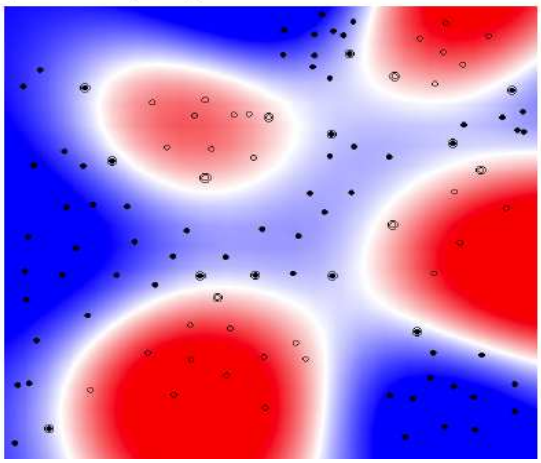
x_1	x_2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Outros exemplos:

$K(\vec{x}_i,\vec{x}_j) = [\vec{x}_i\cdot\vec{x}_j+1]^2$



$K(\vec{x}_i,\vec{x}_j) = \exp(-|\vec{x}_i-\vec{x}_j|^2/\sigma^2)$



2.6. REDES DE BASES RADIAIS (RBF)

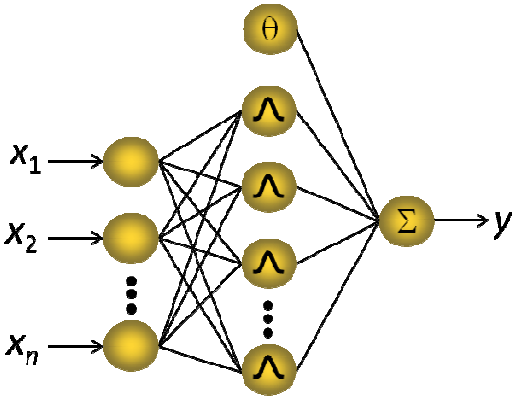
Depois das redes MLP, as redes do tipo RBF (Radial Basis Function - 1987) são as mais utilizadas na literatura.

Estas redes resolvem bem os mesmos tipos de problemas que as redes MLP: classificação, aproximação de funções e previsão de séries temporais. O treinamento das RBF é mais rápido do que de MLP.

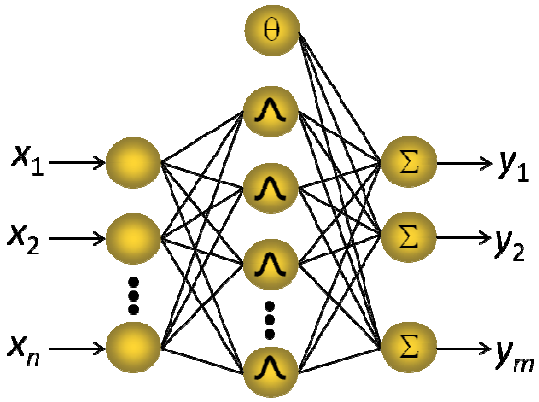
O funcionamento deste tipo de rede difere significativamente, pois elas contêm uma camada escondida com funções de bases radiais e a ativação dos neurônios da camada escondida:

- nas MLPs: produto interno entre o vetor de entrada e o vetor de pesos;
- nas RBF: distância entre o vetor de entrada e um vetor protótipo.

Quando um padrão de entrada é apresentado à rede, a distância entre o neurônio central e o padrão de entrada é calculada, e a saída da rede para o neurônio central é o resultado da aplicação da função de base radial à esta distância.



RBF com *n* entradas e 1 saída



RBF com *n* entradas e *m* saídas

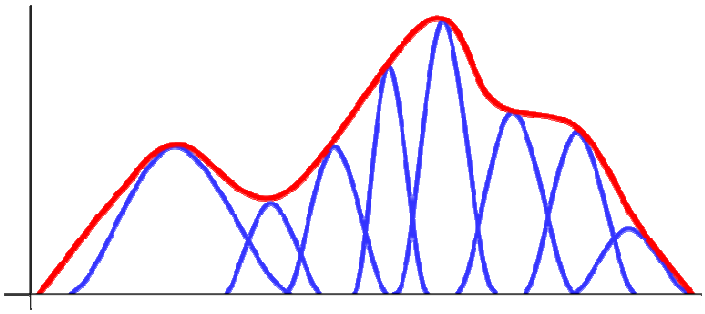
A aprendizagem consiste nos ajustes dos parâmetros para o conjunto de treinamento.

A generalização é a fase de utilização dos parâmetros ajustados na aprendizagem para interpolar dados que não pertencem ao conjunto de aprendizagem, mas que pertençam a alguma vizinhança deste conjunto.

A função gaussiana é a função de base radial mais utilizada neste tipo de rede.

Os valores dos raios das funções podem ser considerados iguais para problemas de aproximação de funções (custo computacional mais baixo).

Possuem cálculos com maior custo computacional do que de outras RNAs.

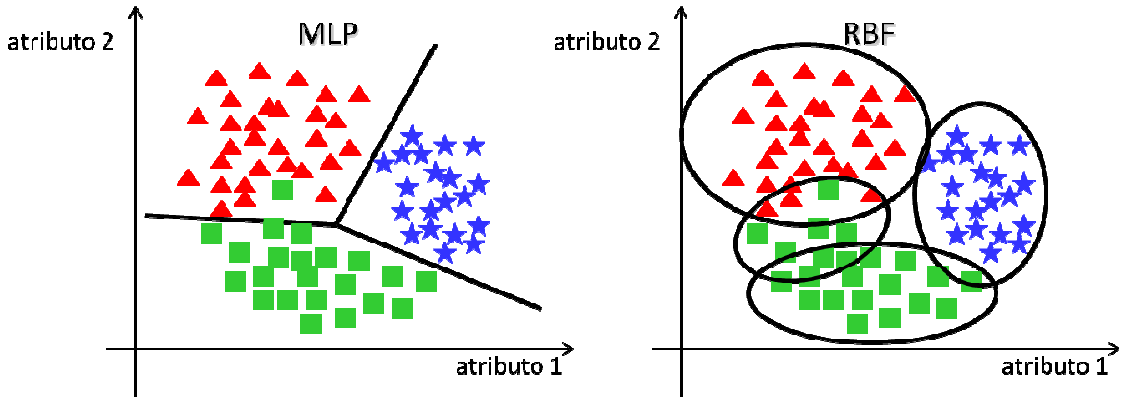


O aprendizado de uma RBF é feito em três etapas:

- selecionar o número de centros (q);
- escolher os valores dos centros (u);
- ajustar os pesos (w).

Outros tipos de redes RBF são:

- Redes Probabilísticas (PNN)
- Redes de Regressão Generalizada (GRNN)



As redes RBF têm melhor desempenho em aproximação de funções e séries temporais do que MLP, enquanto que MLP têm melhor desempenho em problemas de classificação. Estas redes fazem parte do denominado conjunto de técnicas de funções gaussianas.

Vantagens das RBFs:

- aprendizado rápido;
- não começam com pesos aleatórios;
- treinamento incremental;
- sem problemas de paralisia da rede e mínimo local.

Desvantagem:

- Após o treinamento, as redes RBFs são mais lentas para efetuar a recuperação da informação (recall), por serem compostas de mais processadores que as MLP com aprendizado tipo Backpropagation.

A idéia básica de uma RBF pode ser resumida em:

- supondo que os vetores de entrada e pesos definam pontos no espaço *N*-dimensional, o objetivo é ter uma resposta do neurônio que diminua rapidamente conforme esses pontos se distanciem;
- o conjunto de neurônios escondidos é projetado de forma que as suas respostas cubram todas as regiões significativas do espaço de vetores de entrada.

O princípio fundamental de uma RBF é determinar uma função $h(x)$ tal que:

$$h(x_n) = d_n \text{ , } n = 1, \ldots N \text{ [Bishop, 1985]}$$

A aproximação por RBF usa um conjunto de *N* funções base da forma:

$$\varphi(\|x - x_n\|)$$

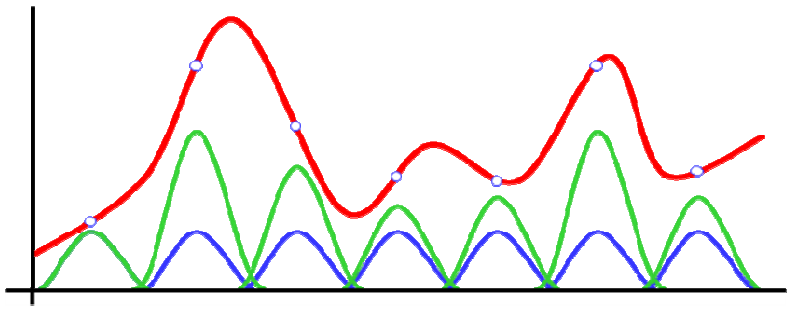
uma para cada ponto da série, onde $\varphi(.)$ é uma função não-linear.

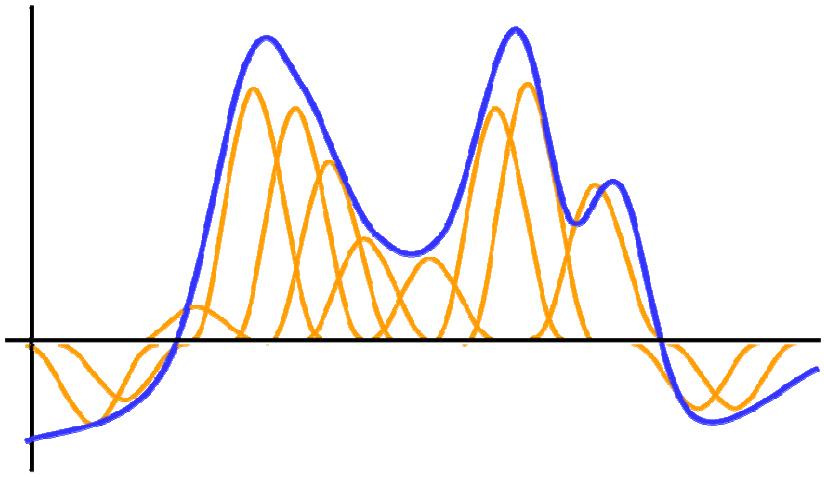
A saída é uma função h:

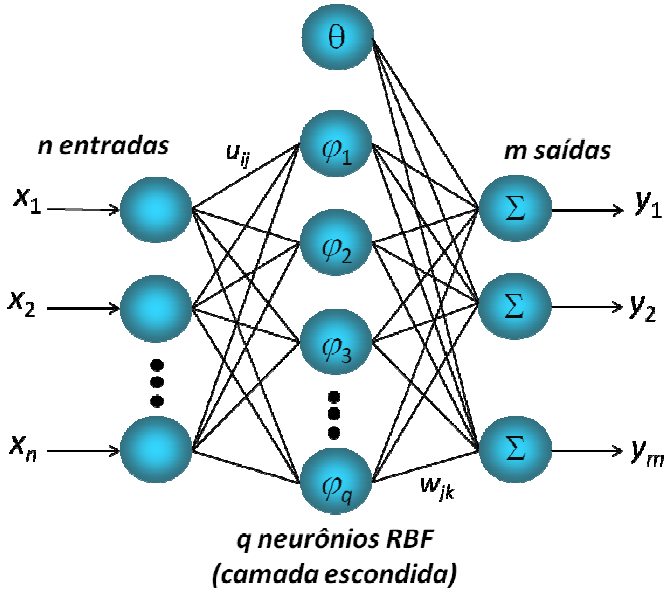
$$h(x) = \Sigma_n w_n \varphi(\|x - x_n\|) \text{ [Powel, 1988]}$$

As redes RBFs são aproximadores universais, isto é, dado um número suficiente de neurônios na camada escondida, as redes RBFs podem aproximar qualquer função contínua com precisão arbitrária.

Exemplos:







A função gaussiana que vamos utilizar é:

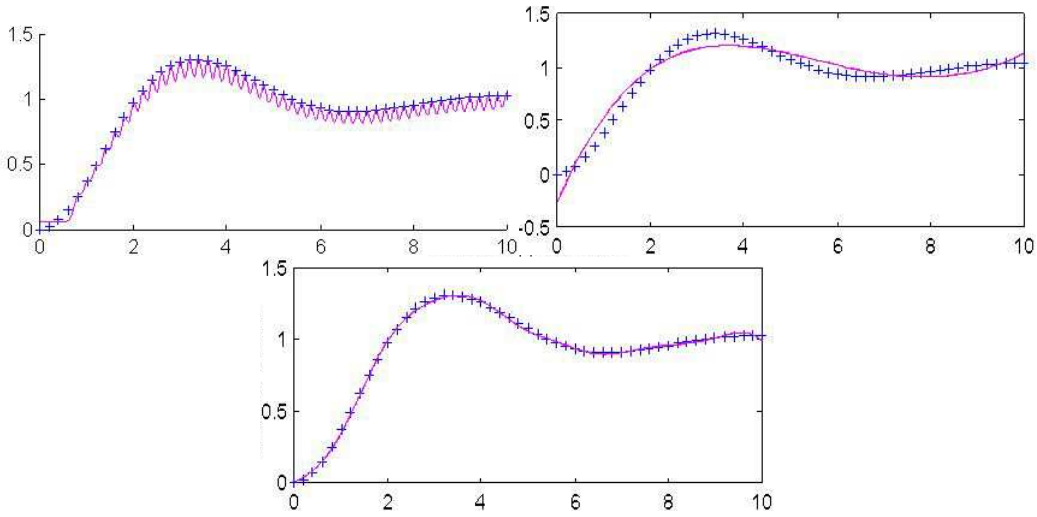
$$\varphi_j(t) = e^{\left(-\frac{1}{2\sigma_j^2(i)}\|x-u\|^2\right)}$$

onde σ_j representa o raio da função base do neurônio j e $u_j(t)$ é o vetor com as coordenadas do centro do neurônio.

Os valores dos raios "extendem" a função de base radial, e a saída da rede é uma combinação das funções de bases radiais com os pesos ajustados na iteração t:

$$y_k(t) = \sum_{j=1}^{q} w_{jk}\varphi_j(t)$$

Exemplos:



Algoritmo Básico de um Projeto para uma rede RBF

Seja uma base de dados (x_i, d_i), i = 1, 2,..., p, onde x_i é um exemplo da base de dados e d_i é o vetor de saídas desejadas correspondentes.

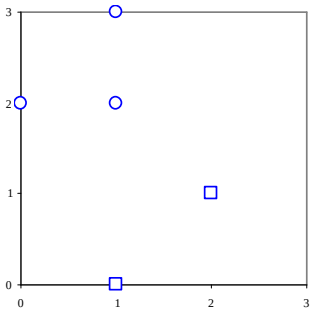
- definir o número q de neurônios ocultos (número de bases radiais), em geral escolhe-se $q \leq n$.
- selecionar aleatoriamente q exemplos do conjunto de dados, e fazer a seguinte atribuição: $u_j = x_j$, j = 1, 2,..., q.
- especificar o valor do raio da função de base radial, σ
- apresentar todos o conjunto de treinamento para a rede, guardando a saída da camada escondida de cada entrada em uma linha da matriz G, onde $g_{ij} = \varphi_j$, e $g_{i(q+1)} = bias$.
- após a apresentação de todas as entradas, calcular os pesos da saída:

$$w = \left(G^T G\right)^{-1} G^T d$$

Exercícios:

1. Utilizando a Rede Neural RBF com centros u = {(0 2), (1 0)}, encontre os pesos desta rede para o problema de classificação dos pontos A_i e B_j dados abaixo:

	x_1	x_2	d
A_1	0	2	1
A_2	1	2	1
A_3	1	3	1
B_1	1	0	0
B_2	2	1	0



Use os parâmetros $\sigma = \sqrt{0,5}$, k = 2 (centros), θ = 1, saída com 1 neurônio.

2. Utilizando a Rede Neural RBF com 3 centros u = {(0 2), (1 0), (2 1)}, encontre os pesos desta rede para o problema de classificação dos pontos A_i e B_j do problema anterior.

3. Utilizando a Rede Neural RBF com 3 centros u = {-5, 3, 7}, encontre os pesos desta rede para aproximar a função $x^3 + 2x^2 + 4x - 12$ através de 21 pontos no intervalo [-3, 3].

4. Utilize uma rede RBF para a função "OU EXCLUSIVO" com os seguintes parâmetros: $\sigma = \sqrt{0,5}$, u = {(1 1), (-1 -1)}, k = 2 (centros), θ = 1, saída com 1 neurônio.

x			
1	-1	-1	1
1	1	-1	-1
d			
1	-1	1	-1

5. Utilizando a Rede Neural RBF com 4 centros, encontre os pesos da rede para aproximar a função sen(x)+ 4cos(x) – 1 através de 21 pontos no intervalo [-2, 4].

6. Utilize uma RBF para resolver o problema de classificação dos 22 pontos da página 19. Utilize σ = 5, u = {(1 3), (4 0), (5 3)}, k = 3 (centros), θ = 1, saída com 1 neurônio.

