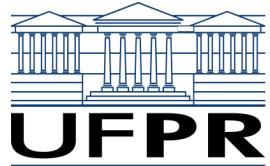




departamento de
**EXPRESSÃO
GRÁFICA**

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
DEPARTAMENTO DE EXPRESSÃO GRÁFICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MÉTODOS NUMÉRICOS EM ENGENHARIA
Prof. Paulo Henrique Siqueira
Disciplina : MNE 740 - Metaheurísticas e aplicações



Atribuição-Não Comercial-Sem Derivações: CC BY-NC-ND

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:

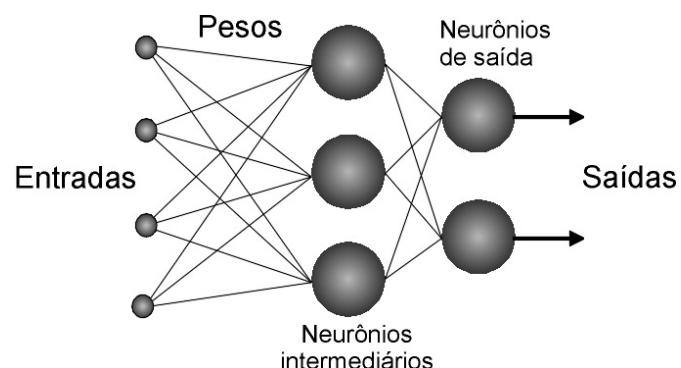
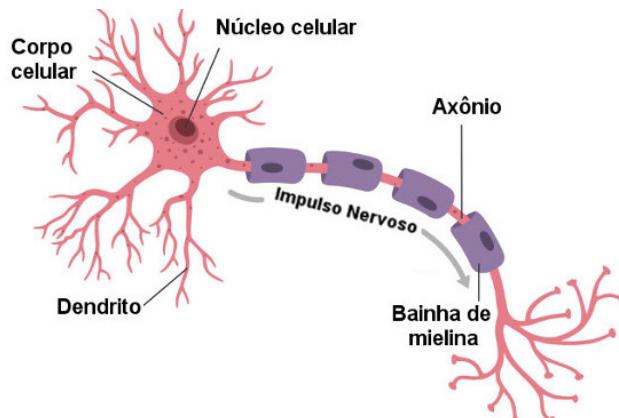
- também chamado de taxa de **aprendizagem** ou de **equilíbrio**;
- varia lentamente para evitar ótimos locais e permitir uma exploração maior do espaço de soluções;
- quanto mais lentamente variar, maior é a probabilidade de encontrar uma solução ótima global;
- existem metaheurísticas para procurar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



Fonte: <https://mundoeducacao.uol.com.br/biologia/neuronios.htm>

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.

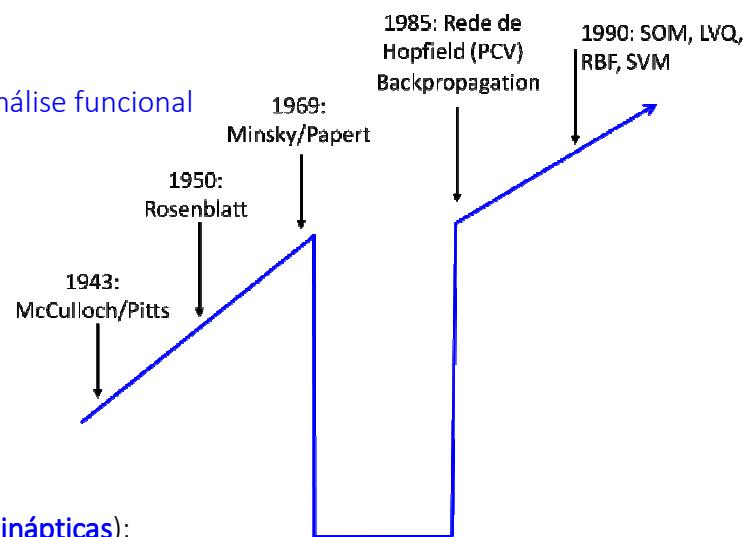
Alguns exemplos de aplicações das RNAs:

- 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, 2007, 2010];
- robótica;
- diagnóstico médico [Boçais et al., 2012], [Rosa, Steiner, Steiner Neto, 2016];
- predição de falência bancária;
- mundo financeiro;
- controle de processos químicos;
- previsão de vazões de hidrelétricas [Teixeira, Teixeira Junior, Siqueira, 2015];
- detecção fraudes com cartões;
- problemas de administração de empresas

Relevância matemática:

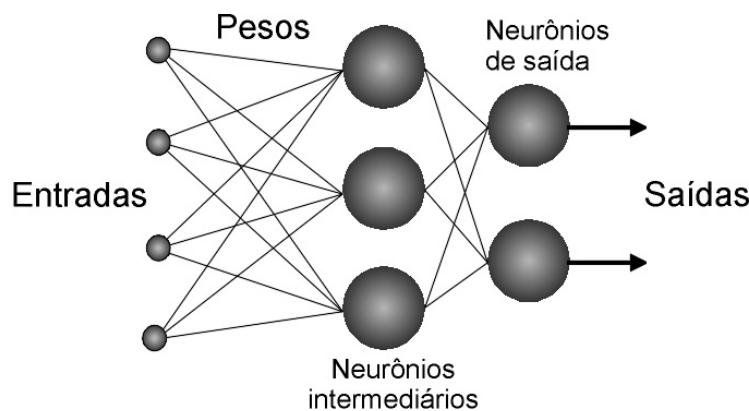
A matemática é utilizada em RNA para:

- 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.



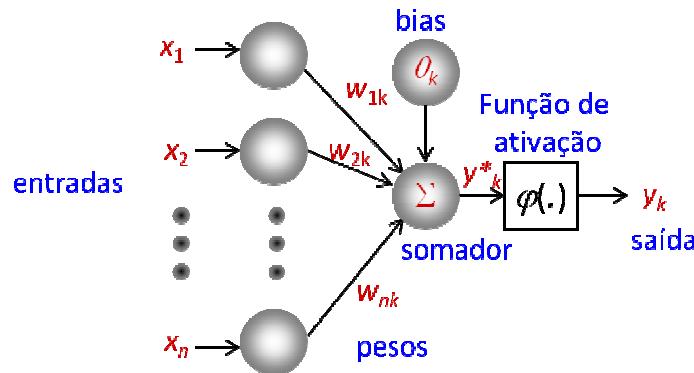
Dendritos: entradas

Corpo celular:

- Soma ponderada
- Função não-linear

Axônio: distribuição aos neurônios

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



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

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)**.

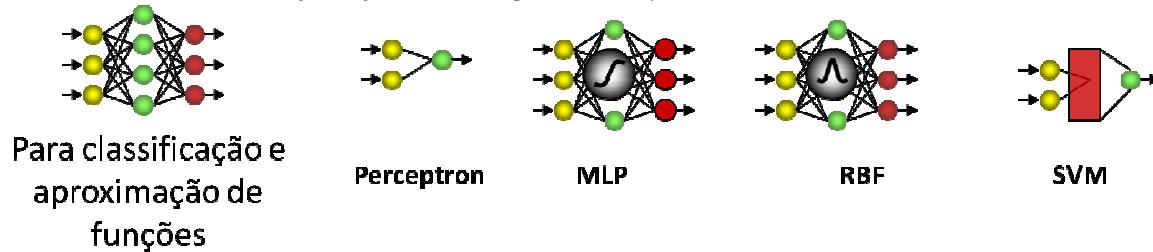
Entretanto, na maioria dos casos, o bias pode ser interpretado como um peso sináptico conectado a uma entrada constante.

Qual o papel do bias?

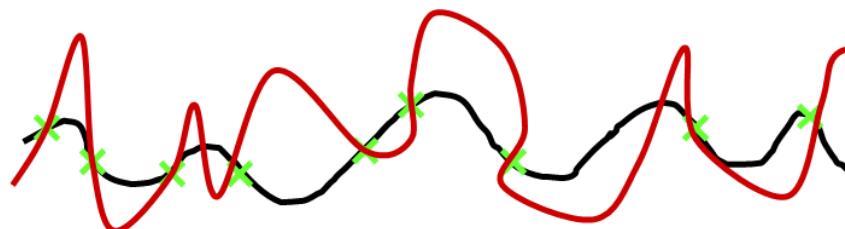
Notações para o bias: θ_k, b_k ou w_0

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:

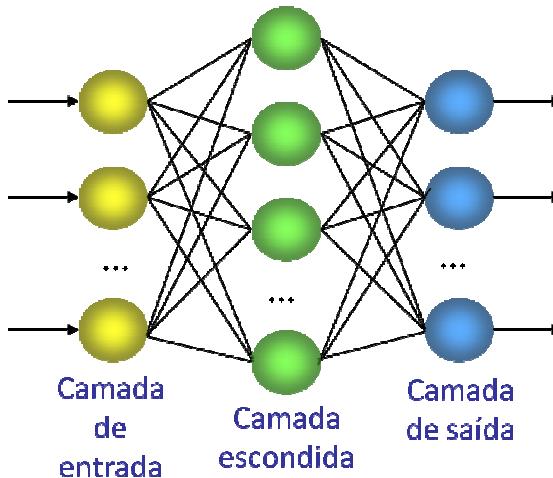


Nestas redes existem conjuntos de nós de **entrada**, de **saída** e os nós **escondidos** (intermediários entre os de entrada e saída).

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

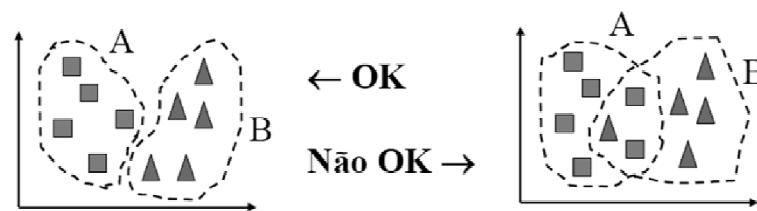
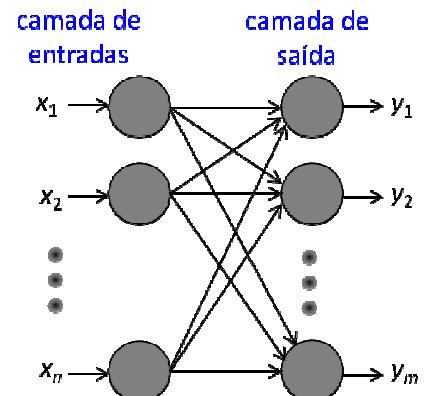
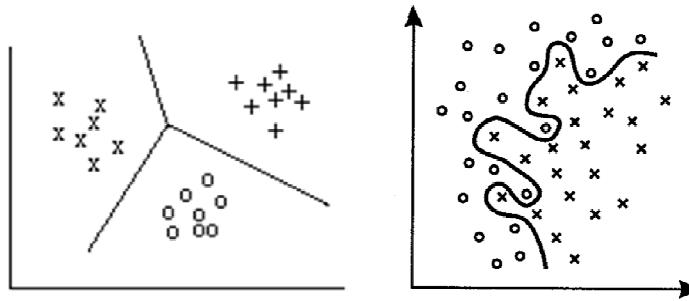
É um tipo de rede neural usado para conjuntos de treinamento [linearmente separáveis](#) (Rosenblatt, 1950).

$$S = w_0 + \sum_{i=1}^n w_i x_i$$

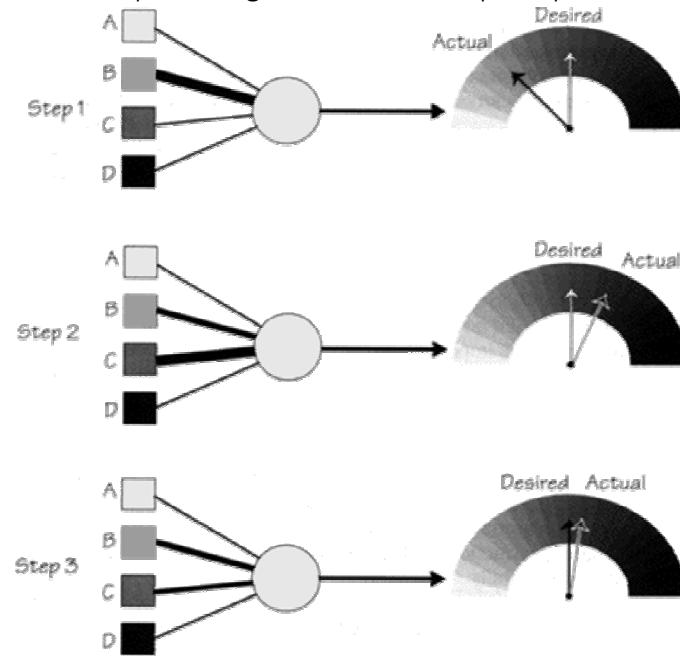
Tem a inclusão de [bias](#) (tendência).

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.



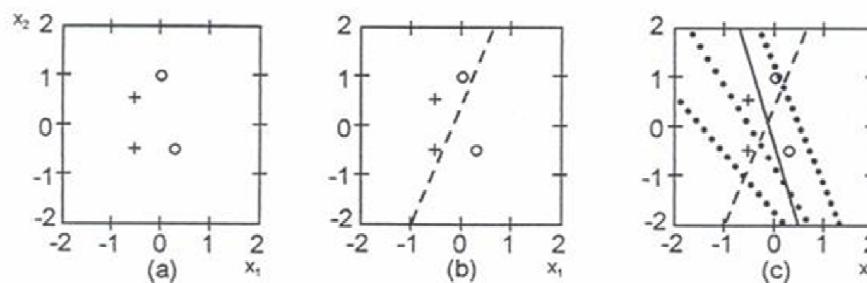
Exemplo de funcionamento da aprendizagem de uma rede perceptron:



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 $y^* = \theta + \sum_i x_i w_i$
 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
 $w_i^{\text{atual}} = w_i^{\text{anterior}} + \alpha d x_i$ e $\theta^{\text{atual}} = \theta^{\text{anterior}} + \alpha d$
Caso contrário
 $w_i^{\text{atual}} = w_i^{\text{anterior}}$ e $\theta^{\text{atual}} = \theta^{\text{anterior}}$
 6. Teste a condição de parada.

Funcionamento de um perceptron:



(a) As entradas (b) Separação inicial do espaço (c) Separação final [Demuth, Beale, 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 quando começam 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:

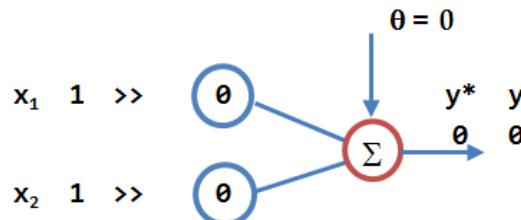
$$w_1 = -2,1642; w_2 = 0,0744; \theta = -0,6433$$

Exercícios:

1. Classificação da função lógica “E” com entradas 0-1 e saídas bipolares:

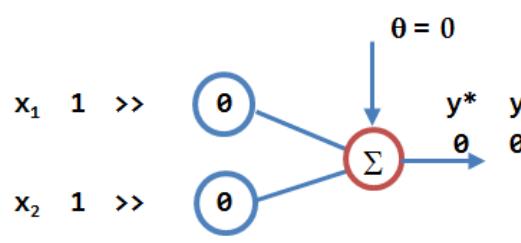
Valores iniciais: $\delta = 0,2$ $\alpha = 1$ $\theta = 0$ $w = 0$

x₁	x₂	d
1	1	1
1	0	-1
0	1	-1
0	0	-1



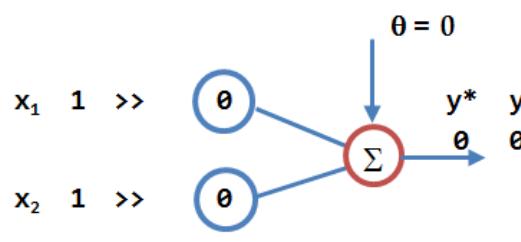
2. Classificação da função lógica “E” com entradas e saídas bipolares:

x₁	x₂	d
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1



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

x₁	x₂	d
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

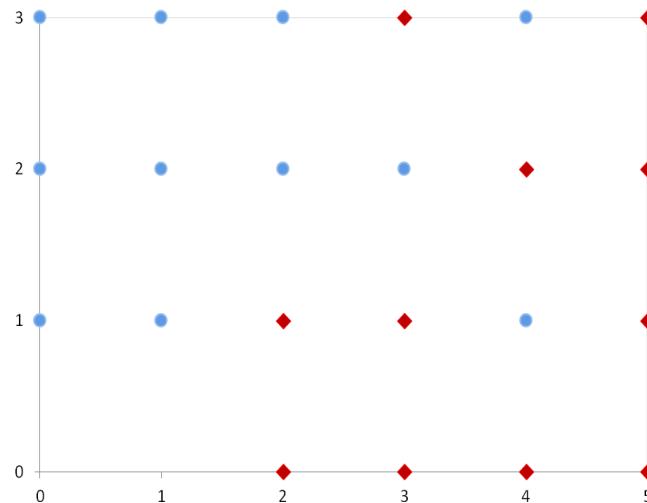


4. Classificação de padrões, conforme tabela abaixo:

x₁	x₂	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	x₁	x₂	d	n	x₁	x₂	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	x₁	x₂	d	n	x₁	x₂	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 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).
- o 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 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:

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

O algoritmo do bolso leva em conta as classificações corretas:

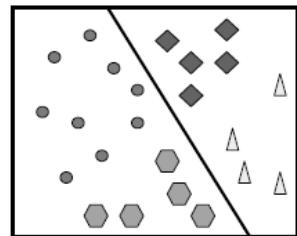
- o mantém um conjunto de pesos em separado W_{bolso} “no bolso”;

- o armazena o número de iterações consecutivas que w^{bolso} classificou corretamente exemplos de treinamento escolhidos.

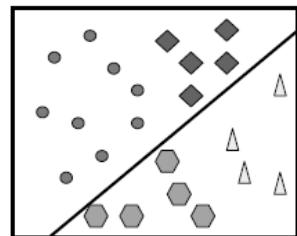
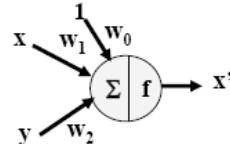
Algoritmo do Bolso:

0. Inicializar os pesos, o bias e a taxa de aprendizado: $w = \theta, w^{bolso} = \theta, \theta = \theta, \alpha = 1$.
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 $y^* = \theta + \sum_i x_i w_i$
 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
 $w_i^{atual} = w_i^{anterior} + \alpha dx_i$ e $\theta^{atual} = \theta^{anterior} + \alpha d$
Caso contrário
 $w_i^{atual} = w_i^{anterior}$ e $\theta^{atual} = \theta^{anterior}$
 6. Se w classifica corretamente mais exemplos do que w^{bolso} :
 $w^{bolso} = w$; grave o número de exemplos corretos
 7. Teste a condição de parada.

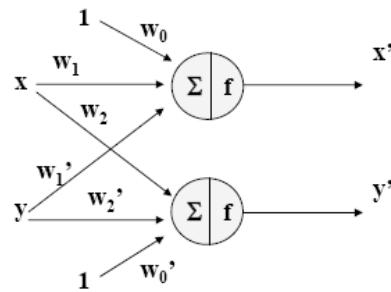
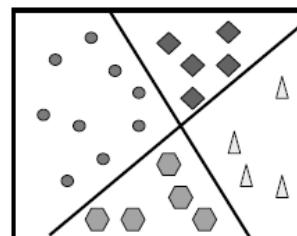
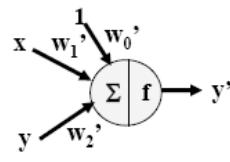
Separação em 4 classes com Perceptron:



Neurônio 1: separa em classes C1 e C2



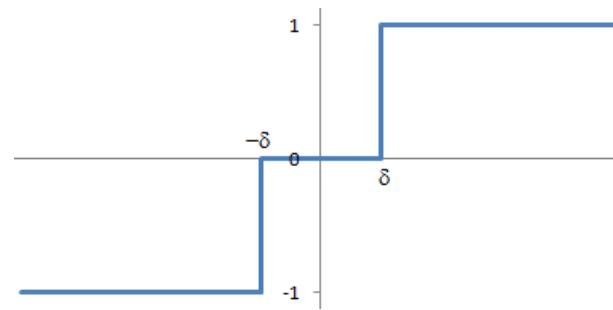
Neurônio 2: separa em classes C1' e C2'



FUNÇÕES DE ATIVAÇÃO

Limiar: saídas em $[0,1]$ ou $[-1,1]$

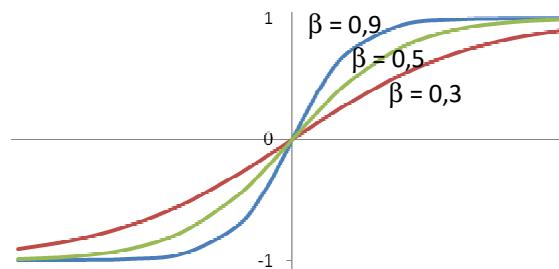
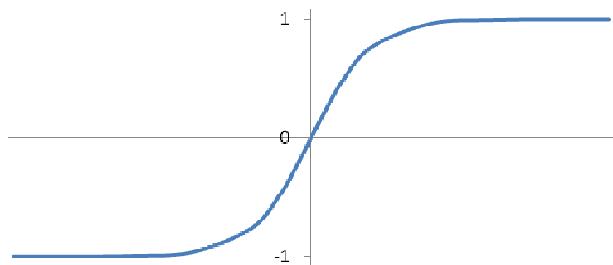
$$f(x_k) = \begin{cases} 1, & \text{se } x_k > \delta \\ 0, & \text{se } -\delta \leq x_k \leq \delta \\ -1, & \text{se } x_k < -\delta \end{cases}$$



Tangente hiperbólica: saídas em $[-1,1]$

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

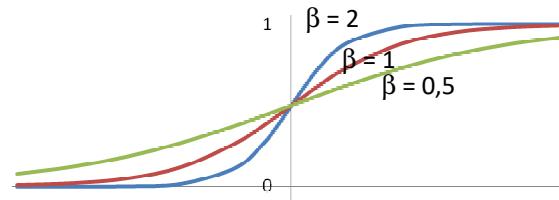
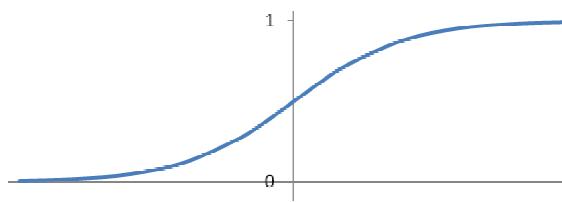
$$\frac{\partial f}{\partial x_k} = \beta(1 - f(x_k)^2) > 0$$



Sigmóide: saídas em $[0,1]$

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

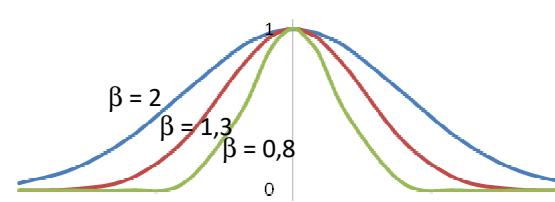
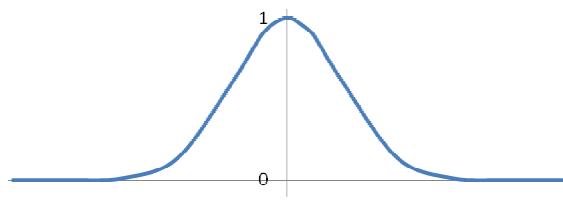
$$\frac{\partial f}{\partial x_k} = \frac{\beta e^{-\beta x_k}}{(1 + e^{-\beta x_k})^2}$$



Gaussiano: saídas em $[0,1]$

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

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



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

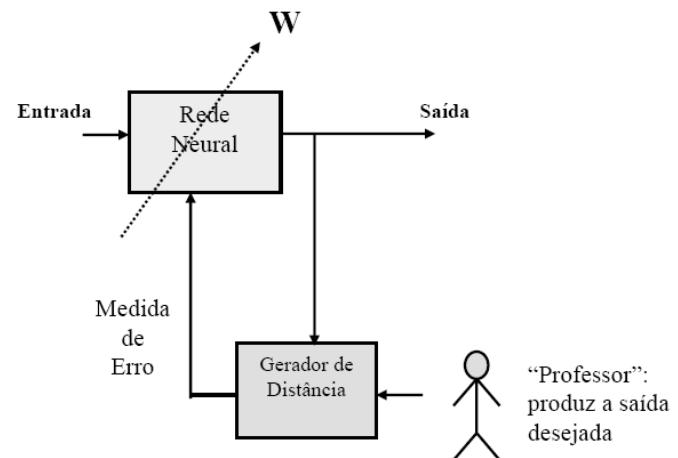
A função limiar também é chamada de degrau.

FORMAS DE APRENDIZAGEM

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

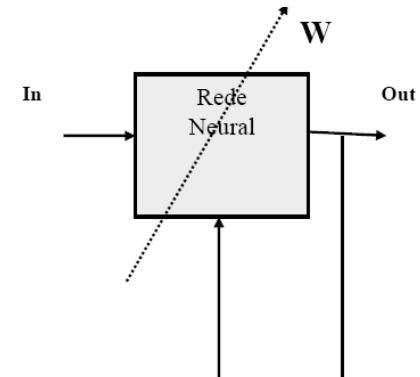
Supervisionada

- 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.



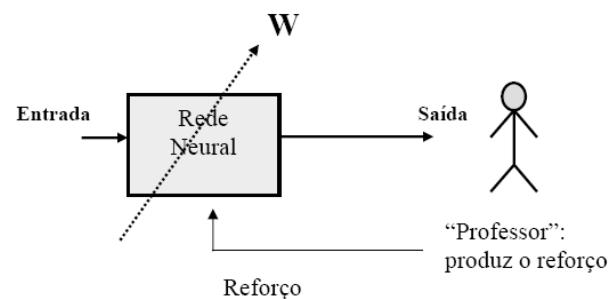
Não-supervisionada

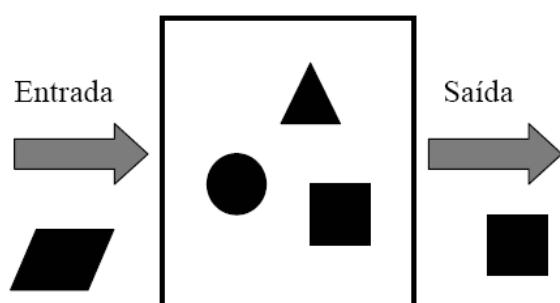
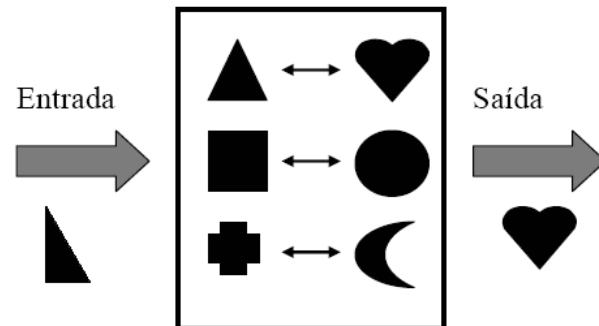
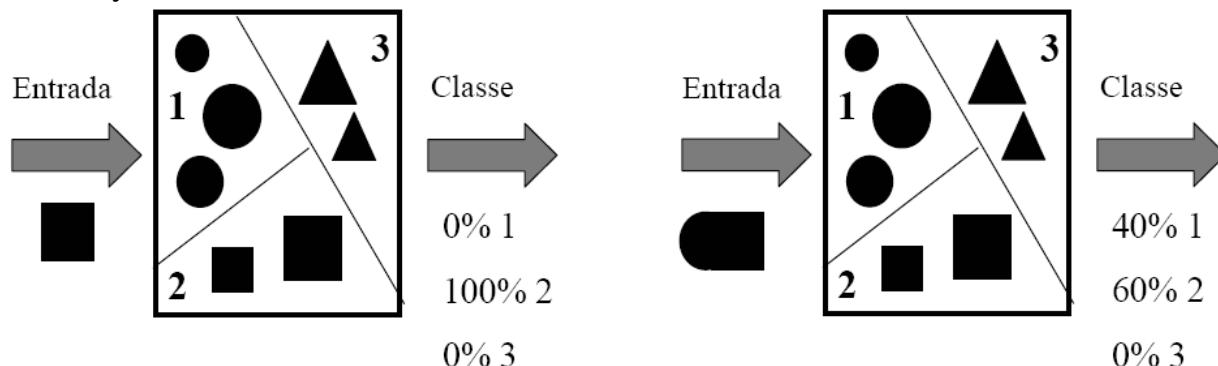
- a rede atualiza os parâmetros e pesos sem o controle de saídas desejadas para os padrões de entrada;
- a rede descobre sozinha as propriedades das entradas, e constrói a saída baseando-se nestas “descobertas”.



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:**

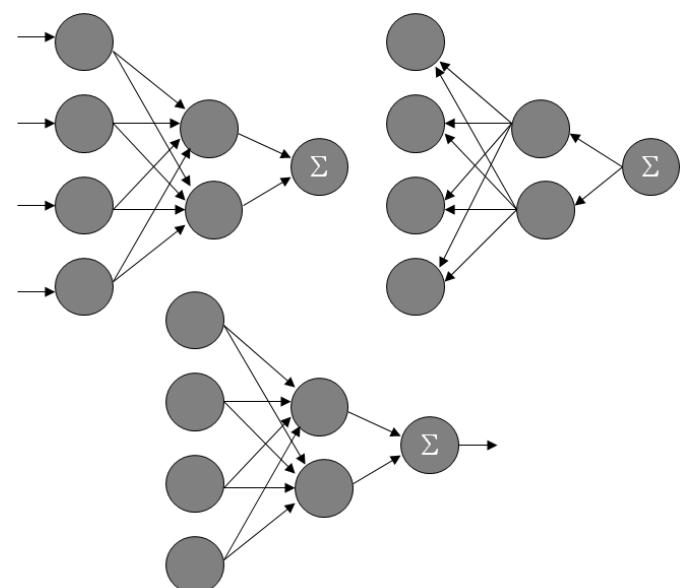
Um determinado conceito A “lembra” o conceito B.

Por exemplo:

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

ARQUITETURAS DAS RNAs**Com realimentação:**

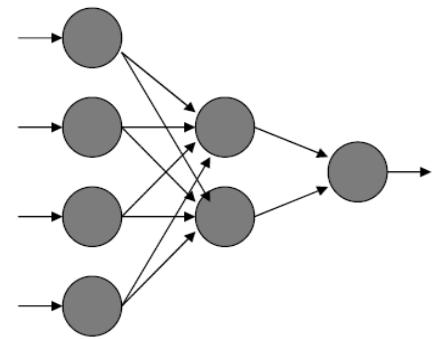
Os sinais dos neurônios percorrem a rede nas duas direções.



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.

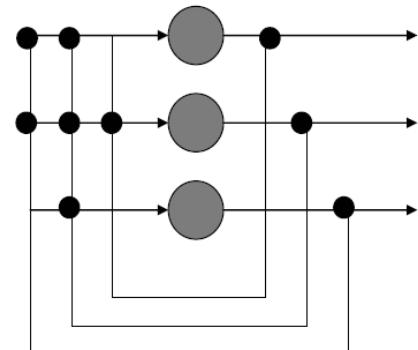


Recorrentes:

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**.

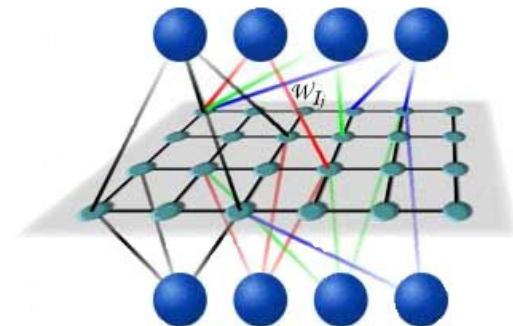
Em alguns casos os **próprios neurônios** são conectados a si mesmos.



Competitivas:

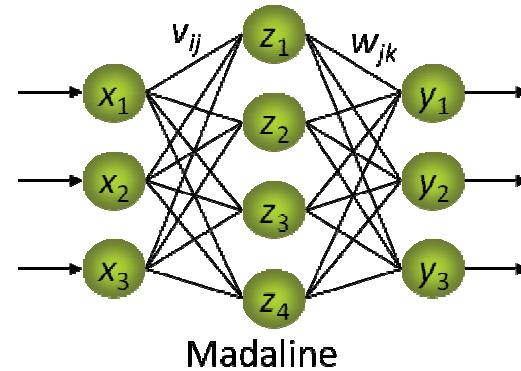
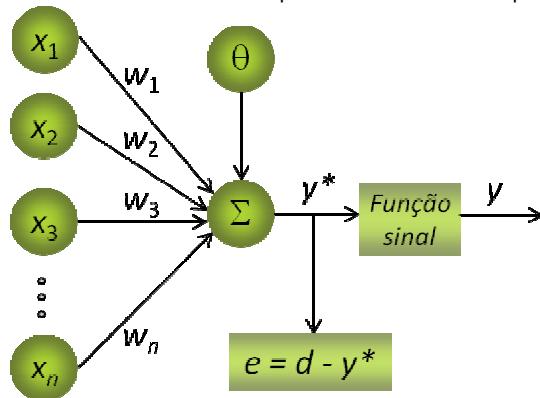
Os dados de entrada são designados a um **mapa de neurônios**, e escolhe-se o **vencedor** (que possui peso mais próximo da entrada).

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.

REGRAS DE APRENDIZAGEM

A ideia básica é minimizar a distância entre as saídas desejadas e as respostas y^* da RNA. Logo, existirá um vetor ótimo de pesos \mathbf{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 todos os p padrões de entrada teremos:

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

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

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

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

Aplicando-se o gradiente em relação ao vetor \mathbf{w} para encontrar o erro mínimo para o erro quadrático médio obtemos:

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

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

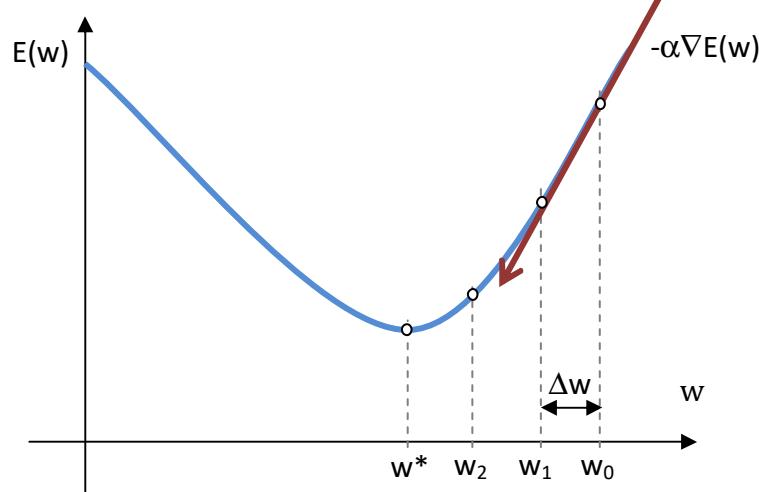
A adaptação da RNA deve ser feita na direção contrária à do gradiente, pois o objetivo é de minimizar o erro médio quadrático, ou seja:

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

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

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

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 $y^* = \theta + \sum_i x_i w_i$
 4. Atualize os pesos e a tendência:

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

$$\theta^{\text{atual}} = \theta^{\text{anterior}} + \alpha(d - y^*)$$
 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: $\alpha = 0,5$ $w_1 = 0,3$ $w_2 = 0,5$ $\theta = 0$

x₁	x₂	d
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

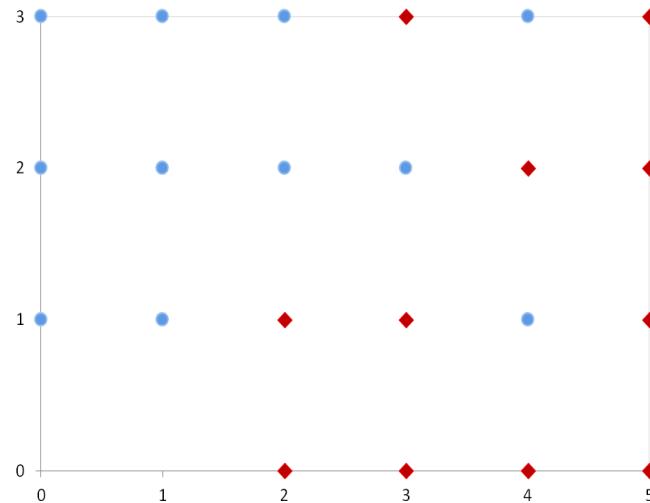
2. Classificação da função lógica “E” com entradas e saídas bipolares:
Valores iniciais: $\alpha = 0,4$ $w_1 = 0,9$ $w_2 = -0,7$

x₁	x₂	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 $w_i = 0$): $p_1 = (1,1,1,1)$, $d_1 = 0,9$; $p_2 = (-1,-1,-1,-1)$, $d_2 = 0,8$; $p_3 = (1,1,-1,-1)$, $d_3 = -0,7$; $p_4 = (1,-1,-1,1)$, $d_4 = -0,8$; $p_5 = (-1,-1,1,1)$, $d_5 = -0,7$.

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	x₁	x₂	d	n	x₁	x₂	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	x₁	x₂	d	n	x₁	x₂	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].

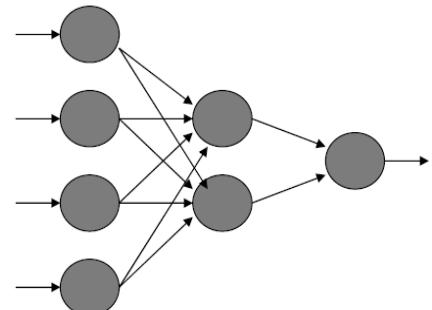
O ajuste dos pesos foi proposto por Rumelhart, Hinton, Williams, Parker, Le Cun e Werbos .

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.



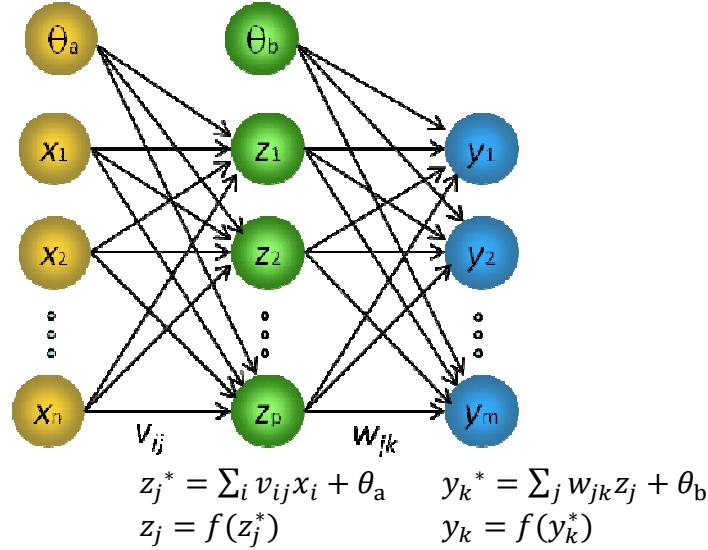
Superfícies de separação

Estrutura	Região de decisão	Problema do "OU" exclusivo	Problemas com regiões mescladas	Regiões de buscas
1 camada	Semi-plano limitado por um hiperplano			
2 camadas	Arbitrária (complexidade definida pelo número de neurônios da camada escondida)			
3 camadas	Arbitrária (complexidade definida pelo número de neurônios da camada escondida)			

REGRA DELTA GENERALIZADA

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

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

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

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

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

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

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

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

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

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 de pesos 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:

$$\frac{\partial E_j}{\partial v_{ij}} = \frac{\partial E_j}{\partial y} \frac{\partial y}{\partial v_{ij}} \quad (3)$$

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

$$\frac{\partial y}{\partial v_{ij}} = \frac{\partial y}{\partial y^*} \frac{\partial y^*}{\partial v_{ij}} \quad (4)$$

Mas

$$\frac{\partial y^*}{\partial v_{ij}} = \frac{\partial y^*}{\partial z_j} \frac{\partial z_j}{\partial v_{ij}} = \sum_k w_{jk} \frac{\partial z_j}{\partial v_{ij}} \quad (5)$$

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:

$$\begin{aligned} \frac{\partial E_j}{\partial v_{ij}} &= - \sum_k [(d_k - y_k) f'(y_k^*) w_{jk}] f'(z_j^*) x_i \\ \therefore \Delta v_{ij} &= \alpha \sum_k [(d_k - y_k) f'(y_k^*) w_{jk}] f'(z_j^*) x_i. \end{aligned} \quad (6)$$

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

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

Substituindo (7) em (6) obtemos:

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

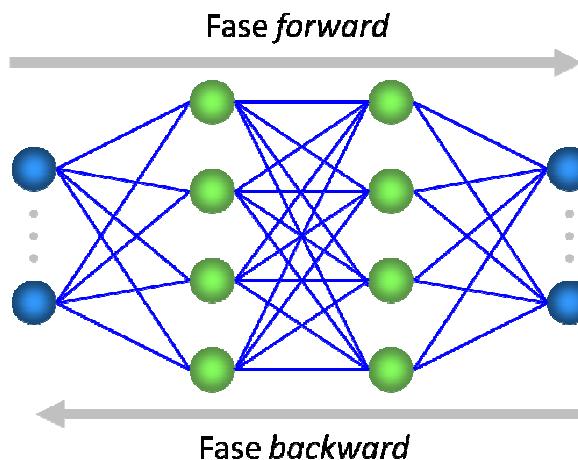
ou seja,

$$\begin{aligned} \frac{\partial E_j}{\partial v_{ij}} &= - \sum_k [(d_k - y_k) y_k (1 - y_k) w_{jk}] z_j (1 - z_j) x_i \\ \therefore \Delta v_{ij} &= \alpha \sum_k [(d_k - y_k) y_k (1 - y_k) w_{jk}] z_j (1 - z_j) x_i. \end{aligned}$$

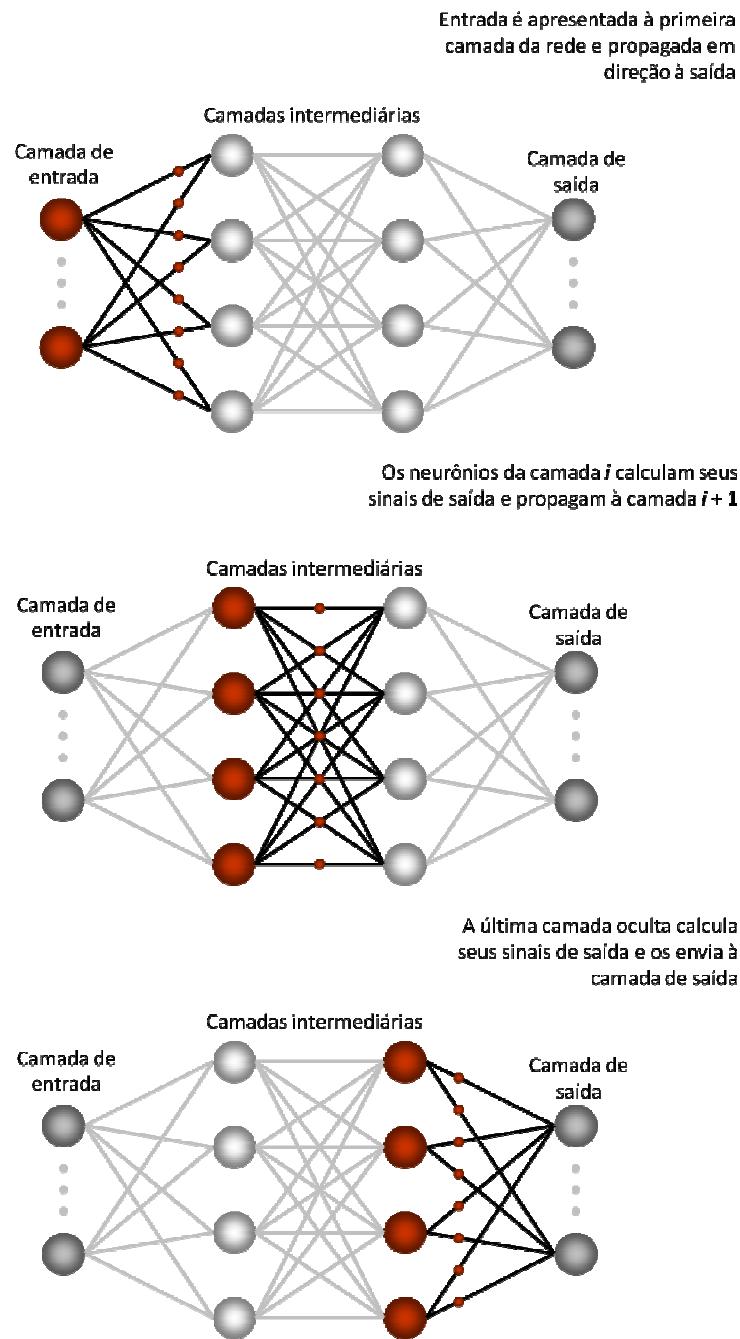
Exercício: Calcule a atualização de Δv_{ij} quando a função de ativação for a tangente hiperbólica.

ALGORITMO BACKPROPAGATION

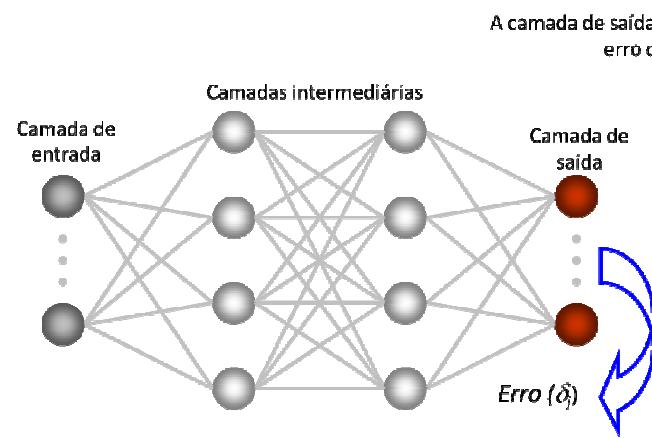
Neste algoritmo, o treinamento da RNA é feito em duas fases:

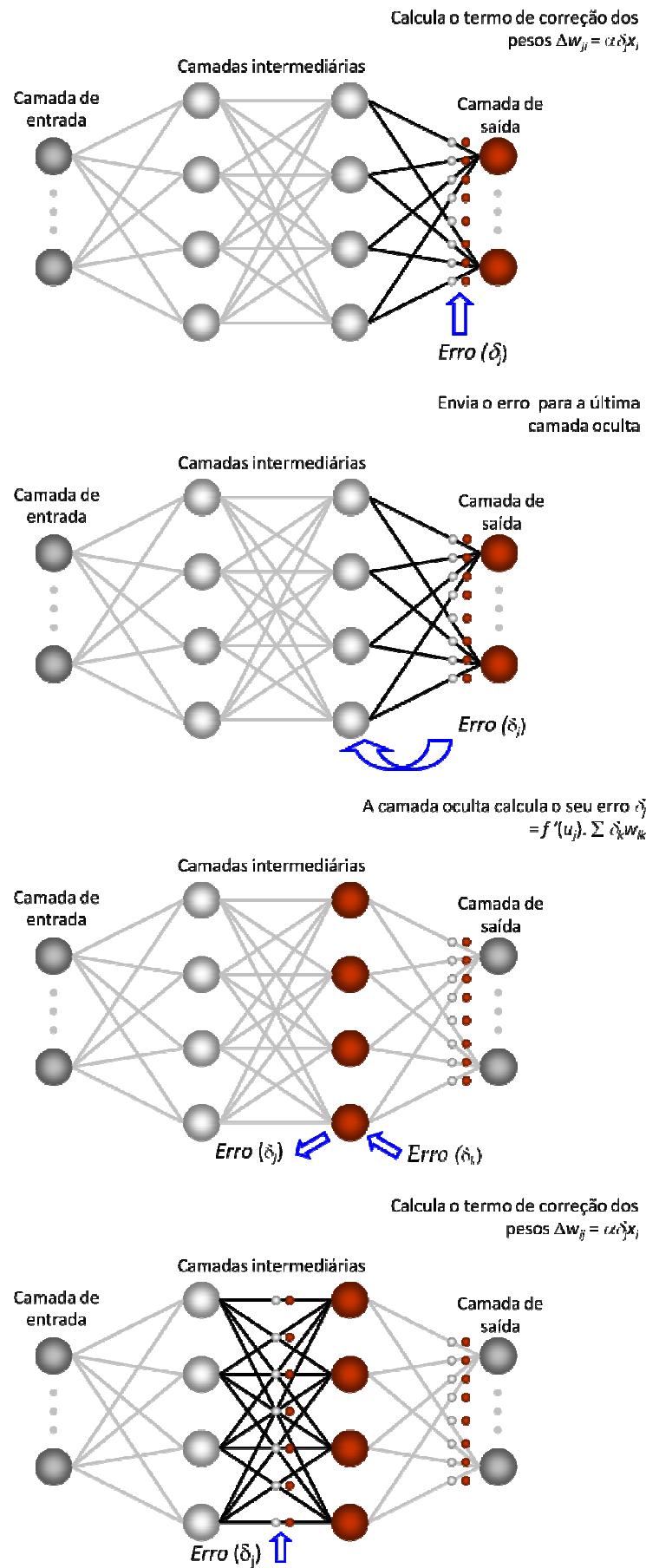


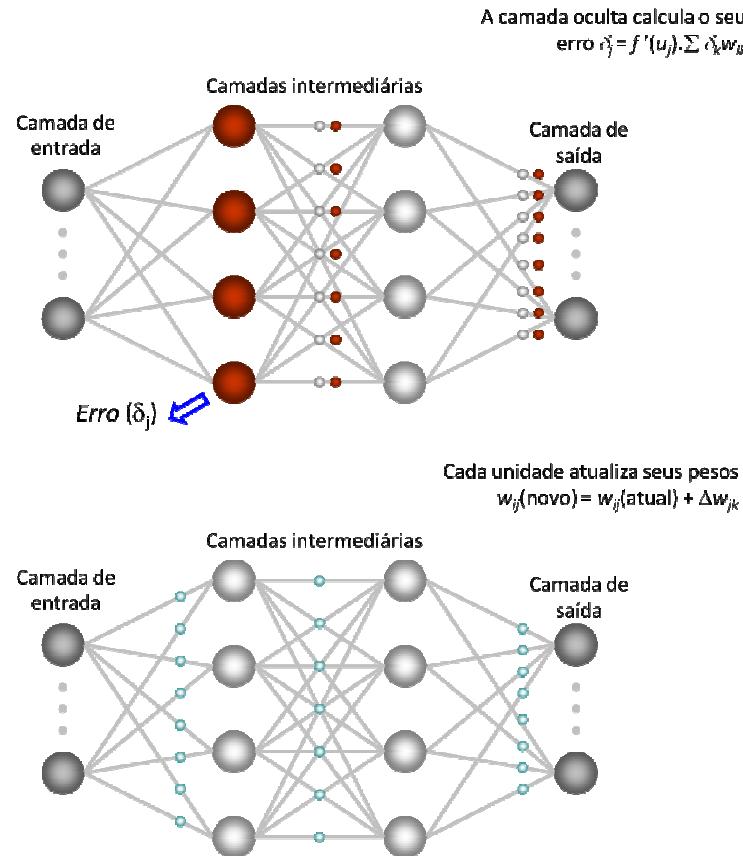
Fase forward:



Fase backward:

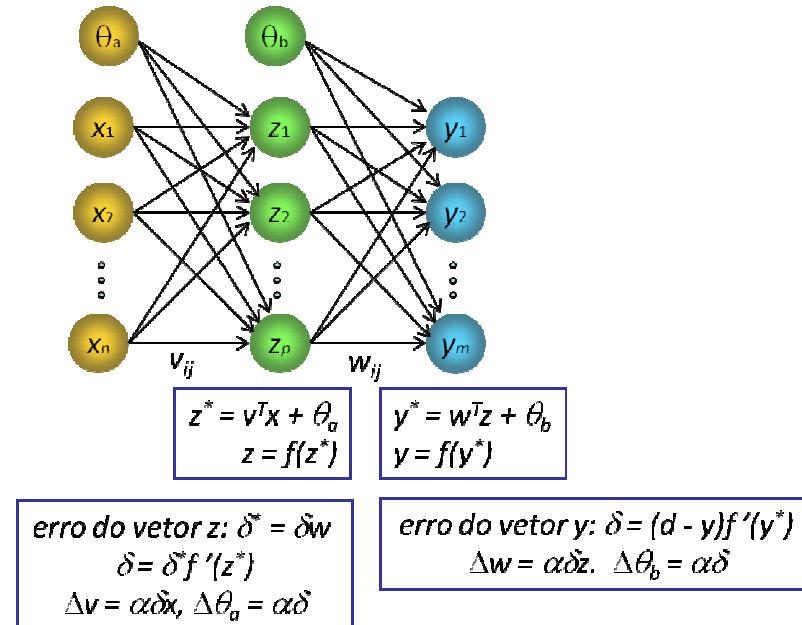






Repete-se o processo enquanto a rede não aprender o padrão de entrada

Resumindo:



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:

$$z_j^* = \sum_i v_{ij}x_i + \theta a_j \Rightarrow z_j = 1/(1 + e^{-z_j^*})$$
, onde $j = 1, \dots, p$, $i = 1, \dots, n$

$$y_k^* = \sum_j w_{jk}z_j + \theta b_k \Rightarrow y_k = 1/(1 + e^{-y_k^*})$$
, onde $k = 1, 2, \dots, m$
 2. Calcule as correções das conexões da camada de saída:

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

$$\Delta \theta b_k = \alpha y_k(1 - y_k)(d_k - y_k) \Rightarrow \theta b_k = \theta b_k + \Delta \theta b_k$$
 3. Calcule as correções das conexões da camada escondida:

$$\Delta v_{ij} = \alpha \sum_k [(d_k - y_k)y_k(1 - y_k)w_{jk}]z_j(1 - z_j)x_i \Rightarrow v_{ij} = v_{ij} + \Delta v_{ij}$$

$$\Delta \theta a_j = \alpha \sum_k [(d_k - y_k)y_k(1 - y_k)w_{jk}]z_j(1 - z_j) \Rightarrow \theta a_j = \theta a_j + \Delta \theta a_j$$
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:

$$z_j^* = \sum_i v_{ij}x_i + \theta a_j \Rightarrow z_j = 1/(1 + e^{-z_j^*})$$
, onde $j = 1, \dots, p$, $i = 1, \dots, n$

$$y^* = \sum_j w_j z_j + \theta b \Rightarrow y = 1/(1 + e^{-y^*})$$
 2. Calcule as correções das conexões da camada de saída:

$$\Delta w_j = \alpha y(1 - y)(d - y)z_j \Rightarrow w_j = w_j + \Delta w_j$$

$$\Delta \theta b = \alpha y(1 - y)(d - y) \Rightarrow \theta b = \theta b + \Delta \theta b$$
 3. Calcule as correções das conexões da camada escondida:

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

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

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

$$y_k^* = \sum_j w_{jk}z_j + \theta b_k \Rightarrow y_k = \tanh(y_k^*)$$
, onde $k = 1, 2, \dots, m$
 2. Calcule as correções das conexões da camada de saída:

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

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

3. Calcule as correções das conexões da camada escondida:

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

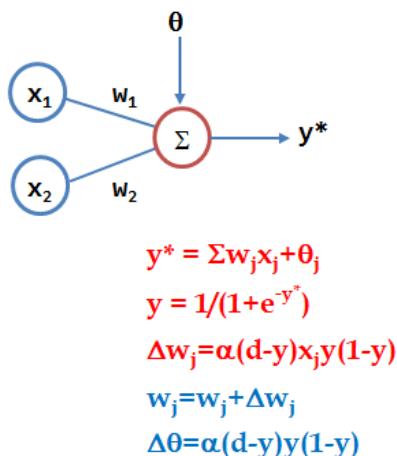
$$\Delta\theta a_j = \alpha \sum_k [(d_k - y_k)(1 - y_k^2)w_{jk}] (1 - z_j^2) \Rightarrow \theta a_j = \theta a_j + \Delta\theta a_j$$

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

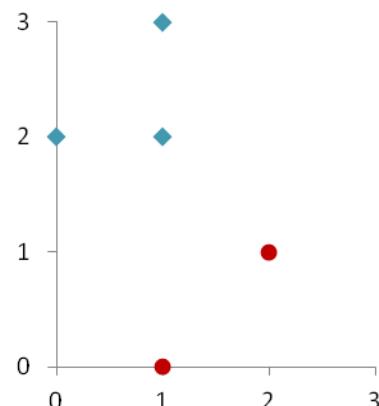
Exercícios:

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:

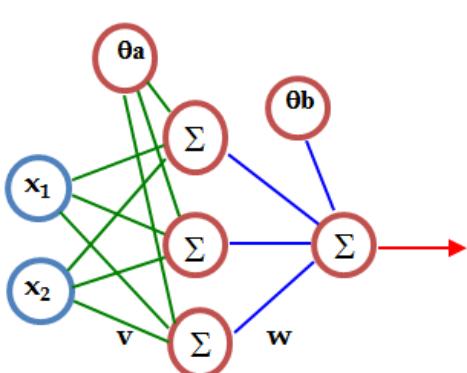
	x ₁	x ₂	d
A ₁	0	2	1
A ₂	1	2	1
A ₃	1	3	1
B ₁	1	0	0
B ₂	2	1	0



Pesos iniciais:
 $w_1=0,9; w_2=-0,9; \theta=0,9; \alpha=1$



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.

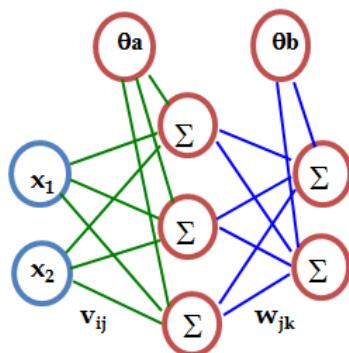


$$\begin{aligned}
 z_j^* &= \Sigma v_{ij} x_i + \theta_a \\
 z_j &= 1/(1+e^{-z_j^*}) \\
 y^* &= \Sigma w_j z_j + \theta_b \\
 y &= 1/(1+e^{-y^*}) \\
 \Delta w_j &= \alpha(d-y)z_jy(1-y) \\
 w_j &= w_j + \Delta w_j \\
 \Delta \theta_b &= \alpha(d-y)y(1-y) \\
 \Delta v_{ij} &= \alpha y(1-y)(d-y)w_j z_j(1-z_j)x_i \\
 v_{ij} &= v_{ij} + \Delta v_{ij} \\
 \Delta \theta_a &= \alpha y(1-y)(d-y)w_j z_j(1-z_j)
 \end{aligned}$$

Pesos iniciais:

θ _a	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
θ _b	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.



$$z_j^* = \sum v_{ij}x_i + \theta a_j; z_j = 1/(1+e^{-z_j^*})$$

$$y_k^* = \sum w_{jk}z_j + \theta b_k; y_k = 1/(1+e^{-y_k^*})$$

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

$$w_{jk} = w_{jk} + \Delta w_{jk}$$

$$\Delta \theta b_k = \alpha(d_k - y_k)y_k(1-y_k)$$

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

$$v_{ij} = v_{ij} + \Delta v_{ij}$$

$$\Delta \theta a_j = \alpha \sum_k [y_k(1-y_k)(d_k - y_k)w_{jk}]z_j(1-z_j)$$

x ₁	x ₂	d ₁	d ₂
-1	-1	0	0
-1	1	0	1
1	-1	0	0
1	1	1	1

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.

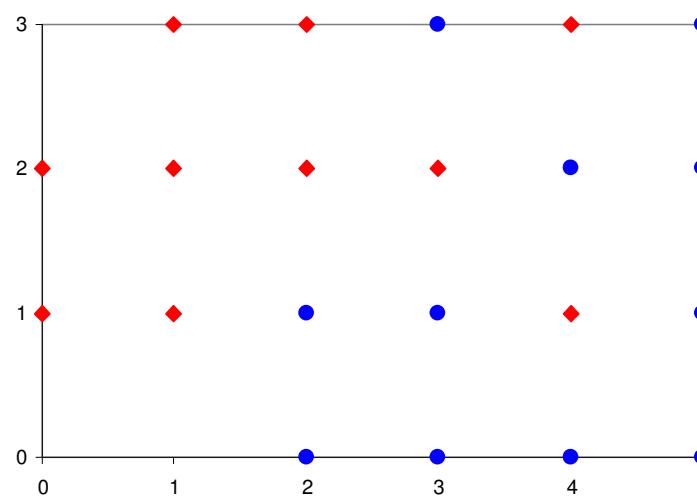
$$X = \begin{pmatrix} A & -0,92 & -0,29 \\ B & -0,28 & -0,32 \\ C & 0,21 & 0,07 \\ D & 0,81 & -1 \\ E & -0,5 & 1 \\ F & 0,62 & 0,87 \\ G & -0,71 & -0,87 \\ H & 0,42 & 0,09 \end{pmatrix}, \text{ onde } d = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

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.

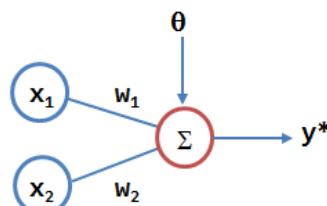
Exemplo:

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

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
x ₁	x ₂	d



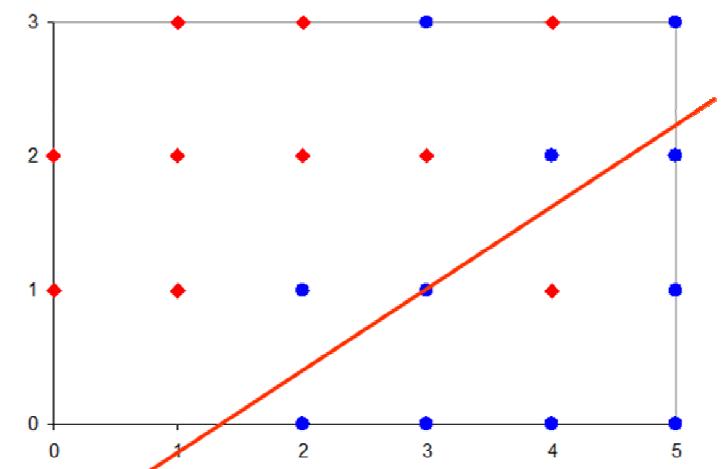
Usando o Perceptron com 2 camadas, e aprendizagem Backpropagation, temos os seguintes resultados:



Na 10^a iteração, temos:

$$w_1=2,95 \quad w_2=-2,79 \quad \theta=-1,94$$

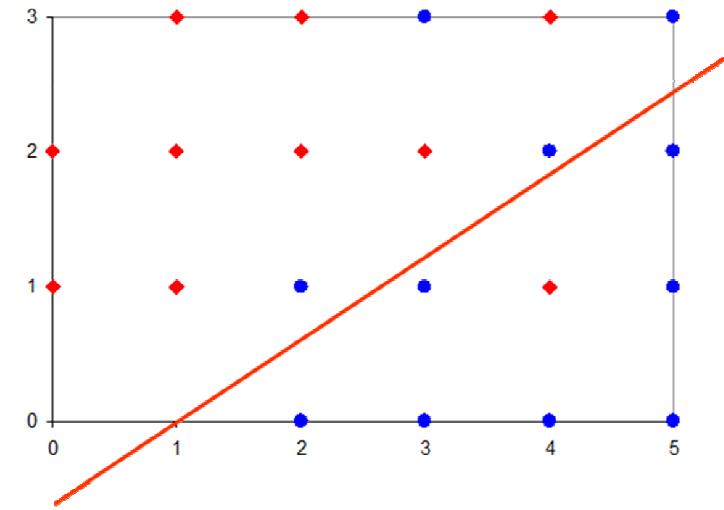
$$\text{erro}=1,52$$



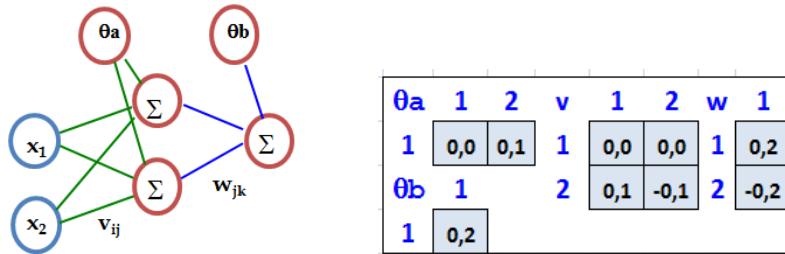
Na 100^a iteração, temos:

$$w_1=4,19 \quad w_2=-4,08 \quad \theta=-4,39$$

$$\text{erro}=1,38$$



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



Na 100^a iteração, temos: erro=1,07

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

Equações discriminantes encontradas:

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

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

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

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

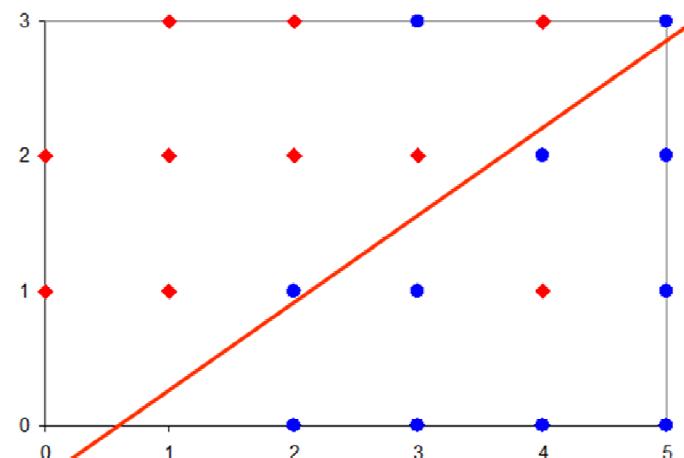
$$-2,74(-2,86x_1+3,71x_2+1,39)-0,19=0$$

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

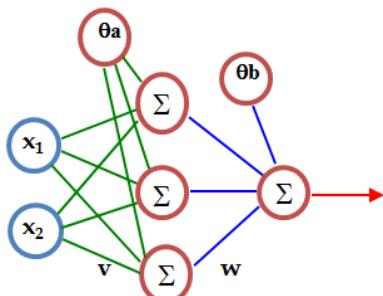
$$15,96x_1-20,57x_2=8,12$$

Na 500^a iteração, temos:

erro=0,966



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

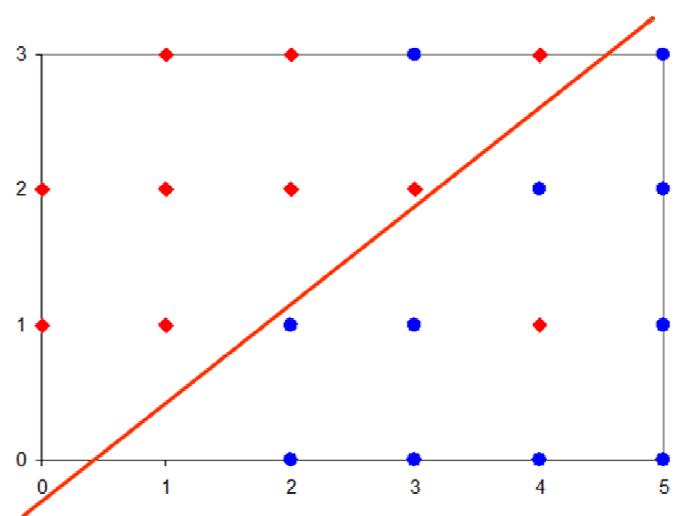


Na 100^a iteração, temos:

erro=1,05

Na 500^a 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;
- Atualização **exponencial**: $\alpha = \alpha_0 e^{-kt}$, onde α_0 é a taxa de aprendizagem inicial e $k \in (0, 1)$ tem valor próximo de 0 (geralmente usamos 0,1 ou 0,05).
- Atualização **linear**: $\alpha = \alpha k$, onde $k \in (0, 1)$. Usamos um número k próximo de 1 (geralmente 0,99 ou 0,95), para a rede aprender “aos poucos”.

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 $0 \leq i \leq k$, 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, $0 \leq i \leq k$, 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, $0 \leq i \leq k$, 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:
 - o indica quando não há mais extração de informação da rede;
 - o porém, em alguns casos pode haver parada prematura da rede.
- Parada com validação cruzada:
 - o dois conjuntos: treinamento e validação;
 - o conjunto de validação: geralmente entre 10 e 25% do total;
 - o com certa freqüência (em geral, de 5 a 10 iterações) verifica-se o desempenho no conjunto de validação;
 - o quando o erro aumenta, pára-se o treinamento .

Algumas heurísticas para melhorar a convergência de uma RNA:

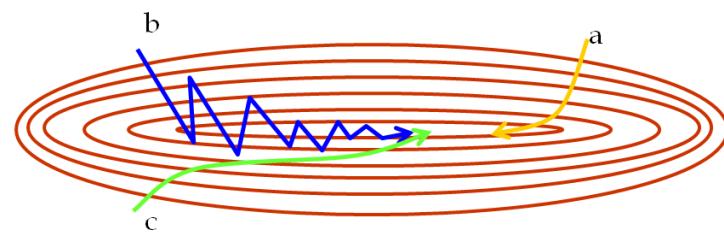
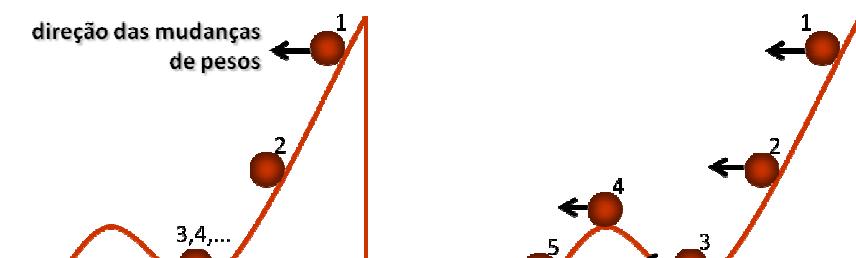
- Normalizar os dados em relação à faixa de ativação da rede;
- Usar não-linearidade do tipo tanh ou sigmóide;
- Normalizar o sinal desejado ligeiramente acima/abaixo do limite (**por exemplo $\pm 0,9$ e não ± 1**);
- Sempre ter mais padrões de treinamento que pesos;
- Usar validação cruzada para parar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

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

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"**);

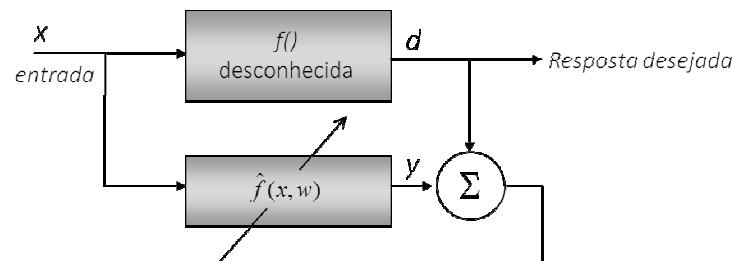
Quando a taxa de aprendizagem é alta, mas o termo momento é considerado, o mínimo é alcançado rapidamente (**trajetória "c"**) [Krose et al., 1993].

Exercício: Resolva o exercício 2 da página 27 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.



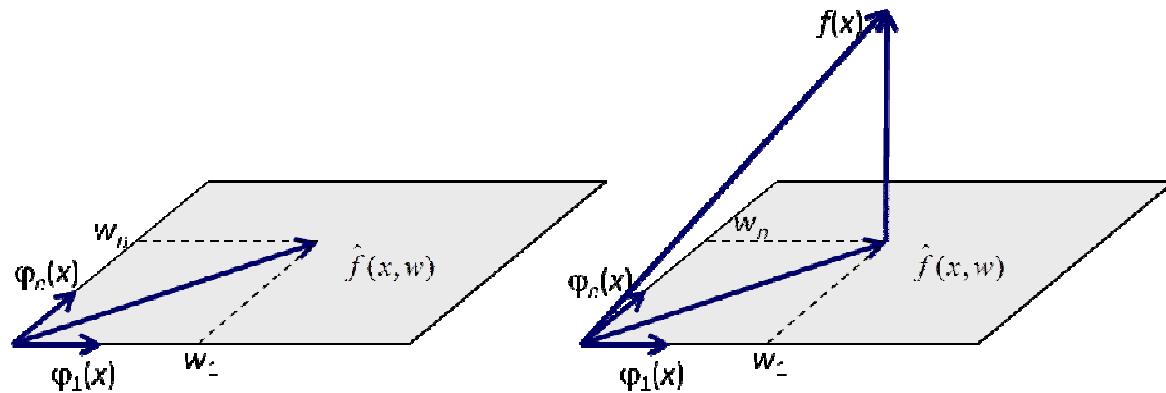
Teorema da projeção linear:

Em uma área compacta S do espaço de entrada descrever uma função $f(x)$, pela [combinação de funções \$\varphi\(x\)\$](#) mais simples:

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

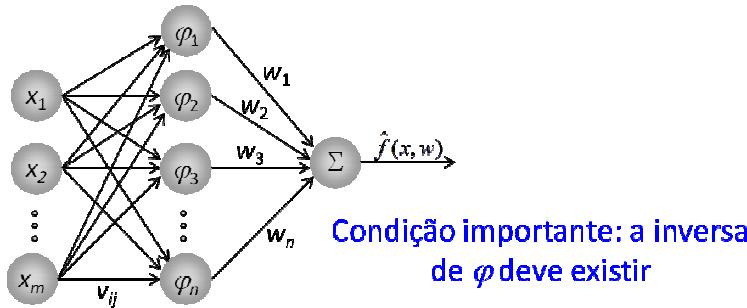
onde w_i são elementos reais do vetor $w = [w_1, \dots, w_n]$ tais que $|f(x) - \hat{f}(x, w)| < \varepsilon$ e ε pode ser arbitrariamente pequeno. A função é chamada de [aproximante](#) e as [funções \$\{\varphi\(x\)\}\$](#) são chamadas de [funções elementares](#) $\hat{f}(x, w)$.

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:

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



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.

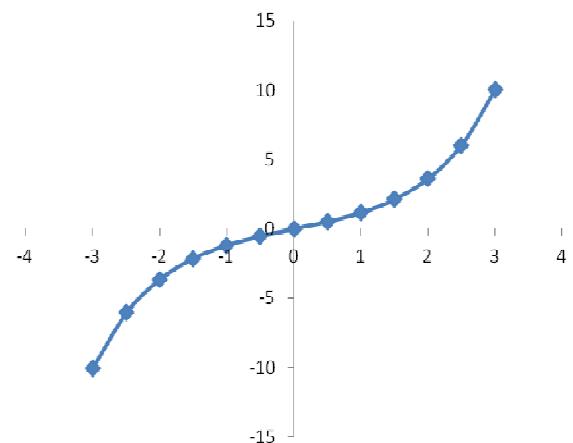
MLPs realizam aproximação de funções com um *conjunto adaptativo de bases*, determinado a partir dos dados *entrada-saída*. As bases são alteradas em função dos dados: o espaço de projeção é dependente dos dados.