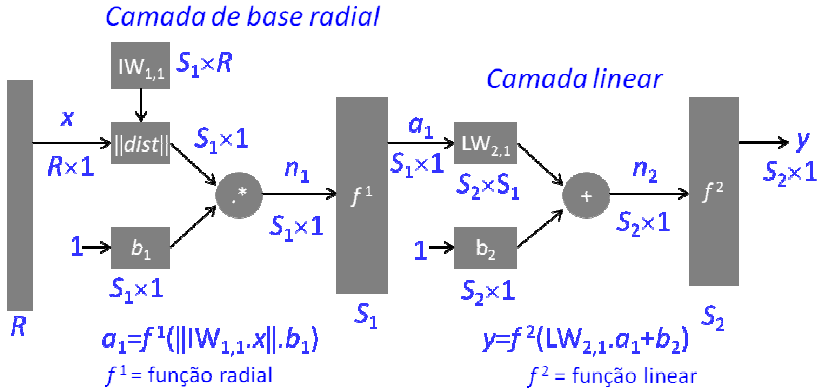
7. Utilize uma RBF com centros u = {(0 0), (1 1), (3 3)} para classificar os dados abaixo. Use σ = 5 e θ = 1.

	x_1	x_2	**d**
A_1	0	1	0
A_2	1	2	0
B_1	0	3	1
B_2	2	3	1
B_3	1	3	1

8. Resolva o problema 7 com saídas desejadas bipolares.

9. Resolva o problema de classificação de 22 pontos (exercício 4 da página 19) usando uma RBF com 3 centros.

Resumindo, a arquitetura de uma RBF é composta dos elementos mostrados na figura abaixo:



2.7. REDE NEURAL DE HEBB

Na rede Neural de Hebb (1949) a comunicação entre dois neurônios é facilitada pela iteração repetida em cada neurônio.

A Regra de Hebb diz que se a saída do i-ésimo processador é y_i e a ativação do j-ésimo processador é x_j, então

$$\Delta w_{ij} = \alpha x_j y_i$$

onde α é o tamanho do passo.

Para aplicar a regra de Hebb, somente os sinais de entrada precisam fluir através da rede. Esta rede é denominada local ao peso e pode ser modificada para treinamento não-supervisionado.

Algoritmo:

0. Inicialize os pesos ($w_i = 0$, onde $i = 1, 2, \ldots, n$)
   1. Para cada par de treinamento (x,d), faça:
   2. $w_i^{atual} = w_i^{anterior} + \alpha x_i d_i$ $(i = 1, 2, \ldots, n)$
      $\theta_i^{atual} = \theta_i^{anterior} + \alpha d_i$
   3. Faça $y^* = w_i x_i + \theta$.
4. Teste a convergência. Se necessário, repita os passos 1-3.

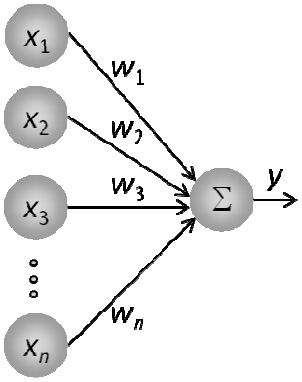
Em notação vetorial a saída do neurônio é

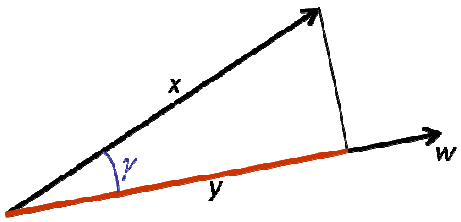
$$y^* = w^T x + \theta$$

Assumindo entradas e pesos normalizados, y maior significa que a entrada está mais "próxima" da direção do vetor peso.

Durante a aprendizagem os dados expostos aos pesos condensam toda informação nos valores dos pesos.

O processador Hebbiano é simples e cria uma medida de similaridade (produto interno) no espaço de entrada de acordo com a informação contida nos pesos. Ele programa um tipo de memória associativa.





Exercícios:

1. Use a rede de Hebb para a função lógica "E" com entradas bipolares.
2. Use a rede de Hebb para a função lógica "OU" com entradas bipolares

3. Use a rede de Hebb para classificar os dados do problema de 5 pontos:

	x_1	x_2
A_1	0	2
A_2	1	2
A_3	1	3
B_1	1	0
B_2	2	1

$$\hat{x}_j = 2\left(\frac{x_j - x_{min}}{x_{max} - x_{min}}\right) - 1 \Rightarrow$$

	x_1	x_2
A_1	-1,00	0,33
A_2	-0,33	0,33
A_3	-0,33	1,00
B_1	-0,33	-1,00
B_2	0,33	-0,33

4. Use a rede de Hebb para classificar os dados do problema de 22 pontos:

	x_1	x_2	d
1	0	1	1
2	0	2	1
3	1	1	1
4	1	2	1
5	1	3	1
6	2	2	1
7	2	3	1
8	3	2	1
9	4	1	1
10	4	3	1
11	2	0	-1
12	2	1	-1
13	3	0	-1
14	3	1	-1
15	3	3	-1
16	4	0	-1
17	4	2	-1
18	5	0	-1
19	5	1	-1
20	5	2	-1
21	5	3	-1
22	0	3	1

$$\hat{x}_j = 2\left(\frac{x_j - x_{min}}{x_{max} - x_{min}}\right) - 1 \Rightarrow$$

x_1	x_2	d

5. Utilizando o algoritmo da Rede de Hebb, encontre os pesos e bias para classificar os seguintes conjuntos de treinamento:

5.1.

x_1	x_2	x_3	d
1	1	1	1
1	1	0	0
1	0	1	0
0	1	1	0

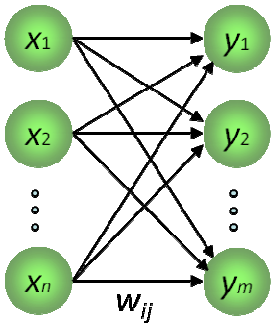
5.2.

x_1	x_2	x_3	d
1	1	1	1
1	1	0	-1
1	0	1	-1
0	1	1	-1

5.3

x_1	x_2	x_3	d
1	1	1	1
1	1	-1	-1
1	-1	1	-1
-1	1	1	-1

2.8. REDE NEURAL HETEROASSOCIATIVA



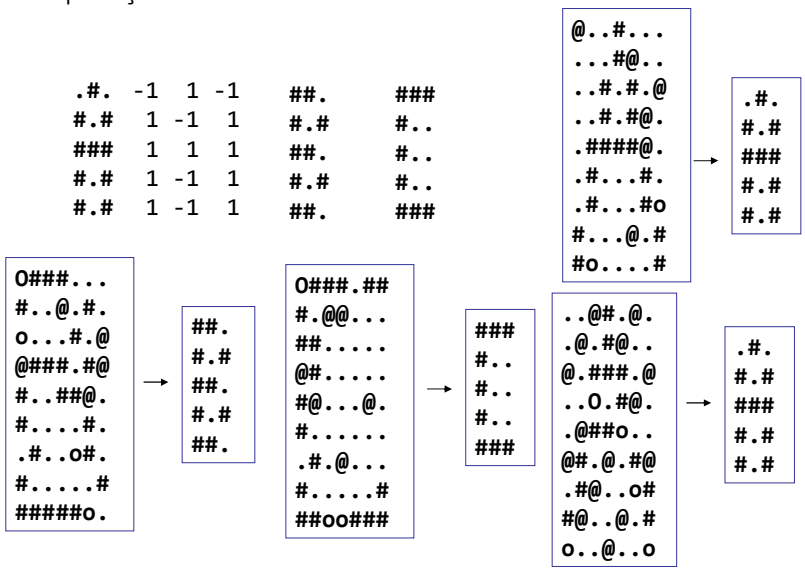
A rede neural heteroassociativa tem os pesos determinados para o armazenamento de P padrões $(x(p), y(p))$, onde os vetores x e y possuem dimensões n e m, respectivamente.

Ela tem a aprendizagem que usa a regra de Hebb.

Algoritmo:

0. Inicialize os pesos ($w_i = 0$, onde $i = 1, 2, \ldots, n$)
1. Para cada par de treinamento (x,d), faça os passos 2-4:
   2. $y_j^* = \Sigma_i x_i w_{ij}$
   3. Se $y_j^* > 0$, $y_j = 1$
      Se $y_j^* = 0$, $y_j = 0$
      Se $y_j^* < 0$, $y_j = -1$
   4. $w_{ij}^{atual} = w_{ij}^{anterior} + \alpha x_i d_j$ $(i = 1, 2, \ldots, n)$
5. Reduza α e teste a convergência. Se necessário, repita os passos de 1-4.

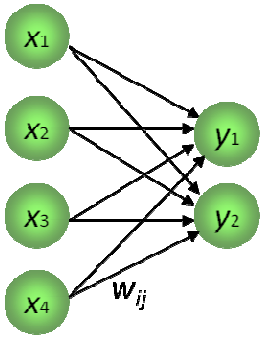
Um exemplo de aplicação desta RNA está no reconhecimento de caracteres:



Exercícios:

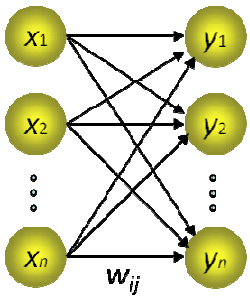
1. Dados dos vetores x e y treine a rede para associar a cada vetor x o respectivo vetor y:

x	d
(1,0,0,0)	(1,0)
(1,1,0,0)	(1,0)
(0,0,0,1)	(0,1)
(0,0,1,1)	(0,1)



2. Resolva o exercício 1 com dados bipolares.

2.9. REDE NEURAL AUTOASSOCIATIVA



A RNA autoassociativa tem os pesos determinados para reconhecer os dados de um conjunto de padrões $x(p)$, e cuja saída consiste no reconhecimento dos próprios vetores de entrada.

O algoritmo é o mesmo da RNA heteroassociativa, com a saída y = x.

É utilizada para reconhecer erros nos vetores de entrada, como falta de dados, ou coordenadas erradas.

A matriz de pesos pode ser iniciada com coordenadas de alguns vetores de entrada para acelerar a convergência.

Exercícios:

1. Utilizando uma rede autoassociativa, calcule a matriz W de pesos que seja capaz de reconhecer os vetores dados abaixo:

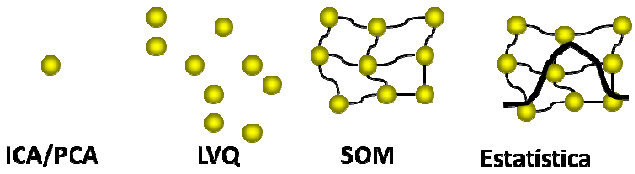
   (1 1 -1 -1)
   (1 -1 1 1)

   Usando a matriz de pesos W, verifique se os vetores acima são mesmo reconhecidos corretamente pela rede.
   O vetor (-1 1 1 -1) é reconhecido corretamente pela rede?

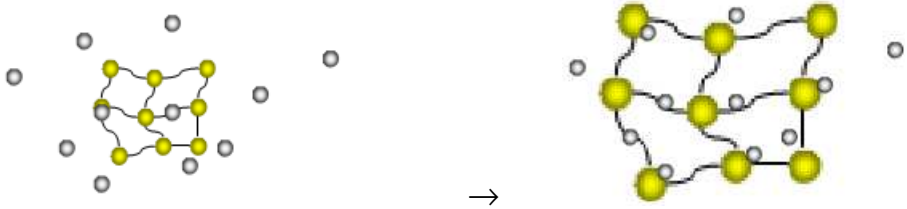
2. Use uma rede autoassociativa para reconhecer o vetor V_1=(1,1,1,1,1,1). Teste a rede com o vetor V_1. Teste a rede com o vetor T=(1,1,1,1,-1,-1). Encontre a rede que reconhece V_2=(1,1,1,-1,-1,-1). Teste a rede com V_1, V_2 e V_3=(1,1,1,-1,0,0)

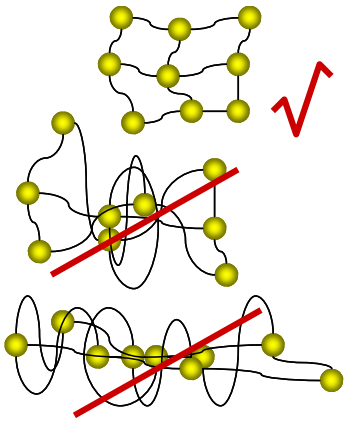
2.10. MAPAS AUTO-ORGANIZÁVEIS

Os principais tipos de RNA com mapas auto-organizáveis são: PCA – Principal Component Analysis, ICA – Independent Component Analysis, LVQ – Learning Vector Quantization, SOM – Self Organizing Maps e redes com mapas e ativações estatísticas.



Treinamento de uma rede auto-organizável:





São redes chamadas de competitivas. Existe uma competição por recursos como maneira de diversificar e otimizar a função dos elementos de um sistema distribuído, além de conduzir à otimização a nível local sem controle global para assinalar recursos do sistema.

Os neurônios de redes competitivas recebem informação idêntica das entradas mas competem pelos recursos:

- através de conexões laterais na topologia; ou
- pela formulação da regra de aprendizagem.

Os neurônios especializam-se em áreas diferentes da entrada. Trata-se de uma operação não-linear, sem um tratamento matemático muito desenvolvido como em outras redes.

Existem dois tipos principais:

- Hard: somente um neurônio ganha os recursos;
- Soft: há um vencedor, mas seus vizinhos também recebem uma parte dos recursos.

O objetivo da aprendizagem competitiva é criar uma regra de aprendizagem que possa ser aplicada a uma topologia de camada única e que atribua os neurônios a áreas diferentes do espaço de entrada.

A aprendizagem competitiva é um paradigma de aprendizagem não-supervisionada, a qual extrai informação somente dos padrões de entrada sem a necessidade de uma resposta desejada.

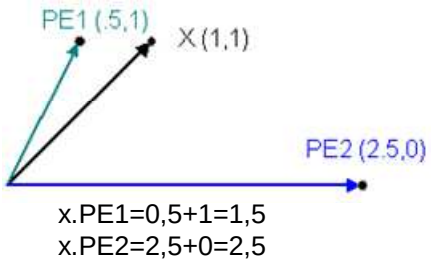
Critérios de competição:

O neurônio mais próximo à entrada atual deve vencer a competição, portanto precisa-se de uma medida de proximidade.

O produto interno é sensível não somente às direções mas também ao comprimento dos vetores, isto é, a entrada e os pesos devem ser normalizados.

Uma alternativa ao produto interno é o uso da distância Euclidiana como métrica para definir o vencedor:

$$\text{vencedor} = \max_i \left(\mathbf{w}_i^T \mathbf{x} \right) \quad \text{ou} \quad \text{vencedor} = \min_i \left(\left| \mathbf{x} - \mathbf{w}_i \right|^2 \right).$$



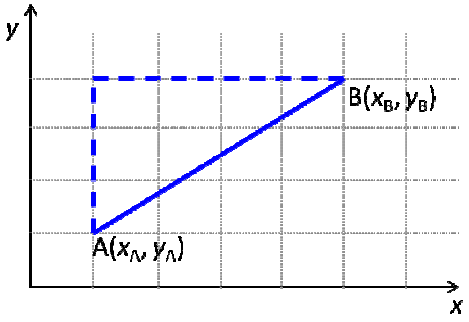
Como a raiz quadrada é uma função computacionalmente cara, a distância métrica é menos eficiente que o produto interno:

$$\left\| \mathbf{x} - \mathbf{w} \right\| = \sqrt{\sum_k \left(x_k - w_k \right)^2} .$$

Às vezes a métrica retangular (conhecida também como métrica de Manhattan) é usada, pois só envolve subtrações e valores absolutos:

Distância Euclidiana: $dE_{AB} = \sqrt{\left(x_A - x_B \right)^2 + \left(y_A - y_B \right)^2}$.

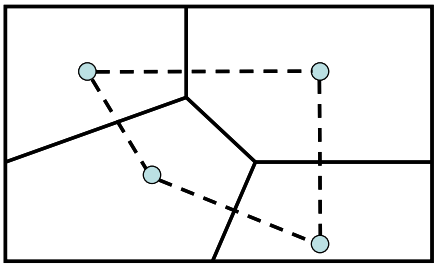
Distância Retangular: $dR_{AB} = \left[\left| x_A - x_B \right| + \left| y_A - y_B \right| \right]$.



Agrupamento

Usando a regra competitiva uma rede linear de camada única agrupa e representa dados que residem em uma vizinhança do espaço de entrada. Cada vizinhança é representada por um único neurônio.

Os pesos de cada neurônio representam pontos no espaço de entrada chamados *vetores protótipos*. Se os vetores forem unidos por uma linha e forem traçadas perpendiculares na metade de cada uma, as mesmas se encontrarão e formarão uma estrutura semelhante a uma colméia de abelhas.