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:

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



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

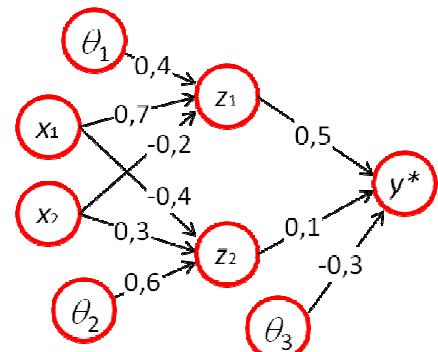
a)	x_1	x_2	x_3	d
	1	1	1	1
	1	1	0	0
	1	0	1	0
	0	1	1	0

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

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 $\alpha = 0,25$ 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 $\alpha = 0,25$, função de ativação tanh e aprendizagem backpropagation)



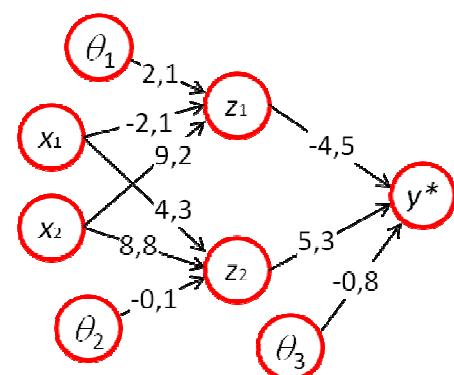
5. Dada a rede MLP ao lado, calcule os novos pesos e bias na seguinte sequência de apresentação de dados x_i de entrada (use $\alpha = 0,4$, função de ativação sigmóide, e calcule o erro de classificação em cada entrada):

 - 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 $d = -0,7$.

```

graph LR
    x1((x1)) -->|2,1| z1((z1))
    x2((x2)) -->|-2,1| z1
    x3((x3)) -->|9,2| z1
    z1 -->|-4,5| "*"
    style x1 fill:#fff,stroke:#000,stroke-width:1px
    style x2 fill:#fff,stroke:#000,stroke-width:1px
    style x3 fill:#fff,stroke:#000,stroke-width:1px
    style z1 fill:#fff,stroke:#000,stroke-width:1px
    style "* fill:#fff,stroke:#000,stroke-width:1px
    style theta1 fill:#fff,stroke:#000,stroke-width:1px
    style theta2 fill:#fff,stroke:#000,stroke-width:1px
    style theta3 fill:#fff,stroke:#000,stroke-width:1px
  
```

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 [Souto, 2020], o nó de saída faz uma combinação de duas funções:

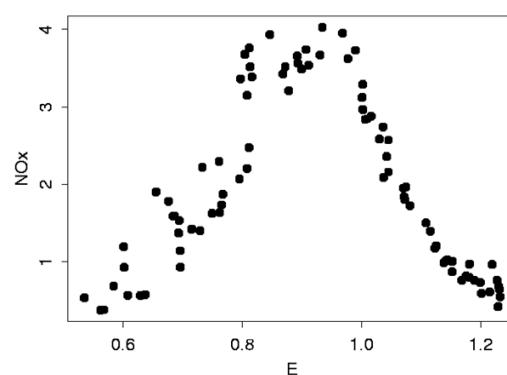
$$y = f \tanh_1(x) + g \tanh_2(x) + a, \text{ onde}$$

$$\tanh_1(x) = \tanh(dx + b)$$

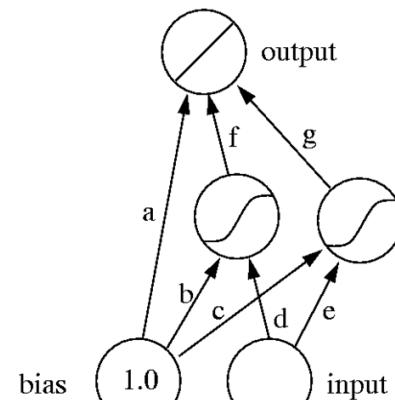
$$\tanh_2(x) = \tanh(ex + c)$$

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

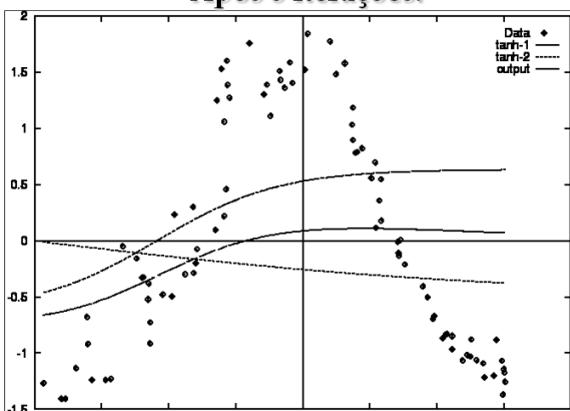
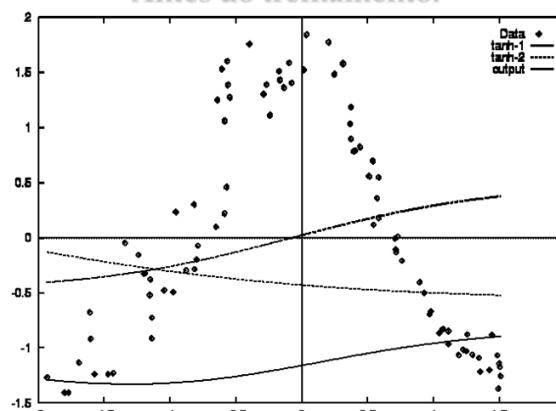
Dados:



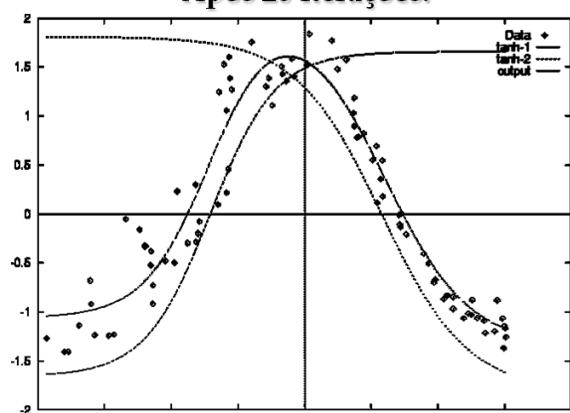
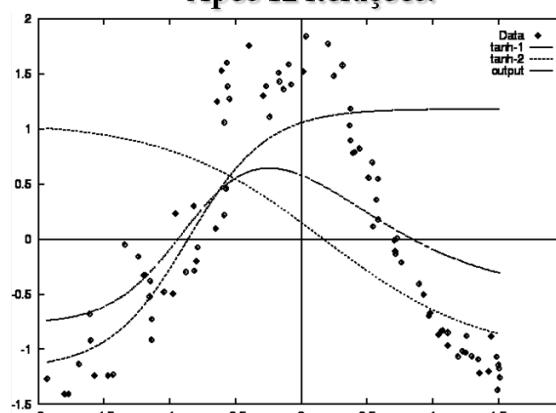
Arquitetura da rede:



Antes do treinamento:



Após 12 iterações:



Exemplo de aplicação de MLP para classificação de imagens

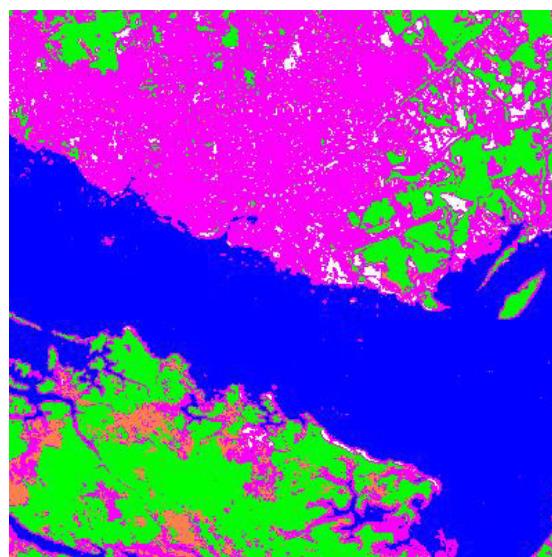
Um dos trabalhos pioneiros neste estudo compara o desempenho de métodos estatísticos de classificação de dados multiespectrais, com o desempenho de classificadores por redes neurais [Benediktsson, Swain e Ersoy, 1989].

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

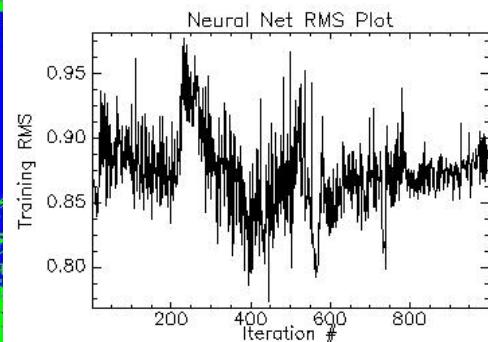
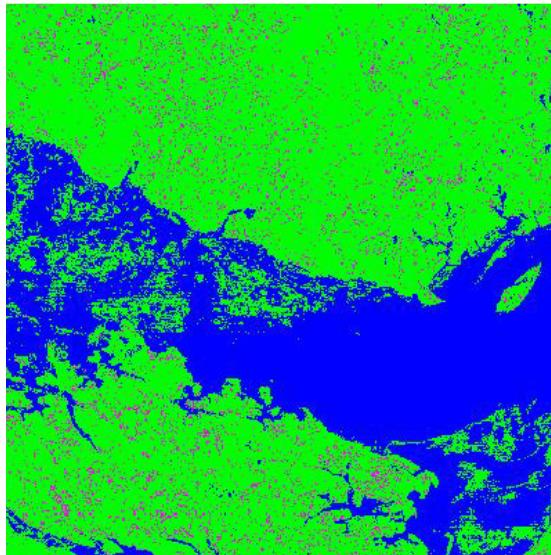
A seguir, temos as imagens e respectivas classificações encontradas pelos autores:



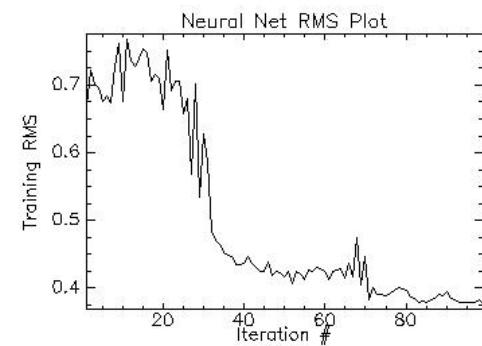
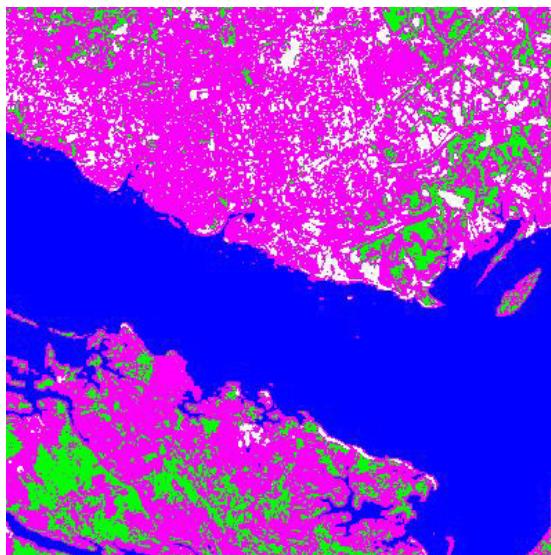
Foto aérea digitalizada, com os dados que devem ser classificados



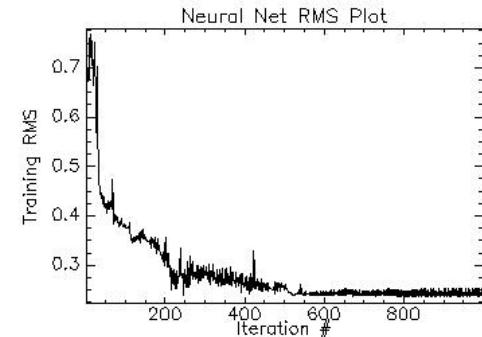
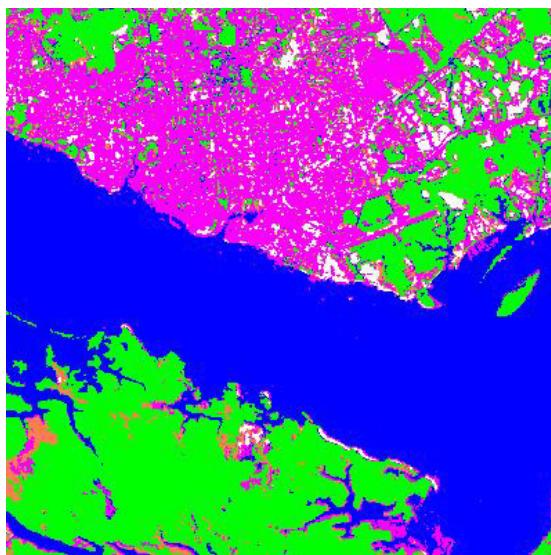
Aplicação do método de Máxima Verossimilhança (MAXVER). Dados classificados corretamente: 71,19%



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



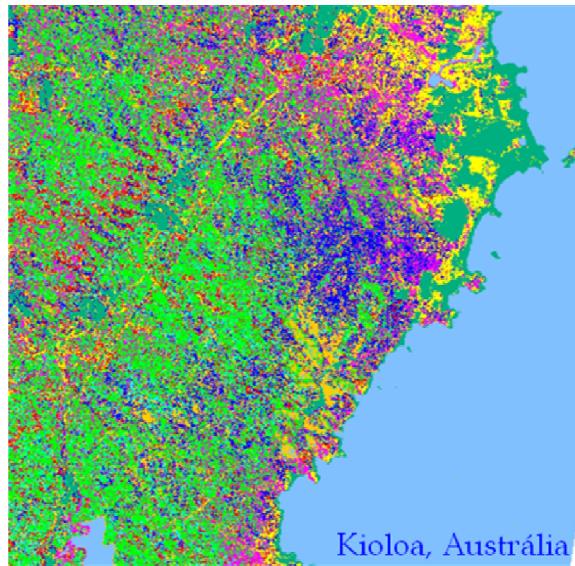
Perceptron com Backpropagation (usando 1 amostra de cada classe que se deseja classificar).
Classificação correta = 58,52%



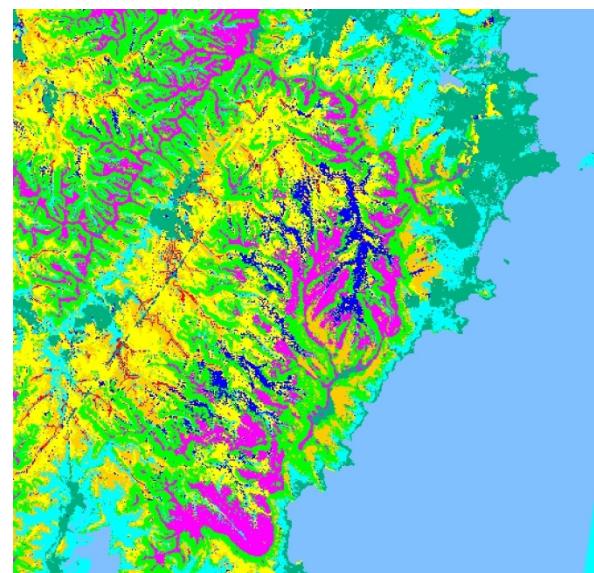
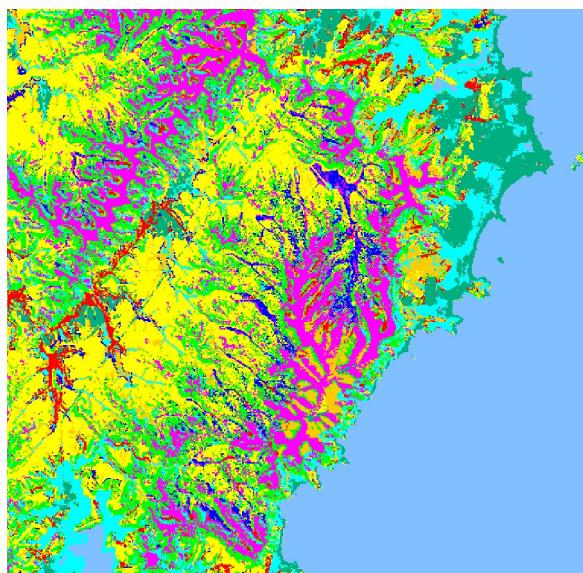
Perceptron com Backpropagation (usando mais amostras de cada classe que se deseja classificar).
Classificação correta = 95,19%

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

As imagens a seguir mostram os resultados encontrados:



Máxima Verossimilhança. Classificação correta: 40%

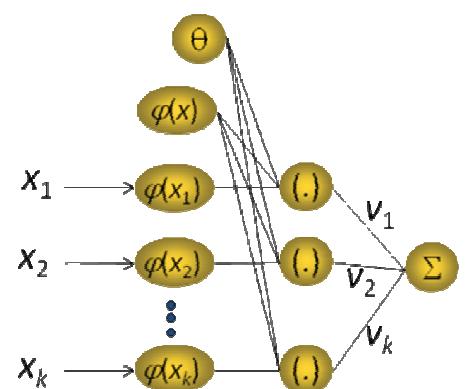


Árvore de decisão. Classificação correta: 75%; MLP + BackPropagation. Classificação correta: 85%

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].

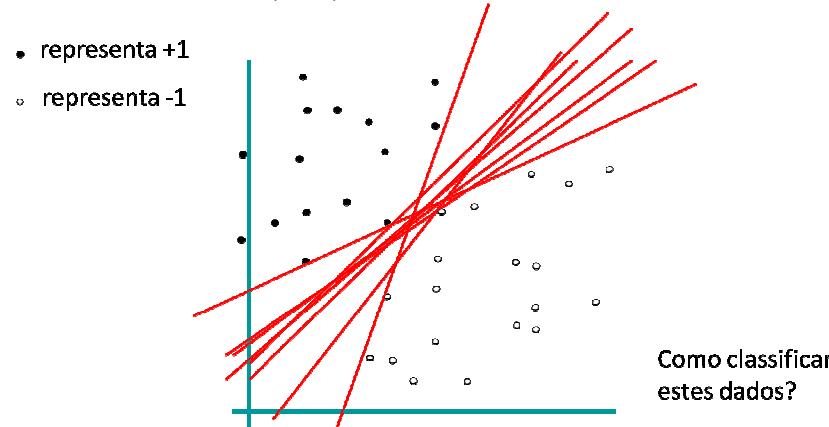
Funções não-lineares são apresentadas para separação não-linear de dados, usando hiperplanos de margens fixas e flexíveis [Cortes e Vapnik, 1995]. Mais detalhes podem ser encontrados na página de Andrew W. Moore: www.cs.cmu.edu/~awm.



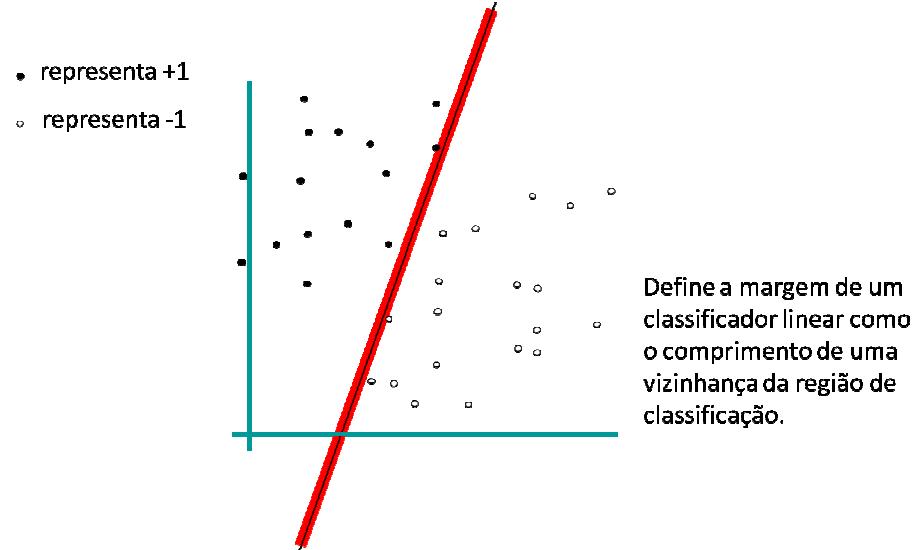
Esta rede neural também pode ser usada na resolução dos seguintes problemas:

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

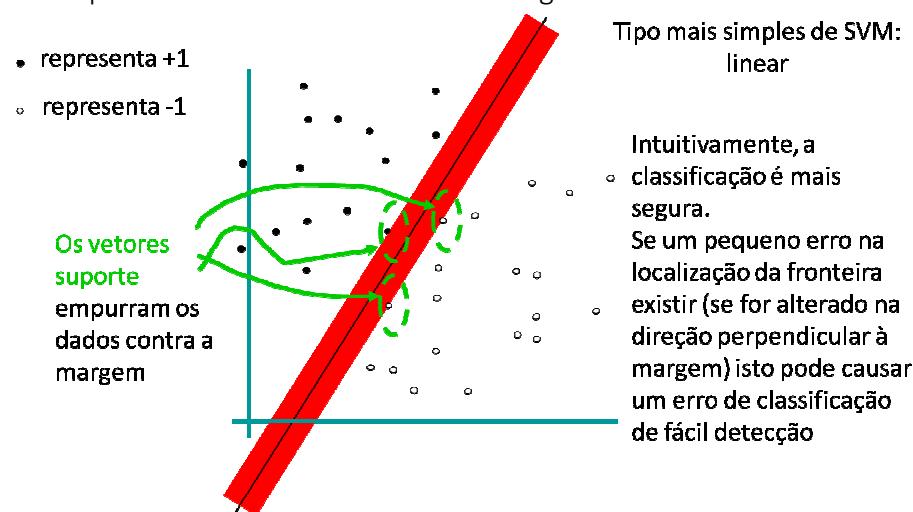
Um classificador simples pode obter diversos resultados, como mostra a figura abaixo:



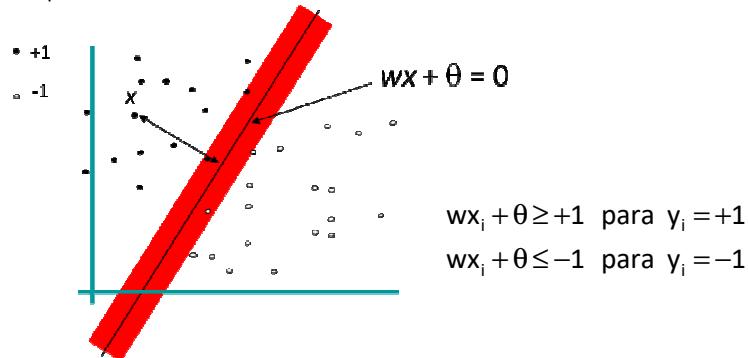
Um classificador com margem pode obtem a seguinte configuração:



Este tipo de classificador define uma margem máxima:



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



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

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

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 (PPQ):

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

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:

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

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

Logo, o hiperplano classificador fica com as seguintes características:

$$\sum_{i=1}^n \alpha_i y_i = 0 \text{ e } w = \sum_{i=1}^n \alpha_i y_i x_i$$

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

$$\alpha_i [y_i(wx_i + \theta) - 1], i = 1, \dots, k.$$

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

Encontrando o Problema de otimização dual do Lagrangeano temos:

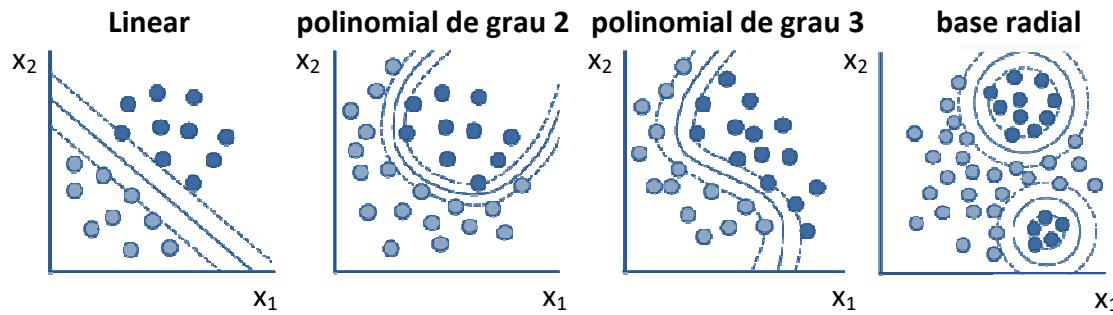
$$\max W(\alpha) = \sum_{i=1}^k \alpha_i - \frac{1}{2} \sum_{i,j=1}^k \alpha_i \alpha_j y_i y_j x_i x_j$$

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

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

As funções básicas de uma SVM podem ser:

- função polinomial de x_k ;
- radial basis function (RBF) de x_k : $\phi(x_k) = e^{-\frac{|x_k - c_j|^2}{\sigma^2}}$;
- neural de x_k : $\phi(x_k) = \tanh(x_k)$;
- funções Kernel (núcleo): $K(x_i, x_j) = \phi(x_i)\phi(x_j)$.



Fonte: <https://www.r-bloggers.com/2019/10/support-vector-machines-with-the-mlr-package/>

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

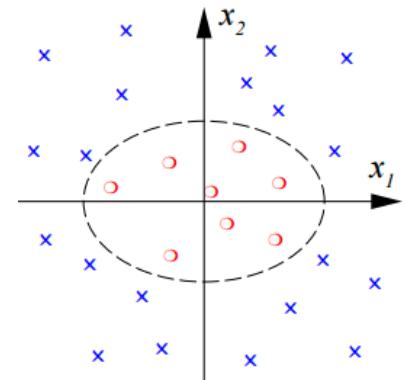
$$\max W(\alpha) = \sum_{i=1}^k \alpha_i - \frac{1}{2} \sum_{i,j=1}^k \alpha_i \alpha_j y_i y_j \phi(x_i)\phi(x_j)$$

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

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

Na função w , como $w = \sum_{i=1}^n \alpha_i y_i x_i$, então $w = \sum_{i=1}^n \alpha_i y_i \phi(x_i) = 0$.

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



Condição de Mercer

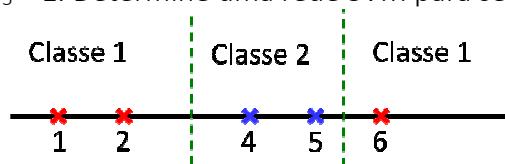
Para decompor uma função *Kernel* através de um produto interno $K(x, y) = \phi(x)\phi(y)$, $K(x, y)$ deve ser semi-definida positiva, ou seja, para qualquer função $f(x)$ que $\int f^2(x) dx$ seja finito:

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

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

Exercícios:

- 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 $y_5 = 1$. Determine uma rede SVM para separar estes dados.



2. Determine uma função Kernel e resolva o problema de classificação da função lógica “OU” exclusiva com uma rede SVM.

x₁	x₂	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



No trabalho de Jason Weston, 60.000 exemplos de treinamento e 10.000 exemplos de testes foram usados para treinar uma SVM com função Kernel linear (com 8,5% de erro de testes) e função polinomial (com 1% de erro de testes).

http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf

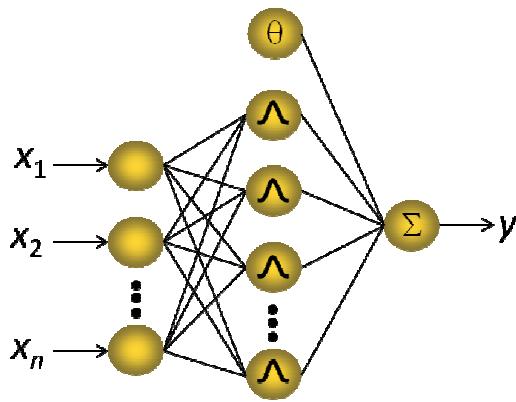
2.6. REDES DE BASES RADIAIS (RBF)

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.

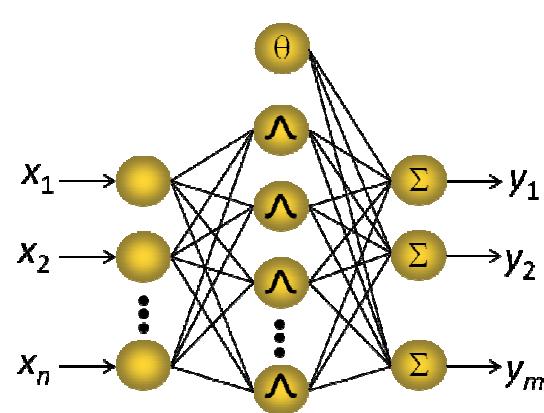
O funcionamento deste tipo de rede difere significativamente, pois elas contêm uma camada escondida com funções de bases radiais e a ativação dos neurônios da camada escondida:

- 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.



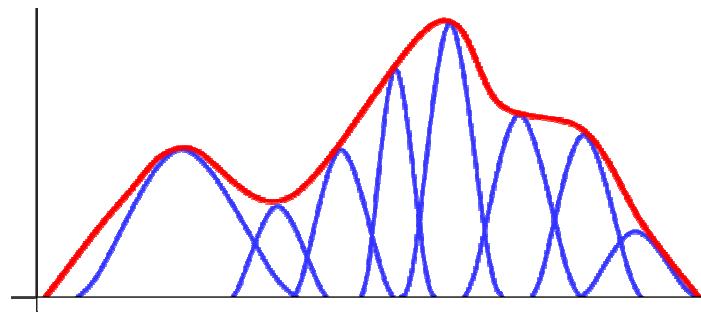
RBF com *n* entradas e 1 saída



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.

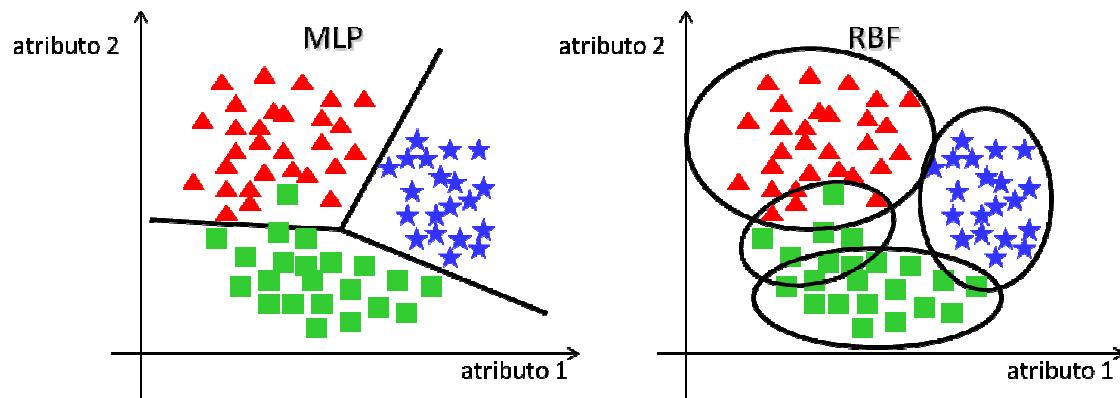


O aprendizado de uma RBF é feito em três etapas:

- 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:

$$h(x_n) = d_n, \quad n = 1, \dots, N.$$

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

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

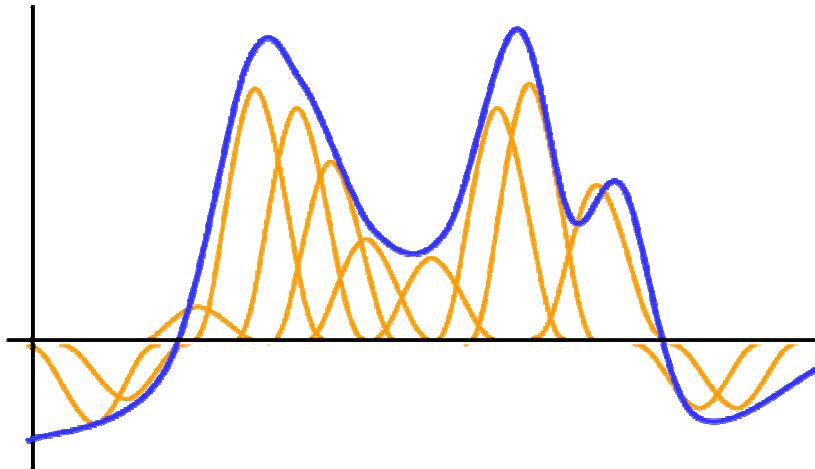
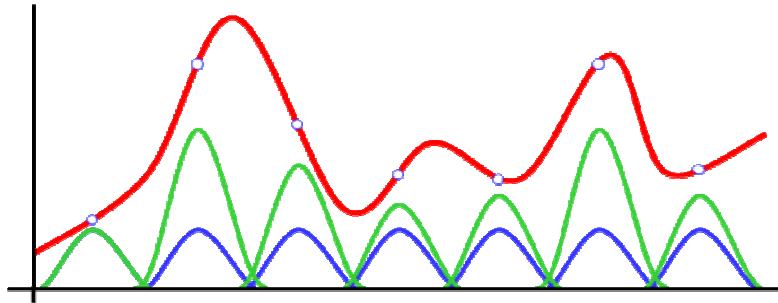
uma para cada ponto da série, onde $\varphi(\cdot)$ é uma função não-linear.

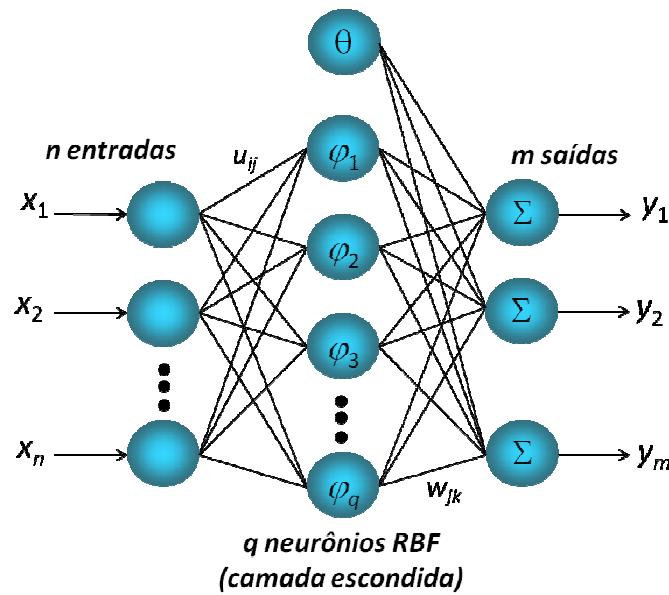
A saída é uma função h :

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

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:

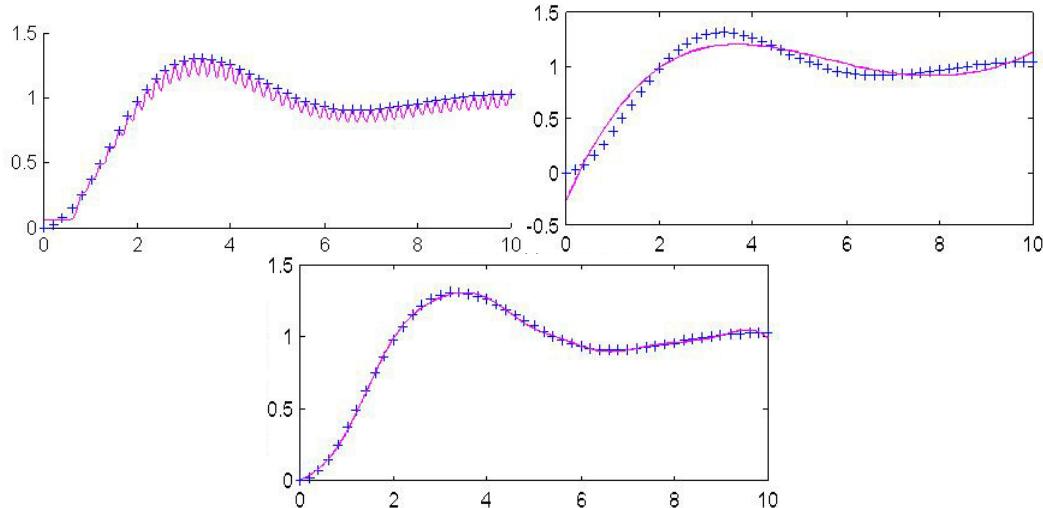




A função gaussiana que vamos utilizar é $\varphi_j = e^{-\frac{1}{2\sigma_j^2} \|x - u_j\|^2}$, onde σ_j representa o raio da função base do neurônio j e u_j é o vetor com as coordenadas do centro do neurônio. Os valores dos raios “extendem” a função de base radial, e a saída da rede é uma combinação das funções de bases radiais com os pesos ajustados:

$$y_k = \sum_{j=1}^{q+1} w_{jk} \varphi_j.$$

Exemplos:



Algoritmo:

0. Considere uma base de dados (x_i, d_i) , $i = 1, 2, \dots, p$, onde x_i é um exemplo da base de dados e d é o vetor de saídas desejadas correspondentes.
1. Defina o número q de neurônios ocultos (bases radiais), em geral escolhe-se $q \leq n$. Selecione aleatoriamente q exemplos do conjunto de dados, e faça a seguinte atribuição: $u_j = x_j$, $j = 1, 2, \dots, q$.
2. Especifique o(s) valor(es) do(s) raio(s) da função de base radial, σ_j ; cada neurônio pode ter um raio diferente, para termos maior diversificação da RBF.

3. Para cada exemplo da base de dados x_i , onde $i = 1, 2, \dots, p$, execute os passos 4 e 5:

4. Calcule a ativação de cada neurônio j da camada escondida:

$$\varphi_j = e^{-\frac{1}{2\sigma_j^2} \|x_i - u_j\|^2}$$

5. Atribua os valores das ativações dos neurônios na matriz G :

$$G_{i,j} = \varphi_j, \text{ e } G_{i,q+1} = \theta$$

6. Após a apresentação de todos os exemplos, calcule os pesos da saída:

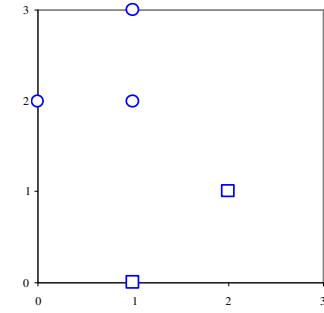
$$w = (G^T G)^{-1} G^T d$$

7. Calcule a saída de cada exemplo: $y_k = \sum_{j=1}^{q+1} w_{jk} \varphi_j$. Calcule o erro de classificação.

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 e B 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 $\sigma = \sqrt{0,5}$, $j = 2$ (centros), $\theta = 1$, 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 e B 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 $x^3 + 2x^2 + 4x - 12$ usando 21 pontos no intervalo $[-3, 3]$.
4. Utilize uma rede RBF para a função “OU EXCLUSIVO” com os seguintes parâmetros: $\sigma = \sqrt{0,5}$, $u = \{(1, 1), (-1, -1)\}$, $j = 2$ (centros), $\theta = 1$, saída com 1 neurônio.

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

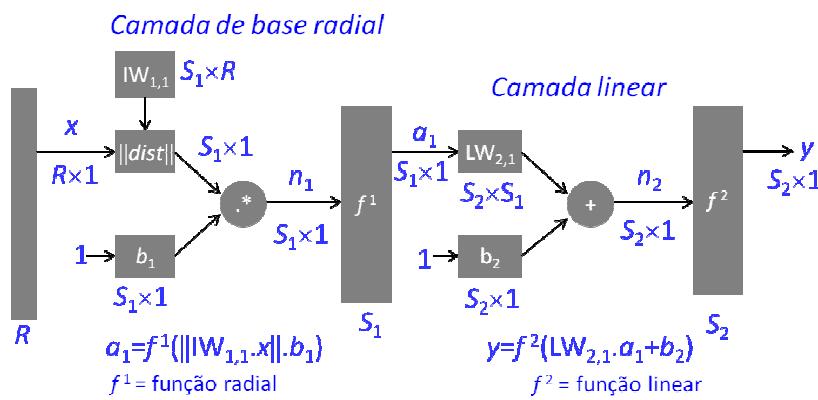
5. Utilizando a Rede Neural RBF com 4 centros, encontre os pesos da rede para aproximar a função $\sin(x) + 4\cos(x) - 1$ usando 21 pontos no intervalo $[-2, 4]$.
6. Utilize uma RBF para resolver o problema de classificação dos 22 pontos da página 19. Utilize $\sigma = 5$, $u = \{(1, 3), (4, 0), (5, 3)\}$, $j = 3$ (centros), $\theta = 1$, saída com 1 neurônio.

7. Utilize uma RBF com centros $\mathbf{u} = \{(0, 0), (1, 1), (3, 3)\}$ para classificar os dados abaixo. Use $\sigma = 5$ e $\theta = 1$.

	x_1	x_2	d
A₁	0	1	0
A₂	1	2	0
B₁	0	3	1
B₂	2	3	1
B₃	1	3	1

8. Resolva o problema 7 com saídas desejadas bipolares.

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.

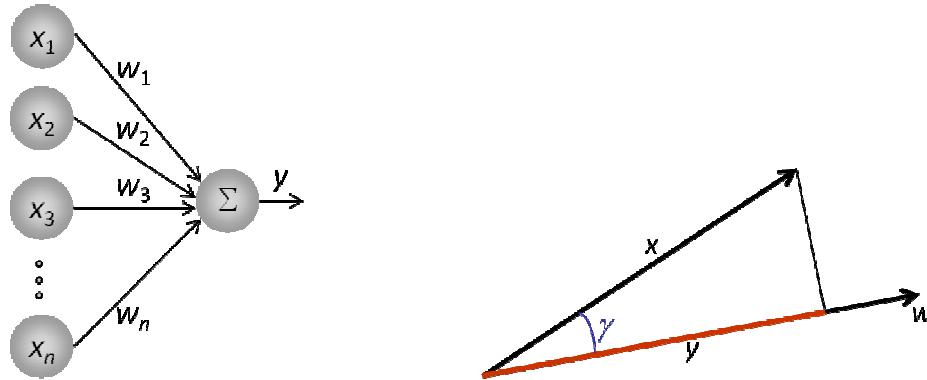
Em notação vetorial a saída do neurônio é

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

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.