A Tesselação de Voronoi é um exemplo de formação de agrupamentos. As amostras de dados que estão nas regiões são assinaladas aos correspondentes vetores-protótipos.

O algoritmo não-neural típico de agrupamento é o k-médio, o qual encontra a melhor divisão de *N* amostras em *K* grupos, tal que a distância total entre as amostras agrupadas e seus respectivos centros, isto é, a variância total, seja minimizada.

Redes competitivas implementam uma espécie de versão do agrupamento k-médio

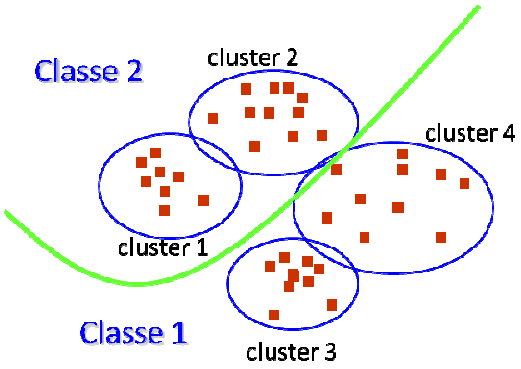
Agrupamento:

- é o processo de agrupar amostras de entradas que são vizinhas espaciais;
- é um processo não-supervisionado.

Classificação:

- consiste na rotulação de amostras de entrada através de algum critério externo;
- é um processo supervisionado.

Como agrupamento é não-supervisionado, ele não pode ser usado diretamente para classificação. Em várias aplicações práticas, os dados de cada classe tendem a ser densos e, portanto, há um vale natural entre as classes. Nestes casos o agrupamento pode ser um pré-processador para a classificação. Com isto obtém-se redes de classificação mais simples.



Se o vetor de pesos de um neurônio está muito distante dos grupos de dados ele nunca vencerá a competição (neurônio morto).

Consciência: penaliza um neurônio se ele vence em demasia ou ajuda-o em caso contrário.

O chamado “mapa de densidade” identifica os neurônios que mais vencem, e auxilia na escolha dos protótipos mais significativos (centróides dos agrupamentos).

O número de centros é controlado pelo número de neurônios de saída:

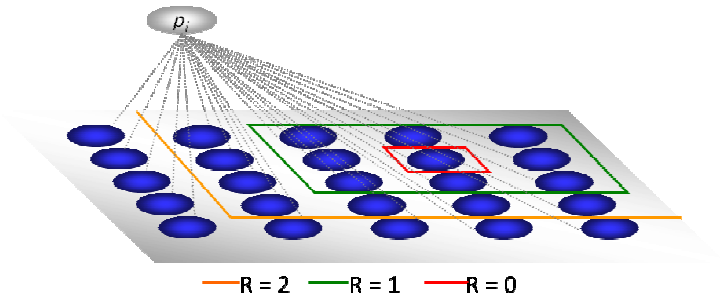
- Se o número de neurônios é menor que o número de grupos reais, cada neurônio representa mais de um grupo e é colocado no centro de massa destes;
- Se o número de neurônios é maior que o número de grupos reais, alguns neurônios podem “congelar” ou mais de um representar o mesmo grupo.

O princípio da competição é diferencial por natureza, ou seja: o neurônio que ganhou a competição é confiável?

O agrupamento cria uma “bolha” de atividade no espaço de saída, onde o neurônio mais próximo é o mais ativo e seus vizinhos são menos ativos.

A rede SOM (*Self Organizing Map)* de Kohonen realiza um mapeamento de um espaço contínuo de entrada para um espaço discreto de saída, onde as propriedades topológicas da entrada são preservadas.

A rede SOM de Kohonen é uma rede linear de camada única totalmente conectada, cuja saída é organizada em uma, duas ou três dimensões.

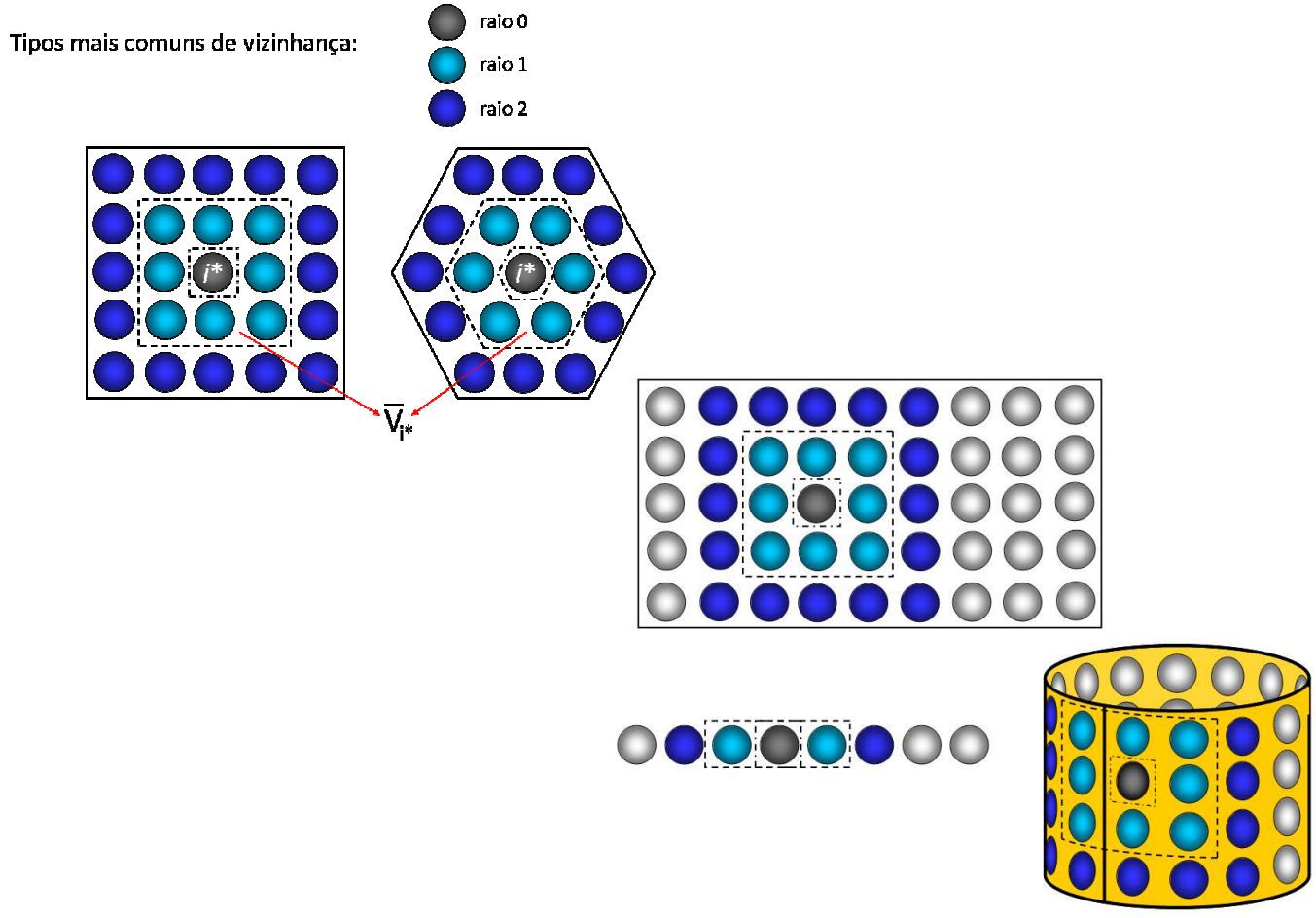


Quando a SOM se adapta a entradas de altas dimensões, ela deve se estender e enrolar para cobrir o espaço de entrada.

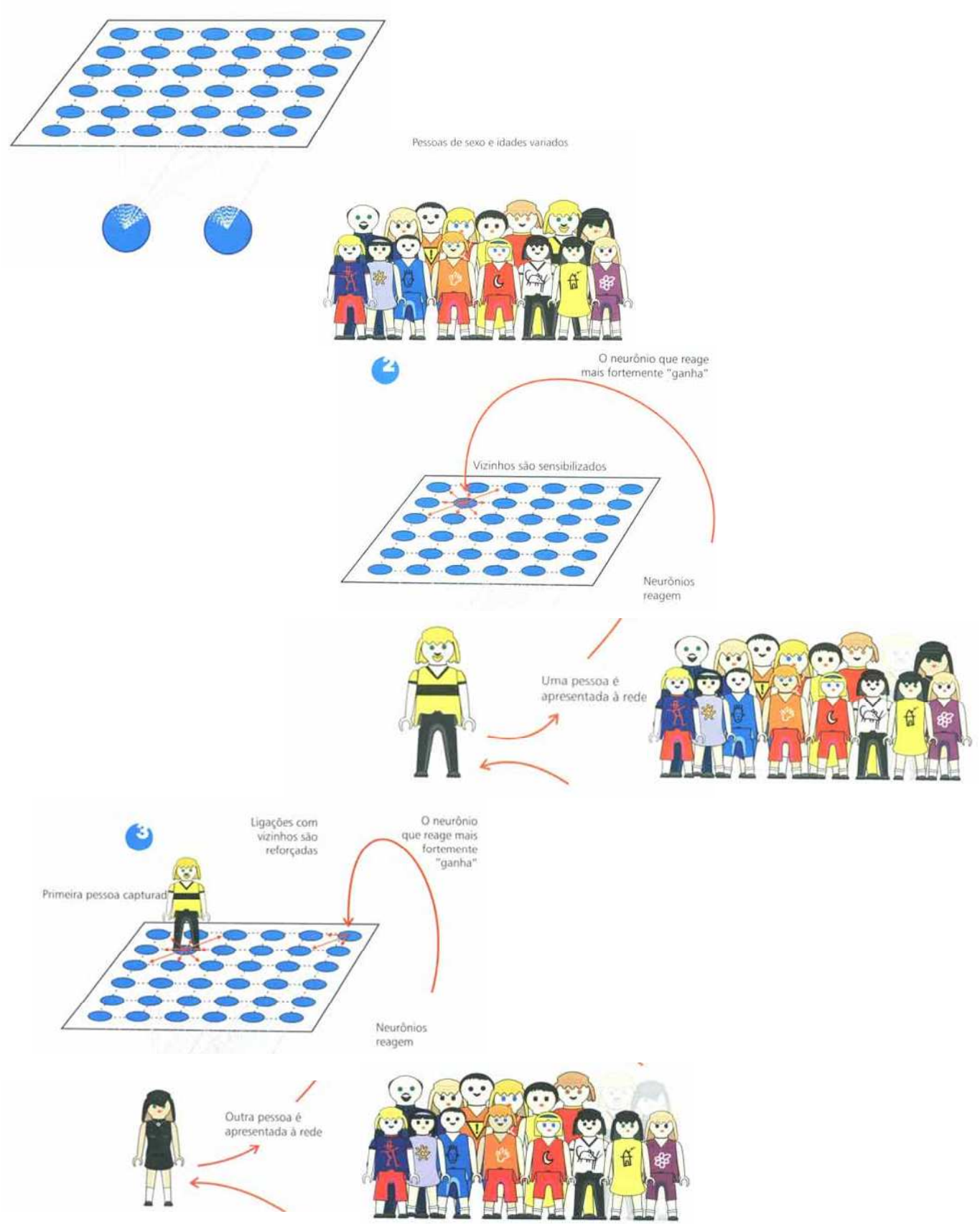
O neurônio vencedor e seus vizinhos são atualizados a cada apresentação de um padrão.

O processo de aprendizado de uma rede de Kohonen consiste em reforçar as ligações que produzem respostas mais eficientes.

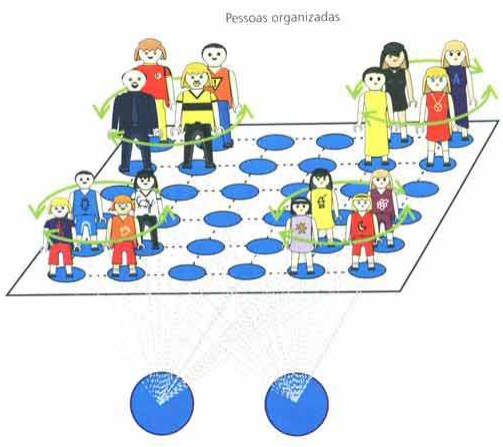
Vizinhanças



Exemplo:


Pessoas de sexo e idades variados
O neurônio que reage mais fortemente "ganha"
Vizinhos são sensibilizados
Neurônios reagem
Uma pessoa é apresentada à rede
Ligações com vizinhos são reforçadas
O neurônio que reage mais fortemente "ganha"
Primeira pessoa capturada
Neurônios reagem
Outra pessoa é apresentada à rede

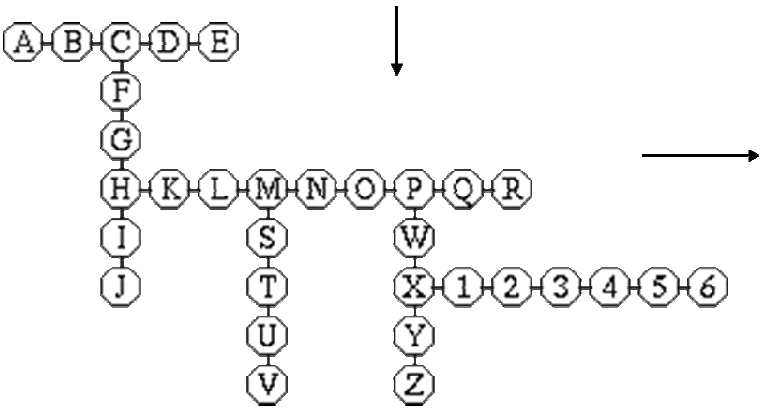
Após o treinamento da rede:



Em um dos primeiros trabalhos de Kohonen sobre as SOMs, o seguinte exemplo foi apresentado:

Considere os 32 vetores abaixo, que representam agrupamentos rotulados pelas letras A-Z e números 1-6:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6



E	D	C	B				6
		F		A		5	
H		G					4
I			K		2		3
J			L		1		
			M		X	Y	Z
U	T	S			W		
V			N	O	P	Q	R

Algoritmo

0. Iniciar os pesos dos n neurônios da rede com valores aleatórios baixos: w_{ij}
1. Apresentar cada entrada x para a rede, e executar os passos 2 e 3:
   2. Determinar o neurônio i que possui a menor distância (euclidiana) do peso sináptico w_j com o vetor x.

$$d_i = \sum_{j=1}^{n} \left(x_j - w_{ij}\right)^2$$

   Este neurônio é denominado "vencedor".

   3. Ajustar os pesos do neurônio vencedor e de todos os neurônios que pertencem a uma vizinhança centrada nele, V_i.

$$w_{ij}^{atual} = w_{ij}^{anterior} + \alpha[x_j - w_{ij}^{anterior}]$$

   onde $i \in V_i$.
4. Ajustar a taxa de aprendizado α e o raio de vizinhança. Se não existirem mais mudanças substanciais no mapa, pare; caso contrário, volte ao passo 1.

Exercícios:

1. Utilizando o conjunto de treinamento *X* e a matriz inicial de pesos *W* dados abaixo, treine a SOM com 3 iterações completas, e depois apresente o conjunto de testes. Interprete geometricamente os resultados de cada iteração.

   Conjunto de treinamento *X*:
   A(-1,5 2,5) B(-2 -2) C(2 2) D(1,5 2,5) E(-2 3) F(-2,5 3) G(-3 -2)
   Conjunto de teste:
   H(0 0) I(10 0) J(2 -7) K(2 3) L(-2 -2)

   taxa de aprendizagem inicial: $\alpha = 0{,}5$

   matriz inicial de pesos: $W^T = \begin{pmatrix} 0{,}1 & -0{,}1 \\ -0{,}1 & 0{,}1 \\ 0 & 0 \\ 0{,}1 & 0{,}1 \end{pmatrix}$

2. Utilizando o exemplo dado anteriormente, faça 4 iterações completas utilizando as coordenadas de A, F, K, R, S, 1 e 2 para treinar um mapa auto-organizável de dimensão 2 x 3. Depois, utilize os dados de V, 6, C e L para testar a rede.