Algoritmo:

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

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

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

	x_1	x_2	d

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

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

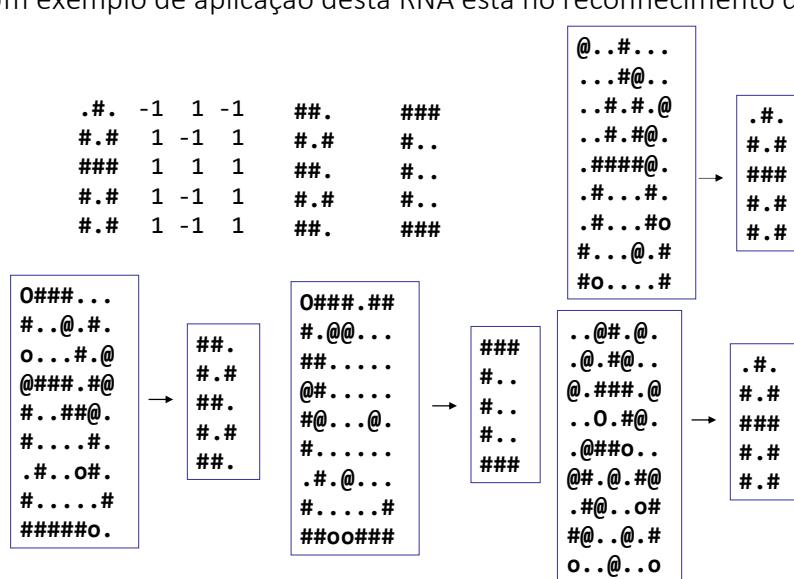
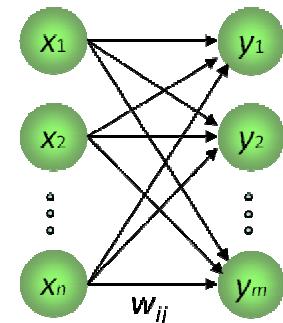
x_1	x_2	x_3	d
1	1	1	1
1	1	-1	-1
1	-1	1	-1
-1	1	1	-1

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.

Um exemplo de aplicacão desta RNA está no reconhecimento de caracteres;



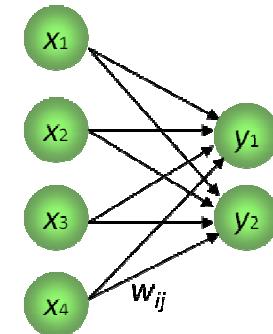
Algoritmo:

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

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.

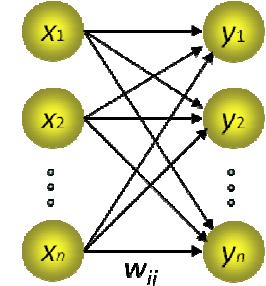
2.9. REDE NEURAL AUTOASSOCIATIVA

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:

$$\begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 \end{pmatrix}$$

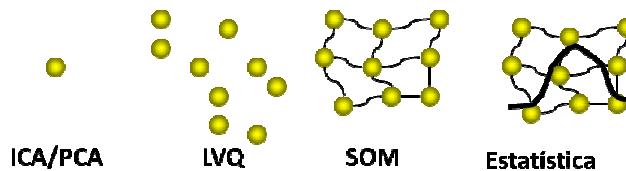
Usando a matriz de pesos W, verifique se os vetores acima são mesmo reconhecidos corretamente pela rede.

O vetor $(-1 \ 1 \ 1 \ -1)$ é reconhecido corretamente pela rede?

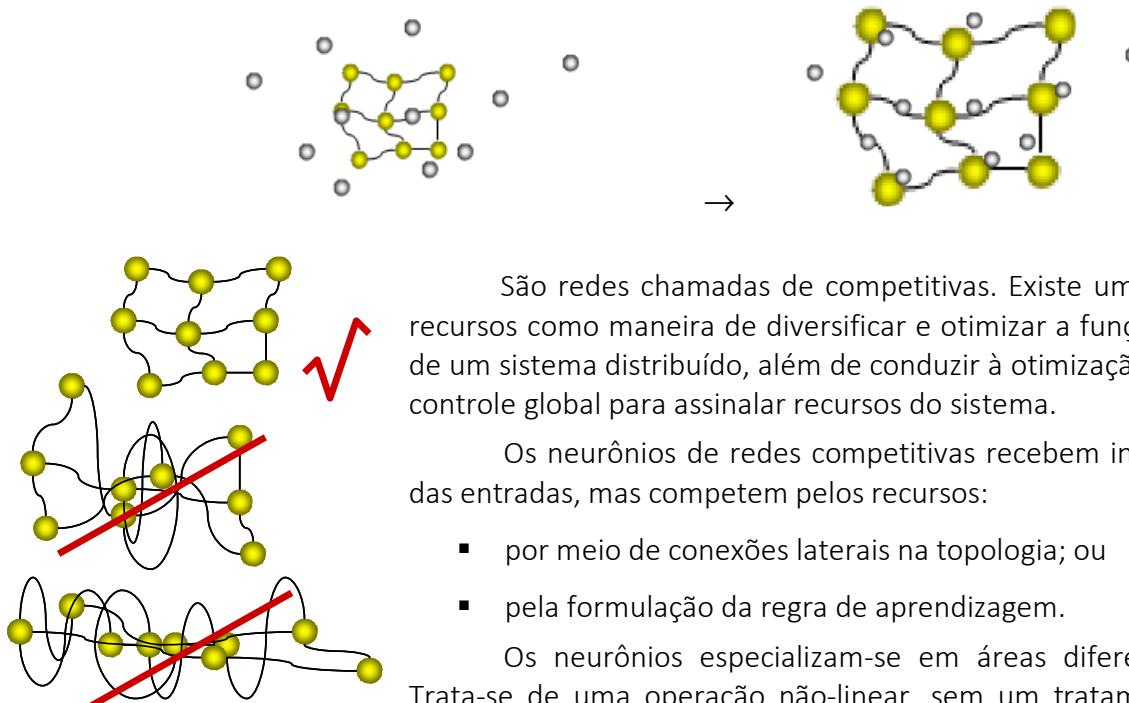
2. Use uma rede autoassociativa para reconhecer o vetor $V_1=(1,1,1,1,1,1)$. Teste a rede com o vetor V_1 . Teste a rede com o vetor $T=(1,1,1,1,-1,-1)$. Encontre a rede que reconhece $V_2=(1,1,1,-1,-1,-1)$. Teste a rede com V_1 , V_2 e $V_3=(1,1,1,-1,0,0)$.

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:

- por meio 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:

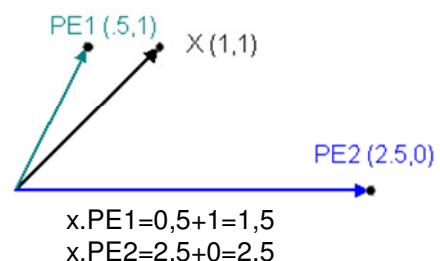
- Hard: somente um neurônio ganha os recursos;
- Soft: há um vencedor, mas seus vizinhos também recebem uma parte dos recursos.

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:

$$\text{vencedor} = \max_i (w_i^T x) \quad \text{ou} \quad \text{vencedor} = \min_i (|x - w_i|^2).$$

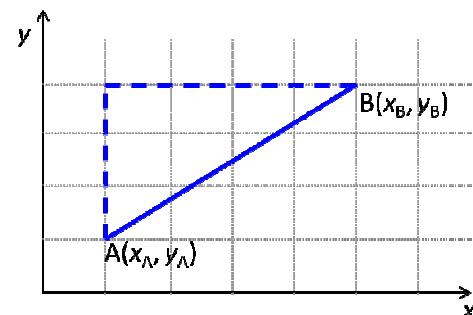
Como a raiz quadrada é uma função computacionalmente cara, a distância métrica é menos eficiente que o produto interno:

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

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

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

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

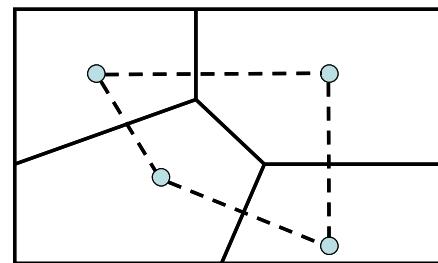


Agrupamento

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.

As redes competitivas fazem uma espécie de versão do agrupamento k-médio

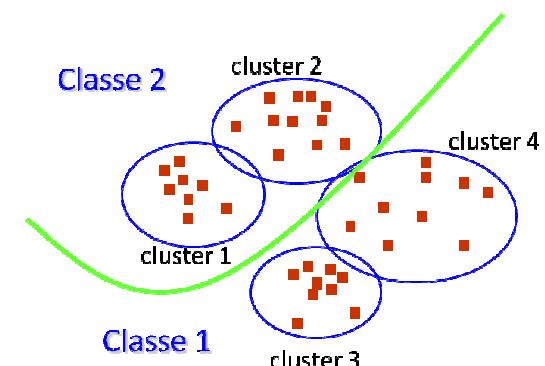
Agrupamento:

- 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.

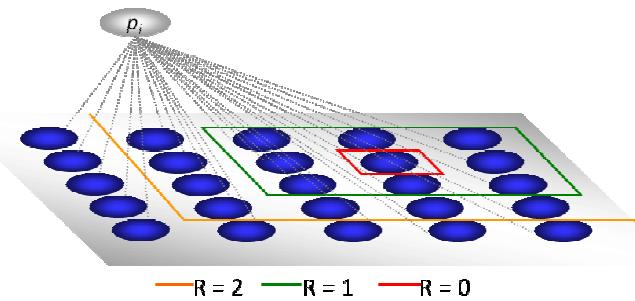
A rede SOM (*Self Organizing Map*) de Kohonen realiza um mapeamento de um espaço contínuo de entrada para um espaço discreto de saída, onde as propriedades topológicas da entrada são preservadas.

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.

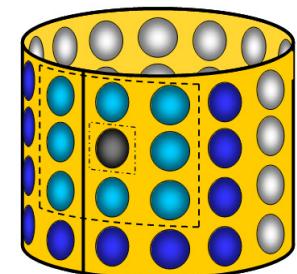
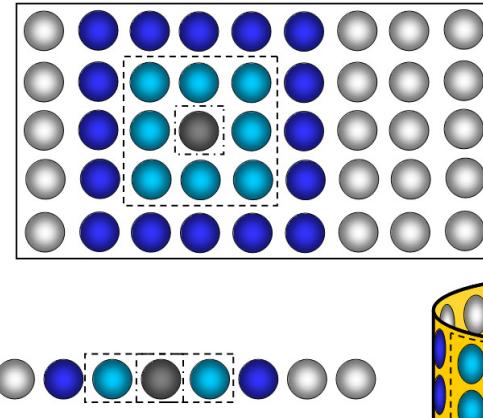
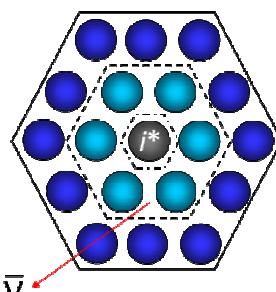
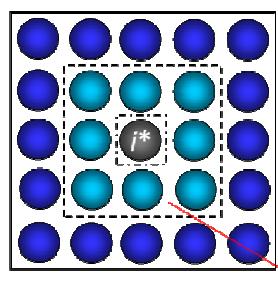
O processo de aprendizado de uma rede de Kohonen consiste em reforçar as ligações que produzem respostas mais eficientes.

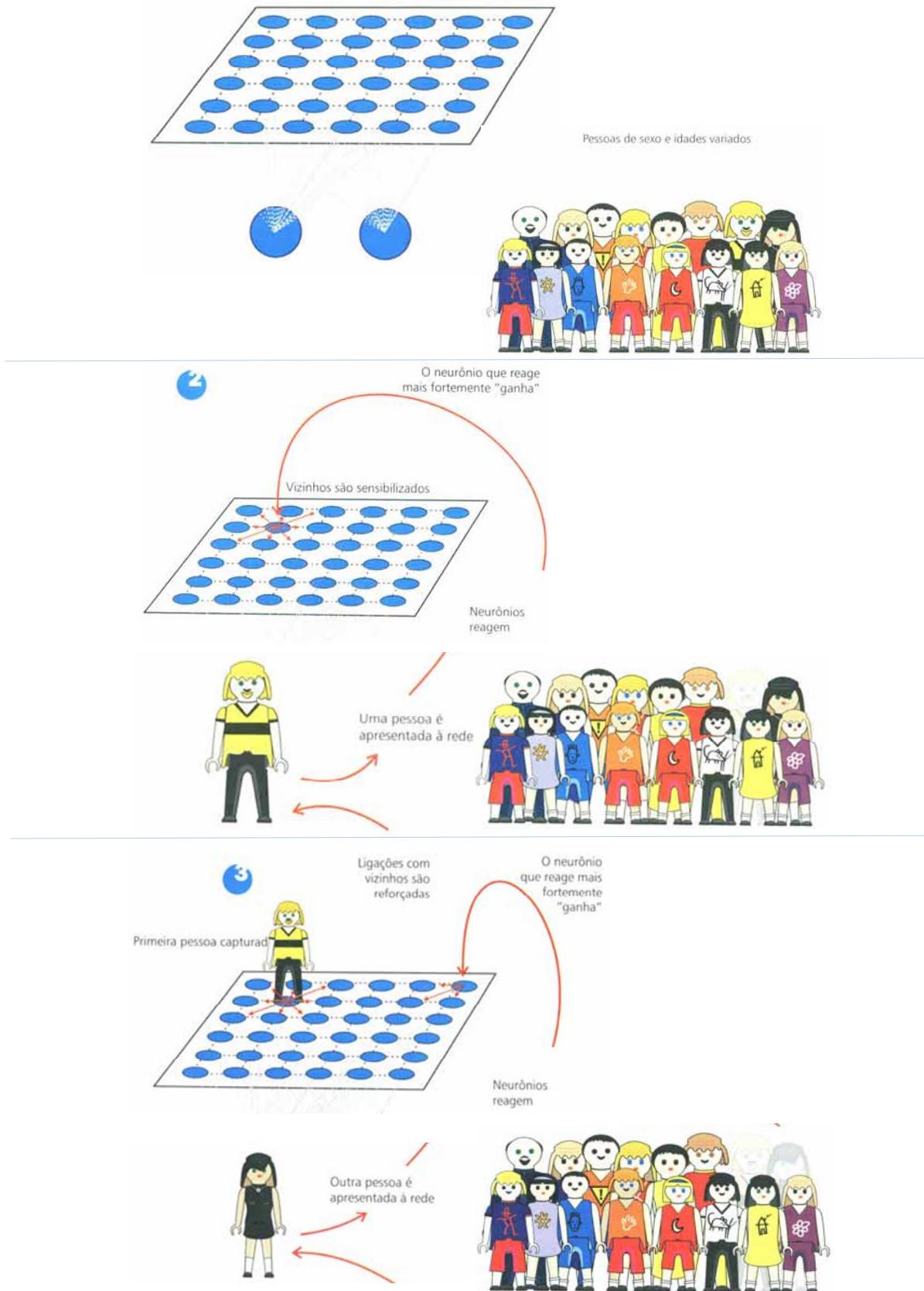


Vizinhanças

Tipos mais comuns de vizinhança:

- raio 0
- raio 1
- raio 2



Exemplo:

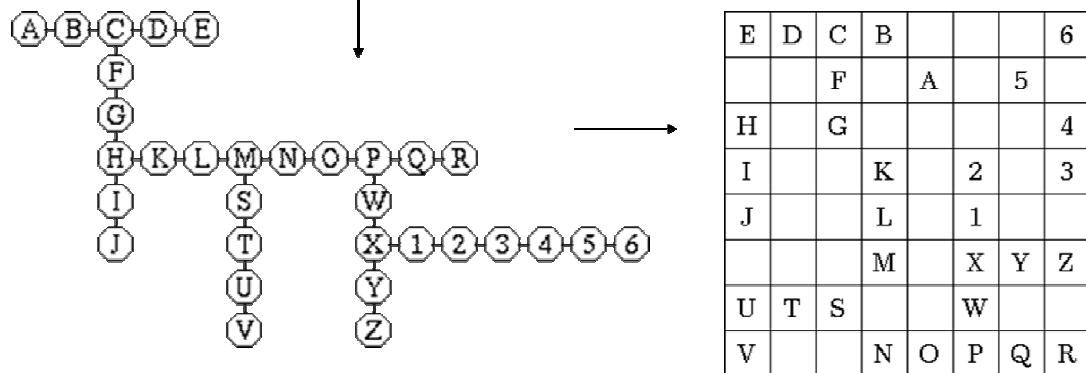
Após o treinamento da rede:



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		
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		
0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	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	2	2	2	2		
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	



Algoritmo:

0. Iniciar os pesos dos 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 com o vetor x .

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

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 .

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

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: $\alpha = 0,5$.

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

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:

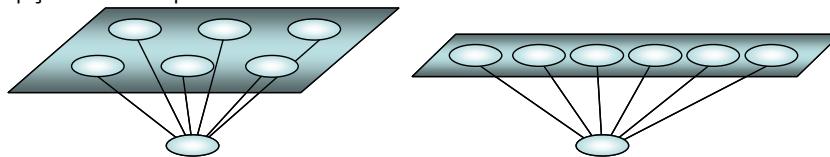
A	F	K	R	S	1	2
1	3	3	3	3	3	3
0	1	3	3	3	3	3
0	0	1	8	3	6	6
0	0	0	0	1	2	2
0	0	0	0	0	1	2

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

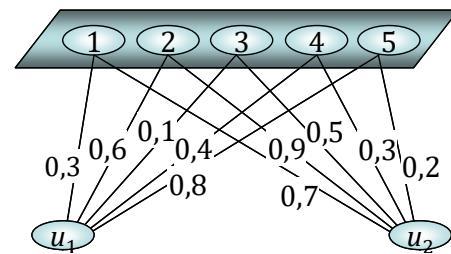
Taxa de aprendizagem inicial: $\alpha = 0,5$

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

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.

Observações:

A Rede de Kohonen tem inibição lateral e 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

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

onde a função Λ_{ii^*} é uma *função de vizinhança* centrada no neurônio vencedor

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

$$\Lambda_{ii^*} = e^{\frac{-d_{ii^*}^2}{2R^2}},$$

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:

$$\alpha = \alpha_0 e^{\frac{-t}{T}}$$

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.

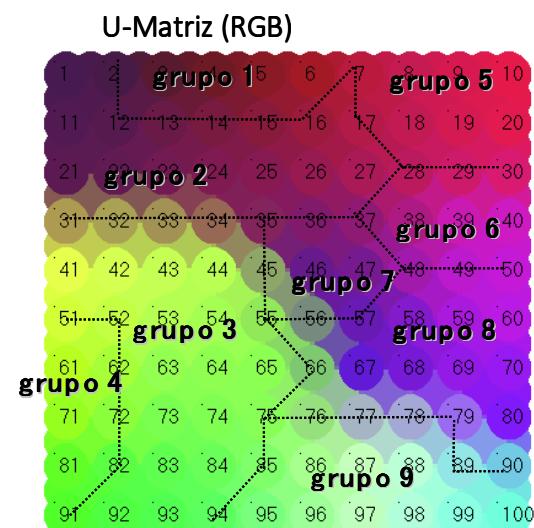
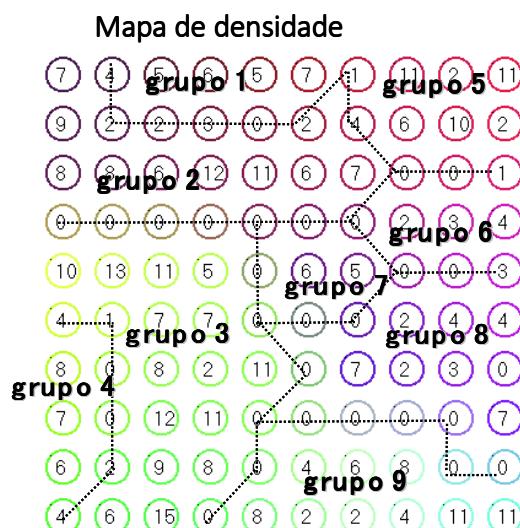
Manutenção da densidade: Regiões no espaço de entrada com maior densidade de pontos são mapeadas para regiões maiores no espaço de saída.

Os cálculos mais comuns dos erros em SOM são:

- Erro de Quantização: $E_Q = \frac{1}{n} \sum_{k=1}^n \|x_k - w^*\|$
- Erro Médio Quadrático: $E_{MQ} = \frac{1}{n} \sum_{k=1}^n \|x_k - w^*\|^2$
- 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:

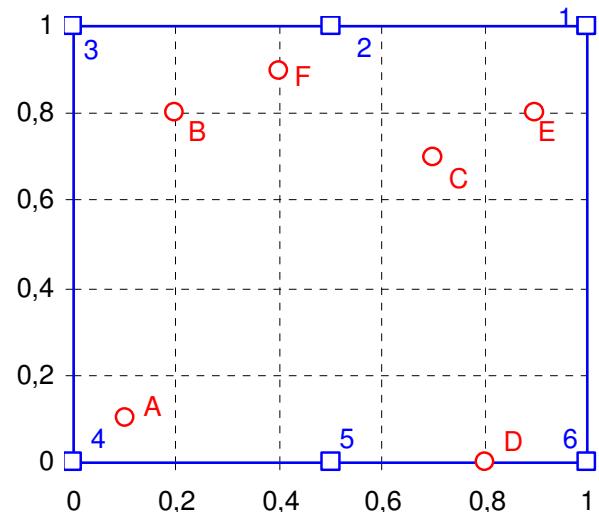


Exercícios:

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



Formulação Matemática do PCV para n cidades

$$\text{Minimizar } C = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

$$\text{Sujeito a } \sum_{i=1}^n x_{ij} = 1, \text{ para cada } j=1, \dots, n$$

$$\sum_{j=1}^n x_{ij} = 1, \text{ para cada } i=1, \dots, n$$

$$\sum_{i \in K} \sum_{j \in K} x_{ij} \leq |K| - 1$$

$$x_{ij} \in \{0,1\},$$

K é o conjunto de todos os subconjuntos de n pontos.

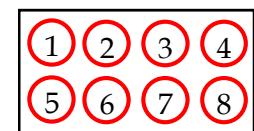
2. Considere o mapa de Kohonen bidimensional dado abaixo, com a matriz de pesos W .

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

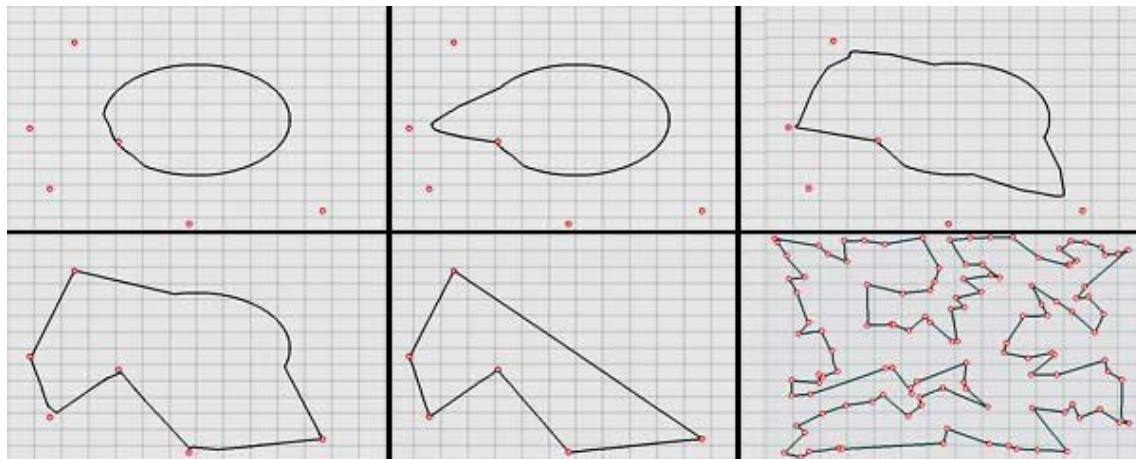
2.1. Utilizando a distância Euclidiana como métrica, encontre o neurônio vencedor para o padrão de entrada $(0,7 \ 0,4 \ 0,1)$;

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 $(0,1 \ -0,4 \ 0,9)$.



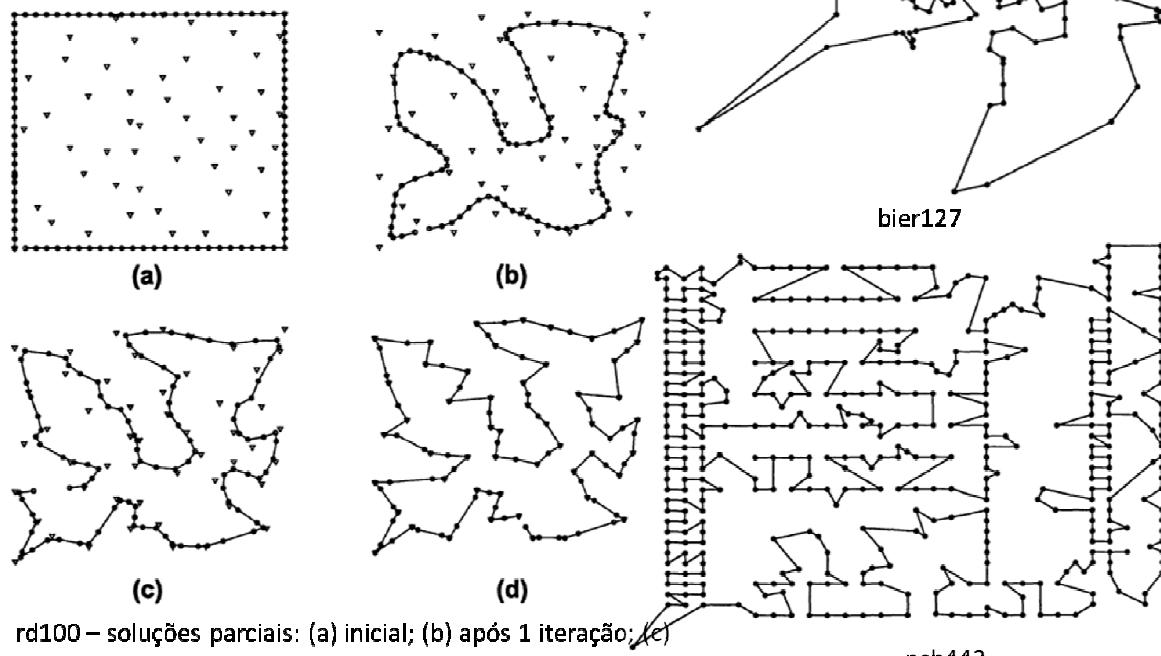
Alguns exemplos de aplicações da SOM para o problema do Caixeiro Viajante:



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

pcb442

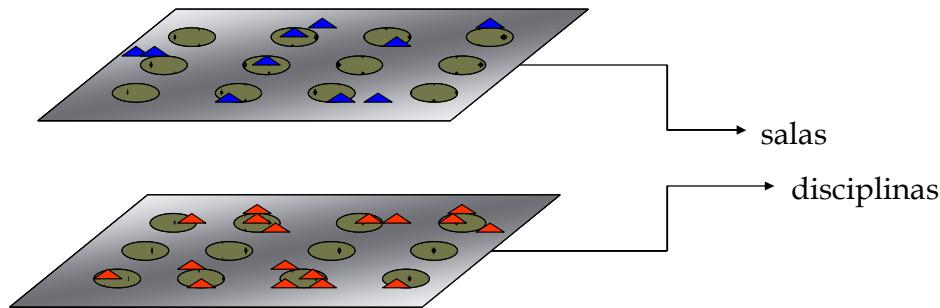
Os padrões de entrada são as coordenadas das cidades.

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.



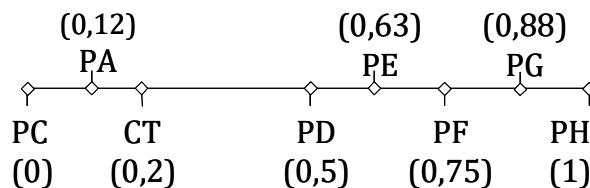
A partir de um arquivo com “modelos” de sala de aula, a Rede de Kohonen classifica tais modelos e cria uma matriz de pesos com as características selecionadas: capacidade, tipo de carteira e bloco.

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.

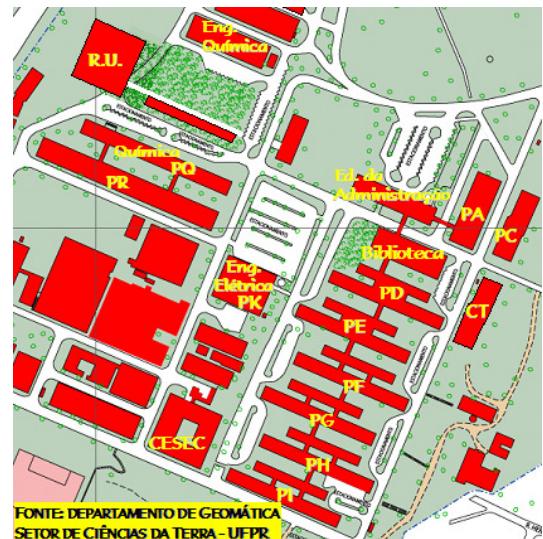


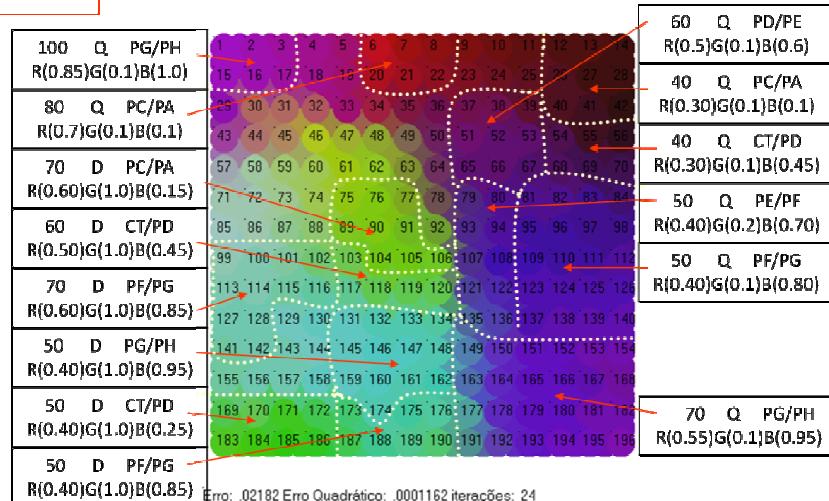
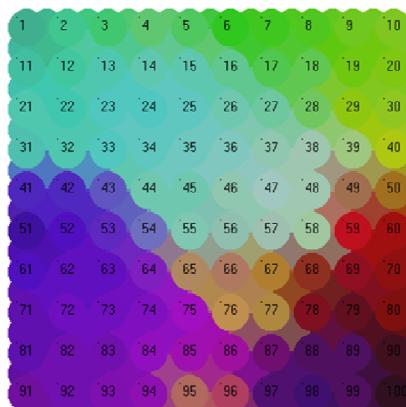
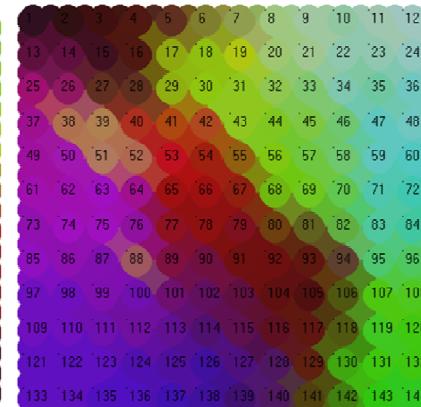
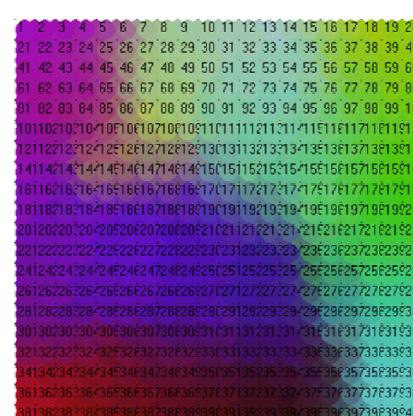
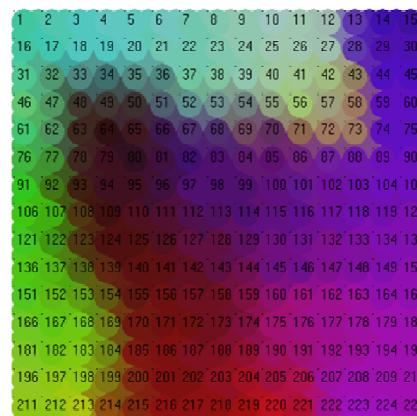
Dados utilizados para o Estudo de Caso: disciplinas de Graduação e Pós-Graduação dos Setores de Ciências Exatas, de Ciências da Terra e de Tecnologia da UFPR (650), e 46 salas de aula.

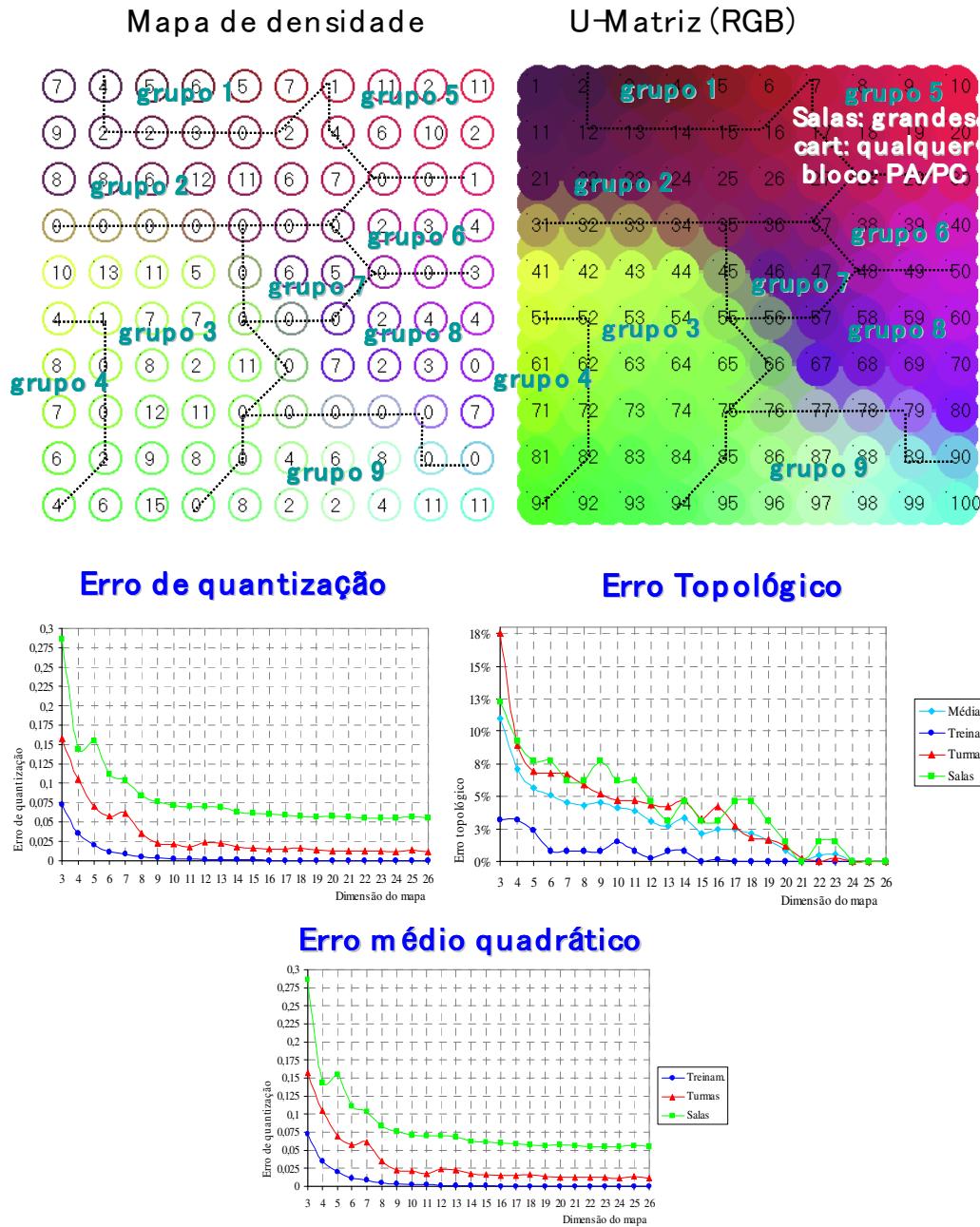
Os padrões de entrada para o treinamento da rede representam 2/3 do número total de disciplinas, ou seja, 433 padrões.

A seguir são mostradas as U-Matrizes de algumas simulações para este problema:

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



196 neurônios**100 neurônios****144 neurônios****400 neurônios****225 neurônios**



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:

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

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:

$$w_1 = (1, 1, 0, 0) \text{ classe 1}$$

$$w_2 = (0, 0, 1, 1) \text{ classe 2}$$

$$w_3 = (0, 0, 1, 0) \text{ classe 2}$$

$$x_1 = (0, 0, 1, 1) \text{ classe 2}$$

$$x_2 = (1, 0, 0, 0) \text{ classe 1}$$

$$x_3 = (0, 1, 1, 0) \text{ classe 2}$$

$$x_4 = (1, 1, 1, 0) \text{ classe 1}$$

Use a taxa de aprendizagem inicial $\alpha = 0,1$

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 .

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 $\alpha=0,5$, 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 ;

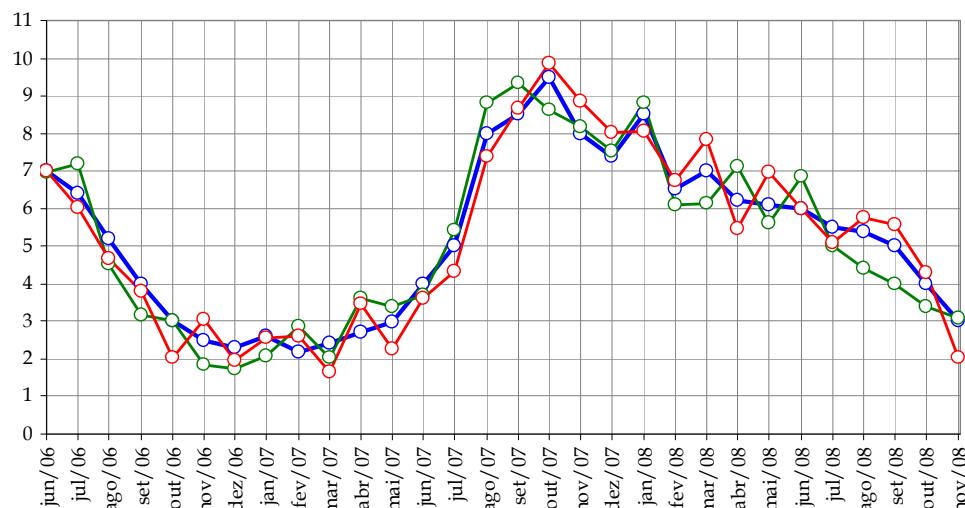
$(0,6 \ 0,65)$ e $(0,75 \ 0,8)$, ambos pertencentes à classe C_4 .

x_2				
1	C_3	C_4	C_1	C_2
0,8	C_1	C_2	C_3	C_4
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

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.

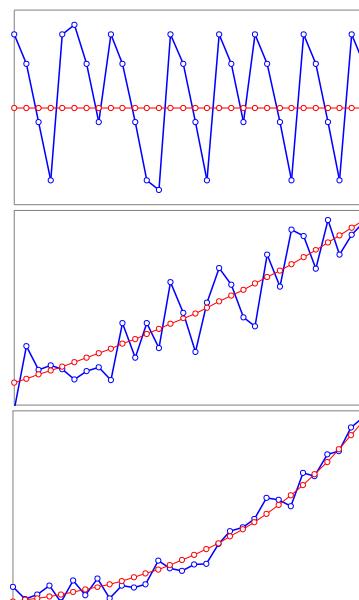
Como inserir a variável **tempo** na estrutura operacional de uma rede neural artificial?