   Conjunto de treinamento:

A	F	K	R	S	1	2
1	3	3	3	3	3	3
0	1	3	3	3	3	3
0	0	1	8	3	6	6
0	0	0	0	1	2	2
0	0	0	0	0	1	2

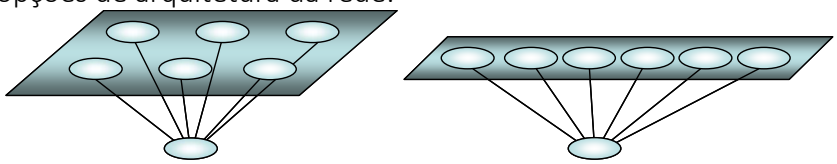
   Conjunto de teste:

V	6	C	L
3	3	3	3
3	3	0	3
3	6	0	2
4	2	0	0
0	6	0	0

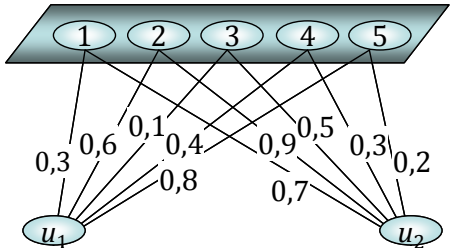
   taxa de aprendizagem inicial: $\alpha = 0{,}5$

matriz inicial de pesos: $W^T = \begin{pmatrix} 0,1 & 0,2 & 0,3 & 0,4 & 0,5 \\ 0,5 & 0,4 & 0,3 & 0,2 & 0,1 \\ 0,3 & 0,1 & 0,5 & 0,2 & 0,4 \\ 0,2 & 0,5 & 0,3 & 0,4 & 0,1 \\ 0,1 & 0,4 & 0,3 & 0,5 & 0,2 \\ 0,5 & 0,3 & 0,2 & 0,1 & 0,4 \end{pmatrix}$

opções de arquitetura da rede:



3. Considere o mapa de Kohonen unidimensional dado.
3.1. Utilizando a distância Euclidiana como métrica, encontre o neurônio vencedor para o padrão de entrada (0,5 0,2);
3.2. Utilize a taxa de aprendizagem 0,2 e raio de vizinhança 1 para encontrar os novos pesos para os neurônios da rede;
3.3. Faça o mesmo para o padrão de entrada (0,5 0,5).



4. Considere o mapa de Kohonen com 2 neurônios e 5 entradas com a matriz de pesos *W*.

$$W^T = \begin{pmatrix} 1 & 0,2 \\ 0,8 & 0,4 \\ 0,6 & 0,6 \\ 0,4 & 0,8 \\ 0,2 & 1 \end{pmatrix}$$

4.1. Use a distância Euclidiana como métrica para encontrar o neurônio vencedor para o padrão (0,5 1 0,5 0 0).;
4.2. Use a taxa de aprendizagem 0,2 com raio de vizinhança 0 para encontrar os novos pesos para os neurônios da rede.

A Rede de Kohonen tem inibição lateral produz uma distribuição gaussiana centrada no neurônio vencedor. Como aplica-se a regra de aprendizagem do tipo instar, que escala a regra competitiva pela atividade de saída de cada neurônio, a regra competitiva SOM de Kohonen torna-se

$$w_i^{atual} = w_i^{anterior} + \Lambda_{ii*}\alpha\left[x - w_i^{anterior}\right]$$

onde a função Λ_{ii*} é uma *função de vizinhança c*entrada no neurônio vencedor

O tamanho do passo e a vizinhança diminuem com o tempo. A função de vizinhança Λ é em geral uma gaussiana:

$$\Lambda_{ii*} = \exp\left(\frac{-d_{ii*}^2}{2R^2}\right),$$

com uma variância (raio) que decresce com a iteração. Inicialmente ela cobre todo o mapa, mas reduz-se progressivamente a uma vizinhança de zero, isto é, somente o neurônio vencedor é atualizado.

Outro tipo de vizinhança que pode ser usada é chamada de vizinhança discreta:

$$\Lambda_{ii^*} = \begin{cases} 1, \text{ se } i \in V_{i^*} \\ 0, \text{ caso contrário} \end{cases}$$

Conforme a vizinhança é reduzida, a rede move-se de uma competição muito soft (quase todo neurônio é atualizado) para uma competição hard (somente o vencedor é atualizado).

Há evidências que a SOM cria um espaço de saída discreto onde relações topológicas dentro das vizinhanças do espaço de entrada são preservadas. A rede SOM é criada de uma maneira não-supervisionada.

A seleção de parâmetros é crucial para a preservação de topologia.

Existem duas fases na aprendizagem SOM:

- Fase de ordenação topológica dos pesos, ou seja, definição das vizinhanças;
- Fase de convergência com o ajuste fino da distribuição de entrada.

Com *t* iterações, a função de vizinhança decresce, em geral, com um raio definido por:

$$R = R_0\left(1 - \frac{t}{R_0}\right).$$

Normalmente a taxa de aprendizagem é alta (acima de 0,1) para permitir à rede se auto-organizar. Ela é ajustada da seguinte forma:

$$\alpha = \alpha_0 \exp\left(-t/T\right)$$

onde α_0 é a taxa de aprendizagem inicial e *T* é uma aproximação do número total de iterações (segundo HAYKIN, geralmente *T* =1000).

A fase de convergência é a mais demorada, onde se mantém uma taxa de aprendizagem pequena (0,01) e usa-se a menor vizinhança (somente o neurônio ou seus vizinhos mais próximos).

A escolha do número de neurônios é feita experimentalmente. O número de saídas afeta a precisão do mapeamento e o tempo de treinamento. O aumento do número de neurônios aumenta a resolução, mas aumenta em muito o tempo de treinamento.

Aproximação do Espaço de Entrada: A SOM é capaz de preservar a estrutura do espaço de entrada relativamente bem.

Ordenamento Topológico: Os neurônios na saída da SOM estão topologicamente ordenados no sentido de que neurônios vizinhos correspondem a regiões similares no espaço de entrada.

Manutenção da densidade: Regiões no espaço de entrada com maior densidade de pontos são mapeadas para regiões maiores no espaço de saída.

Os cálculos mais comuns dos erros em SOM são:

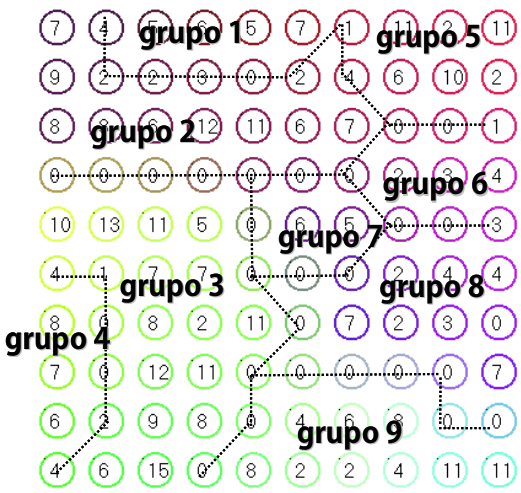
- Erro de Quantização: $E_Q = \frac{1}{n}\sum_{k=1}^{n}\left\|x_k - w^*\right\|$

- Erro Médio Quadrático: $E_{MQ} = \frac{1}{n}\sum_{k=1}^{n}\left\|x_k - w^*\right\|^2$
- Erro Topológico: $E_T = \frac{1}{n}\sum_{k=1}^{n} u_k$, onde $u_k = \begin{cases} 1, \text{ se } i^{**} \notin \bar{V}_{i^*} \\ 0, \text{ caso contrário} \end{cases}$

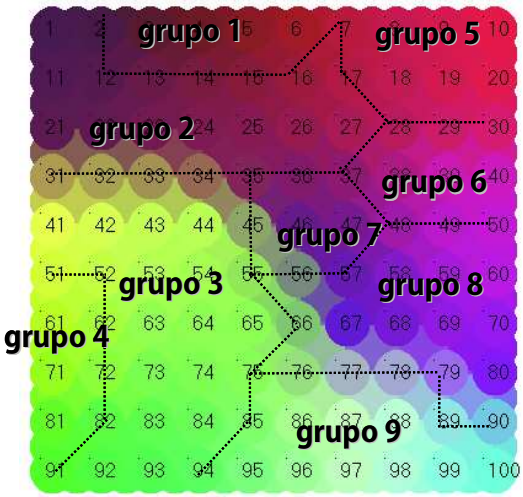
Visualização de agrupamentos

Existem duas formas de visualização de resultados de agrupamentos das RNA do tipo SOM:

Mapa de densidade



U-Matriz (RGB)

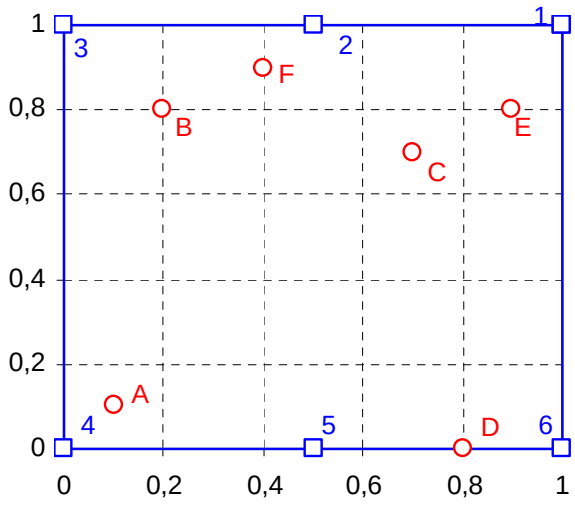


Exercícios:

1. Considere o conjunto de cidades A, B, C, D, E e F, e o conjunto de pesos iniciais dos neurônios 1, 2, 3, 4, 5 e 6. Use 2 iterações de uma rede tipo SOM para aproximar uma solução do PCV (problema do Caixeiro Viajante) usando as coordenadas abaixo:

pesos	x	y
1	1	1
2	0,5	1
3	0	1
4	0	0
5	0,5	0
6	1	0

cidades	x	y
A	0,1	0,1
B	0,2	0,8
C	0,7	0,7
D	0,8	0
E	0,9	0,8
F	0,4	0,9



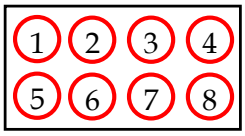
2. Considere o mapa de Kohonen bidimensional dado abaixo, com a matriz de pesos *W*.

$$W^T = \begin{pmatrix} 0,8 & 0,7 & 0,1 & -0,2 & -0,9 & 0 & 0 & 0,5 \\ 0,6 & 0,7 & 0,5 & 0,7 & -0,7 & 0,6 & 0,5 & 0,1 \\ 0 & -1 & -0,7 & 0,5 & 0,7 & -0,1 & -0,2 & -0,9 \end{pmatrix}$$

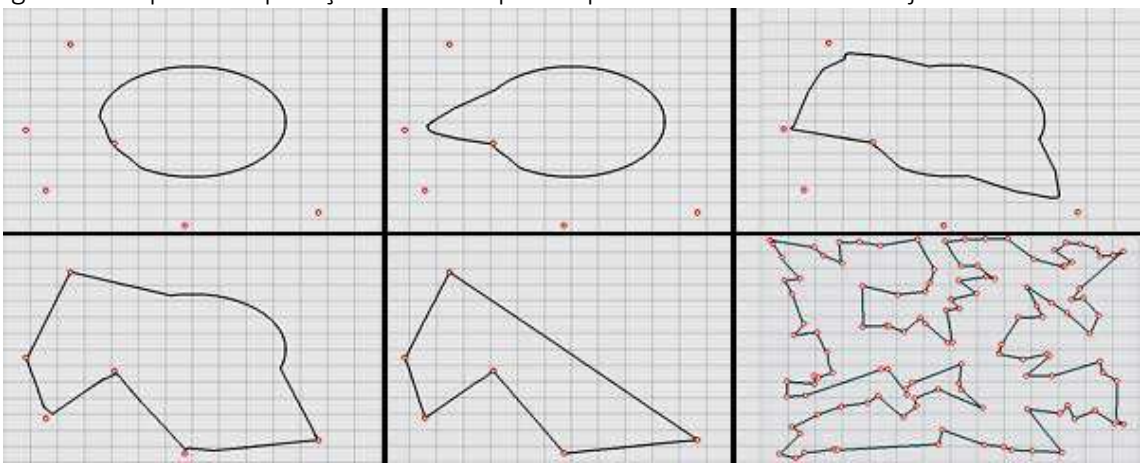
2.1. Utilizando a distância Euclidiana como métrica, encontre o neurônio vencedor para o padrão de entrada (0,7 0,4 0,1);

2.2. Utilize a taxa de aprendizagem 0,2, raio de vizinhança 1, e a função gaussiana de vizinhança para encontrar os novos pesos para os neurônios da rede;

2.3. Faça o mesmo para o padrão de entrada (0,1 -0,4 0,9).



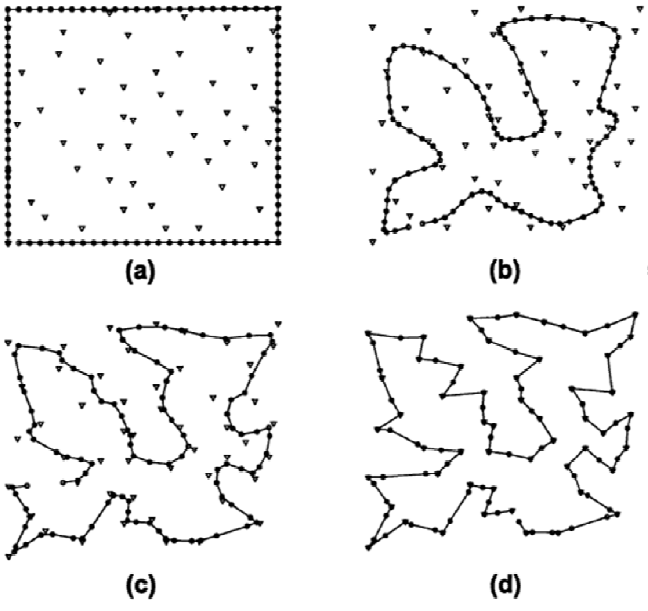
Alguns exemplos de aplicações da SOM para o problema do Caixeiro Viajante:



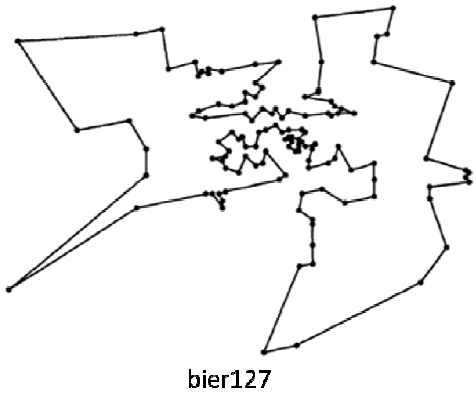
Problema rd100 – TSPLib

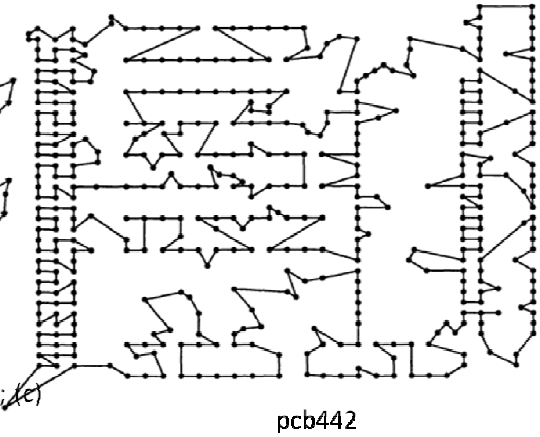
An Efficient Approach to the Travelling Salesman Problem Using Self-Organizing Maps

[Vieira, Dória Neto, Costa, 2003]



rd100 – soluções parciais: (a) inicial; (b) após 1 iteração; (c) Após 8 iterações; (d) após 16 iterações

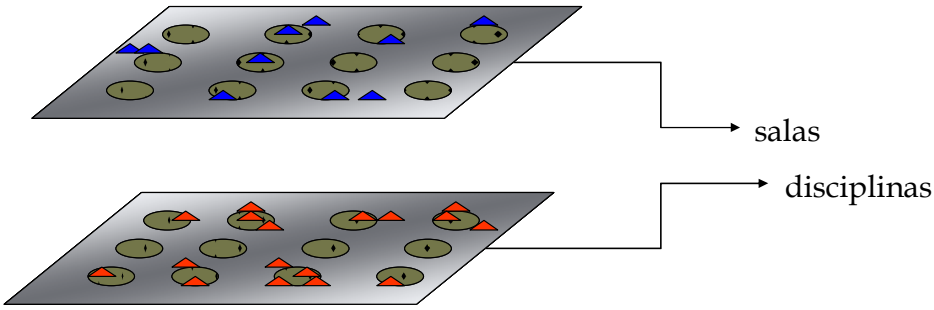




Os padrões de entrada são as coordenadas das cidades.
Os pesos são aleatórios no início, que se aproximam aos valores das coordenadas das cidades.
O número de neurônios deve ser maior do que o número de cidades.

Aplicação de uma SOM para calcular custos para designação: problema do Ensalamento

Os cálculos dos custos para executar a designação podem ser feitos através da classificação das salas e das disciplinas com Redes de Kohonen.



A partir de um arquivo com "modelos" de sala de aula, a Rede de Kohonen classifica tais modelos e cria uma matriz de pesos com as características selecionadas: capacidade, tipo de carteira e bloco.