- 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

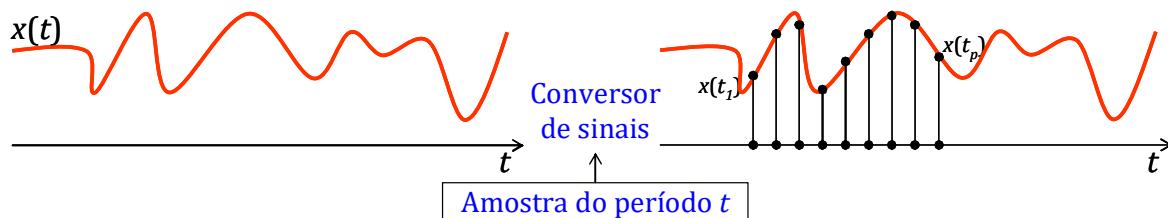
Efeito tendencioso aditivo

Efeito tendencioso multiplicativo

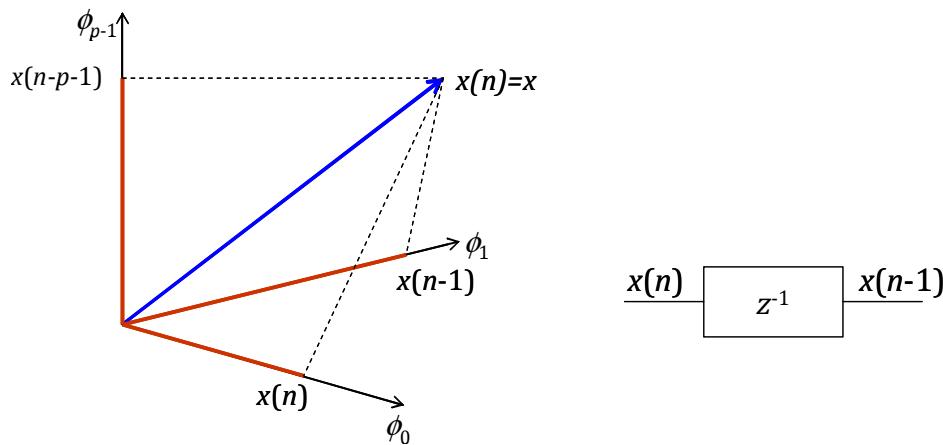
Problemas que não envolvem o tempo (tais como classificação de padrões) são chamados **estáticos**. O tempo estabelece uma ordem nos dados de entrada. O tempo e as variáveis físicas são contínuas, compondo os sinais analógicos.

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*.

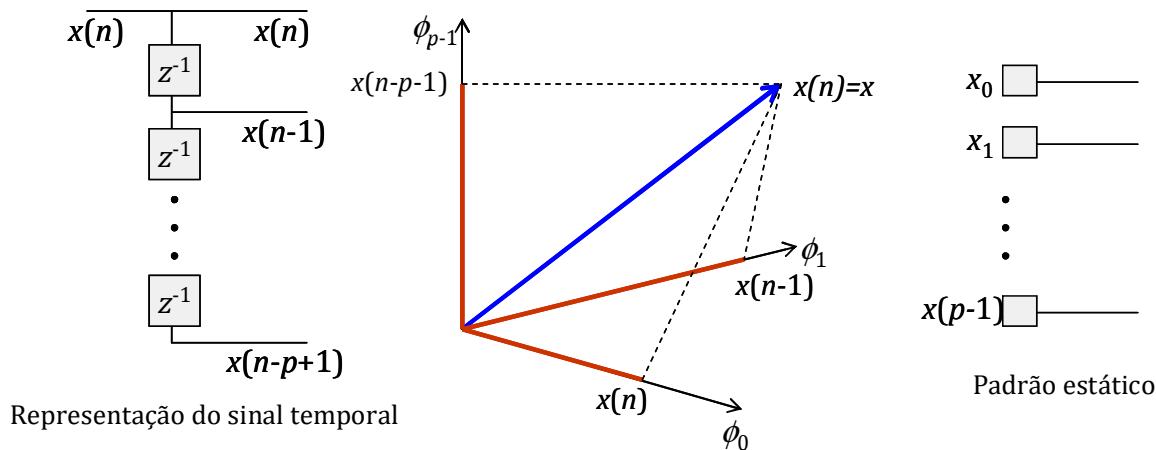
$$x(t_p) = [x(t_1), x(t_2), x(t_3), \dots, x(t_p)].$$



Vetor com atraso:

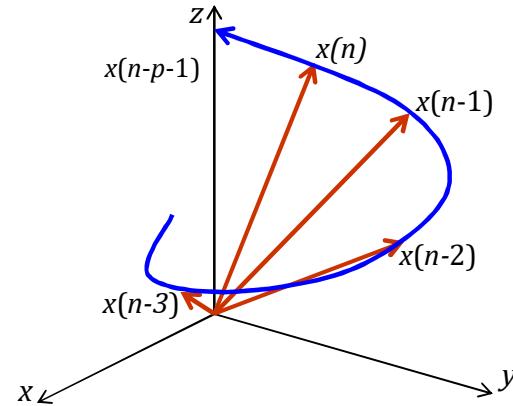
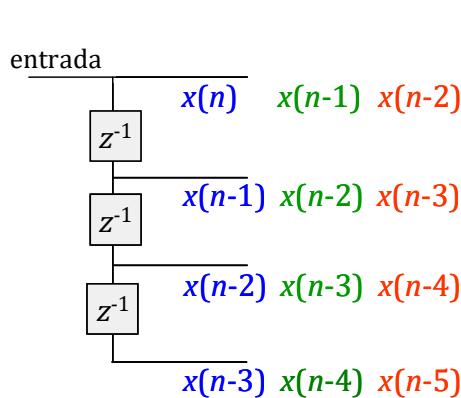


Um *elemento de atraso ideal*, denotado por z^{-1} , atrasa o sinal em uma amostra. Uma *linha de atraso* é um sistema de uma entrada e várias saídas, composto pela ligação em cascata de vários operadores de atraso.



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*.



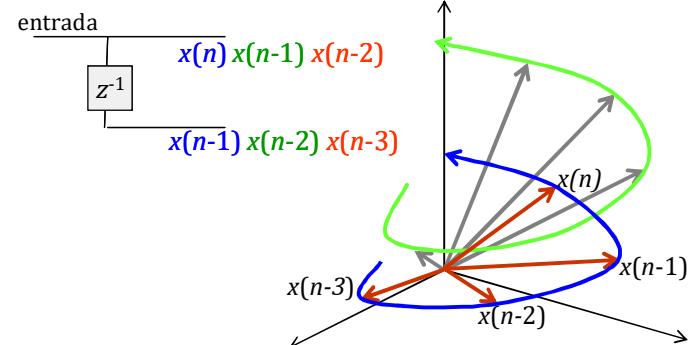
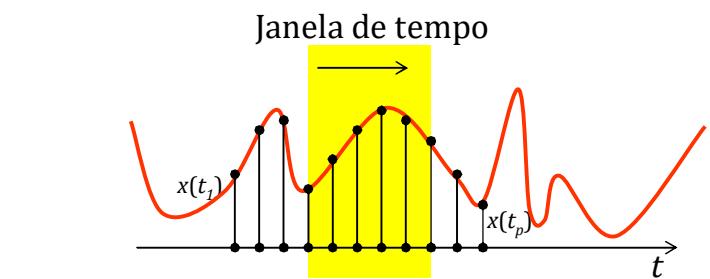
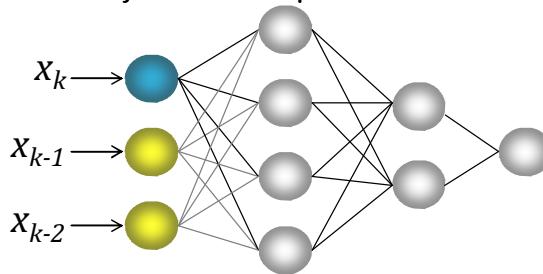
A construção de uma janela de tempo é essencial para a solução de um problema de Séries Temporais com RNA:

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

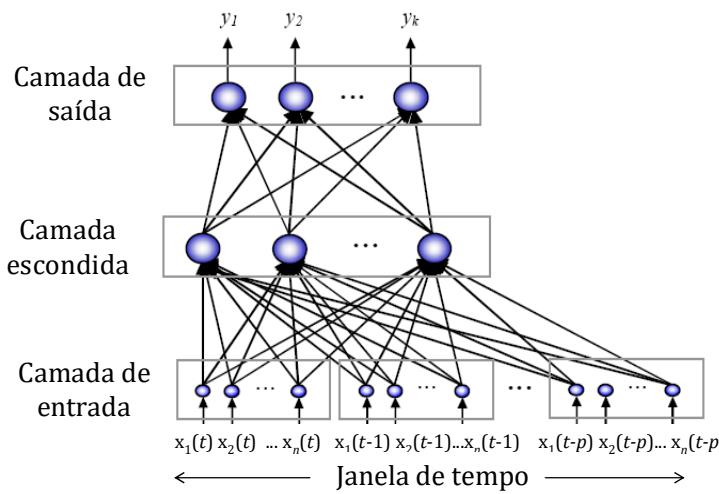
Este comprimento corresponde ao tamanho da linha de atraso e estabelece a dimensionalidade do espaço de reconstrução.

Algumas arquiteturas que podem ser usadas em Séries Temporais são:

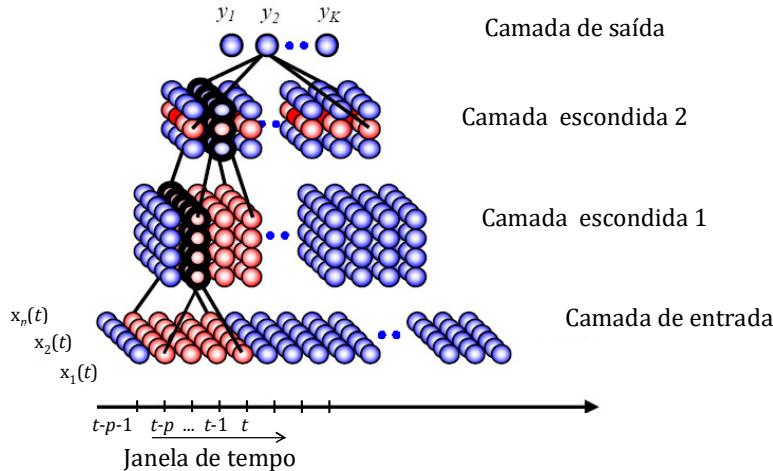
MLP com janela de tempo e 1 saída:



MLP com janela de tempo e k saídas:

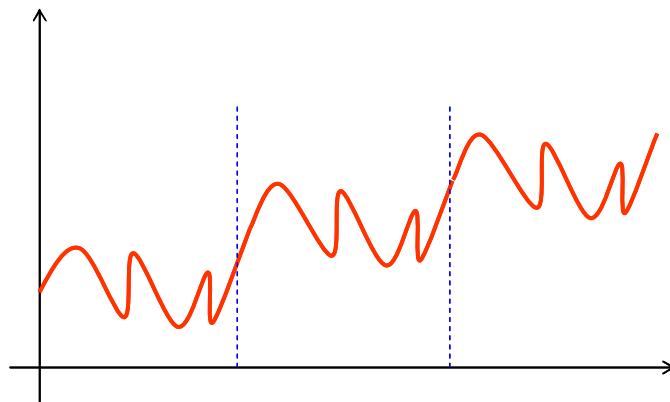


Rede Neural com atraso de tempo:



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 possuir 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 $[x_1(t), x_2(t), \dots, x_k(t)]$

saída $y_k(t)$

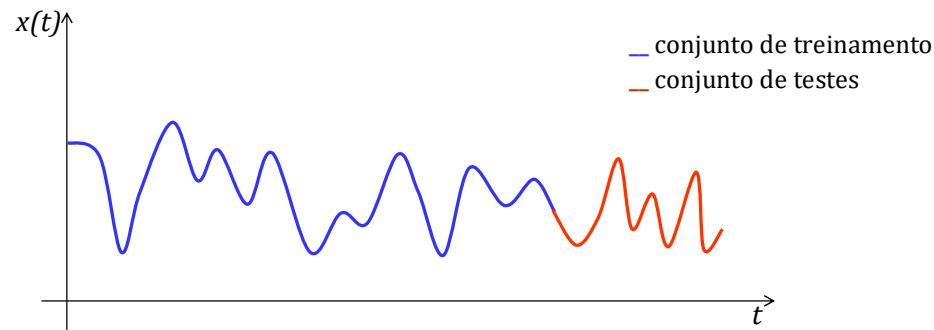
Metodologia 2: Rede Neural com **p** vetores de entrada (com atrasos) e **k** saídas (*Focused Time-Delay Neural Network*).

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

saída $\{y_1, y_2, \dots, y_k\}$

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

Segue um exemplo da metodologia 1:

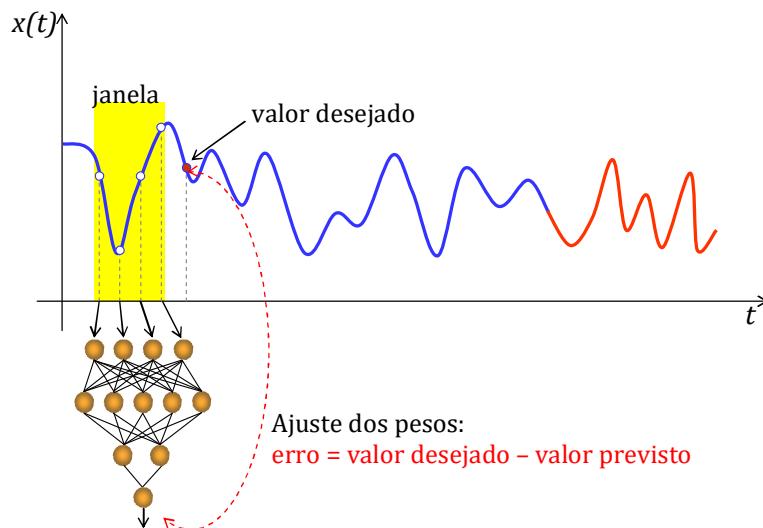
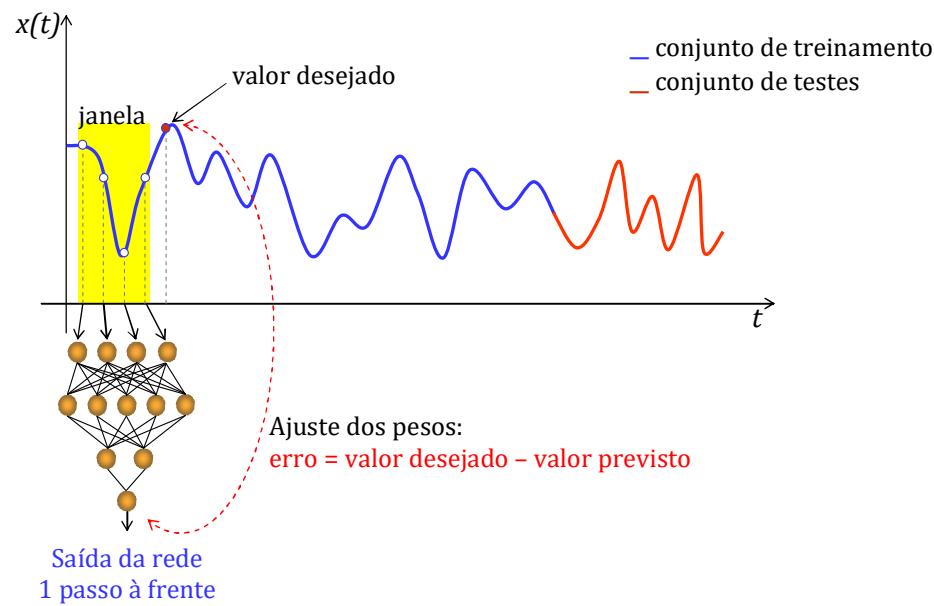


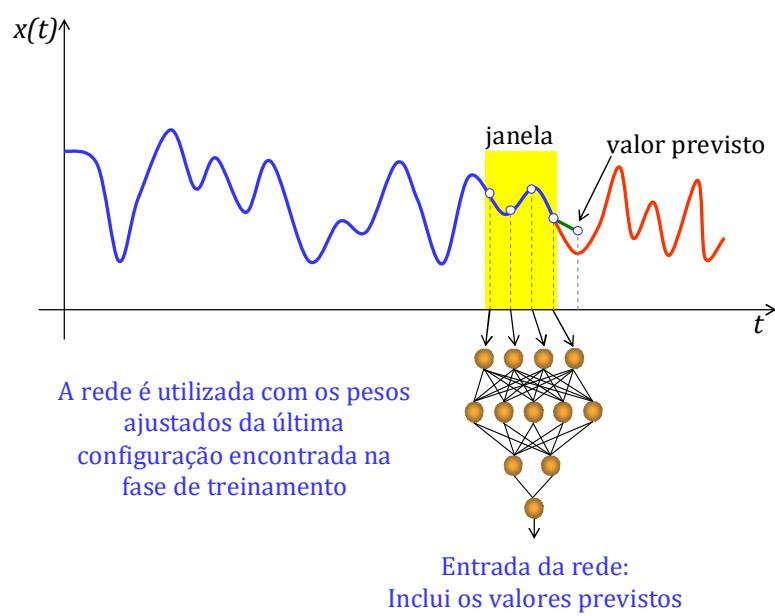
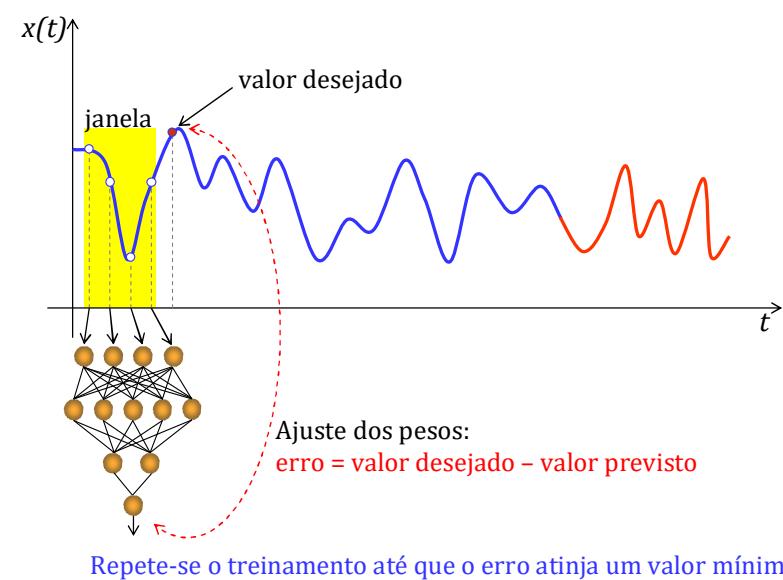
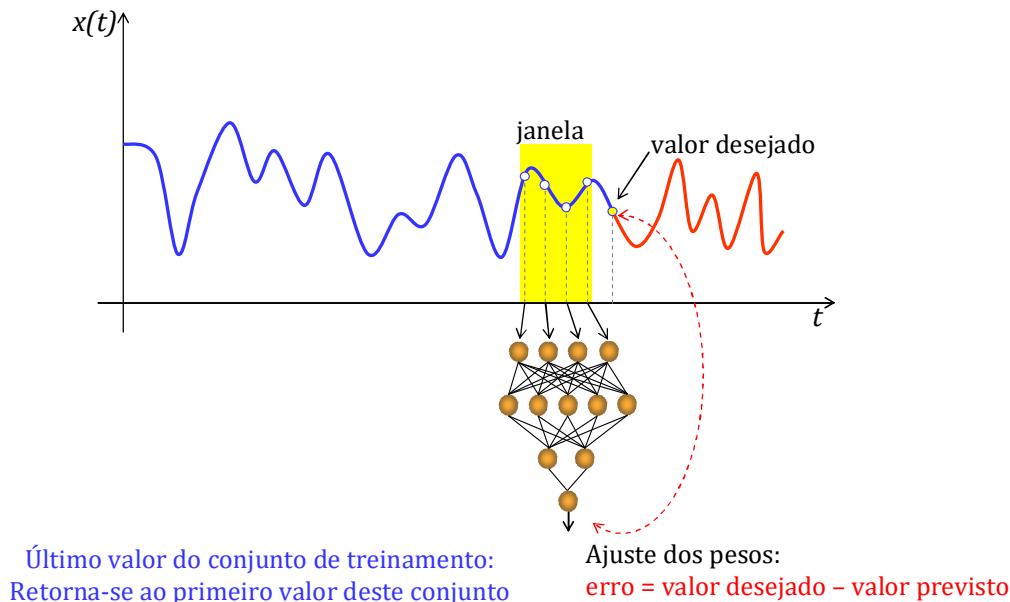
Entradas da rede:

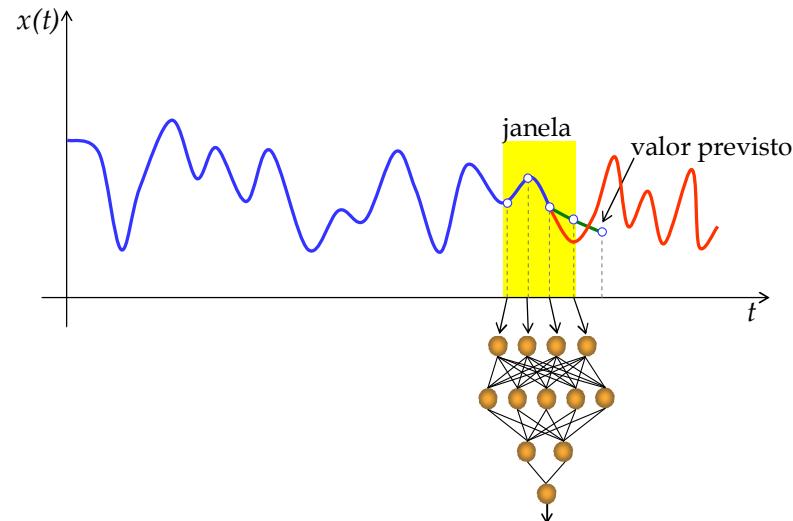
p valores passados (por exemplo, 4 valores)

Saída desejada:

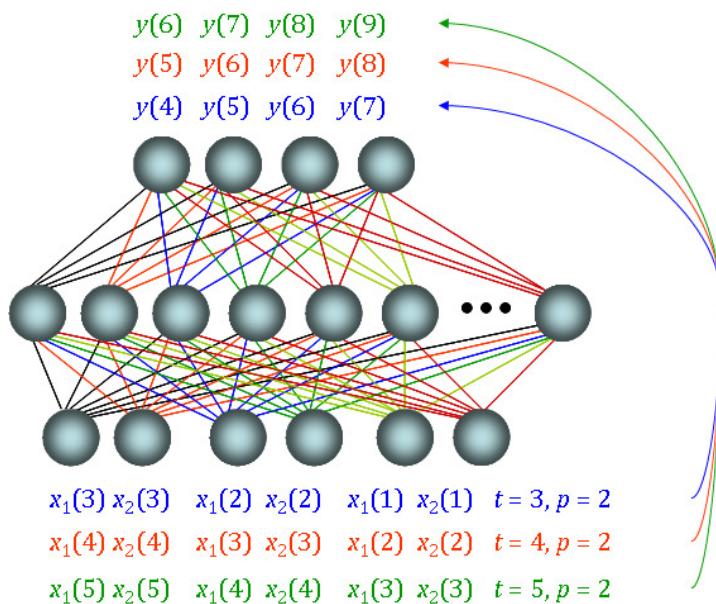
valor da série k passos à frente (por exemplo, 1 passo)







Exemplo de funcionamento de uma RNA temporal:

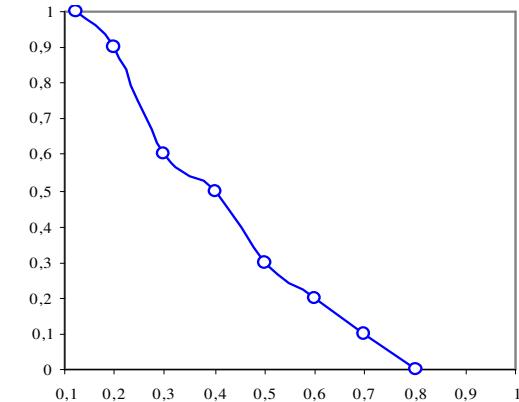
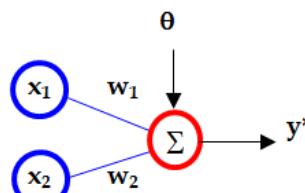


Exercícios:

- 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
	0,4	0,5
	0,5	0,3
	0,6	0,2
	0,7	0,1
	0,8	0

$$\begin{aligned}
 w_1 &= 1,0 & w_2 &= 1,0 & \theta &= 0,5000 & y^* &= \sum w_j x_j + \theta_j \\
 &&&&&& y &= 1/(1+e^{-y^*}) \\
 &&&&&& \Delta w_j &= \alpha(d-y)x_jy(1-y) \\
 &&&&&& w_j &= w_j + \Delta w_j \\
 &&&&&& \Delta \theta &= \alpha(d-y)y(1-y)
 \end{aligned}$$



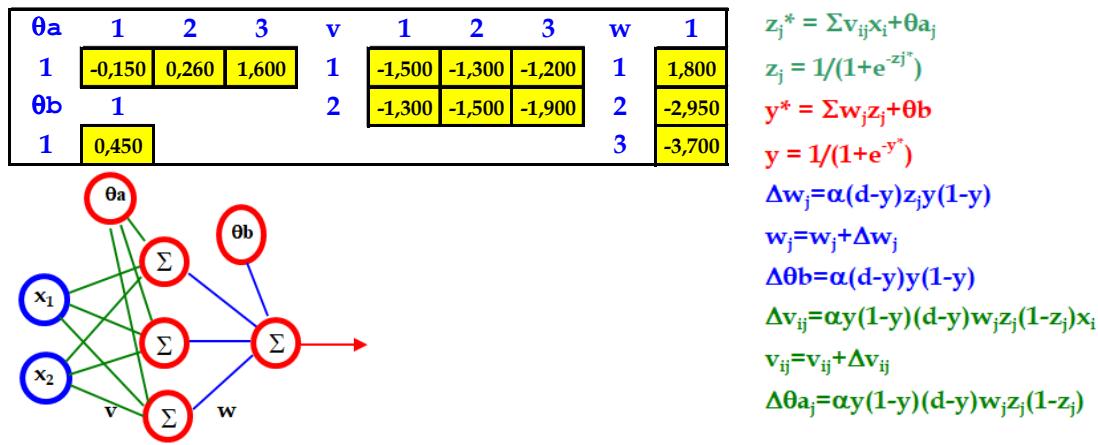
Esta é a aplicação da metodologia 1, com entradas x.

2. Resolva o problema anterior usando entradas t, e os parâmetros dados a seguir.

	t	x
x_1	x	y
x_1	0,1	1
x_2	0,2	0,9
	0,3	0,6
	0,4	0,5
	0,5	0,3
	0,6	0,2
	0,7	0,1
	0,8	0

$x_1 = -1,0 \quad x_2 = -1,0 \quad \theta = 0,8000$

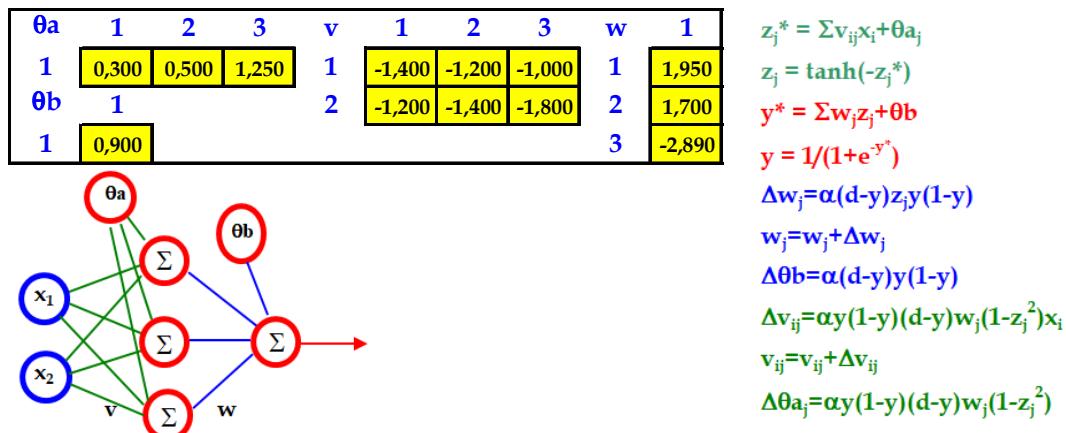
3. Utilize uma rede MLP com uma camada de entrada de 2 neurônios ($k=2$), uma camada escondida com 3 neurônios (sigmoidais) e saída com um neurônio (sigmoidal) para a série temporal do exercício 1. Use entradas x com os seguintes parâmetros:



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

θ_a	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
θ_b	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:



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

θ_a	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
θ_b	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: $\sigma = 1$; $\alpha = 1$; $u = \{(1 \ 0,9), (0,6 \ 0,4)\}$; $k = 2$; $\theta = 1$; saída com 1 neurônio.

	x	y
x1	0,1	1
x2	0,2	0,9
	0,3	0,75
	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

y1

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: $\sigma = 0,5$; $\alpha=1$; $k=2$; $\theta = 1$; $u = \{(1 \ 0,9), (0,55 \ 0,6)\}$; saída com 2 neurônios.

	x	y
x1	0,1	1
x2	0,2	0,9
	0,3	0,75
	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

y1

y2

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.

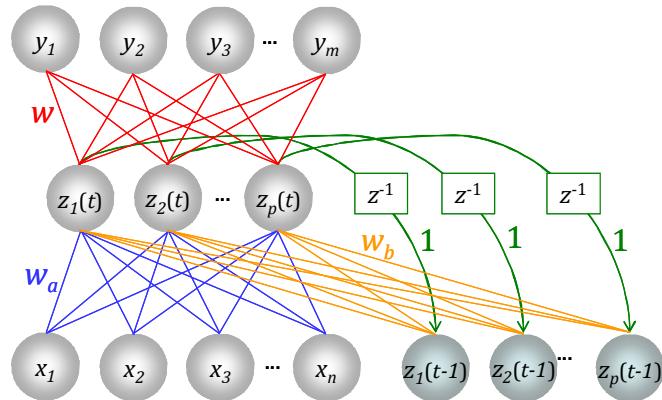
São redes que têm maior estabilidade, também chamadas de TLFN (*time layer focused neural network*). Elas tornam o mecanismo de memória de curto prazo estável.

A recorrência torna os cálculos mais trabalhosos.

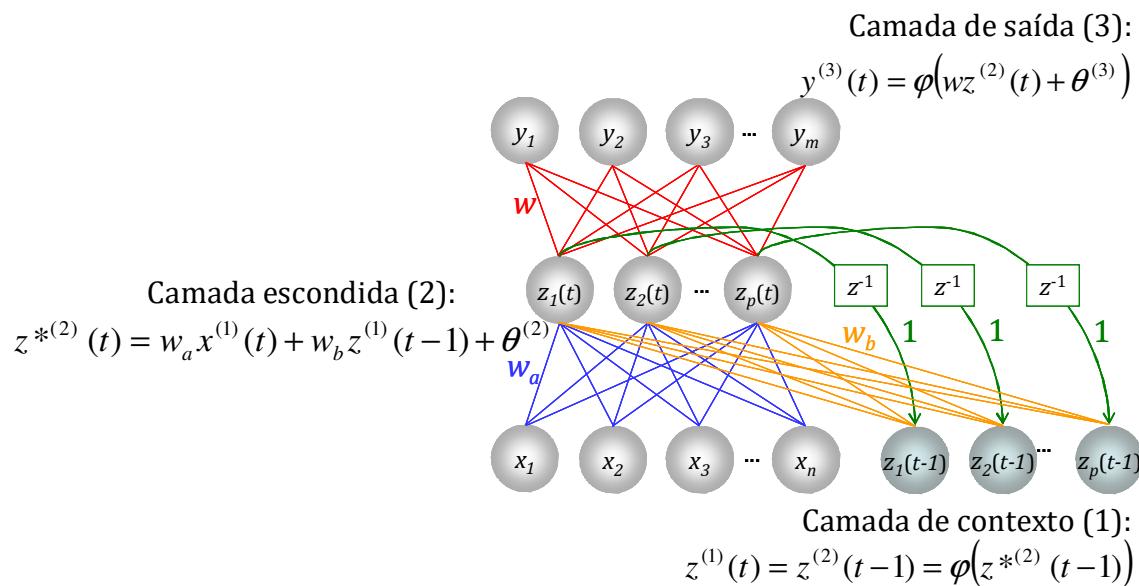
As redes recorrentes mais conhecidas são simples, baseadas em neurônios de contexto, fáceis de treinar (com parâmetros de realimentação fixos), com correções de erros usando o algoritmo backpropagation.

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

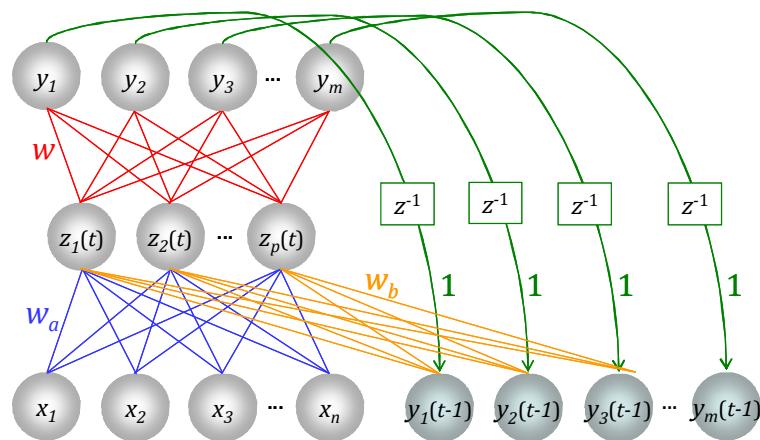
A **rede neural de Elman** (1990) tem cada um dos neurônios da camada oculta com realimentação para as unidades de contexto. É conhecida como “Perceptron de múltiplas camadas recorrentes”.



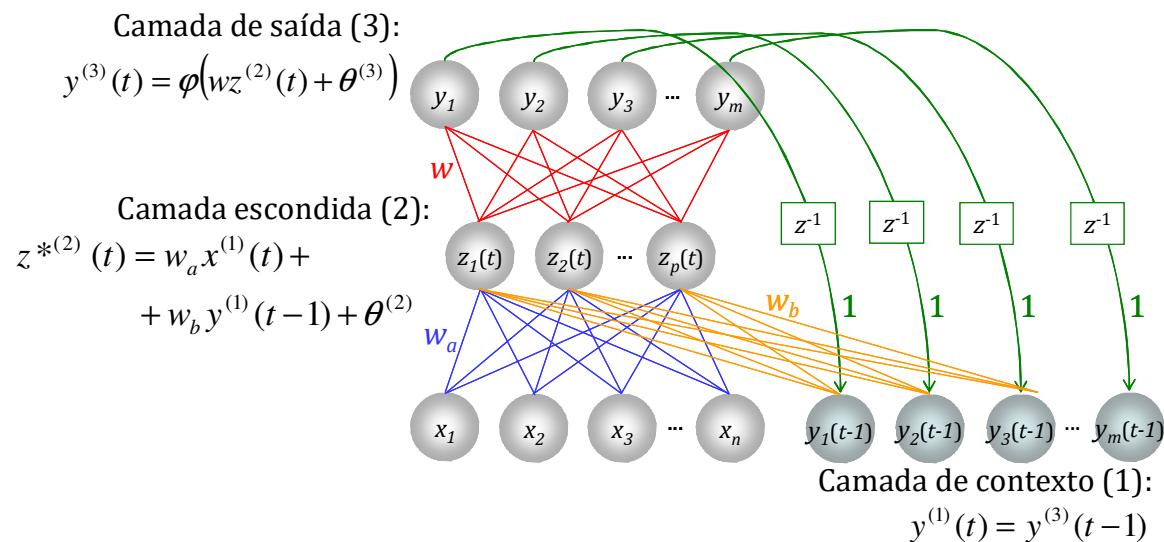
O cálculo da saída de uma rede neural de Elman é feito de maneira similar à MLP, com as seguintes atribuições:



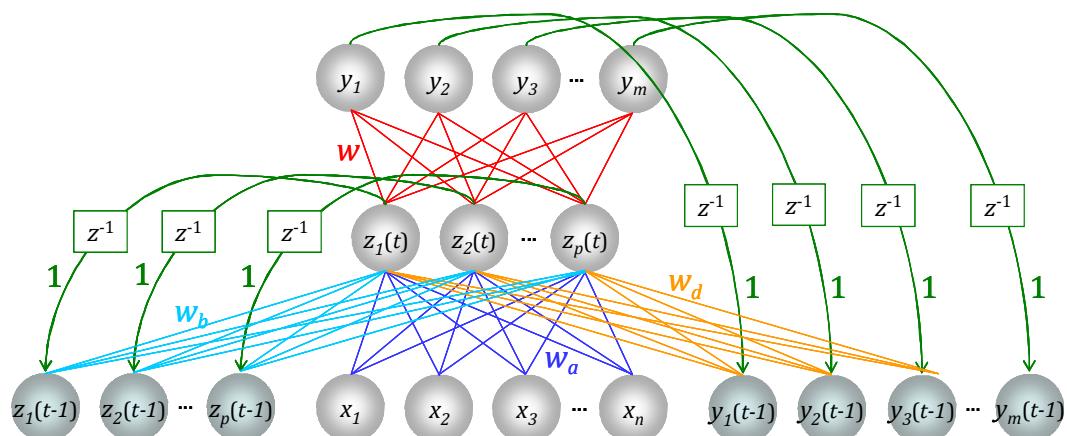
A **rede neural de Jordan** (1986) considera somente realimentação dos valores de ativação de saída para as unidades de contexto.