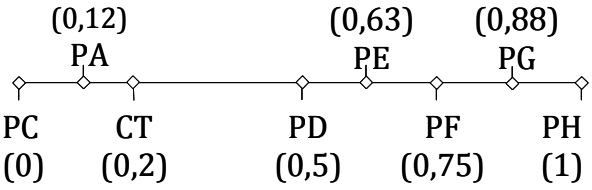
Alunos	cart	Bloco	nBloco	class	erro
90	Q	PH	Q	90	0,0086129
70	D	PH	Q	213	0,0031832
40	D	PH	Q	217	0,0030573
50	Q	PH	Q	45	0,0029618
60	Q	PH	Q	60	0,0029451
70	Q	PH	Q	75	0,0029362
90	D	PH	Q	211	0,0023920
60	D	PH	Q	215	0,0020528
70	D	PG	Q	213	0,0019005
40	D	PG	Q	216	0,0018594
70	Q	PG	Q	74	0,0017608
50	Q	PG	Q	30	0,0017114

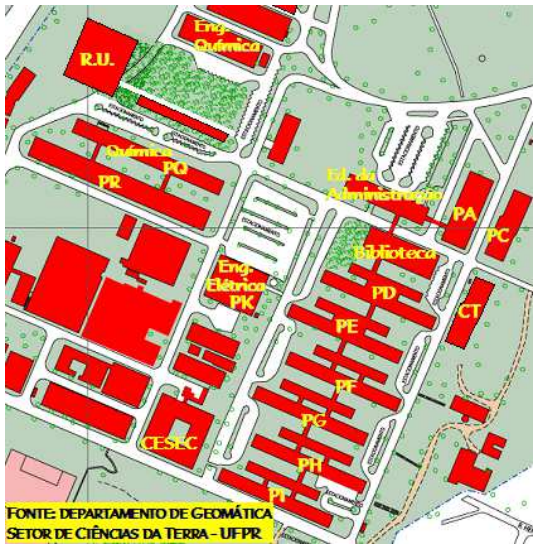
Número de "modelos" para o treinamento: 72

Matriz de características:

- tipo de carteira:
  - 0,1 (tipoc = "Q");
  - 1 (tipoc = "D")
- capacidade: capac = capac / $capac_{max}$

A terceira coordenada dos padrões de entrada foi determinada de acordo com a localização de cada bloco.



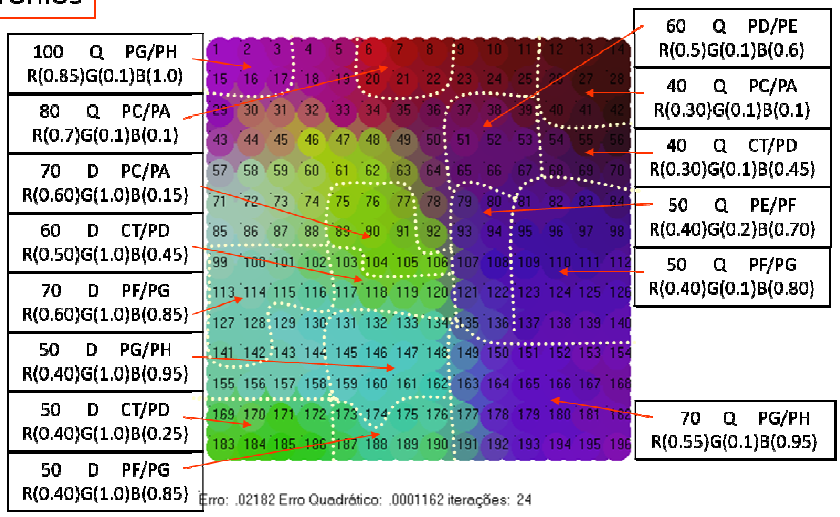


Dados utilizados para o Estudo de Caso: disciplinas de Graduação e Pós-Graduação dos Setores de Ciências Exatas, de Ciências da Terra e de Tecnologia da UFPR (650), e 46 salas de aula.

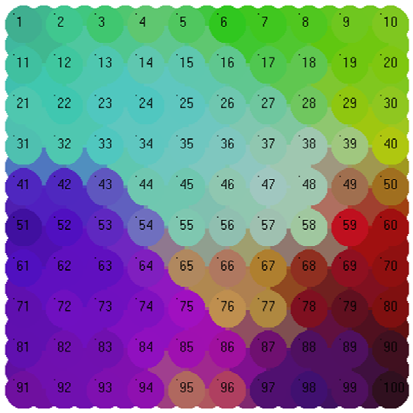
Os padrões de entrada para o treinamento da rede representam 2/3 do número total de disciplinas, ou seja, 433 padrões.

Seguem abaixo as U-Matrizes de algumas simulações para este problema:

196 neurônios

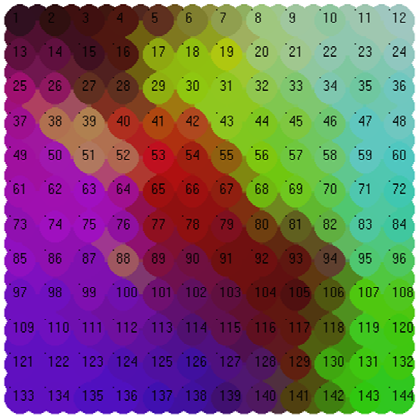


100 neurônios



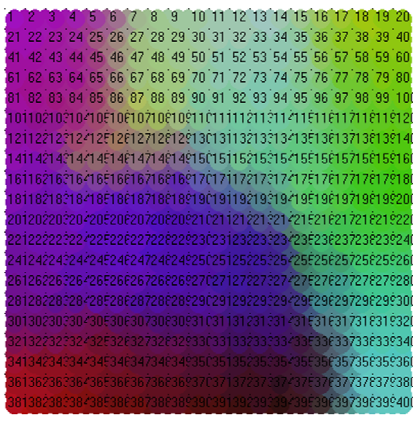
Erro: .15067 Erro Quadrático: .0027821 iterações: 24

144 neurônios



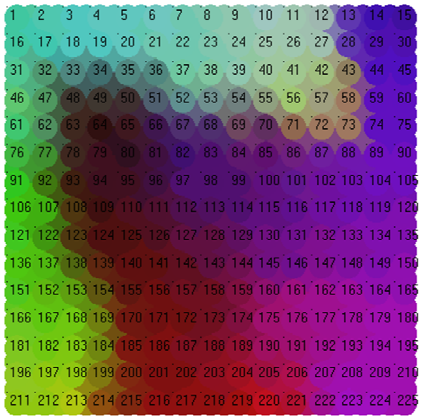
Erro: .05859 Erro Quadrático: .000502 iterações: 24

400 neurônios

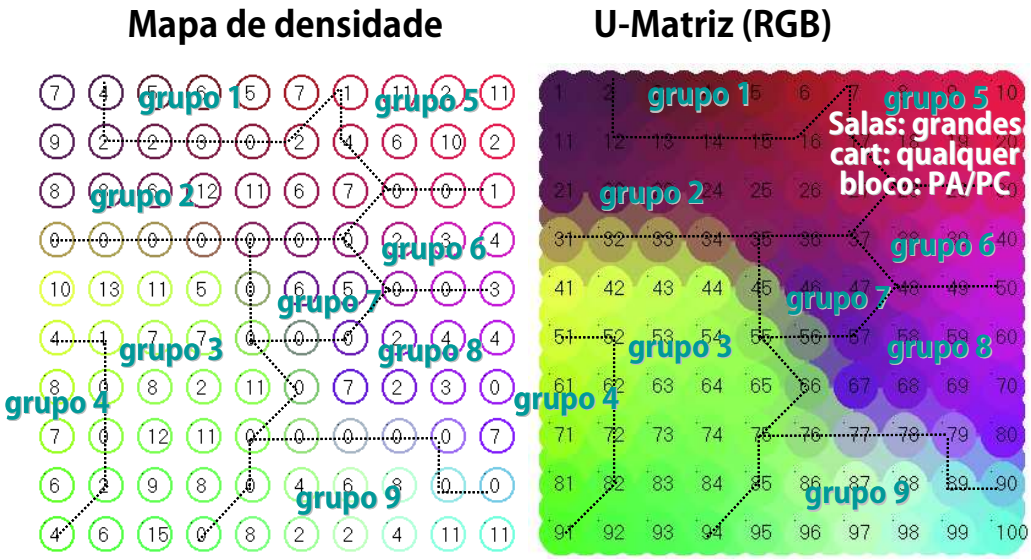


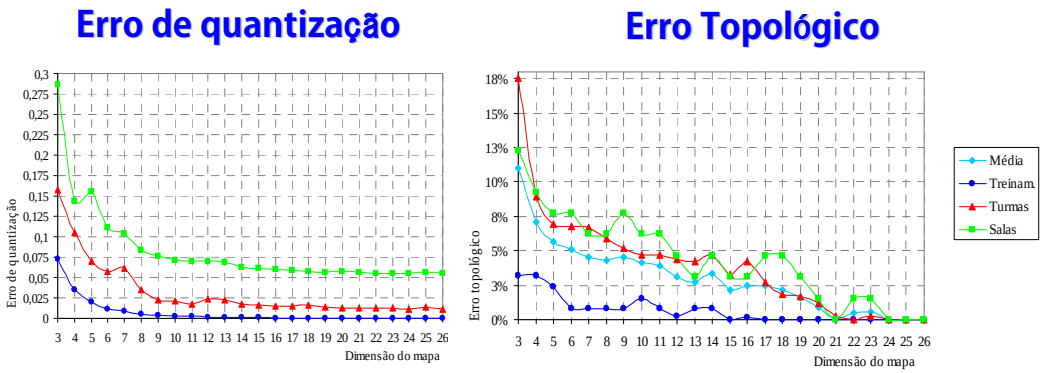
Erro: .08419 Erro Quadrático: .0010203 iterações: 24

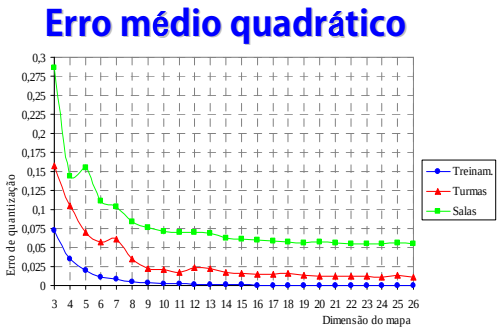
225 neurônios



Erro: .05404 Erro Quadrático: .0005096 iterações: 24







2.11. LEARNING VECTOR QUANTIZATION (LVQ)

A rede do tipo Learning Vector Quantization utiliza informações de classes para ajustar os limites das regiões de classificação.

Se a regra competitiva produz a saída certa, não há alteração; se a saída estiver errada, os pesos dos neurônios devem ser "repelidos" do agrupamento atual, pela regra:

$$\Delta w_{i^*j} = \begin{cases} \alpha\left(x_j - w_{i^*j}\right), & \text{para classe correta} \\ -\alpha\left(x_j - w_{i^*j}\right), & \text{para classe incorreta} \end{cases}$$

A arquitetura idêntica ao SOM, com exceção da representação topológica dos mapas. É um tipo de RNA supervisionada.

Exercícios:

1. Utilize os vetores iniciais *w* para classificar os vetores *x* em 2 classes:
   w_1 = (1, 1, 0, 0) classe1
   w_2 = (0, 0, 1, 1) classe 2
   w_3 = (0, 0, 1, 0) classe 2
   x_1 = (0, 0, 1, 1) classe 2
   x_2 = (1, 0, 0, 0) classe 1
   x_3 = (0, 1, 1, 0) classe 2
   x_4 = (1, 1, 1, 0) classe 1
   Use a taxa de aprendizagem inicial $\alpha(0) = 0{,}1$

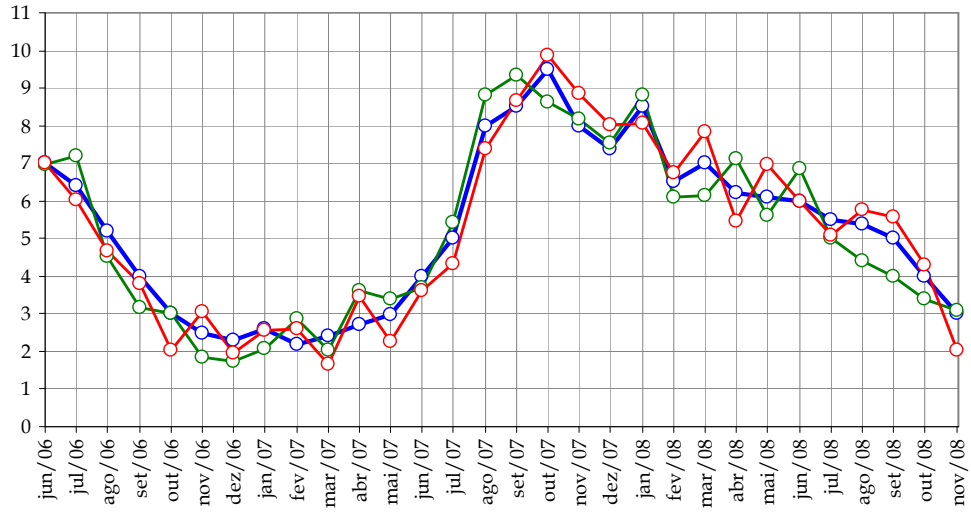
2. Considere o LVQ com as 4 classes 'mapeadas' abaixo, onde a posição de uma classe depende dos valores de x_1 e x_2.

x_2							
1							
0,8		C_3	C_4	C_1	C_2		
0,6		C_1	C_2	C_3	C_4		
0,4		C_3	C_4	C_1	C_2		
0,2		C_1	C_2	C_3	C_4		
0							
	0	0,2	0,4	0,6	0,8	1	x_1

   Utilize a distância Euclidiana como métrica e a posição de cada classe no 'mapa' ao lado para determinar novas classes e atualizar as existentes.
   2.1. Apresente o vetor (0,25 0,25) com classe C_1.
   2.2. Atualize o LVQ com taxa de aprendizagem α=0.5, e determine as mudanças no 'mapa' (pesos);
   2.3. Faça o mesmo com os vetores:
   (0,4 0,35) e (0,4 0,45), ambos pertencentes à classe C_1;
   (0,6 0,65) e (0,75 0,8), ambos pertencentes à classe C_4.

2.12. REDES NEURAIS TEMPORAIS

As simulações de Séries Temporais através das Redes Neurais Artificiais envolvem os seguintes elementos: Períodos de amostragem, Conjuntos de teste e treinamento, Validação da rede e Metodologias de aprendizado.



A maior dificuldade é determinar o número ideal de informações para um bom aprendizado da rede, onde informações redundantes dificultam a aprendizagem e, por outro lado, poucas informações garantem aprendizagem fraca.

As principais Redes Neurais para resolver problemas de Séries Temporais são:

- MLP (Perceptron com múltiplas camadas)
- Redes estáticas (Adaline, Hebb)
- Redes RBF
- Redes dinâmicas
- Redes recorrentes
- Redes competitivas

Uma alternativa inicial é a aproximação por memória finita através de mapas, onde encontram-se classes de mapas que aproximam as séries com estruturas simples e não-lineares, geralmente com funções sigmoidais e gaussianas.

As aproximações de RNA para Séries Temporais envolvem 2 estágios:

- Pré-processamento com implementação de memória nas estruturas;
- Aprendizagem com exemplos.

Podem ser construídas redes com "filtros de memória" conectados aos neurônios com aprendizagem com retropropagação do erro.

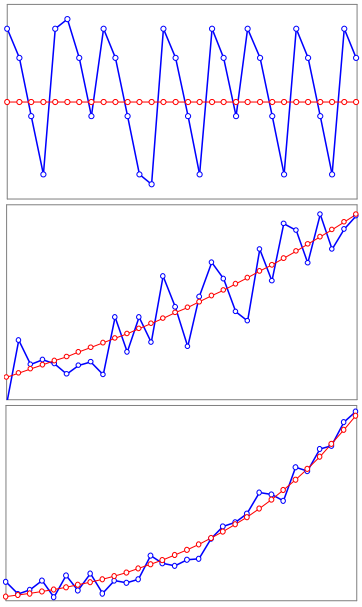
Como inserir a variável ***tempo*** na estrutura operacional de uma rede neural artificial?

- Representação implícita: sinal colocado na estrutura da RNA (na entrada, por exemplo);
- Representação explícita: tempo colocado como vetor na estrutura.

As redes estáticas podem resolver problemas de séries temporais. Uma rede é considerada dinâmica quando possui memória.

Para variáveis temporais, a RNA deve possuir memória de curto prazo para funcionar adequadamente, onde esta memória pode ser considerada como atrasos de tempo colocados na entrada da rede.