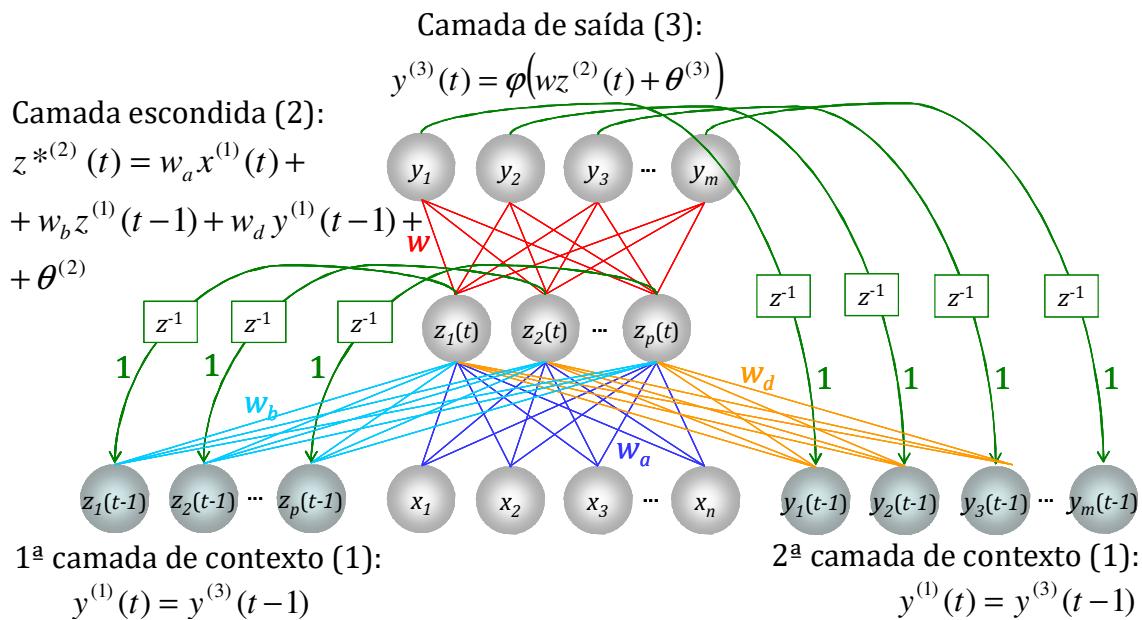
O cálculo da saída de uma rede neural de Jordan é feito de maneira similar à rede de Elman, com as seguintes atribuições:



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:

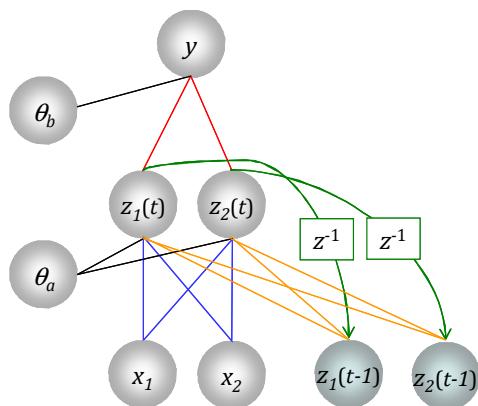


Uma aplicação interessante das Redes recorrentes é a solução de problemas de Séries Temporais, por suas características dinâmicas.

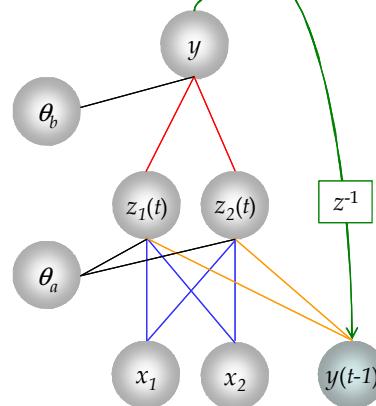
Exercícios:

- 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



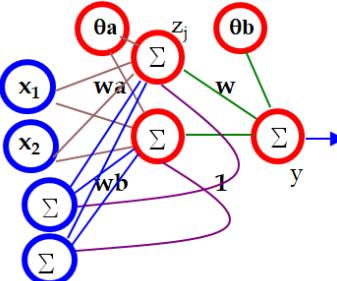
Arquitetura - Jordan



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

θ_a	1	2	w_a	1	2	w_b	1	2	w	1
1	-0,500	2,000	1	-1,000	-3,000	1	-0,500	-0,500	1	2,000
θ_b	1		2	3,000	-3,000	2	0,500	0,500	2	-3,000
1	0,500									
x_1			x_2							
1			0,9							
0,9			0,6							
0,6			0,5							
0,5			0,3							
0,3			0,2							
0,2			0,1							

$z_1(t-1) = 1 \quad z_2(t-1) = 1$



$$z_j^* = \sum w_{ai}x_i + \sum w_{kj}z_k(t-1) + \theta_a$$

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

$$y^* = \sum w_j z_j + \theta_b$$

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

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

$$w_j = w_j + \Delta w_j$$

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

$$\Delta w_{ai} = \alpha y(1-y)(d-y)w_jz_j(1-z_j)x_i$$

$$w_{ai} = w_{ai} + \Delta w_{ai}$$

$$\Delta w_{kj} = \alpha y(1-y)(d-y)w_jz_k(1-z_k)z_j(t-1)$$

$$w_{kj} = w_{kj} + \Delta w_{kj}$$

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 Jordan com entradas t .

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

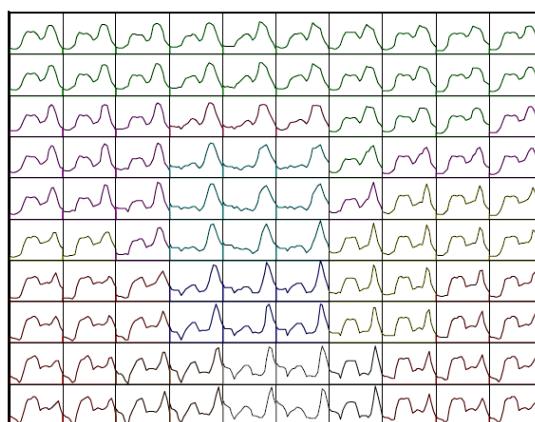
Uma das RNAs é a *Rede de Kohonen*, com MLPs conectadas a cada agrupamento feito pelo mapa de Kohonen [Le Coadou & Benabdeslem, 2006]. Outra configuração possível é de uma rede de Kohonen dupla.

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

- Classificar todos os vetores de entrada

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

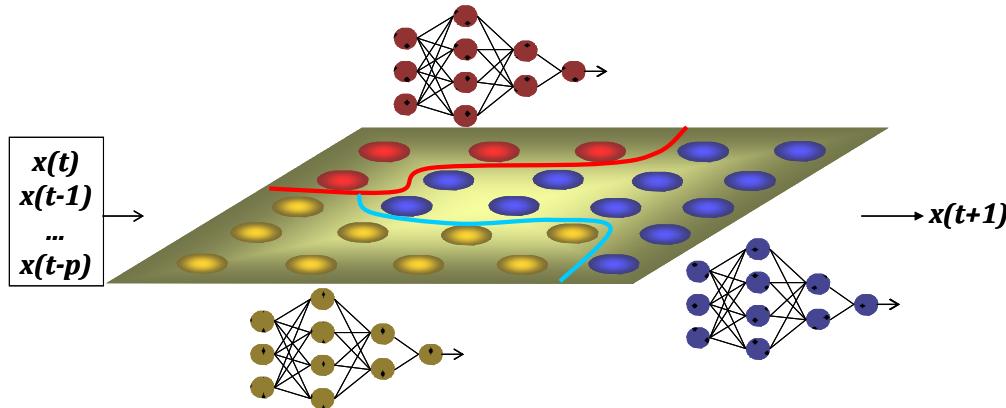
- Criar um mapa com protótipos similares aos vetores apresentados para a rede:



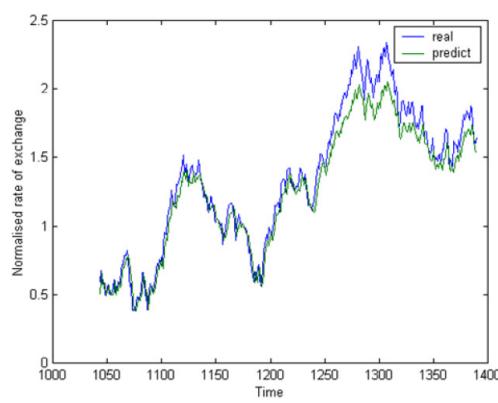
- Associar a cada agrupamento (representado pelo centróide G_k) uma rede do tipo MLP.

Os vetores do agrupamento são considerados como entrada para a MLP.

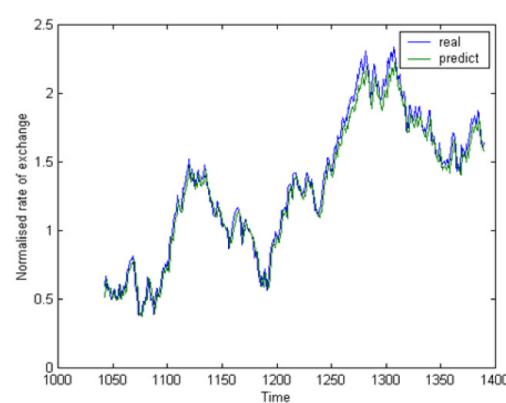
- ❑ A previsão de cada entrada apresentada à rede é o resultado do processamento de cada MLP.



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 Comprehensive 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.
- BOÇOIS, A., OLIVEIRA, A. A., SIQUEIRA, P. H., TELLES, F. Q. *Diagnóstico de Doenças Dermatológicas usando a Rede Neural de Kohonen*. In: IX Encontro Nacional de Inteligência Artificial (ENIA 2012), 2012, Curitiba. Proceedings Brazilian Conference on Intelligent Systems, v. 1. p. 1-8, 2012.
- SIQUEIRA, P. H., SCHEER, S., STEINER, M. T. A. *Application of the Winner Takes All Principle in Wang's Recurrent Neural Network for the Assignment problem*. Lecture Notes in Computer Science, Berlin, v. 3496, n. 1, p. 731-738, 2005.
- SIQUEIRA, P. H., SCHEER, S., STEINER, M. T. A. *A new approach to solve the Traveling Salesman Problem*. Neurocomputing (Amsterdam), v. 70, p. 1013-1021, 2007.

- SIQUEIRA, P. H., STEINER, M. T. A., SCHEER, S. *Recurrent Neural Network with Soft 'Winner Takes All' principle for the TSP*. In: ICNC 2010 - International Conference on Neural Computation, 2010, Valencia. Proceedings of the International Conference on Fuzzy Computation and International Conference on Neural Computation, v. 1. p. 265-270, 2010.
- ROSA, C. R. M. ; STEINER, M. T. A. ; STEINER NETO, P. J. *Técnicas de Mineração de Dados aplicadas a um Problema de Diagnóstico Médico*. Espacios (Caracas), v. 37, p. 1, 2016.
- TEIXEIRA, L. L., TEIXEIRA JUNIOR, L. A., SIQUEIRA, P. H. *Previsões de vazões mensais via combinação híbrida ARIMA_NEURAL com encolhimento e decomposição Wavelet*. REVISTA DE ENGENHARIA E TECNOLOGIA, v. 7, p. 144-161, 2015.
- DEMUTH, H., BEALE, M. *Neural Network Toolbox User's Guide (For Use with MATLAB)*, The MathWorks, Inc., MA, USA, 1994.
- SOUTO, M. *Multi-layer Perceptrons e Backpropagation*. DIMAp/UFRN, 2020. Disponível em: <<https://slideplayer.com.br/slide/3258057/>>
- BENEDIKTSSON, J. A., SWAIN, P. H.,ERSOY, O. K. *Neural network approaches versus statistical methods in classification of multisource remote sensing data*. In: 12th Canadian Symposium on Remote Sensing Geoscience and Remote Sensing Symposium, IEEE, 1989. p. 489-492.
- HEPNER, G., LOGAN, T., RITTER, N., BRYANT, N. *Artificial neural network classification using a minimal training set- Comparison to conventional supervised classification*. Photogrammetric Engineering and Remote Sensing, v. 56, n. 4, p. 469-473, 1990.
- GORNI, A. A. *Redes neurais artificiais - uma abordagem revolucionária em inteligência artificial*. São Paulo, Micro Sistemas, 1993.
- KRÖSE, B., KRÖSE, B., SMAGT, P. *An introduction to neural networks*, 1993.
- GAHEGAN, M., WEST, G. *The classification of complex geographic datasets: An operational comparison of artificial neural network and decision tree classifiers*. In: Third International Conference on GeoComputation, p. 17-19, 1998.
- BOSER, B. E., GUYON, I. M., VAPNIK, V. N. *A training algorithm for optimal margin classifiers*. In: Proceedings of the fifth annual workshop on Computational learning theory, p. 144-152, 1992.
- CORTES, C., VAPNIK, V. *Support-vector networks*. Machine learning, v. 20, n.3, p. 273-297, 1995.
- POWELL, M. J. D. *Radial basis functions for multivariate interpolation: a review*. In: J.C. Mason, M.G. Cox (Eds.), *Algorithms for Approximation*, Clarendon Press, Oxford, 1987
- VIEIRA, F. C., DÓRIA NETO, A. D., COSTA, J. A. F. *An Efficient Approach to the Travelling Salesman Problem Using Self-Organizing Maps*. International Journal of Neural Systems, London, UK, v. 13, n.2, p. 59-66, 2003
- Le COADOU, BENABDELEM, K. *Optimizing local modeling for times series prediction*. International Journal of Computational Intelligence Research, v. 2, n. 1, p. 81-85, 2006.

PARTE III
OUTRAS METAHEURÍSTICAS

3.1 BUSCA TABU

A técnica de Busca Tabú (*Tabu Search*) foi proposta por Glover em 1986 para solução de problemas de Programação Inteira. Em 1987, Glover fez uma extensão da metaheurística para problemas de Pesquisa Operacional.

É uma heurística computacional de busca, conhecida por superar (em geral) o problema da convergência local em problemas de otimização.

Elementos da técnica:

Tabu - proibição;

Restrições tabu - inibe certos movimentos;

Critérios de aspiração - decidem quando os movimentos classificados como tabu podem ser executados;

As restrições tabu são controladas por uma lista que memoriza os últimos movimentos executados. A implementação deve decidir como os movimentos são realizados, ou seja, como é gerenciada a memória dos movimentos.

Pode ser caracterizada como busca através das soluções vizinhas (GLOVER, 1991). Considerando-se X – conjunto de soluções

Cada $x \in X$, tem um conjunto associado de soluções vizinhas $V(x) \subset X$. Toda solução $x' \in V(x)$ pode ser gerada a partir de x por um certo tipo de operação denominada **movimento** (m).

Geralmente em Busca Tabú (BT), as soluções vizinhas são simétricas, ou seja: x' é solução vizinha a x se, e somente se, x é solução vizinha a x' .

Critérios de aspiração

São introduzidos para determinar quando uma restrição tabu pode ser quebrada, ou seja, a restrição é ignorada e o movimento, mesmo classificado como proibido, é executado.

Um critério de aspiração bastante utilizado é o seguinte: **Ignorar a restrição tabu** se a solução formada por um determinado **movimento proibido** for melhor que a melhor solução encontrada até o momento.

Estratégias de Oscilação

Admite oscilação entre soluções factíveis e infactíveis durante o processo de busca, pois considerar soluções infactíveis é um meio de tentar escapar de ótimos locais. O fato de permitir temporariamente soluções infactíveis tende a direcionar a busca mais rapidamente para soluções factíveis cada vez melhores.

Listas Tabu

A lista de memória (lista tabu) armazena cada movimento executado. O movimento deverá permanecer nesta lista durante as três próximas iterações. Enquanto um determinado movimento permanecer na lista tabu, será considerado proibido e só poderá ser executado, se este resultar num valor de isolamento melhor que o melhor valor obtido em todas as iterações anteriores (critério de aspiração). No caso de maximizar a função F procura-se sempre o maior valor de troca possível.

Algoritmo

Faça $i = 1$ e crie aleatoriamente uma solução S_i .

Enquanto $iteração_{atual} \leq max_iterações$, faça:

$iteração_{atual} = iteração_{atual} + 1$.

Crie uma lista de movimentos $M = \{m_1, m_2, \dots, m_k\}$.

Calcule a função objetivo do problema S_i considerando a aplicação de cada movimento $m_j \in M$.

Verifique se o critério de aspiração será usado (solução na lista tabu pode ser aceita?).

Escolha $m_j \in M$ que produz a melhor solução S_{i+1} , tal que $tabu(m_j) = 0$.

Se $f(S_{i+1}) \leq f(S_i)$, então

$S_i = S_{i+1}$

$tabu(m_j) = 3$

$i = i + 1$

Fim

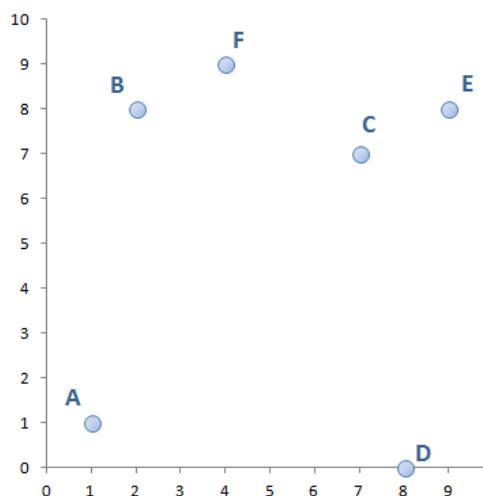
Atualize a lista tabu: $tabu(m_q) = tabu(m_q) - 1$, onde $tabu(m_q) > 0$.

Fim

Fim

Exercícios:

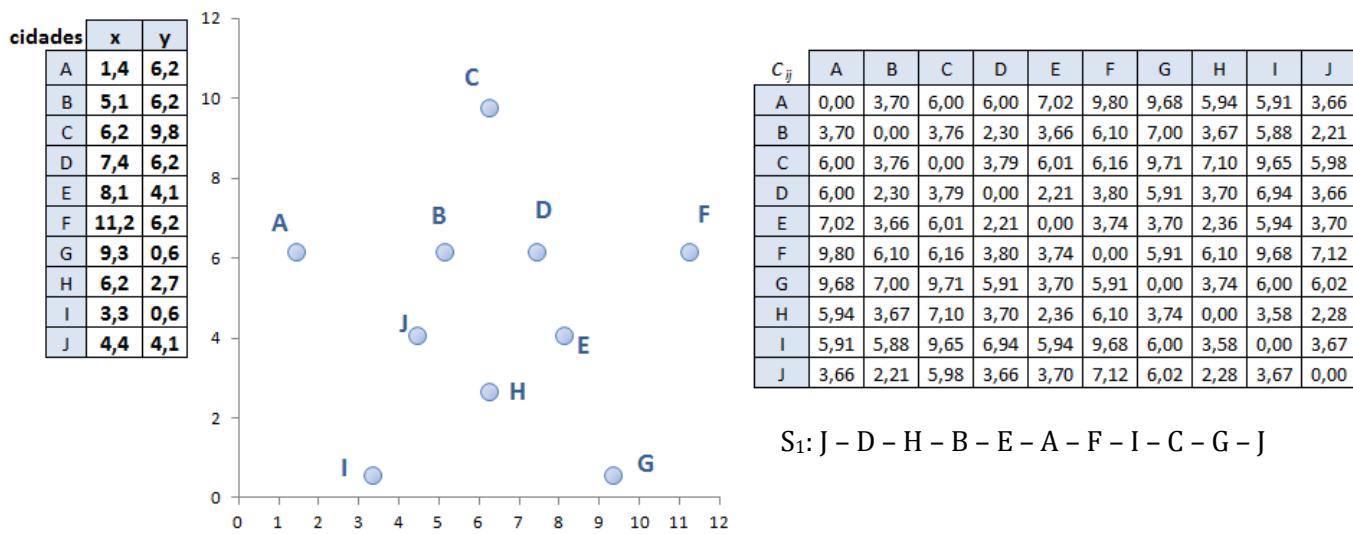
- Utilize a técnica Busca Tabu ($k = 3$) para encontrar rotas factíveis para o problema do Caixeiro Viajante com matriz de custos dada abaixo.



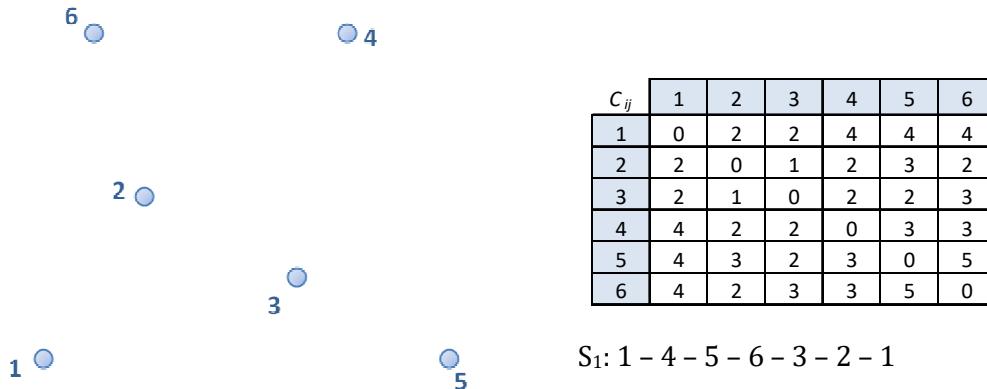
C_{ij}	A	B	C	D	E	F
A	0,00	7,07	8,49	7,07	10,63	8,54
B	7,07	0,00	5,10	10,00	7,00	2,24
C	8,49	5,10	0,00	7,07	2,24	3,61
D	7,07	10,00	7,07	0,00	8,06	9,85
E	10,63	7,00	2,24	8,06	0,00	5,10
F	8,54	2,24	3,61	9,85	5,10	0,00

$S_1: A - B - C - D - E - F - A$

2. Utilize a técnica Busca Tabu ($k = 3$) para encontrar rotas factíveis para o problema do Caixeiro Viajante com matriz de custos dada abaixo.



3. Utilize a técnica Busca Tabu ($k = 2$) para encontrar soluções para o problema do Caixeiro Viajante dado pela matriz de distâncias abaixo.



4. Utilize Busca Tabu ($k = 3$) para resolver o problema da Mochila dado abaixo, onde $n = 6$ e $P = 20$.

$$\text{Maximizar } \sum_{i=1}^n v_i z_i$$

$$\text{Sujeito a } \sum_{i=1}^n p_i z_i \leq P$$

Objeto i	1	2	3	4	5	6
Peso p_i	4	5	7	9	6	3
Valor v_i	2	2	3	4	4	2

$$S_1: (1, 0, 0, 0, 0, 0)$$

$$z_i \in \{0,1\}$$

Função de avaliação de cada solução: $f = \sum_{i=1}^n v_i z_i - \gamma \max\{0, \sum_{i=1}^n p_i z_i - P\}$, onde $\gamma = \sum_{i=1}^n v_i$.

5. Utilize Busca Tabu ($k = 3$) para resolver o problema da Mochila dado abaixo, onde $n = 15$ e $P = 275$.

Objeto i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Peso p_i	63	21	2	32	13	80	19	37	56	41	14	8	32	42	7
Valor v_i	13	2	20	10	7	14	7	2	2	4	16	17	17	3	21

3.2 NUVEM DE PARTÍCULAS

A técnica de Otimização por Nuvem de Partículas (ONP) ou *Particle Swarm Optimization (PSO)* foi desenvolvida por James Kennedy (psicólogo) e Russell Eberhart (engenheiro) em 1995, com base no comportamento de pássaros em revoadas e modelado pelo biólogo Frank Heppner.

Baseia-se no comportamento social dos pássaros em revoadas, cardumes de peixes e enxames de abelhas. Algorítmicamente, tem-se um conjunto de partículas que percorrem o espaço de busca apresentando comportamentos aleatórios em relação à **individualidade** e à **sociabilidade**.

A **individualidade** de uma partícula está relacionada à ênfase dada, em seus movimentos, à melhor solução já encontrada por ela mesma, enquanto sua **sociabilidade** reflete o grau de importância dado por ela à melhor solução já encontrada por seus vizinhos.

O conceito de vizinhança em *PSO* não é o mesmo utilizado pelas metaheurísticas de busca por entornos, pois cada partícula, associada a uma solução que evolui, é vizinha de um conjunto de partículas que nunca é alterado.

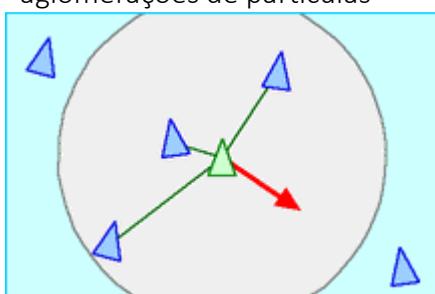
A estrutura de vizinhanças é construída de forma que os progressos obtidos em cada região tenham influência, potencialmente, em todas as partículas.

Aplicações comuns de PSO:

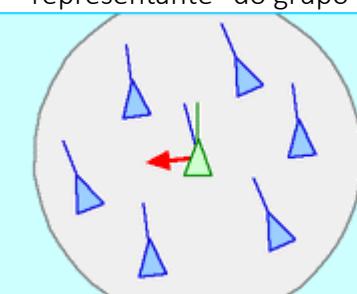
- Evolução de redes neurais artificiais;
- Extração de regras de RNAs;
- Escalonamento de tarefas (*Multi-objective Job shop scheduling*);
- Problema do Caixeiro Viajante;
- Roteamento de veículos (*Capacitated Vehicle Routing*);
- Caminho ótimo para operações de perfuração automatizadas;
- Mineração de dados para tarefas de classificação;
- Posicionamento de bases em computação móvel; e
- Aproximação poligonal ótima de curvas digitais.

Conceitos da técnica, imitando a natureza:

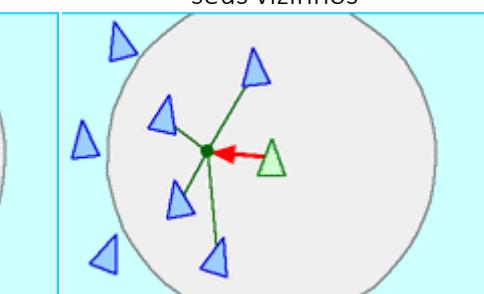
Separação: usada para evitar aglomerações de partículas



Alinhamento: encaminha a busca para a partícula "representante" do grupo



Coesão: uma partícula movimenta-se na "média" dos seus vizinhos



O PSO é um método baseado em população, como o Algoritmo Genético, entretanto, o conceito básico é **cooperação** em vez da **rivalidade**.

PSO trata-se de uma técnica atrativa, porque é computacionalmente “barata”, robusta e simples.

Outras aplicações:

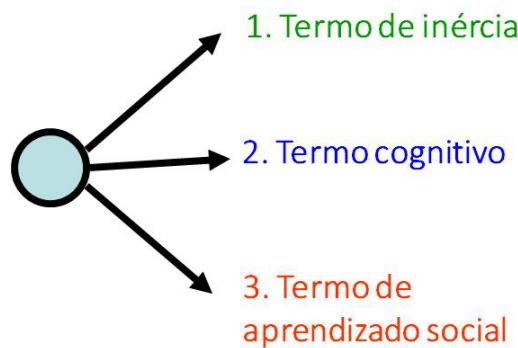
- Controle de veículos por controle remoto;
- Controle para detectar e eliminar tumores;
- O filme da Disney “O Rei Leão” foi o primeiro a usar tecnologia de nuvem de partículas para fazer a cena da debandada de pássaros. O filme da New Line Cinema “O Senhor dos Anéis” também utilizou técnicas semelhantes nas cenas de batalhas;
- Réplica de dados em grade.

Apesar da semelhança com AG, esta técnica não usa operadores genéticos (crossover, mutação, etc), uma partícula movimenta-se com velocidade usando a própria experiência, além da experiência de todas as partículas.

A ideia é similar ao bando de pássaros (ou cardume de peixes ou enxame de abelhas) procurando comida, usando habilidades de:

- troca de informações entre vizinhos;
- memorizar uma posição anterior; e
- usar informações para tomada de decisões.

Três termos definem uma nova velocidade para uma partícula:



- | |
|--|
| <input type="checkbox"/> Força a partícula a mover-se na mesma direção
<input type="checkbox"/> Tendência para seguir a própria direção com a mesma velocidade |
| <input type="checkbox"/> Melhora o indivíduo
<input type="checkbox"/> Força a partícula a voltar a uma posição anterior que seja melhor do que a atual
<input type="checkbox"/> Tendência conservativa |
| <input type="checkbox"/> Força a partícula a seguir a direção de seus melhores vizinhos
<input type="checkbox"/> Como em todo rebanho, mas seguindo os melhores |

Elementos do algoritmo:

A : população de agentes.

x_i : posição do agente a_i no espaço de soluções.

f : função de avaliação.

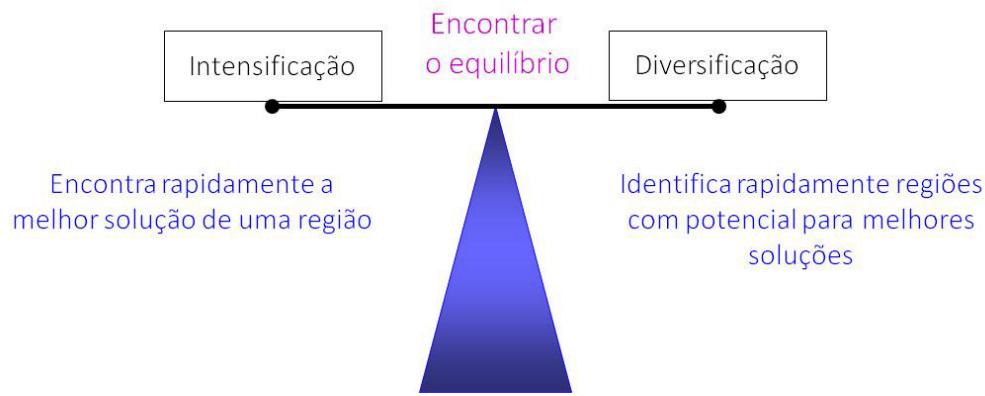
v_i : velocidade do agente a_i .

$V(a_i)$: conjunto **fixo** de vizinhos do agente a_i .

Todos os agentes estão conectados, direta ou indiretamente.

O PSO é lembrado pelas seguintes características de busca de soluções:

- **Intensificação** é a exploração das soluções encontradas em proezas anteriores;
- **Diversificação** é a busca por soluções ainda não visitadas.



Para cada agente a_i , temos:

$$\begin{array}{c} \text{inércia} \quad \text{cognitivo} \quad \text{aprendizado social} \\ \downarrow \qquad \downarrow \qquad \downarrow \\ v_i = wv_i + \eta_1 \cdot \text{rnd.}(pbest_i - x_i) + \eta_2 \cdot \text{rnd.}(gbest - x_i) \\ x_i = x_i + v_i \end{array}$$

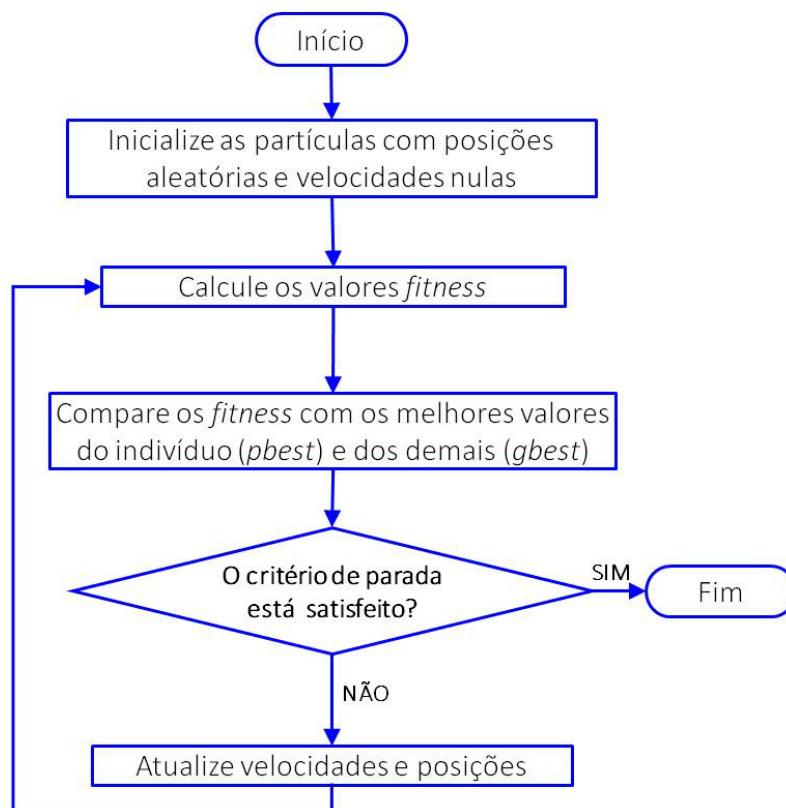
onde:

$pbest_i$ é a melhor posição em que a partícula a_i já esteve;

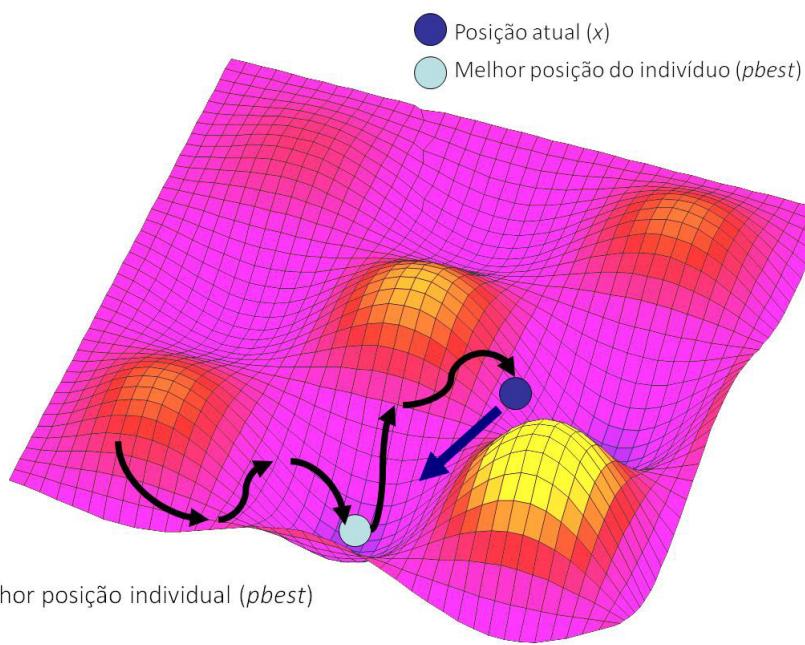
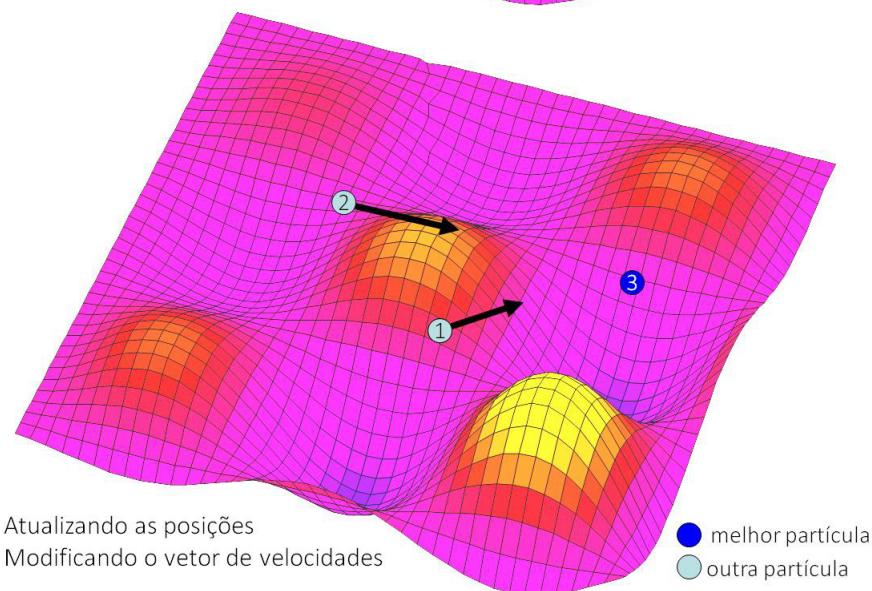
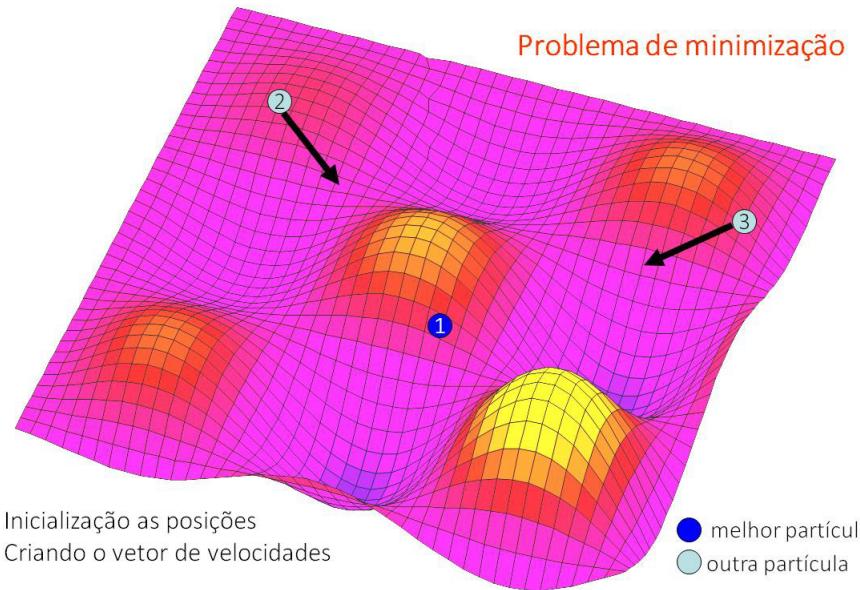
$gbest$ é a melhor posição em que algum vizinho de a_i já esteve;

w, η_1 e η_2 são os pesos de inércia, cognitivo e de aprendizado social, respectivamente.

Fluxograma do algoritmo PSO



Exemplo de evolução do PSO



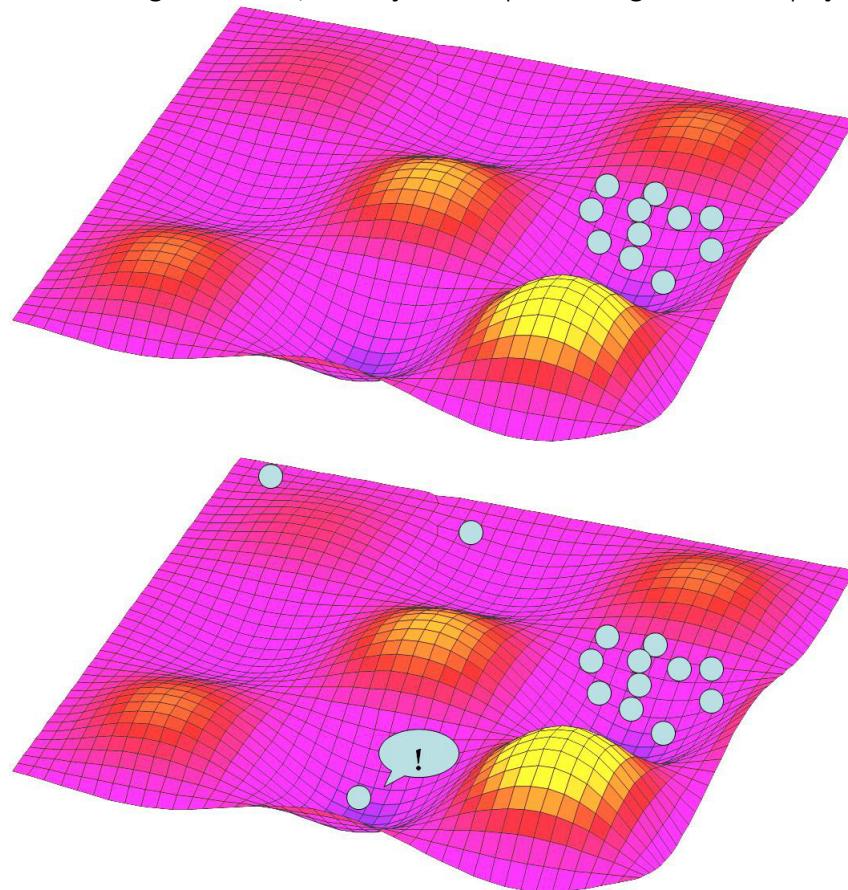
Armadilhas do PSO

As partículas tendem a se agrupar, ou seja, devido a uma convergência rápida demais, a solução fica presa em um ponto ótimo local.

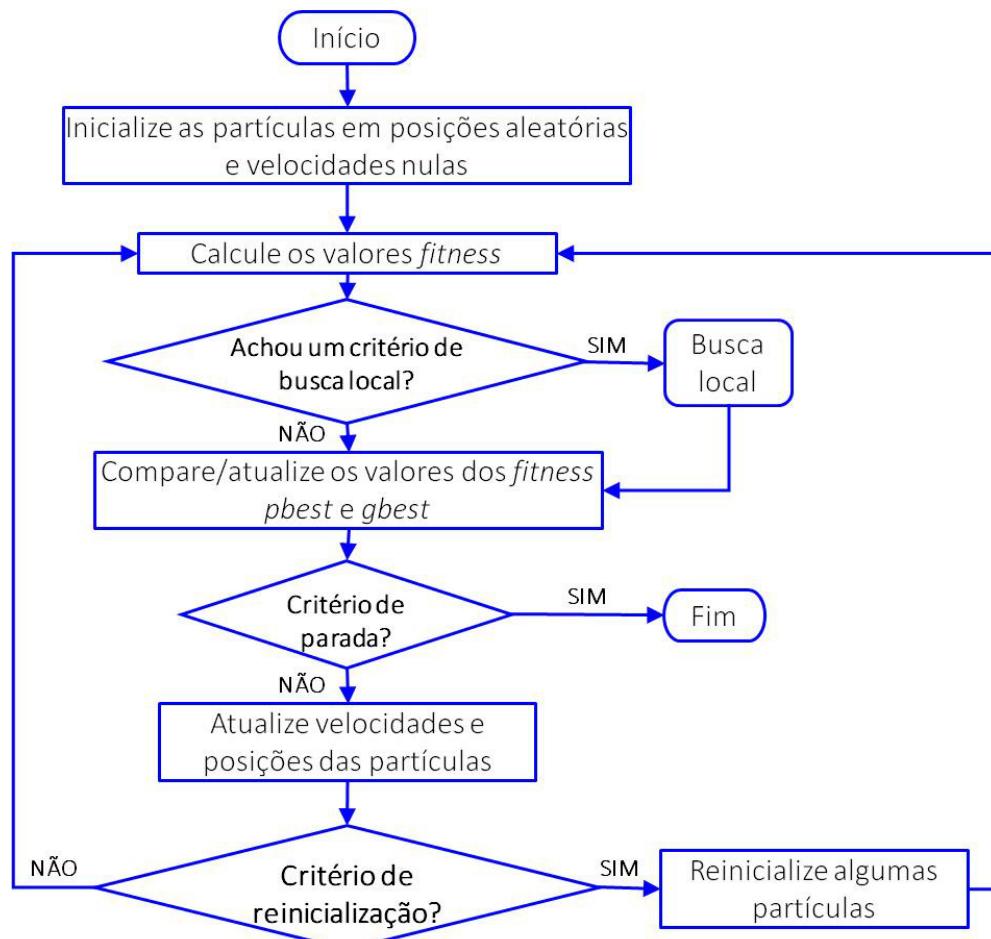
O movimento de alguma partícula pode ser levado a um ponto de solução infactível.

As partículas podem mapear um espaço inapropriado de soluções factíveis.

Quando as partículas ficarem aglomeradas, a solução é “espalhar” algumas no espaço de busca.



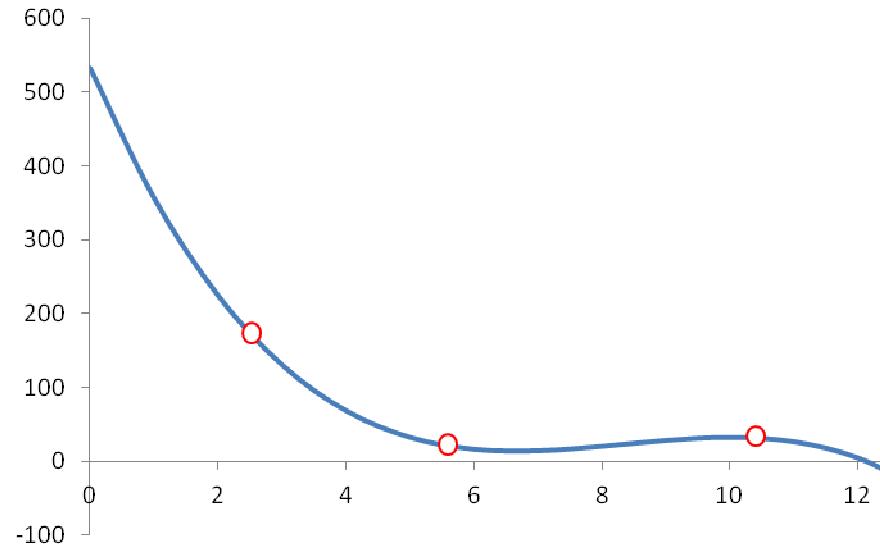
Fluxograma do algoritmo PSO com reinicialização de partículas



Exercícios:

1. Utilize o algoritmo PSO para encontrar uma aproximação do valor mínimo da função

$f(x) = (x - 4)^2 - (x - 8)^3 + 5$ com as três partículas dadas abaixo, usando 10 iterações completas.



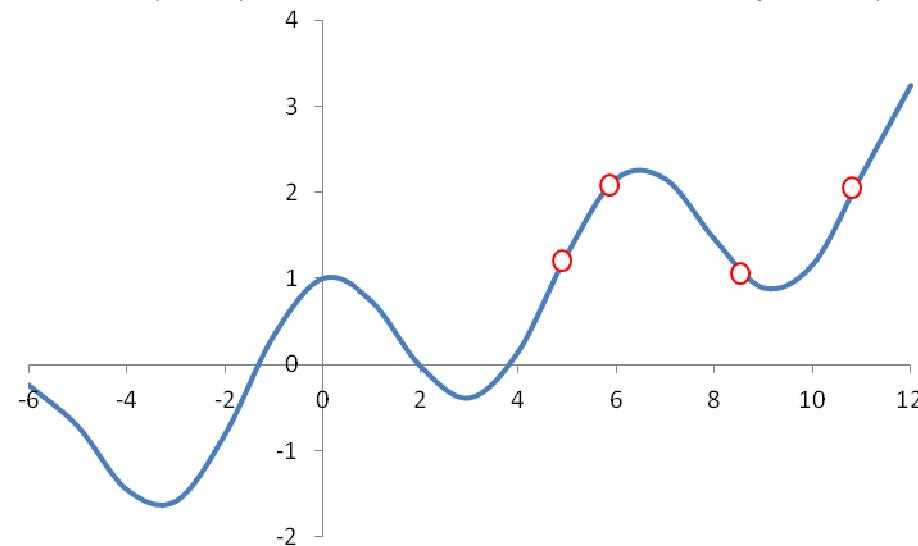
partículas:	x	velocidades:	v
1	2,5	1	0
2	5,6	2	0
3	10,5	3	0

$$w = 0,3$$

$$\eta_1 = 0,7$$

$$\eta_2 = 0,7$$

2. Utilize o algoritmo PSO para encontrar uma aproximação do valor mínimo da função $f(x) = \cos(x) + \frac{x}{5}$ com as quatro partículas dadas abaixo, usando 10 iterações completas.



partículas:	x	velocidades:	v
1	5	1	0
2	6	2	0
3	8,5	3	0
4	11	4	0

$$w = 0,2$$

$$\eta_1 = 0,3$$

$$\eta_2 = 0,5$$

3. Utilize o algoritmo PSO para encontrar uma aproximação do valor máximo da função

$f(x_1, x_2) = \sin(x_1) + \cos(x_2) + \frac{x_2}{5}$ com as quatro partículas dadas abaixo, usando 10 iterações.

partículas:	x_1	x_2	velocidades:	v
1	-2	-1	1	0
2	-1	-2	2	0
3	0	2	3	0
4	1	2	4	0

$$w = 0,4$$

$$\eta_1 = 0,3$$

$$\eta_2 = 0,4$$

Adaptações do PSO para resolver o problema do Caixeiro Viajante

Partículas

Em PCV com N cidades, procuramos ciclos com $N + 1$ vértices. Logo uma partícula consiste em um ciclo com as cidades do PCV:

$$x = (n_1, n_2, \dots, n_N, n_{N+1}),$$

esta partícula somente é aceita se todos os arcos (n_i, n_{i+1}) existem.

Função de *fitness*

O custo de cada partícula é dado por: $\sum_{i=1}^N c_{n_i, n_{i+1}}$

Velocidade

Podemos definir um operador v que quando aplicado a uma partícula retorna uma outra posição. Definimos então uma lista de transposições de elementos da partícula, que são trocas de 2 a 2:

$$v = \{(i_k, j_k)\}, \text{ onde } i_k, j_k \in \{1, 2, \dots, N\},$$

que significa a troca dos elementos (i_1, j_1) , depois (i_2, j_2) , até k .

Movimento

O movimento da partícula é obtido aplicando-se a velocidade à partícula: $x = x + v$.

Por exemplo: $x = (1, 2, 3, 4, 5, 1)$, $v = \{(1, 2), (2, 3)\}$

com a primeira troca $(1, 2)$, temos: $x' = (2, 1, 3, 4, 5, 2)$

com a segunda troca $(2, 5)$, temos $x' = (2, 5, 3, 4, 1, 2)$

Subtração

A diferença $x_i - x_j$ é definida como a velocidade que deve ser aplicada na posição x_j para obter a posição x_i . Quando $x_i = x_j$, temos $v = 0$.

Adição

O resultado da soma das velocidades $v_i + v_j$ é a lista de transposições primeiro de v_i , depois de v_j .

Multiplicação de escalar por vetor

- quando $c = 0$, temos $cv = \emptyset$

- quando $c \in (0, 1)$, truncase v .

Exemplo:

$$c = 0,6; \text{ e } v = \{(1, 2), (3, 5), (17, 23), (7, 3), (8, 19)\}$$

$$\text{logo, } cv = \{(1, 2), (3, 5), (17, 23)\}$$

- quando $c > 1$, defina $c = k + c'$, onde $k \in \mathbb{N}$, e $c' \in (0, 1)$.

$$\text{logo, } cv = \underbrace{v + v + \dots + v}_{k \text{ vezes}} + c'v$$

$$v_i = wv_i + \eta_1 \cdot \text{rnd.}(pbest_i - x_i) + \eta_2 \cdot \text{rnd.}(gbest - x_i)$$

$$x_i = x_i + v_i$$

Exercícios:

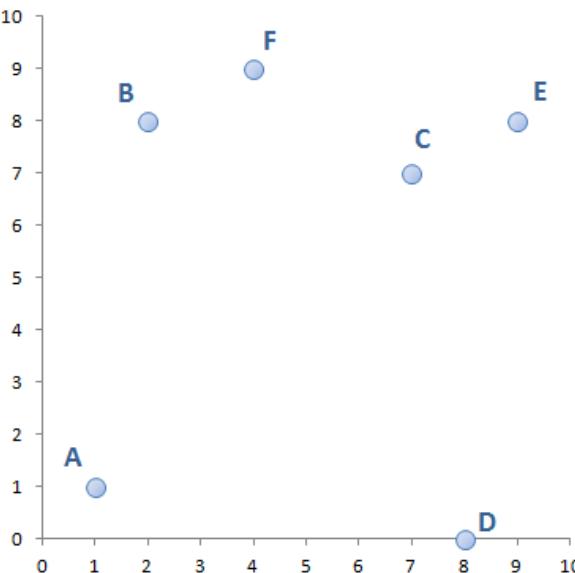
1. Resolva o problema do Caixeiro Viajante com 6 cidades dado abaixo, usando as partículas e velocidades indicadas.

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

$w = 0,3$

$\eta_1 = 0,7$

$\eta_2 = 0,2$



C_{ij}	A	B	C	D	E	F
A	0,00	7,07	8,49	7,07	10,63	8,54
B	7,07	0,00	5,10	10,00	7,00	2,24
C	8,49	5,10	0,00	7,07	2,24	3,61
D	7,07	10,00	7,07	0,00	8,06	9,85
E	10,63	7,00	2,24	8,06	0,00	5,10
F	8,54	2,24	3,61	9,85	5,10	0,00

$v_i = wV_i + \eta_1 * rnd*(pbest-x_i) + \eta_2 * rnd*(gbest-x_i)$

$v_i = 0,3V_i + 0,7*rnd*(pbest-x_i) + 0,2*rnd*(gbest-x_i)$

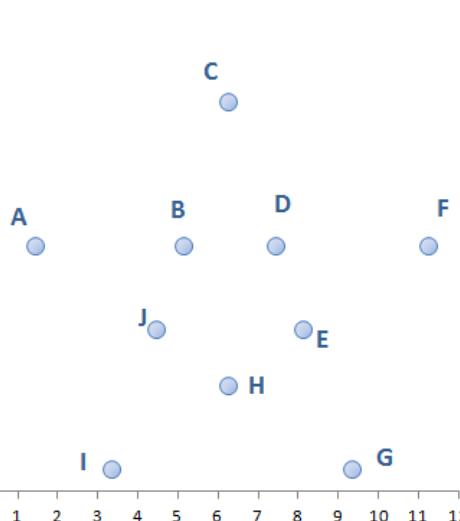
2. Resolva o problema do Caixeiro Viajante com 10 cidades dado abaixo, usando as partículas e velocidades indicadas.

cidades	x	y
A	1,4	6,2
B	5,1	6,2
C	6,2	9,8
D	7,4	6,2
E	8,1	4,1
F	11,2	6,2
G	9,3	0,6
H	6,2	2,7
I	3,3	0,6
J	4,4	4,1

$w = 0,3$

$\eta_1 = 0,7$

$\eta_2 = 0,2$



C_{ij}	A	B	C	D	E	F	G	H	I	J
A	0,00	3,70	6,00	6,00	7,02	9,80	9,68	5,94	5,91	3,66
B	3,70	0,00	3,76	2,30	3,66	6,10	7,00	3,67	5,88	2,21
C	6,00	3,76	0,00	3,79	6,01	6,16	9,71	7,10	9,65	5,98
D	6,00	2,30	3,79	0,00	2,21	3,80	5,91	3,70	6,94	3,66
E	7,02	3,66	6,01	2,21	0,00	3,74	3,70	2,36	5,94	3,70
F	9,80	6,10	6,16	3,80	3,74	0,00	5,91	6,10	9,68	7,12
G	9,68	7,00	9,71	5,91	3,70	5,91	0,00	3,74	6,00	6,02
H	5,94	3,67	7,10	3,70	2,36	6,10	3,74	0,00	3,58	2,28
I	5,91	5,88	9,65	6,94	5,94	9,68	6,00	3,58	0,00	3,67
J	3,66	2,21	5,98	3,66	3,70	7,12	6,02	2,28	3,67	0,00

$v_i = wV_i + \eta_1 * rnd*(pbest-x_i) + \eta_2 * rnd*(gbest-x_i)$

$v_i = 0,3V_i + 0,7*rnd*(pbest-x_i) + 0,2*rnd*(gbest-x_i)$

x_1	A	B	D	F	C	E	H	J	G	I
x_2	A	J	I	H	G	E	D	C	B	F
x_3	H	B	E	D	J	A	C	F	G	I

3. Utilize Nuvem de partículas para resolver o problema da Mochila dado, onde $n = 6$ e $P = 20$.

$\text{Maximizar } \sum_{i=1}^n v_i z_i$

$\text{Sujeito a } \sum_{i=1}^n p_i z_i \leq P$

$z_i \in \{0,1\}$

Objeto i	1	2	3	4	5	6
Peso p_i	4	5	7	9	6	3
Valor v_i	2	2	3	4	4	2

$w = 0,3$

$\eta_1 = 0,7$

$\eta_2 = 0,2$

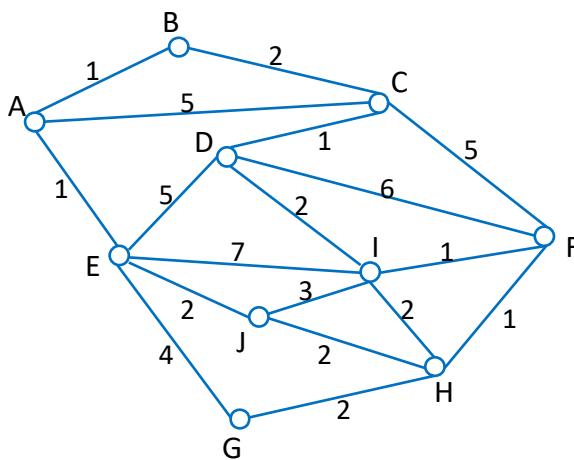
Função de avaliação de cada solução: $f = \sum_{i=1}^n v_i z_i - \gamma \max(0, \sum_{i=1}^n p_i z_i - P)$, onde $\gamma = \sum_{i=1}^n v_i$.

partícula 1	1	0	0	1	0	0
partícula 2	0	1	1	0	0	0
partícula 3	0	0	1	0	1	1

4. Utilize Nuvem de partículas para resolver o problema da Mochila dado abaixo, onde $n = 15$ e $P = 275$. Crie aleatoriamente 3 partículas factíveis para inicialização do algoritmo.

Objeto i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Peso p_i	63	21	2	32	13	80	19	37	56	41	14	8	32	42	7
Valor v_i	13	2	20	10	7	14	7	2	2	4	16	17	17	3	21

5. Utilize Nuvem de partículas para resolver o problema de caminho mínimo entre os vértices A e F dados no grafo a seguir. Utilize a mesma ideia que usamos no PCV para calcular o fitness, e a mesma ideia do problema da Mochila pra criar penalidades para arcos inexistentes.



partícula 1	A	B	C	D	E	F	G	H	I	J
partícula 2	A	E	D	C	B	G	H	I	F	J
partícula 3	A	E	C	D	B	I	J	F	H	G

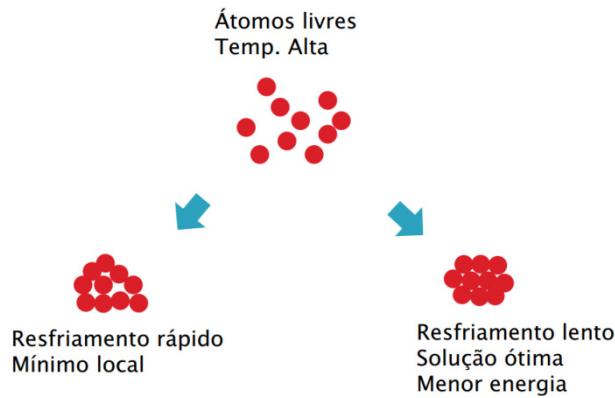
3.3 SIMULATED ANNEALING

A origem da técnica de otimização conhecida por Simulated Annealing (têmpera simulada ou anelamento simulado) vem de 1953, quando foi usada para simular em um computador o processo de annealing de cristais.

O annealing (anelamento ou têmpera) de certos materiais consiste em submetê-los inicialmente a altas temperaturas e reduzi-las gradualmente até atingirem, com aumentos e reduções do estado de energia, o equilíbrio térmico, tornando-os assim, consistentes e rígidos.

A técnica matemática de *Simulated Annealing* faz uma simulação algorítmica do processo físico da têmpera de materiais. A idéia de aplicar este método para resolver problemas de otimização combinatória surgiu bem mais tarde [Kirkpatrick et al., 1983], [Aragon et al., 1984].

O resfriamento gradativo de um material a partir de uma alta temperatura inicial leva o material a estados mínimos de energia. Informalmente esses estados são caracterizados por uma perfeição estrutural do material congelado que não se obteria caso o resfriamento não tivesse sido gradativo.



Sob outras condições menos cuidadosas de resfriamento, o material se cristalizaria com uma energia “localmente mínima”, apresentando imperfeições estruturais.

A técnica utiliza o Método de Monte Carlo para gerar sequências de estados do material caracterizados pelas posições de suas partículas. Sob altas temperaturas as partículas estão totalmente “desorganizadas” correspondendo a uma configuração aleatória de um problema de otimização (em geral, distante da configuração ótima desejada). Uma pequena alteração (perturbação) nas posições de algumas destas partículas resulta numa variação de energia, que equivale a uma alteração no valor da função objetivo.

Simulação

A referida simulação a uma temperatura fixa T , consiste em dar um pequeno deslocamento a um dos átomos, computando a variação ΔE da energia do sistema, considerando os casos:

- Se $\Delta E \leq 0$, o deslocamento é incorporado ao estado do sistema, que é utilizado no passo seguinte;
- Caso contrário, isto é, se $\Delta E > 0$, a aceitação ou não do deslocamento passa a ser uma decisão probabilística.

Os estados possíveis de um metal correspondem a soluções do espaço de busca. A energia em cada estado corresponde ao valor da função objetivo, considerando: **Energia mínima** (se o problema for de **minimização** ou **máxima**, se de **maximização**) corresponde ao valor de uma solução ótima local, possivelmente global.

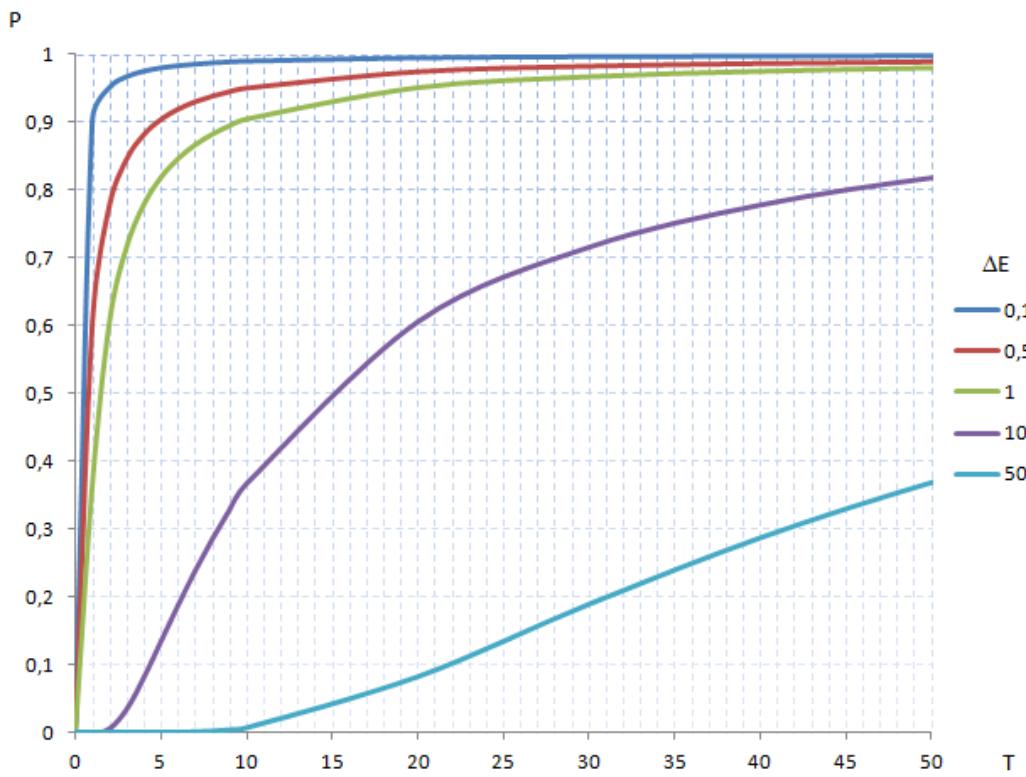
Considerando os estados sucessivos de energia E_{i+1} e E_i , podem ocorrer os seguintes casos:

- Se o novo estado é de energia menor que o estado atual ($\Delta E \leq 0$), esse novo estado passa a ser o estado atual;
- Se o novo estado tem uma energia maior que o estado atual ($\Delta E > 0$), a probabilidade de se mudar do estado atual para o novo estado é:

$$e^{-\Delta E/T}, \text{ onde } T = \text{temperatura atual.}$$

Este procedimento é repetido até se atingir o equilíbrio térmico (passo de Metropolis).

Quando $\Delta E = 0$, temos uma situação de estabilidade ou equilíbrio. Neste caso tem-se duas configurações vizinhas com o mesmo custo (coincidência pouco provável de acontecer na prática).



Passos para utilização do método:

- Identificar a função energia do sistema com a função objetivo que se quer otimizar, por exemplo, minimizar;
- Associar os átomos do sistema às variáveis do problema;
- Para cada temperatura de uma sequência de temperaturas decrescentes realiza-se a simulação descrita;
- No final do processo, espera-se que o sistema estacione em um estado de energia globalmente mínima, por analogia com a física do problema.

Parâmetros

O fato do método *Simulated Annealing* permitir a aceitação de configurações intermediárias do problema, na qual o valor da função objetivo aumenta quando que se deseja minimizar é importante. Essa aceitação temporária de soluções significa que o método permite pesquisar todo o espaço de soluções, para evitar ótimos locais.

Laarhoven e Aarts (1987) publicaram um estudo detalhado da teoria e aplicações de *Simulated Annealing*, onde apresentam várias formas de se obter o fator de decréscimo da temperatura, dentre as quais se tem a seguinte:

$$T_i = \alpha T_{i-1}.$$

Mitra e Romeo (1985) propõem que α seja igual a 0,9.

Laarhoven e Aarts (1987) indicam várias formas de se obter a temperatura inicial, dentre as quais:

$$T_0 = -\Delta E^+ / \ln(\xi_0),$$

proposta por Johnson et al. (1987), onde:

- ΔE^+ é a média aritmética, para um número randômico de perturbações, dos incrementos da função objetivo. (por exemplo, gerar 10 soluções aleatórias para o PCV, calcular a variação obtida na função objetivo, de uma para outra solução gerada: da 1^a para a 2^a; da 2^a para a 3^a, ...);

- ξ_0 é um valor empírico, aproximadamente igual a 0,8.

Pode-se estimar previamente o número de iterações necessárias (m) para variação da temperatura, pois em geral as fórmulas possuem, implicitamente, um fator α de seu decréscimo.

Se $T_i = \alpha T_{i-1}$, então $T_1 = \alpha T_0$; $T_2 = \alpha T_1 = \alpha(\alpha T_0) = \alpha^2 T_0$; ...; $T_m = \alpha^m T_0$; assim:

$$m = \log \alpha (T_m/T_0)$$

Valores sugeridos: $\alpha = 0,9$, se $m = 100$ iterações, para T_0 valores entre 0,5 a 100.

Algoritmo

Inicialização: S_0 (solução inicial), M (máximo de iterações), V (máximo de vizinhos), L (máximo de sucessos), $S = S_0$, $T = T_0$, iteração = 1

Repete

i = 1 (número de soluções vizinhas encontradas), *nsucess = 0*

Repete

Crie uma solução S_{i+1} , vizinha de S_i

Calcule a função objetivo para S_{i+1}

Calcule a probabilidade P de aceitação de nova solução: $P = e^{-\Delta E/T}$

Se $\Delta E = f(S_{i+1}) - f(S_i) \leq 0$ ou $P > rnd$, então

$S = S_{i+1}$ (melhor solução)

nsucess = nsucess + 1

fim

i = i + 1 (tentativas)

Até *nsucess ≥ L* ou *i > V*

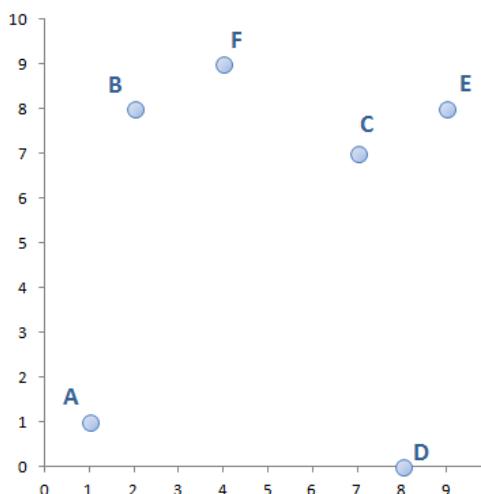
$T = \alpha T$

iteração = iteração + 1

Até *nsucess = 0* ou *iteração ≥ M*

Exercícios:

- Utilize a técnica SA para encontrar rotas factíveis para o problema do Caixeiro Viajante com matriz de custos dada abaixo.



C_{ij}	A	B	C	D	E	F
A	0,00	7,07	8,49	7,07	10,63	8,54
B	7,07	0,00	5,10	10,00	7,00	2,24
C	8,49	5,10	0,00	7,07	2,24	3,61
D	7,07	10,00	7,07	0,00	8,06	9,85
E	10,63	7,00	2,24	8,06	0,00	5,10
F	8,54	2,24	3,61	9,85	5,10	0,00

$$\alpha = 0,9$$

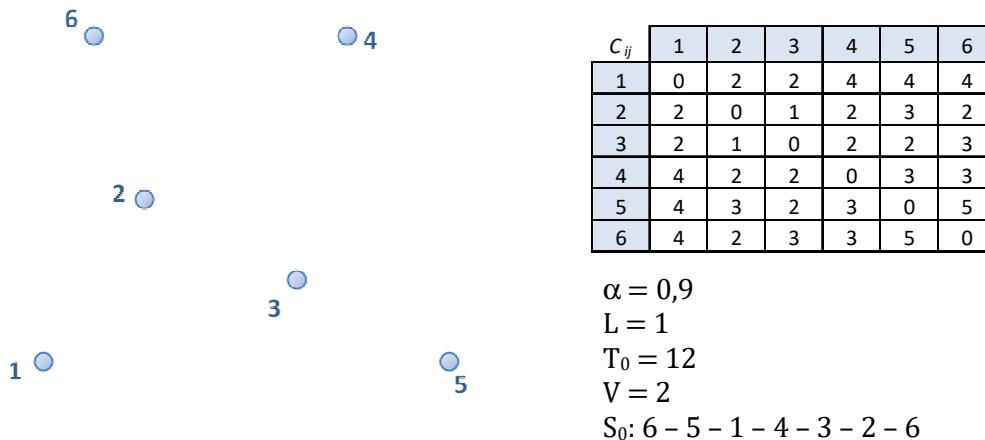
$$L = 1$$

$$T_0 = 17$$

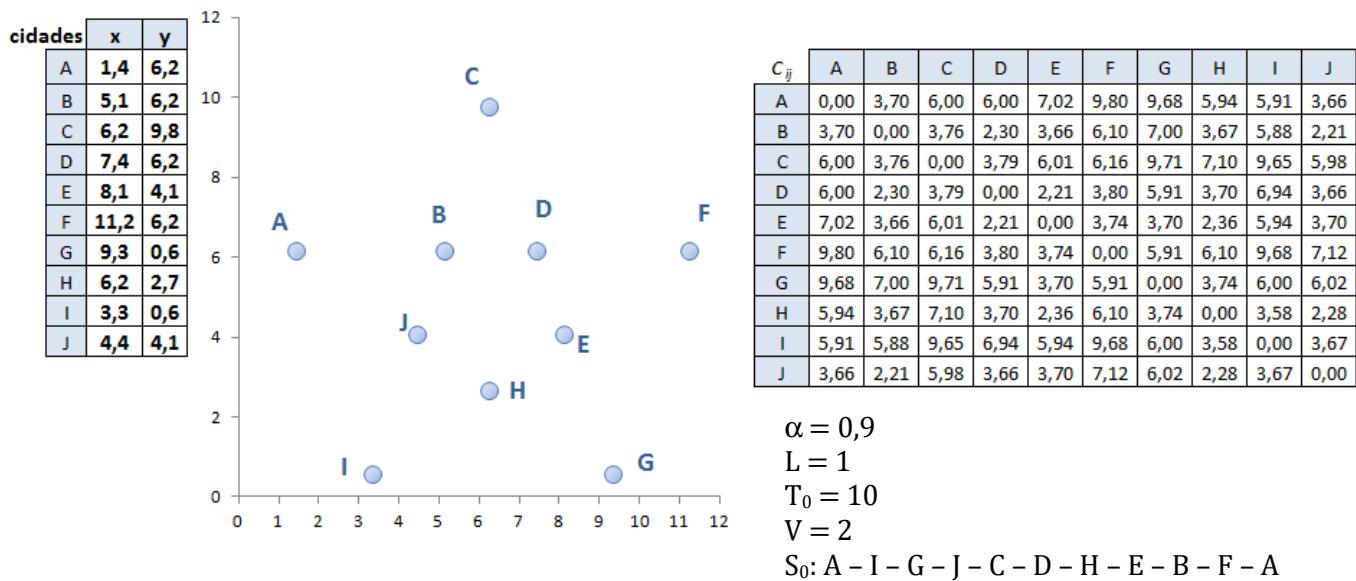
$$V = 2$$

$$S_0: B - D - F - A - E - C - B$$

2. Utilize SA para encontrar soluções para o problema do Caixeiro Viajante dado pela matriz de distâncias abaixo.



3. Utilize SA para encontrar soluções para o problema do Caixeiro Viajante dado pela matriz de distâncias abaixo.



4. Utilize SA para resolver o problema da Mochila dado abaixo, onde $n = 6$ e $P = 20$.

$$\text{Maximizar } \sum_{i=1}^n v_i z_i$$

$$\text{Sujeito a } \sum_{i=1}^n p_i z_i \leq P$$

$$z_i \in \{0,1\}$$

Objeto i	1	2	3	4	5	6
Peso p_i	4	5	7	9	6	3
Valor v_i	2	2	3	4	4	2

$$\alpha = 0,9$$

$$L = 1$$

$$T_0 = 19$$

$$V = 3$$

$$S_0: (1,0,0,0,0,0)$$

Função de avaliação de cada solução: $f = \sum_{i=1}^n v_i z_i - \gamma \max(0, \sum_{i=1}^n p_i z_i - P)$, onde $\gamma = \sum_{i=1}^n v_i$.

5. Utilize SA para resolver o problema da Mochila dado abaixo, onde $n = 15$ e $P = 275$.

Objeto i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Peso p_i	63	21	2	32	13	80	19	37	56	41	14	8	32	42	7
Valor v_i	13	2	20	10	7	14	7	2	2	4	16	17	17	3	21

$$\alpha = 0,9$$

$$L = 1$$

$$T_0 = 19$$

$$V = 3$$

3.4 ITERATED LOCAL SEARCH

O método *ILS – Iterated Local Search* – parte do princípio de que os ótimos locais de um problema de otimização podem ser gerados a partir de perturbações na solução ótima local corrente. A perturbação precisa ser suficientemente forte para permitir que a busca local explore diferentes soluções e fraca o suficiente para evitar um reinício aleatório.

Este método faz uma combinação entre intensificação com diversificação da seguinte forma:

Intensificação: fazendo “pequenas” perturbações na solução ótima corrente;

Diversificação: aceitando quaisquer soluções x' e aplicando-se “grandes” perturbações na solução ótima corrente.

Algoritmo

```

 $x_0 = \text{Solução_Inicial}$ 
 $x = \text{busca_local}(x_0)$  aplica uma melhoria na solução inicial
Repita
     $x' = \text{perturbação}(x, \text{histórico})$  encontra uma nova solução, guiada por um
        histórico de trocas
     $x'' = \text{busca_local}(x')$  aplica uma melhoria na solução  $x'$ 
    Se  $f(x'') < f(x)$ , então
         $x = x''$  (aceita a melhor solução)
    Caso contrário, se  $f(x') < f(x)$ , então
         $x = x'$  (aceita a melhor solução)
    Fim
Enquanto o critério de parada não for satisfeito

```

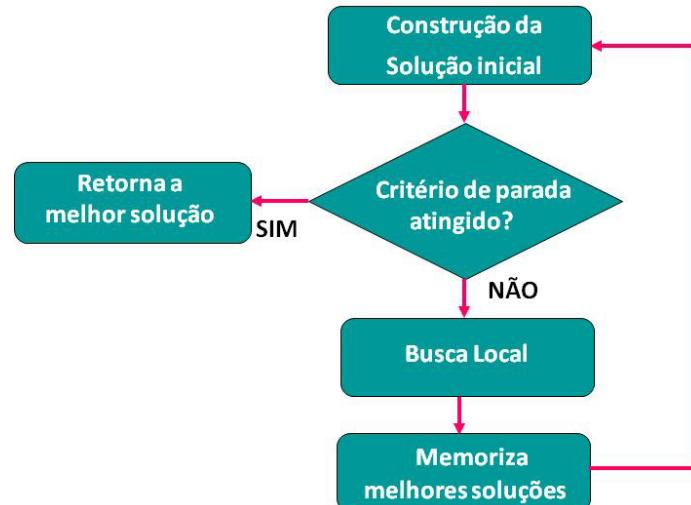
3.5 GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) é um método de otimização combinatorial que foi desenvolvido por Feo e Resende (1989 e 1995).

Trata-se de um processo iterativo, no qual a cada iteração uma nova solução inicial é gerada aleatoriamente e cada iteração consiste em 2 fases:

1. Construtiva: Geração Gulosa, Randômica e Adaptativa;
2. Busca local: gera alguma melhoria na solução corrente, através de uma busca local na vizinhança para encontrar o ótimo local.

Fluxograma do algoritmo



Algoritmo

`Melhor_solução = M`, função de avaliação: `f`.

Repete

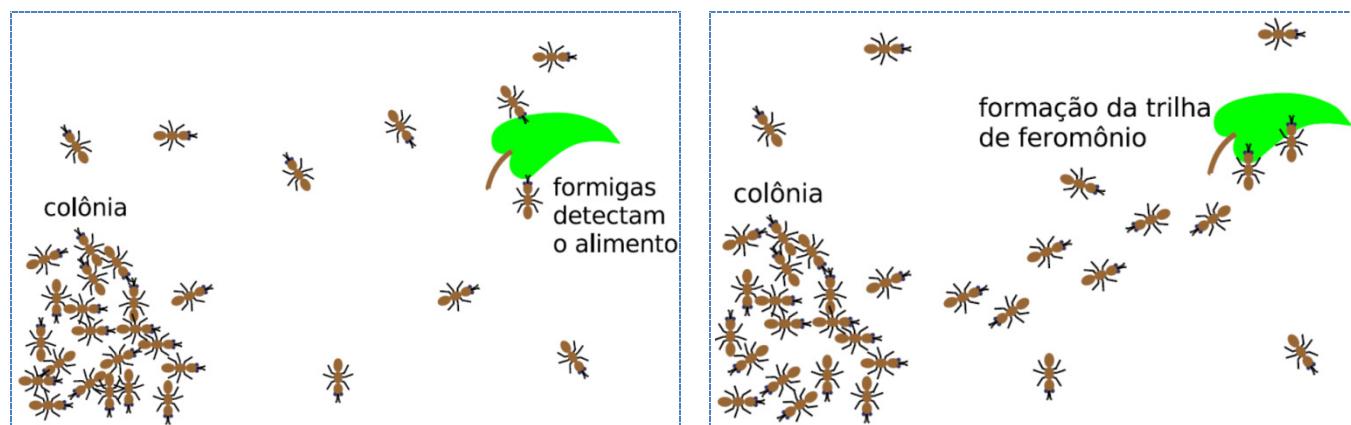
```

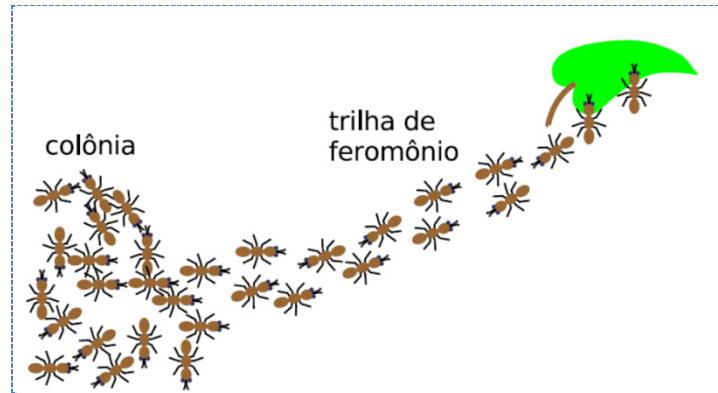
X = solução_grasp (criar uma solução aleatória por inserção gulosa de elementos)
X = busca_local(X) (aplicar uma busca de vizinhança para melhorar a solução X:
trocas de arcos)
Se f(X) < Melhor_solução, então
  Melhor_solução = X (aceita a melhor solução)
Fim
  
```

Enquanto o critério de parada não for satisfeito

3.6 COLÔNIA DE FORMIGAS

A técnica de Otimização por Colônia de Formigas (ACO: Ant Colony System Optimization) foi desenvolvida por Dorigo e Gambardella em 1997. Tem inspiração biológica, no comportamento das formigas em busca de alimento.