Alguns efeitos que as RNAs acompanham com dados de uma série temporal são:



Sem efeitos tendenciosos

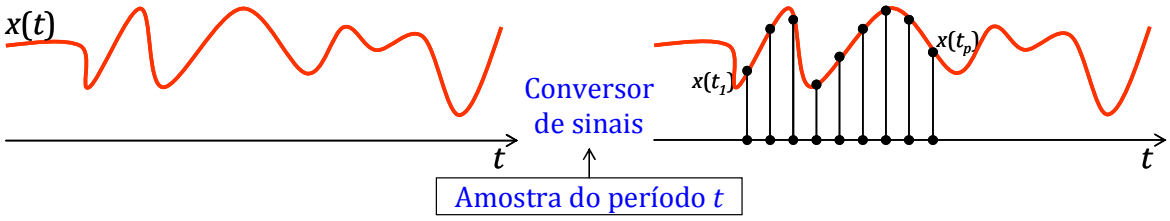
Efeito tendencioso aditivo

Efeito tendencioso multiplicativo

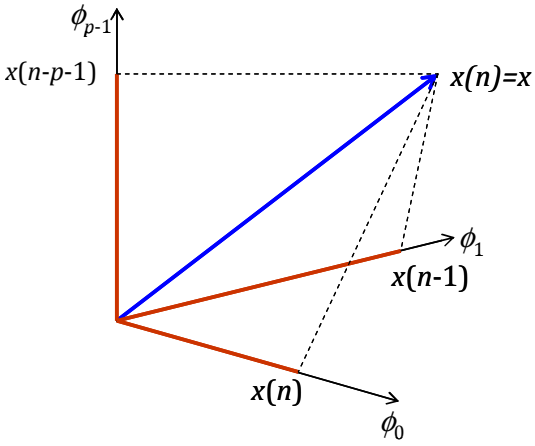
Problemas que não envolvem o tempo (tais como classificação de padrões) são chamados ***estáticos***. O tempo estabelece uma ordem nos dados de entrada. O tempo e as variáveis físicas são contínuas, compondo os sinais analógicos.

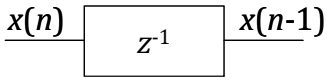
A cada *T* segundos, chamado *período de amostragem*, o sinal analógico *x(t)* é medido, produzindo um sinal $x(t_p)$, chamado *sequência* ou *série temporal*.

$$x(t_p) = \left[x(t_1), x(t_2), x(t_3), \cdots, x(t_p) \right].$$

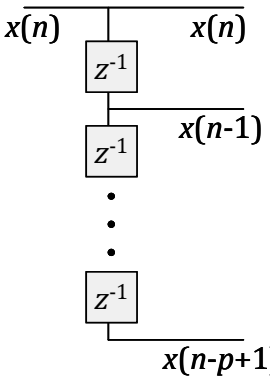


Vetor com atraso:

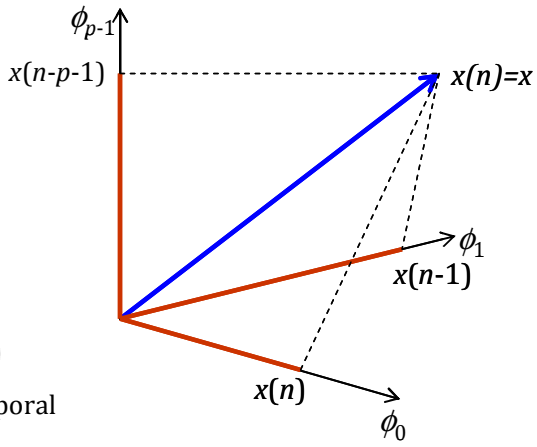


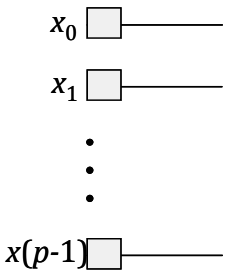


Um *elemento de atraso ideal*, denotado por z^{-1}, atrasa o sinal em uma amostra. Uma *linha de atraso* é um sistema de uma entrada e várias saídas, composto pela ligação em cascata de vários operadores de atraso.



Representação do sinal temporal

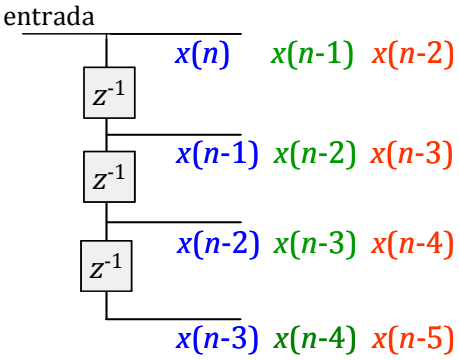


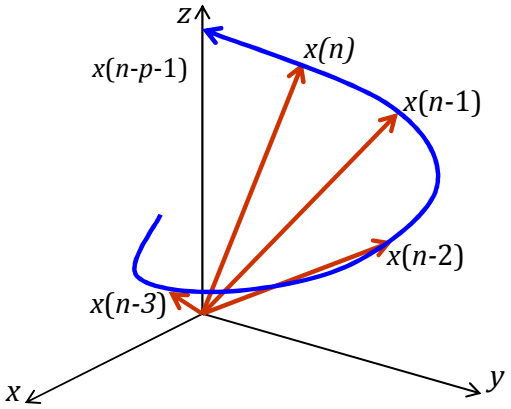


Padrão estático

O espaço cujos eixos são os sinais dos terminais da linha de atraso é chamado *espaço de sinais* ou de *reconstrução*.

A cada instante de tempo, o vetor do sinal muda sua posição, criando uma *trajetória do sinal*.

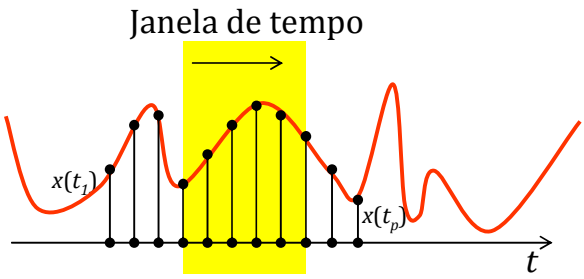


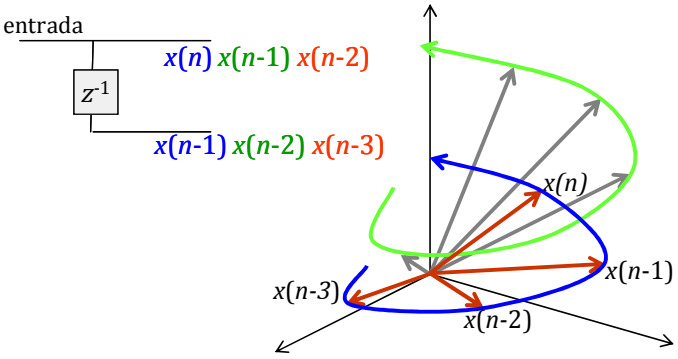


A construção de uma janela de tempo é essencial para a solução de um problema de Séries Temporais com RNA:

O tamanho do espaço de reconstrução determina o comprimento ***n*** de uma janela de tempo que desliza sobre a série temporal completa.

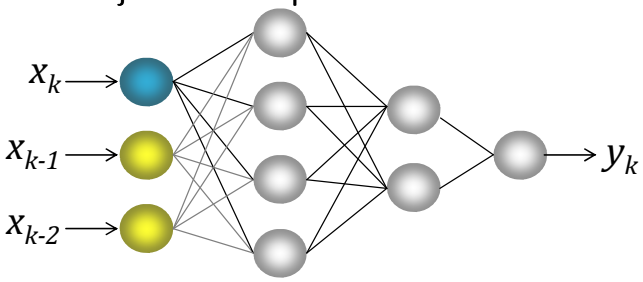
Este comprimento corresponde ao tamanho da linha de atraso e estabelece a dimensionalidade do espaço de reconstrução.



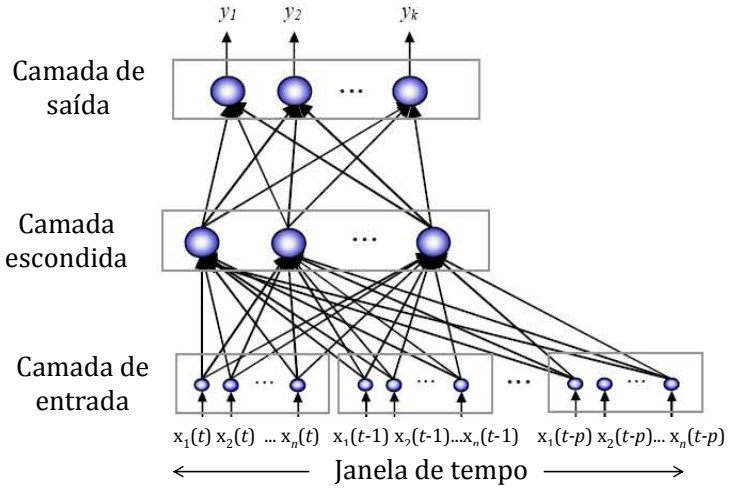


Algumas arquiteturas que podem ser usadas em Séries Temporais são:

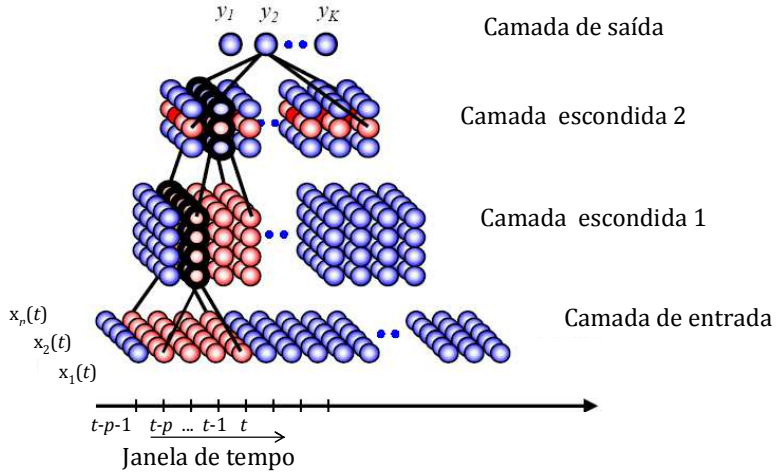
MLP com janela de tempo e 1 saída:



MLP com janela de tempo e *k* saídas:

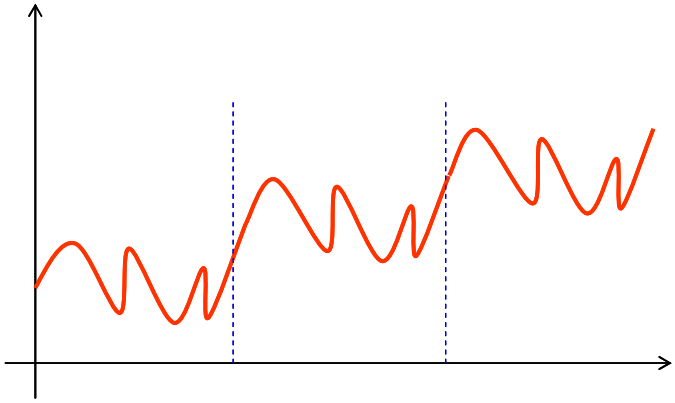


Rede Neural com atraso de tempo:



A camada 2 recebe uma janela de tempo da camada 1, e a saída recebe uma janela de tempo da camada 2

O *tempo* ajuda a remover a ambiguidade dos dados. É necessária *memória de curto prazo*. Se o problema a ser resolvido possui dinâmica, deve-se usar uma topologia com memória



Outro fator que auxilia uma RNA para Séries Temporais é a padronização dos dados de entrada:

Procedimento 1: Indicado quando os neurônios utilizam a função de ativação ***sigmoidal*** (logística)

$$\hat{x}_j = \frac{x_j}{x_{max}} \Rightarrow \hat{x}_j \in [0, 1]$$

Procedimento 2: Indicado quando os neurônios utilizam a função de ativação ***tangente hiperbólica.***

$$\hat{x}_j = 2\left(\frac{x_j - x_{min}}{x_{max} - x_{min}}\right) - 1 \Rightarrow \hat{x}_j \in [-1, +1]$$

Algumas metodologias que podemos usar são as seguintes:

Metodologia 1: Rede Neural com *k* entradas em janela de tempo, e 1 saída.

entrada $[x_1(t), x_2(t),..., x_k(t)]$

saída $y_k(t)$

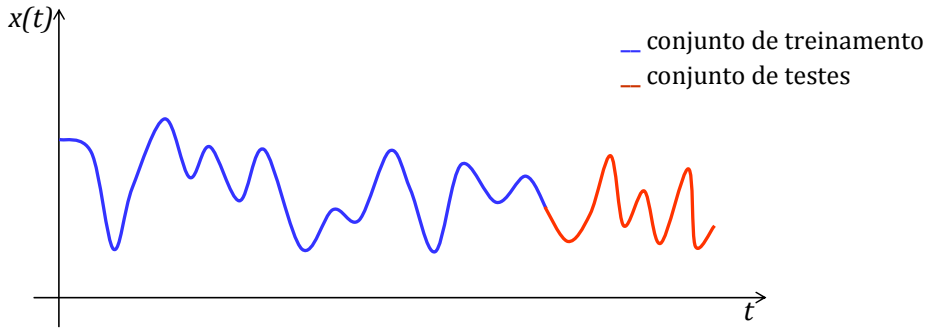
Metodologia 2: Rede Neural com *p* vetores de entrada (com atrasos) e *k* saídas (*Focused Time-Delay Neural Network*).

entrada $\{[x_1(t), x_2(t),..., x_n(t)], [x_1(t\text{-}1), x_2(t\text{-}1),..., x_n(t\text{-}1)], ..., [x_1(t\text{-}p), x_2(t\text{-}p),..., x_n(t\text{-}p)]\}$

saída $\{y_1, y_2, ..., y_k\}$

Metodologia 3: Rede com *N* vetores de entrada e *N* saídas.

Segue um exemplo da metodologia 1:

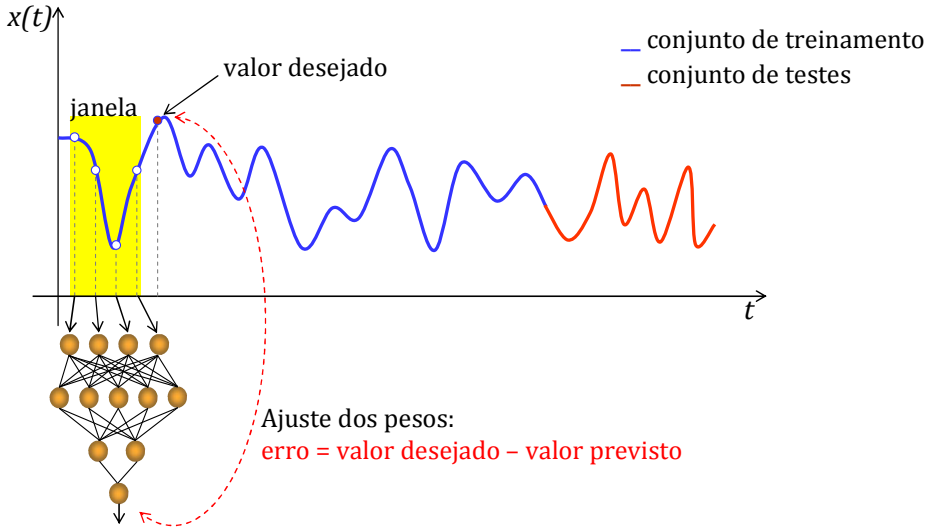


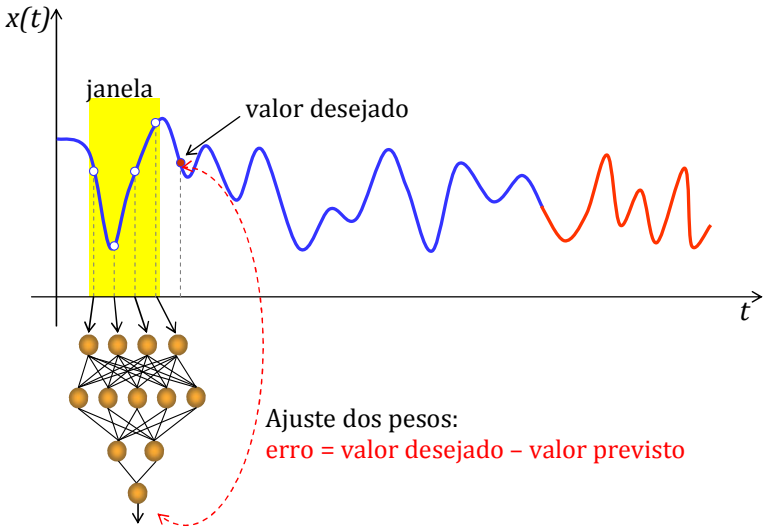
Entradas da rede:

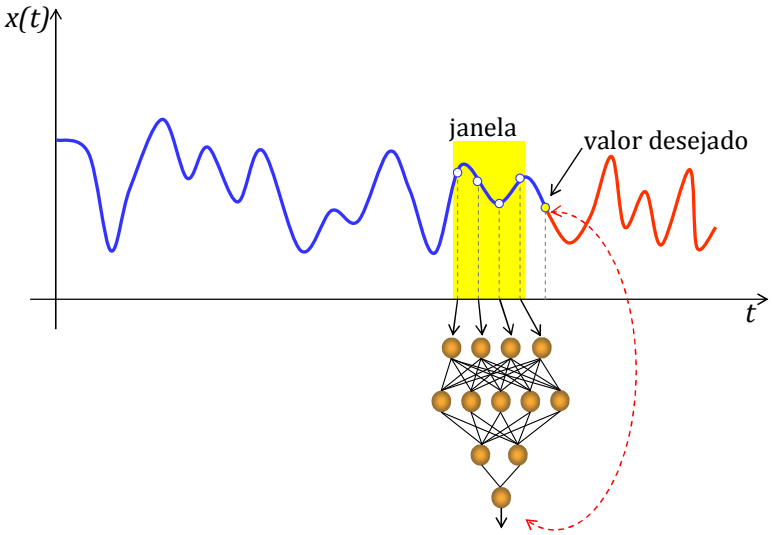
p valores passados (por exemplo, 4 valores)

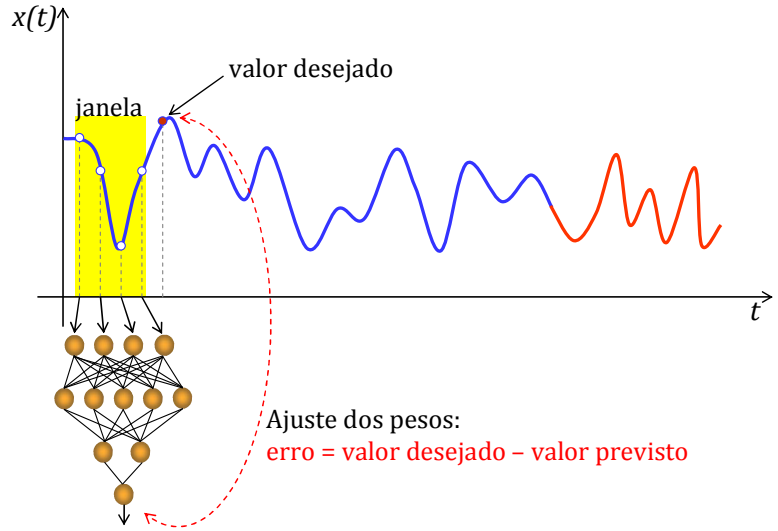
Saída desejada:

valor da série *k* passos à frente (por exemplo, 1 passo)

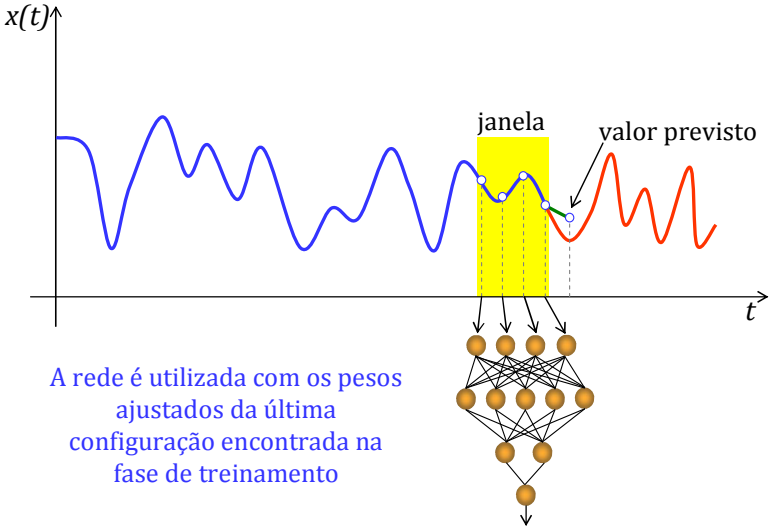

x(t)
conjunto de treinamento
conjunto de testes
valor desejado
janela
t
Ajuste dos pesos:
erro = valor desejado – valor previsto
Saída da rede
1 passo à frente


x(t)
janela
valor desejado
t
Ajuste dos pesos:
erro = valor desejado – valor previsto


x(t)
janela
valor desejado
t
Último valor do conjunto de treinamento:
Retorna-se ao primeiro valor deste conjunto
Ajuste dos pesos:
erro = valor desejado – valor previsto

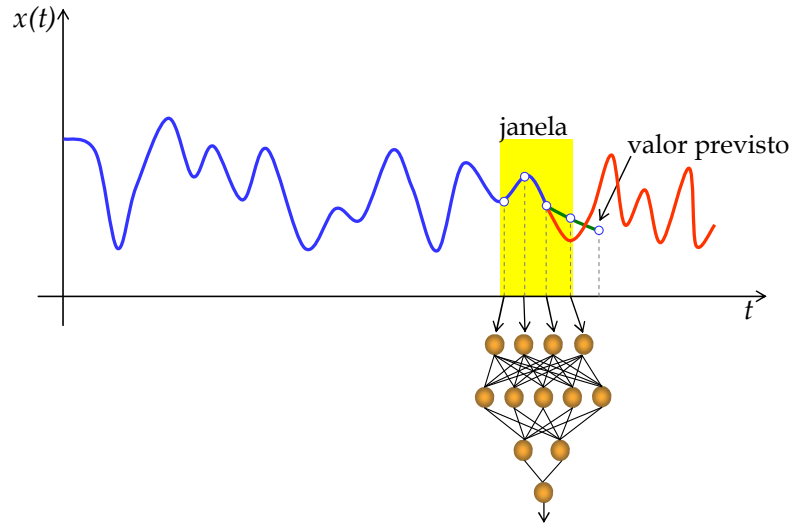


Repete-se o treinamento até que o erro atinja um valor mínimo

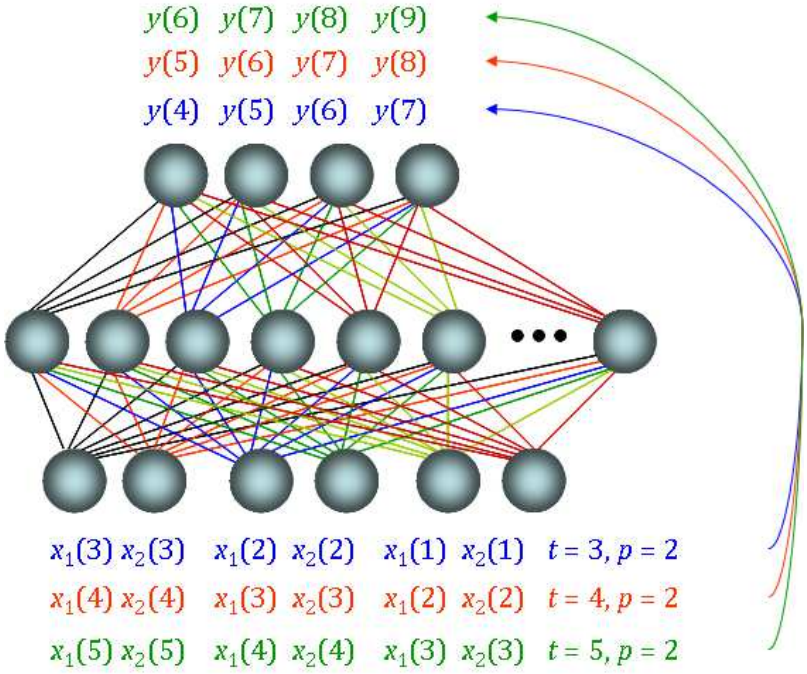


A rede é utilizada com os pesos ajustados da última configuração encontrada na fase de treinamento

Entrada da rede:
Inclui os valores previstos



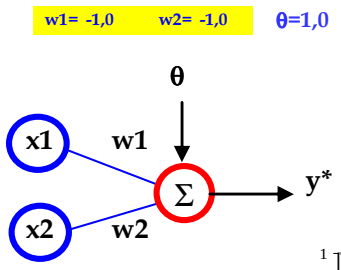
Exemplo de funcionamento de uma RNA temporal:



Exercícios:

1. Utilize uma rede MLP com uma camada de entrada de 2 neurônios (k=2) e saída com um neurônio para a seguinte série temporal:

	x	y	
x1	0,1	1	
x2	0,2	0,9	
	0,3	0,6	y1
	0,4	0,5	
	0,5	0,3	
	0,6	0,2	
	0,7	0,1	
	0,8	0	



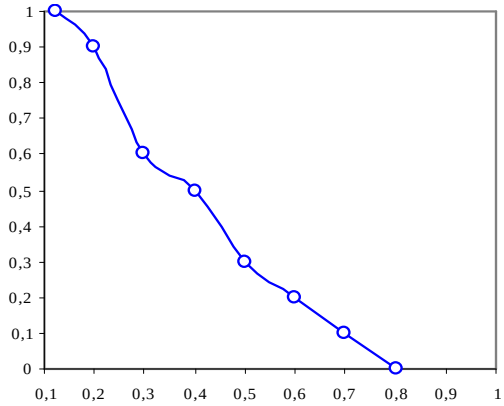
$y^* = \Sigma w_j x_j + \theta_j$

$y = 1/(1+e^{(-y^*)})$

$\Delta w_j = \alpha(d-y)x_j y(1-y)$

$w_j = w_j + \Delta w_j$

$\Delta\theta = \alpha(d-y)y(1-y)$



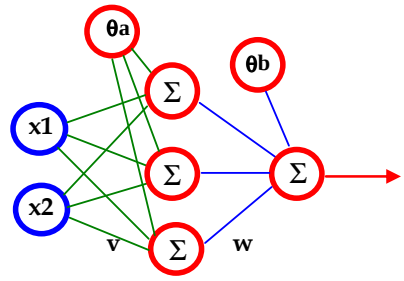
Esta é a aplicação da metodologia 1, com entradas x.

2. Resolva o problema anterior usando entradas t, e θ = 0,8.

	t / x / y		
x1	0,1	1	x1
x2	0,2	0,9	x2
	0,3	0,6	y1
	0,4	0,5	
	0,5	0,3	
	0,6	0,2	
	0,7	0,1	
	0,8	0	

3. Utilize uma rede MLP com uma camada de entrada de 2 neurônios (k=2), uma camada escondida com 3 neurônios (sigmoidais) e saída com um neurônio (sigmoidal) para a série temporal do exercício 1. Use entradas x com os seguintes parâmetros:

θa	1	2	3	v	1	2	3	w	1
1	-0,150	0,260	1,600	1	-1,500	-1,300	-1,200	1	1,800
θb	1			2	-1,300	-1,500	-1,900	2	-2,950
1	0,450							3	-3,700



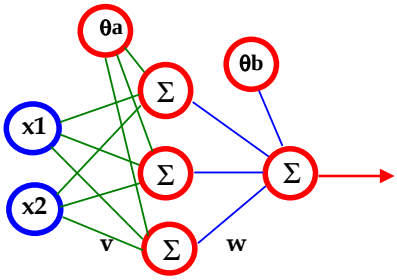
$z_j^* = \Sigma v_{ij} x_i + \theta a_j$
$z_j = 1/(1+e^{(-z_j^*)})$
$y^* = \Sigma w_j z_j + \theta b$
$y = 1/(1+e^{(-y^*)})$
$\Delta w_j = \alpha(d-y) z_j y(1-y)$
$w_j = w_j + \Delta w_j$
$\Delta \theta b = \alpha(d-y)y(1-y)$
$\Delta v_{ij} = \alpha y(1-y)(d-y) w_j z_j (1-z_j) x_i$
$v_{ij} = v_{ij} + \Delta v_{ij}$
$\Delta \theta a_j = \alpha y(1-y)(d-y) w_j z_j (1-z_j)$

4. Resolva o problema anterior com entradas t e os seguintes parâmetros:

θa	1	2	3	v	1	2	3	w	1
1	-1,100	-0,900	-1,050	1	-4,500	-2,500	-3,250	1	7,200
θb	1			2	-2,100	-6,752	-2,725	2	5,400
1	-1,050							3	6,700

5. Utilize uma rede MLP com uma camada de entrada de 2 neurônios (k=2), uma camada escondida com 3 neurônios (tanh) e saída com um neurônio (sigmoidal) para a série temporal do exercício 1. Use entradas x com os seguintes parâmetros:

θa	1	2	3	v	1	2	3	w	1
1	0,300	0,500	1,250	1	-1,400	-1,200	-1,000	1	1,950
θb	1			2	-1,200	-1,400	-1,800	2	1,700
1	0,900							3	-2,890



$z_j^* = \Sigma v_{ij} x_i + \theta a_j$
$z_j = \tanh(z_j^*)$
$y^* = \Sigma w_j z_j + \theta b$
$y = 1/(1+e^{(-y^*)})$
$\Delta w_j = \alpha(d-y) z_j y(1-y)$
$w_j = w_j + \Delta w_j$
$\Delta \theta b = \alpha(d-y)y(1-y)$
$\Delta v_{ij} = \alpha y(1-y)(d-y) w_j (1-z_j^2) x_i$
$v_{ij} = v_{ij} + \Delta v_{ij}$
$\Delta \theta a_j = \alpha y(1-y)(d-y) w_j z_j (1-z_j^2)$

6. Resolva o problema anterior com entradas t e os seguintes parâmetros:

θa	1	2	3	v	1	2	3	w	1
1	0,950	1,150	0,870	1	-1,150	-0,950	-1,000	1	2,100
θb	1			2	-1,050	-1,200	-1,750	2	1,900
1	0,980							3	-2,540

7. Utilize uma rede RBF para a série temporal dada a seguir, com entradas x e os seguintes parâmetros: $\sigma = 1$; $\alpha = 1$; u = {(0,2 0,3), (0,5 0,6)}; $k = 2$; $\theta = 1$; saída com 1 neurônio.

	x	y	
x1	0,1	1	
x2	0,2	0,9	
	0,3	0,75	y1
	0,4	0,6	
	0,5	0,55	
	0,6	0,6	
	0,7	0,4	
	0,8	0,3	
	0,9	0,2	
	1	0,05	

8. Resolva o problema anterior com entradas t e os mesmos parâmetros.

9. Utilize uma rede RBF para a série temporal dada no exercício 7, com entradas x e os seguintes parâmetros: $\sigma = 0,5$; $\alpha=1$; $k=2$; $\theta = 1$; u = {(0,1 0,2), (0,4 0,5)}; saída com 2 neurônios.

	x	y	
x1	0,1	1	
x2	0,2	0,9	
	0,3	0,75	y1
	0,4	0,6	y2
	0,5	0,55	
	0,6	0,6	
	0,7	0,4	
	0,8	0,3	
	0,9	0,2	
	1	0,05	

10. Resolva o problema anterior com entradas t e os mesmos parâmetros.

2.13. REDES NEURAIS RECORRENTES

As RNA são chamadas recorrentes quando possuem uma ou mais conexões de realimentação as quais proporcionam comportamento dinâmico à rede.

A realimentação pode ser:

- Local se está dada por apenas um neurônio
- Global se a realimentação engloba alguma(s) camada(s) completa(s).

A realimentação armazena, indiretamente, os valores prévios apresentados à rede, constituindo uma memória.

Existem dois usos funcionais para as redes recorrentes:

- Memória associativa
- Mapeamento de entrada-saída

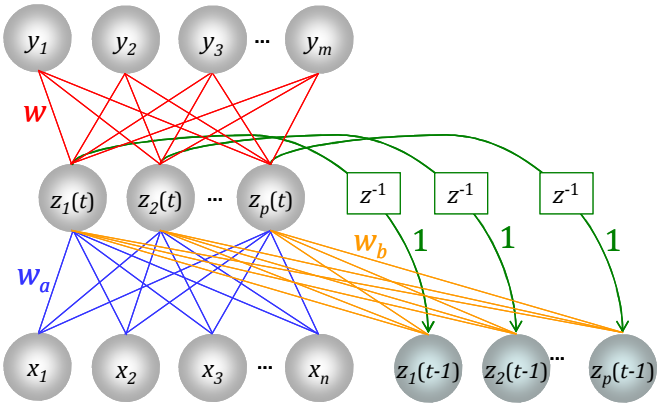
São redes que têm maior estabilidade, também chamadas de TLFN (*time layer focused neural network*). Elas tornam o mecanismo de memória de curto prazo estável.

A recorrência torna mais trabalhosos os cálculos.

As redes recorrentes mais conhecidas são simples, baseadas em neurônios de contexto, fáceis de treinar (com parâmetros de realimentação fixos), com correções de erros usando o algoritmo backpropagation.

Elas realizam o mapeamento com topologias pequenas e não há recorrência no caminho entrada-saída.

A **rede neural de Elman** (1990) tem cada um dos neurônios da camada oculta com realimentação para as unidades de contexto. É conhecida como "Perceptron de múltiplas camadas recorrentes".



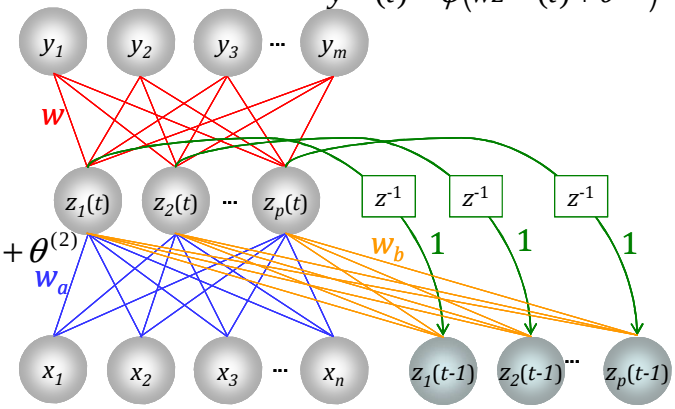
O cálculo da saída de uma rede neural de Elman é feito de maneira similar à MLP, com as seguintes atribuições:

Camada de saída (3):

$$y^{(3)}(t) = \varphi\left(wz^{(2)}(t) + \theta^{(3)}\right)$$

Camada escondida (2):

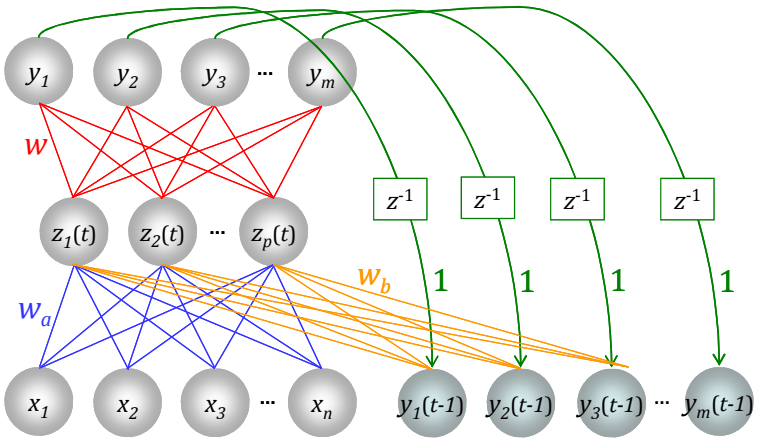
$$z^{*(2)}(t) = w_a x^{(1)}(t) + w_b z^{(1)}(t-1) + \theta^{(2)}$$



Camada de contexto (1):

$$z^{(1)}(t) = z^{(2)}(t-1) = \varphi\left(z^{*(2)}(t-1)\right)$$

A **rede neural de Jordan** (1986) considera somente realimentação dos valores de ativação de saída para as unidades de contexto.



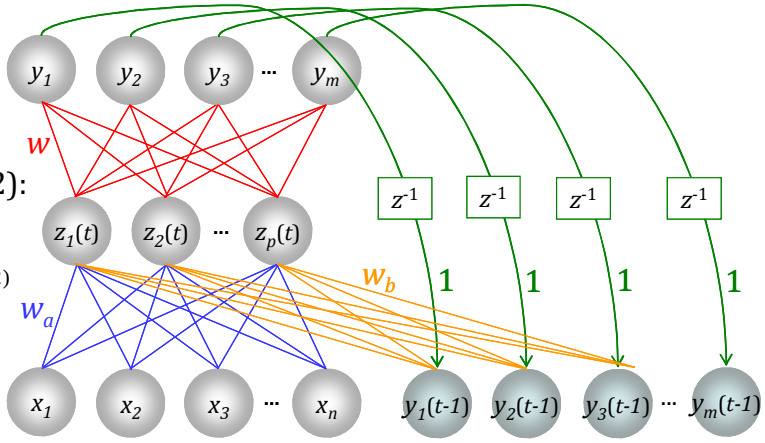
O cálculo da saída de uma rede neural de Jordan é feito de maneira similar à rede de Elman, com as seguintes atribuições:

Camada de saída (3):

$$y^{(3)}(t) = \varphi\left(wz^{(2)}(t) + \theta^{(3)}\right)$$

Camada escondida (2):

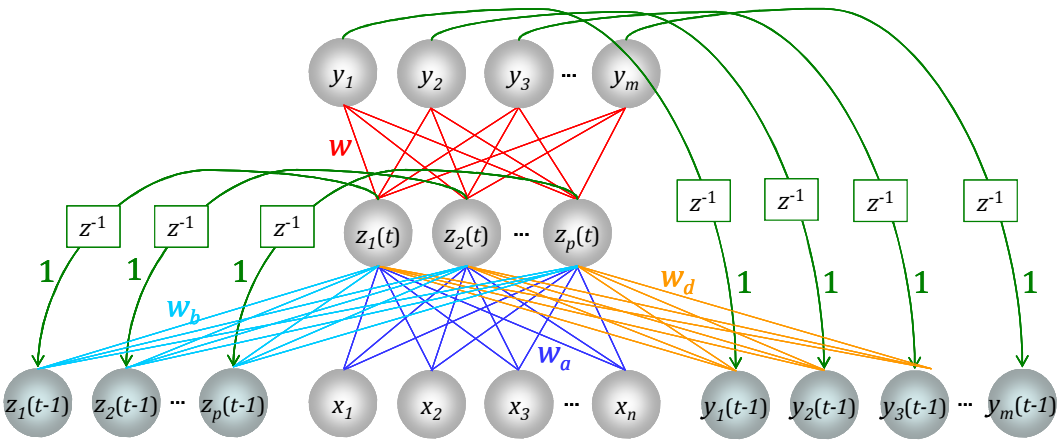
$$z^{*(2)}(t) = w_a x^{(1)}(t) + w_b y^{(1)}(t-1) + \theta^{(2)}$$



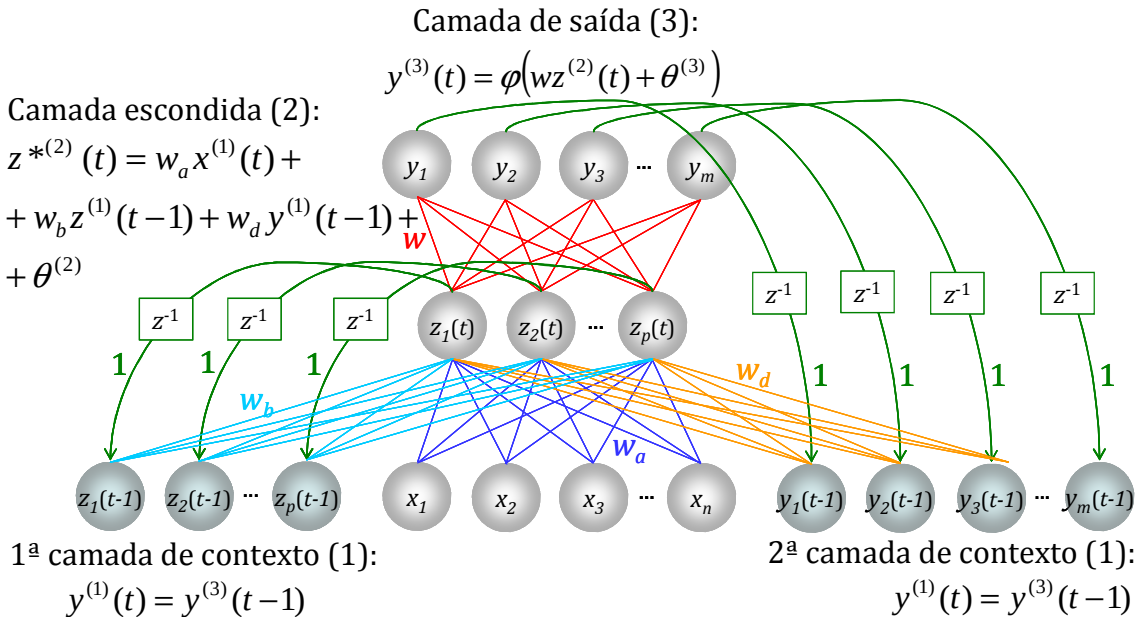
Camada de contexto (1):

$$y^{(1)}(t) = y^{(3)}(t-1)$$

A **rede neural de Williams-Zipser** (1988) tem a(s) camada(s) oculta(s) com unidades de contexto e um laço de realimentação da saída da rede para as unidades escondidas.



O cálculo da saída de uma rede neural de Williams-Zipses é feito de maneira similar às redes de Elman e de Jordan, com as seguintes atribuições:

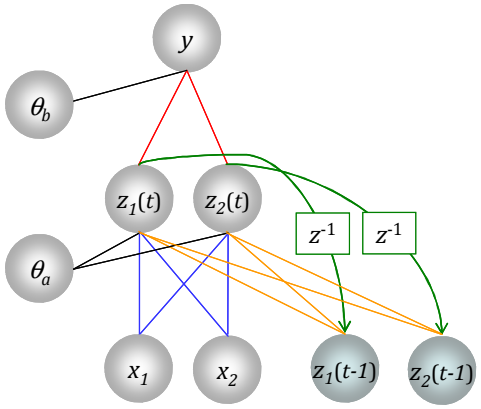


Uma aplicação interessante das Redes recorrentes é a solução de problemas de Séries Temporais, por suas características dinâmicas.

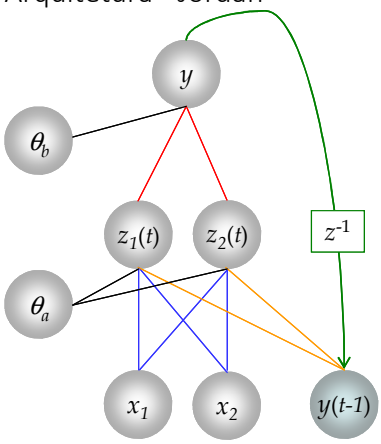
Exercícios:

1. Utilize as redes recorrentes de Elman e Jordan para resolver os problemas de classificação "OU" exclusivo e "E". Use arquitetura de entrada com 2 neurônios x, 2 neurônios na camada escondida z(t) e 2 neurônios de contexto z(t-1).

Arquitetura – Elman



Arquitetura - Jordan

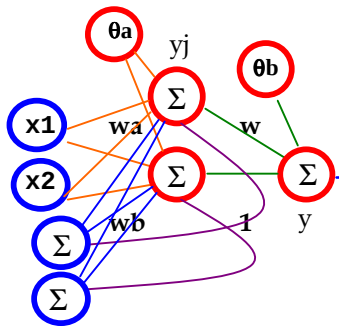


2. Utilize uma Rede de Elman para resolver o problema de séries temporais dado abaixo com entradas x. Considere os seguintes parâmetros: α=1, 1 camada escondida, com 2 neurônios, aprendizagem backpropagation, pesos iniciais:

θa	1	2	wa	1	2	wb	1	2	w	1
1	-1,00	1,00	1	1,00	-1,00	1	-1,00	1,00	1	-1,00
θb	1		2	-1,00	1,00	2	1,00	-1,00	2	1,00
1	1,00									

x1	x2	d
1	2	0,6
2	3	0,5
3	4	0,3
4	5	0,2
5	6	0,1

y1(t-1)= 1 **y2(t-1)= 1**



$y_j^* = \Sigma_i wa_{ij} x_i + \Sigma_\kappa wb_{kj} y_k(t-1) + \theta a_j$
$y_j = 1/(1+e^{(-y_j^*)})$
$y^* = \Sigma w_j y_j + \theta b$
$y = 1/(1+e^{(-y^*)})$
$\Delta w_j = \alpha(d-y) y_j y(1-y)$
$w_j = w_j + \Delta w_j$
$\Delta \theta b = \alpha(d-y) y(1-y)$
$\Delta wa_{ij} = \alpha y(1-y)(d-y) w_j y_j (1-y_j) x_i$
$wa_{ij} = wa_{ij} + \Delta wa_{ij}$
$\Delta \theta a = \alpha y(1-y)(d-y) w_j y_j (1-y_j)$
$\Delta wb_{kj} = \alpha y(1-y)(d-y) w_j y_k (1-y_k) y_j(t-1)$
$wb_{ij} = wb_{ij} + \Delta wb_{ij}$

3. Resolva o problema 2 com entradas t e os mesmos parâmetros de entrada.

4. Resolva o problema 2 com uma rede de Jordan com entradas x.

5. Resolva o problema 2 com uma rede de Williams-Zipses com entradas x.

Além das RNA recorrentes, uma rede híbrida também pode resolver o problema de séries temporais.

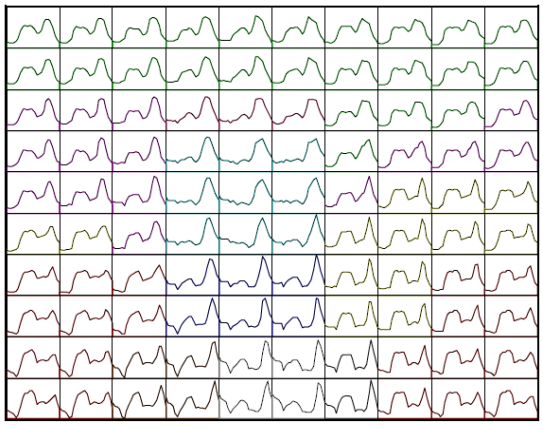
Uma das RNAs é a *Rede de Kohonen*, com MLPs conectadas a cada agrupamento feito pelo mapa de Kohonen [Le Coadou & Benabdeslem, 2006]. Outra configuração possível é de uma rede de Kohonen dupla.

Os seguintes passos são feitos para a rede híbrida SOM+MLP:

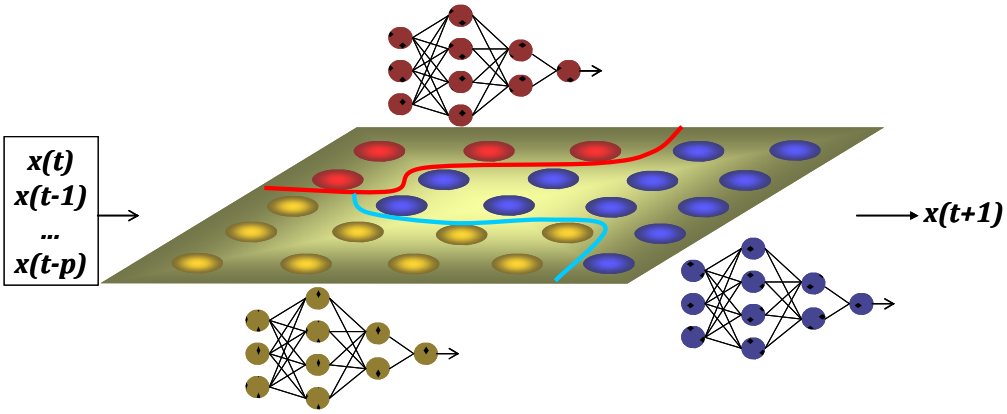
- ❑ Classificar todos os vetores de entrada

$$(x_t, x_{t-1}, \ldots, x_{t-N-1})$$

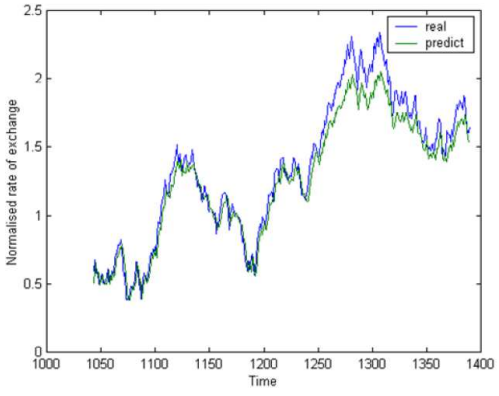
- ❑ Criar um mapa com protótipos similares aos vetores apresentados para a rede:



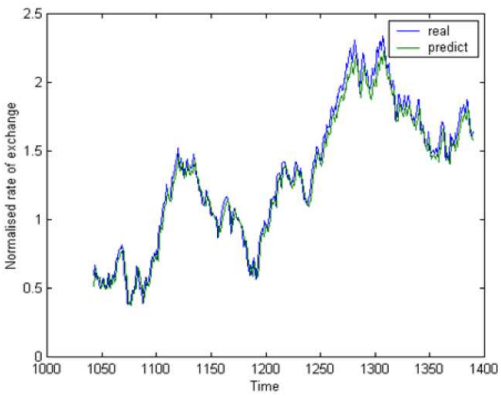
- Associar a cada agrupamento (representado pelo centróide G_k) uma rede do tipo MLP. Os vetores do agrupamento são considerados como entrada para a MLP.
- A previsão de cada entrada apresentada à rede é o resultado do processamento de cada MLP.



Resultados:



Global MLP
8 neurônios na camada escondida
Mse: 0,00504



SOM + MLP
mapa 10x12, 40 agrupamentos
40 MLPs
Mse: 0,003

REFERÊNCIAS

FAUSETT, L. *Fundamentals of Neural Networks.* Prentice Hall, 1994

HAYKIN, S. *Neural Networks – A Compreensive Foundation.* Macmillan College Publishing, 1994

KOHONEN, T. Self-Organizing Maps. Springer, 1995

SILVA, I.N.; SPATTI, D.H.; FLAUZINO, R.A. *Redes Neurais Artificiais para engenharia e ciências aplicadas*. Artliber, 2010

TAFNER, M.A.; XEREZ, M.; RODRÍGUEZ FILHO, I.W. Redes Neurais Artificiais: introdução e princípios da neurocomputação. FURB, 1996