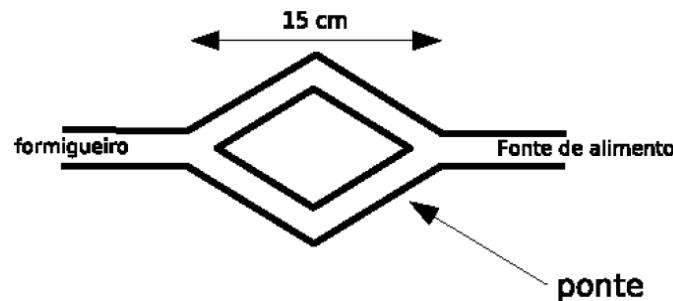
A principal aplicação desta técnica está no problema do Caixeiro Viajante e similares, como roteirização de veículos.

A relação da técnica está ligada ao comportamento forrageiro das formigas em busca de alimento ou deslocamento. Muitas espécies de formigas são quase cegas e a comunicação delas é feita através de feromônios (usados para criar caminhos – trilhas de formigas).

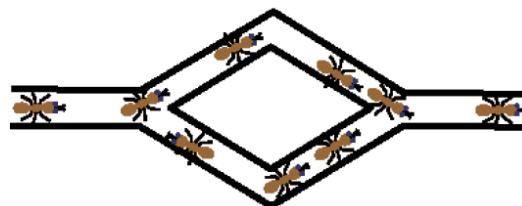
Ao caminhar, as formigas depositam no chão o feromônio, formando, uma trilha. Através do olfato, as formigas escolhem, conforme a probabilidade, o caminho com maior feromônio.

Esta trilha auxilia a formiga a encontrar o alimento e a volta ao formigueiro, além de ajudar as outras formigas a encontrar o alimento.

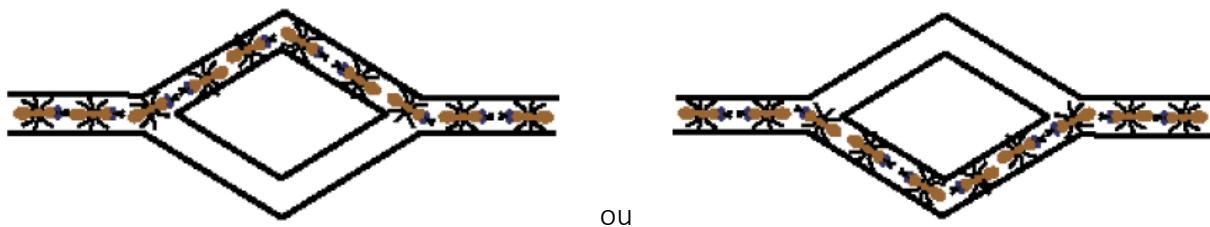
O experimento da ponte binária foi realizado por Denebourg et al., 1990, para estudar o comportamento forrageiro das formigas.



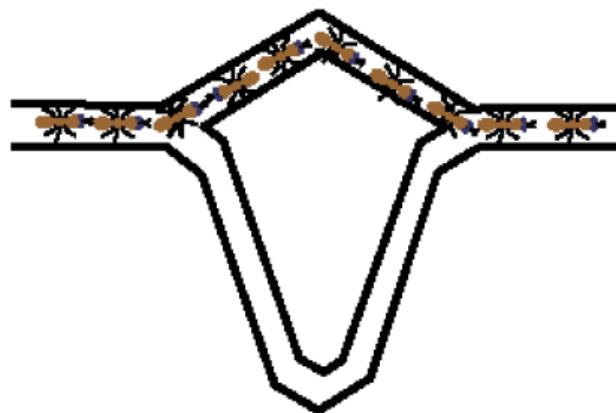
No início, as formigas são deixadas livres para escolher o caminho. Não há feromônio ainda. As formigas convergem para um dos caminhos com igual probabilidade.



Devido a flutuações, uma das pontes terá mais feromônio e atrairá as formigas com maior probabilidade.



Usando pontes de tamanhos diferentes, as formigas convergem para a ponte mais curta. A ponte curta é percorrida em menos tempo, fazendo com que mais formigas atravessem por ela. Logo, mais feromônio é depositado na ponte mais curta.

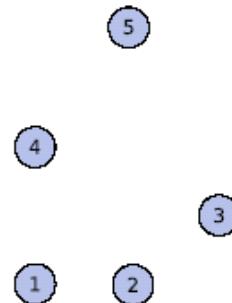


As formigas escolhem, com maior probabilidade, a ponte mais curta (com mais feromônio).

Formigas artificiais são heurísticas construtivas. As soluções são contruídas de forma probabilística utilizando duas informações: a trilha de feromônios (artificial), que muda dinamicamente durante a execução do programa; e a informação heurística específica do problema a ser resolvido.

Aplicação do ACO ao problema do Caixeiro Viajante

Considere o exemplo de 5 cidades dado abaixo:



Cada formiga construirá uma solução movendo-se de uma cidade para outra. No início, cada formiga é colocada em uma cidade diferente (ou colocada aleatoriamente).

Começando de uma cidade i , a formiga move-se escolhendo probabilisticamente a cidade vizinha j (entre os vizinhos factíveis).



A probabilidade da formiga k que está na cidade i de escolher a cidade j é dada pela regra

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in N_j^k} \tau_{il}^\alpha \eta_{jl}^\beta}, \text{ quando } j \in N_i^k.$$

onde:

τ_{ij} é o feromônio associado à aresta (i, j)

α e β são parâmetros para determinar a influência do feromônio e da informação heurística

N_j^k é a vizinhança factível da formiga k (isto é, o conjunto de cidades ainda não visitadas pela formiga k).

Associado a aresta (i, j) existe uma valor heurístico η_{ij} dado por

$$\eta_{ij} = 1/c_{ij}$$

que representa a atratividade da formiga visitar a cidade i depois de visitar a cidade j .

O valor η_{ij} é inversamente proporcional à distância c_{ij} entre as cidades i e j .

A partir de uma cidade i , a escolha da cidade candidata j é feita de acordo com a probabilidade de transição, com idéia similar à escolha por roleta de algoritmos genéticos.

No feromônio τ_{ij} associado à aresta (i, j) ocorrem dois eventos:

1. Evaporação

- evita que o feromônio acumulado cresça indefinidamente;
- permite esquecer pobres decisões do passado de busca; e
- permite soluções diferentes.

2. Depósito de feromônio de todas as formigas que passaram sobre (i, j)

Depois que todas as formigas contruíram suas viagens, o feromônio é atualizado:

$\Delta\tau_{ij}^k$ é a quantidade de feromônio que a formiga k deposita sobre a aresta (i, j) :

$$\begin{cases} \Delta\tau_{ij}^k = Q/L_k & \text{quando a aresta } (i, j) \text{ pertence } S_k \\ \Delta\tau_{ij}^k = 0 & \text{em caso contrário} \end{cases} \text{ onde } Q \text{ é uma constante}$$

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sum_{k=1}^m \Delta\tau_{ij}^k$$

↑
Evaporação
↓
Depósito

Critérios de parada:

- número máximo de iterações;
- estagnação ou convergência;
- situação na qual todas as formigas seguem sempre o mesmo percurso;

A estagnação é causada pelo excessivo crescimento de feromônio nas arestas de uma viagem subótima. Apesar da natureza estocástica do algoritmo, uma forte concentração de feromônio nas arestas força a formiga a fazer sempre o mesmo percurso.

Algoritmo do ACO

Coloque cada formiga em uma cidade aleatória

Para $t = 1$ até o número máximo de iterações

Para $k = 1$ até m (nº de formigas)

Enquanto a formiga k não construir a viagem S_k

Seleciona a próxima cidade pela regra da probabilidade:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in N_j^k} \tau_{il}^\alpha \eta_{jl}^\beta}, \text{ quando } j \in N_i^k.$$

Fim

Calcule a distância L_k da viagem S_k

Se $L_k < L^*$ então

$$S^* = S_k, L^* = L_k$$

Fim

Fim

Atualize os feromônios: $\tau_{ij} = (1 - \rho)\tau_{ij} + \sum_{k=1}^m \Delta\tau_{ij}^k$, onde:

$$\begin{cases} \Delta\tau_{ij}^k = Q/L_k & \text{quando a aresta } (i, j) \text{ pertence } S_k, \text{ onde } Q \text{ é uma constante} \\ \Delta\tau_{ij}^k = 0 & \text{em caso contrário.} \end{cases}$$

Fim

O resultado é a rota S^* .

Exercícios:

1. Resolva 2 iterações para resolver o problema do Caixeiro Viajante com 5 cidades dado abaixo, usando a técnica ACO com os parâmetros e tabelas de distâncias e feromônios dados abaixo.

5

c_{ij}	1	2	3	4	5
1	0,0	1,0	2,2	2,0	4,1
2	1,0	0,0	1,4	2,2	4,0
3	2,2	1,4	0,0	2,2	3,2
4	2,0	2,2	2,2	0,0	2,2
5	4,1	4,0	3,2	2,2	0,0

$\eta_{ij} = 1/c_{ij}$	1	2	3	4	5
1	∞	1,00	0,45	0,50	0,24
2	1,00	∞	0,71	0,45	0,25
3	0,45	0,71	∞	0,45	0,31
4	0,50	0,45	0,45	∞	0,45
5	0,24	0,25	0,31	0,45	∞

$$\alpha = 0,5$$

$$\beta = 0,5$$

4

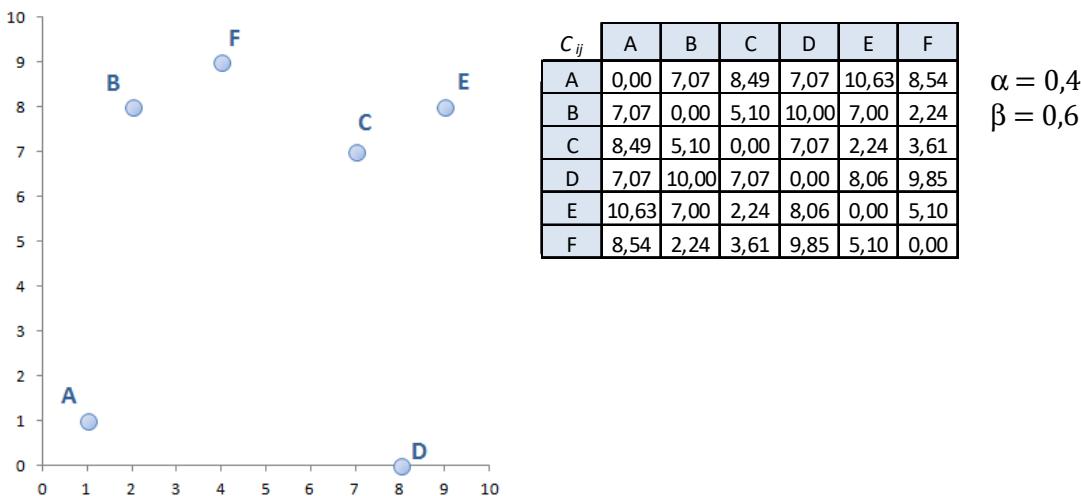
3

1

2

τ_{ij}	1	2	3	4	5
1	∞	0,30	0,25	0,20	0,30
2	0,30	∞	0,20	0,20	0,30
3	0,25	0,20	∞	0,10	0,15
4	0,20	0,20	0,10	∞	0,45
5	0,30	0,30	0,15	0,45	∞

2. Resolva 2 iterações para resolver o problema do Caixeiro Viajante de 6 cidades dado abaixo, usando a técnica ACO com tabela de feromônios aleatória.



3. Utilize Colônia de formigas para resolver o problema da Mochila dado abaixo, onde $n = 6$ e $P = 20$.

$$\text{Maximizar } \sum_{i=1}^n v_i z_i$$

$$\text{Sujeito a } \sum_{i=1}^n p_i z_i \leq P$$

Objeto i	1	2	3	4	5	6
Peso p_i	4	5	7	9	6	3
Valor v_i	2	2	3	4	4	2

$$z_i \in \{0,1\}$$

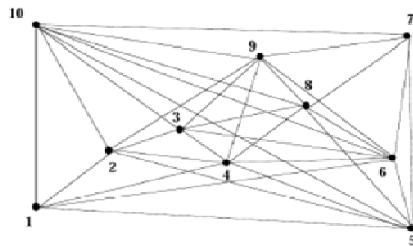
Função de avaliação de cada solução: $f = \sum_{i=1}^n v_i z_i - \gamma \max(0, \sum_{i=1}^n p_i z_i - P)$, onde $\gamma = \sum_{i=1}^n v_i$.

4. Utilize Colônia de formigas para resolver o problema da Mochila dado abaixo, onde $n = 15$ e $P = 275$.

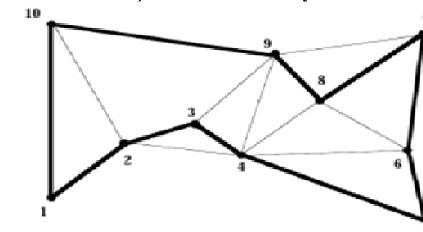
Objeto i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Peso p_i	63	21	2	32	13	80	19	37	56	41	14	8	32	42	7
Valor v_i	13	2	20	10	7	14	7	2	2	4	16	17	17	3	21

Nas formigas artificiais, existe um estado interno ou memória, para que não haja sobreposição de movimentos. O depósito de feromônio no mundo artificial ocorre com base na qualidade da solução encontrada, diferentemente do mundo real, onde formigas depositam feromônio sob demanda.

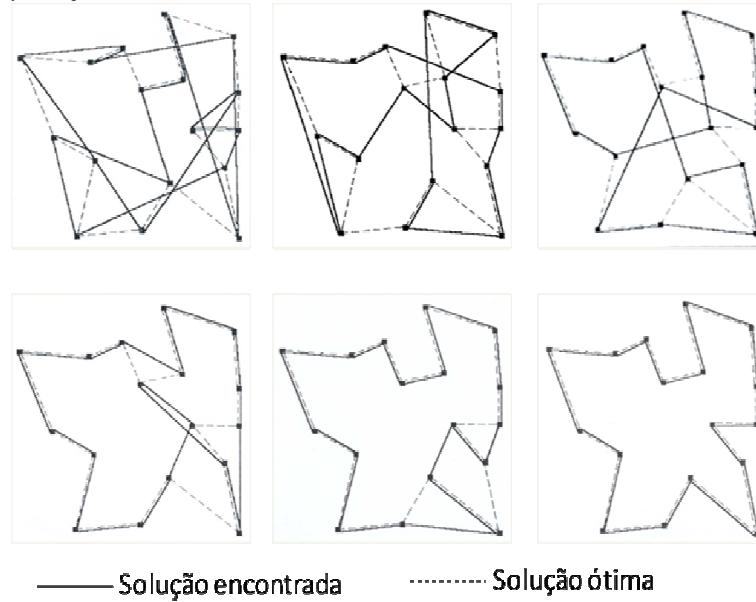
Distribuição de feromônio no início da busca



Distribuição de feromônio após 100 iterações



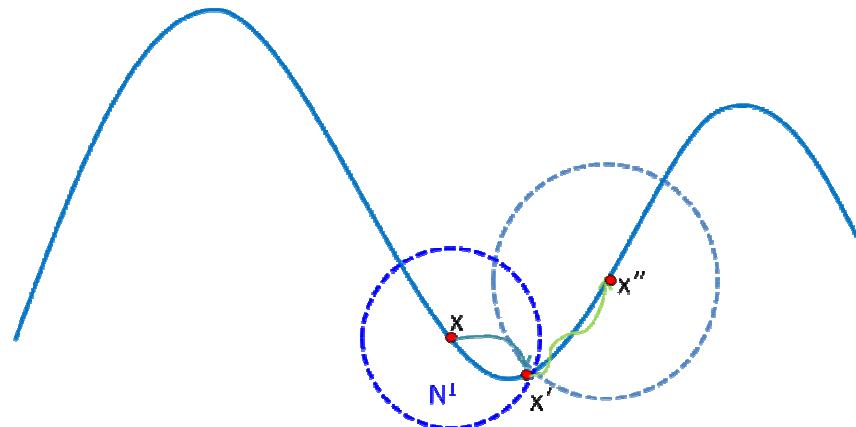
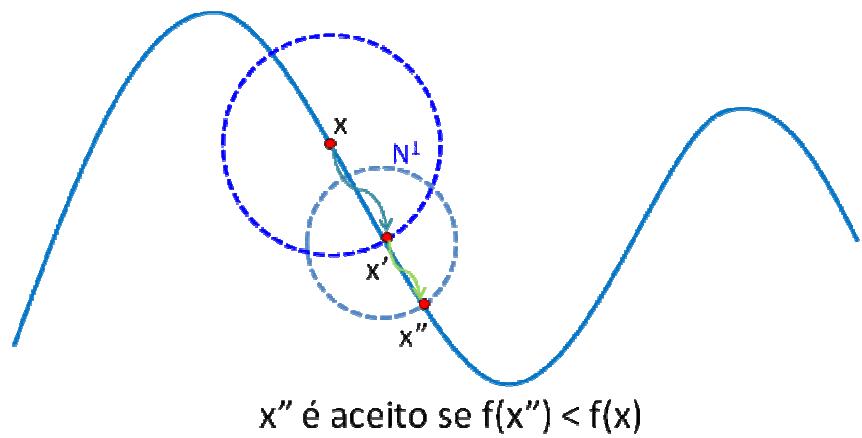
Resultados parciais da aplicação do ACO:



3.7 VARIABLE NEIGHBORHOOD SEARCH

O método *VNS – Variable Neighborhood Search* – foi proposto por Nenad Mladenovic & Pierre Hansen em 1997. Trata-se de uma metaheurística de busca local que explora o espaço de soluções através de trocas sistemáticas de soluções vizinhas.

O VNS explora vizinhanças gradativamente mais “distantes” e focaliza a busca em torno de uma nova solução somente se um movimento de melhora é realizado.



Algoritmo

```

 $x_0 = \text{Solução_Inicial}$ 
 $x = \text{busca_local}(x_0)$  aplica uma melhoria na solução inicial
Repita
     $x' = \text{vizinho}(x)$  encontra uma nova solução, vizinha de  $x$  através de 1 troca de
    arcos
     $x'' = \text{busca_local}(x')$  aplica uma melhoria na solução  $x'$ 
    Se  $f(x'') < f(x)$ , então
         $x = x''$  (aceita a melhor solução)
    Caso contrário, se  $f(x') < f(x)$ , então
         $x = x'$  (aceita a melhor solução)
    Fim
Enquanto o critério de parada não for satisfeito

```

3.8 ALGORITMOS GENÉTICOS

A Computação Evolucionária é a área da Inteligência Artificial que engloba um conjunto de métodos computacionais. São métodos inspirados na Teoria da Evolução das Espécies de Charles Darwin para a solução de problemas, usando as seguintes hipóteses:

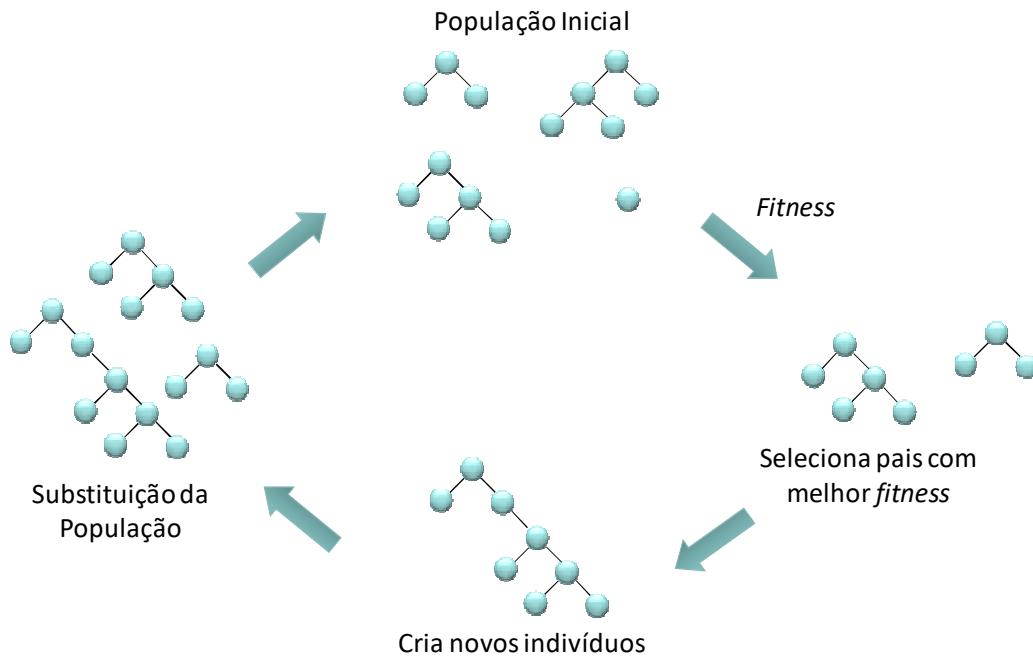
- Na natureza sobrevivem os indivíduos que possuem maior capacidade de se adaptar ao meio ambiente;
- Suas características genéticas são repassadas para as gerações seguintes e melhoradas;
- A nova geração será composta por indivíduos com material genético melhor do que os da população anterior.

Na Programação Evolucionária o aprendizado é decorrente de um processo evolutivo, a função de avaliação mede diretamente a aptidão (adaptação do indivíduo ao ambiente), os indivíduos são alterados através da mutação e não há recombinação.

Algoritmos Genéticos (AG, desenvolvida por Holland em 1975) e Programação Genética (PG, desenvolvida por Koza em 1992) são as duas principais frentes de pesquisa em Computação Evolucionária. A diferença essencial entre AG e PG é que em PG as estruturas manipuladas são bem mais complexas, assim como várias das operações realizadas pelo algoritmo. Ambas as técnicas compartilham a mesma base teórica, inspirada na competição entre indivíduos pela sobrevivência.

Algumas aplicações:

- Otimização:
 - Indústria, solução de problemas: máquinas x processos, alocação de recursos, roteamento de veículos.
- Busca:
 - Mineração de Dados, descoberta de conhecimento em bases de dados, indução de classificadores (características x doenças, estrutura de proteínas).
- Aprendizado e adaptação.



Elementos principais de AG e PG:

- População inicial de indivíduos;
- Função de aptidão (*fitness*);
- Um ciclo de nascimento e morte baseados na função de *fitness*;
- A noção de herança genética.

Os Algoritmos Genéticos simulam processos naturais de sobrevivência e reprodução das populações, essenciais em sua evolução. Consistem em métodos de busca e otimização inspiradas no princípio Darwiniano de seleção natural e reprodução genética dos seres vivos.

Os Algoritmos Genéticos são considerados procedimentos iterativos que mantém uma população de estruturas (indivíduos) - possíveis soluções para um problema. A cada iteração (“geração”), os indivíduos da população passam por uma avaliação que verifica sua capacidade em oferecer uma solução satisfatória para o problema. Esta avaliação é feita conforme uma função que recebe o nome de função de aptidão, ou função de *fitness*.

Embora os AGs nem sempre possam encontrar a solução ótima global para um determinado problema, na maioria das vezes são capazes de encontrar uma solução quase ótima. Os AGs encontram soluções aceitáveis para problemas complexos de Otimização combinatória (métodos convencionais são inviáveis em razão do esforço computacional necessário para resolvê-los).

Requisitos para implementação de Algoritmos genéticos:

- Codificação genética das possíveis soluções do problema;
- População inicial com diversidade suficiente para permitir ao algoritmo combinar características e produzir novas soluções;
- Definição de um método para medir a qualidade de uma solução potencial;
- Definição de procedimentos de combinação de soluções para gerar novos indivíduos na população;

- Temos que definir:
 - Critério de escolha das soluções que permanecerão na população ou que serão retiradas da mesma;
 - Procedimento para introduzir, periodicamente, alterações em algumas soluções da população. Desse modo mantém-se a diversidade da população e a possibilidade de se produzir soluções inovadoras para serem avaliadas pelo critério de seleção dos mais aptos.

Fluxograma básico de Algoritmos Genéticos:



Critérios de parada:

- alcançar um número máximo de gerações;
- solução ótima encontrada (quando esta é conhecida);
- perda de diversidade;
- Convergência: nas últimas k gerações não houve melhoria na aptidão.

AG Geracional:

- toda a população é substituída pelos novos indivíduos criados depois do processo de seleção e aplicação dos operadores genéticos;
- considerando que neste processo toda a população é substituída pela nova, corre-se o risco de perder bons indivíduos;
- para evitar isto, pode-se utilizar um procedimento conhecido como **elitismo**, que consiste em passar para a geração seguinte uma cópia de alguns dos melhores indivíduos.

AG Steady-State:

- cria apenas um indivíduo de cada vez, sendo que o indivíduo gerado pode ou não ser passado para a geração seguinte;
- é transmitido para a próxima geração, se o seu valor de *fitness* for melhor do que o pior valor de *fitness* da população antiga;

A primeira etapa na resolução de um problema utilizando AG é a codificação dos indivíduos;

- Fenótipos - elementos do espaço de busca;
- Genótipo - código que os representa

Matematicamente, a escolha da codificação para um dado problema é a função ou regra que associa os elementos do espaço de **genótipos** aos do espaço de busca, os **fenótipos**.

A codificação corresponde a utilizar cadeias (**strings**) de comprimento l , formadas por caracteres de um determinado alfabeto. O caso mais comum é o **binário**: 0/1.

A cadeia 10010101 - possível solução de um dado problema ($l = 8$). O conjunto dos **genótipos** é formado por todos os números binários de 00000000 a 11111111.

$$2^l = 2^8 = 256.$$

Conjunto de indivíduos candidatos à solução:

- Populações pequenas - grandes chances de perder a diversidade necessária (exploração de todo o espaço de soluções);
- Populações grandes – perda de eficiência pelo alto custo computacional (demora em avaliar a função de *fitness*).

A geração da População inicial:

- Realizada **aleatoriamente** utilizando-se um gerador de números aleatórios com distribuição uniforme dos indivíduos numa faixa previamente definida pelo usuário;
- Esta faixa é definida levando-se em consideração algum conhecimento prévio do problema a ser otimizado.
- No caso de codificação binária, por exemplo, é fácil obter uma boa distribuição de “0’s” e “1’s” na população inicial.
- Pode-se também introduzir na população inicial um ou mais **indivíduos “interessantes”** como, por exemplo, soluções aproximadas conhecidas ou contendo algum tipo de informação prévia.
- O número de elementos que comporá a população depende muito da experiência do usuário e do seu conhecimento prévio sobre a função a ser otimizada.
- O número de elementos na população, a probabilidade de ocorrer cruzamento e a probabilidade de acontecer mutação são denominados de **parâmetros de controle do AG**.

A função de *fitness* avalia a capacidade de que o indivíduo ofereça uma solução satisfatória para o problema (adaptação ao ambiente), diferenciando soluções boas e ruins. Além disso, de acordo com esta avaliação, alguns indivíduos são selecionados (com a regra probabilística) para passar por um processo de recombinação (operadores genéticos).

A fase de seleção serve para escolher os indivíduos sobre os quais serão aplicados os operadores genéticos. Existem diversas formas de seleção, as mais utilizadas são: Roleta, Torneio e Ranking.

Seleção por Roleta:

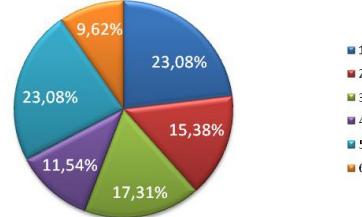
Especifica a probabilidade de que cada indivíduo seja selecionado para a próxima geração.

$$p_i = \frac{f_i}{\sum_{j=1}^n f_j}$$

onde f_i é o valor de aptidão do indivíduo e $\sum_{j=1}^n f_j$ é o valor acumulado de aptidão de todos os indivíduos da população n.

Cada indivíduo da população recebe uma porção da roleta proporcional ao seu valor $p_i = \frac{f_i}{\sum_{j=1}^n f_j}$.

Fitness	% Fitness
12	23,08
8	15,38
9	17,31
6	11,54
12	23,08
5	9,62



O sorteio dos elementos é feito através de um “jogo de roleta”, onde a probabilidade de cada indivíduo ser selecionado é proporcional ao seu *fitness*.

Pode ocorrer que os indivíduos que possuem melhor *fitness* não sejam selecionados, pois sua chance de escolha não é de 100%.

- Elitismo [Michalewicz and Schoemauer, 1996], uma porcentagem da população com os melhores *fitness* é preservada para a próxima geração automaticamente.

Seleção por Torneio:

Um número p de indivíduos da população é escolhido aleatoriamente para formar uma sub-população temporária. Deste grupo, é selecionado o melhor indivíduo.

Seleção por Ranking:

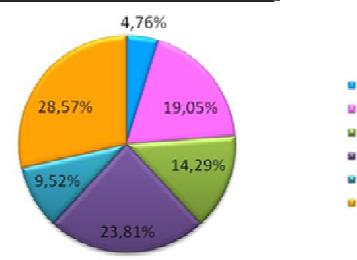
Os indivíduos são ordenados de acordo com seu *fitness*:

- Indivíduo menos adaptado → valor 1.
- Indivíduo mais adaptado → valor igual número de indivíduos da população.

Em seguida, a cada indivíduo i é associada uma probabilidade p_i de ser escolhido.

Problema de minimização
menor fitness = maior
probabilidade

Fitness	Ranking	Ranking (%)
12	1	5%
8	4	19%
9	3	14%
6	5	24%
12	2	10%
5	6	28%



Operadores Genéticos:

São necessários para transformar a população através de sucessivas gerações até chegar a um resultado satisfatório. Desta forma, a população se diversifica e mantém características de adaptação adquiridas nas gerações anteriores.

Os operadores de cruzamento e de mutação têm um papel fundamental em um algoritmo genético. Os principais operadores genéticos são:

- Reprodução;
- Cruzamento;
- Mutação.

Reprodução

Seleciona-se um indivíduo da população atual e o mesmo é copiado para a próxima geração sem alteração em sua estrutura. Este indivíduo é novamente inserido na população, ou seja, haverá duas versões do mesmo indivíduo na população.

Cruzamento (crossover)

Dois indivíduos pais são selecionados e seu material genético é combinado, permutando uma parte de um dos pais por uma parte do outro, gerando um novo indivíduo, esperando-se que seja melhor do que os anteriores, pois foram criados a partir da combinação das melhores partes de cada indivíduo.

Visa guiar a solução de maneira a combinar as melhores soluções na busca da solução ótima, e funciona da seguinte maneira:

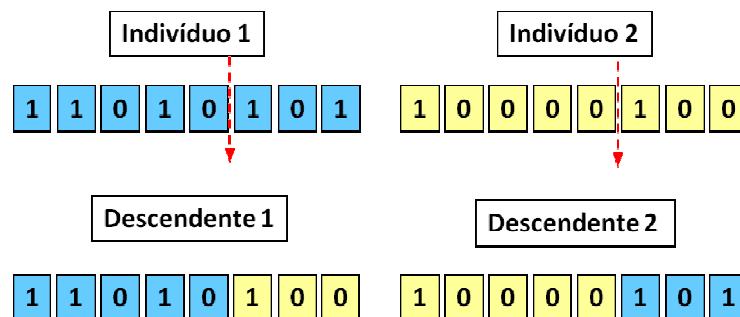
- Escolhe-se dois indivíduos através do valor de sua função de aptidão;
- Seleciona-se, aleatoriamente, em cada indivíduo, um ou dois pontos de cruzamento;
- Permuta-se as sub-árvores dos dois indivíduos gerando os filhos, que farão parte da nova população.

Exemplo com 1 ponto de cruzamento:

São gerados dois números aleatórios n_1 e n_2 .

$n_1 \in [0, 1]$ indica a probabilidade de ocorrer o cruzamento entre dois indivíduos.

$n_2 \in [0, g - 1]$ indica o local de realização do cruzamento, onde g é o número de bits de cada indivíduo.

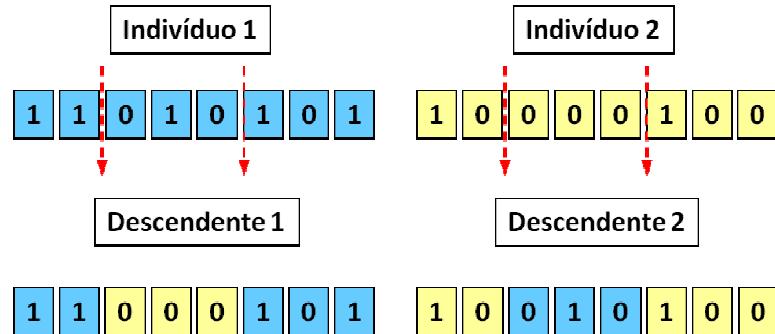


Exemplo com 2 pontos de cruzamento:

São gerados três números aleatórios n_1 , n_2 e n_3 .

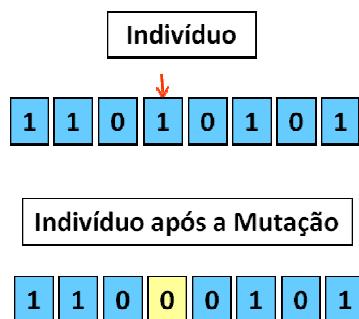
$n_1 \in [0, 1]$ indica a probabilidade de ocorrer o cruzamento entre dois indivíduos.

$n_2, n_3 \in [0, g - 1]$ indicam os locais de realização do cruzamento, onde g é o número de bits de cada indivíduo.



Mutação

É o operador genético mais simples de ser implementado. No sistema binário, uma posição do cromossomo é sorteada e o *bit* correspondente é invertido, isto é, se for “1” ele passa a ser “0” e vice-versa. A probabilidade de se efetuar uma mutação deve ser relativamente baixa, caso contrário o algoritmo fará uma busca aleatória, dificultando a convergência.



Codificação binária

Decimal → Binário

O número 25 escrito em código binário, fica representado pelos

0's e 1's, lendo-se de baixo para cima: **(11001)**

Binário → Decimal

$$(11001) = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 8 + 0 + 0 + 1 = 25$$

25	2			
		1	12	2
			0	2
			6	0
			3	2
		1	1	2
			1	0

- Passo 1:** Gerar População Inicial;
Passo 2: Avaliar o *fitness*;
Passo 3: Critério de Parada;
Passo 4: Seleção;
Passo 5: Operador Genético;
Passo 6: Substituir a população

Exercícios:

1. Utilize AG para encontrar o valor inteiro **máximo** da função $f(x) = x^2$ no intervalo $[0, 31]$ com os quatro indivíduos dados abaixo, usando 5 gerações.

indivíduos:	i_1	(1 1 0 0 1)
	i_2	(0 1 1 1 1)
	i_3	(0 1 1 1 0)
	i_4	(0 1 0 1 0)

2. Utilize AG para encontrar o valor inteiro **máximo** da função $f(x) = \cos(x) + \frac{x}{5}$ no intervalo $[0, 11]$ com os quatro indivíduos dados abaixo, usando 5 gerações.

indivíduos:	i_1	(1 0 0 1)
	i_2	(1 0 0 0)
	i_3	(0 0 1 0)
	i_4	(0 0 0 1)

3. Utilize AG para encontrar o valor inteiro **mínimo** da função $f(x) = (x - 4)^2 - (x - 8)^3 + 5$ no intervalo $[0, 15]$ com os quatro indivíduos dados abaixo, usando 5 gerações.

indivíduos:	i_1	(1 0 0 1)
	i_2	(1 0 0 0)
	i_3	(0 0 1 0)
	i_4	(0 0 0 1)

4. Utilize AG com os quatro indivíduos indicados para resolver o problema da Mochila dado abaixo, onde $n = 6$ e $P = 20$.

$$\text{Maximizar } \sum_{i=1}^n v_i z_i$$

$$\text{Sujeito a } \sum_{i=1}^n p_i z_i \leq P$$

$$z_i \in \{0,1\}$$

Objeto i	1	2	3	4	5	6
Peso p_i	4	5	7	9	6	3
Valor v_i	2	2	3	4	4	2

indivíduos:	i_1	(1 1 0 1 0 0)
	i_2	(1 0 0 0 0 0)
	i_3	(0 0 1 1 0 0)
	i_4	(0 0 0 1 1 1)

$$\text{Função de avaliação de cada solução: } f = \sum_{i=1}^n v_i z_i - \gamma \max(0, \sum_{i=1}^n p_i z_i - P), \text{ onde } \gamma = \sum_{i=1}^n v_i.$$

5. Utilize AG para resolver o problema da Mochila dado abaixo, onde $n = 15$ e $P = 275$. Crie uma população com 3 indivíduos.

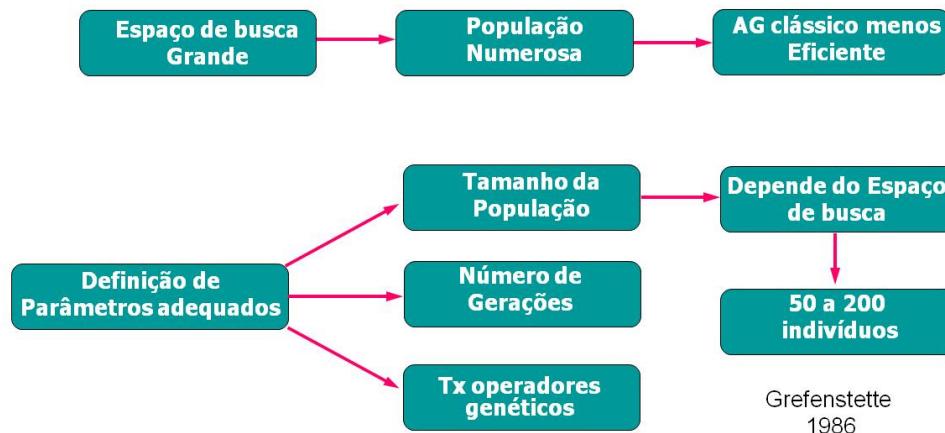
Objeto i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Peso p_i	63	21	2	32	13	80	19	37	56	41	14	8	32	42	7
Valor v_i	13	2	20	10	7	14	7	2	2	4	16	17	17	3	21

Na maior parte das aplicações, a probabilidade de ocorrer mutação é bem menor que a de ocorrer cruzamento. Os indivíduos selecionados para serem manipulados pelos operadores genéticos são inicialmente divididos em casais e a estes é aplicado o operador genético de cruzamento. A seguir, a cada novo indivíduo gerado aplica-se o operador genético da mutação. Ao final, tem-se uma nova geração que se espera tenha melhor adaptação que a anterior e assim sucessivamente.

As populações seguintes, geralmente apresentam o mesmo número de indivíduos, porém a composição pode ser totalmente diferente. A população inicial tem seus indivíduos gerados aleatoriamente e a população seguinte é obtida através da manipulação genética da população inicial.

Para uma boa convergência dos Algoritmos Genéticos podemos considerar a redução de incesto, técnica que reduz a perda de diversidade, evitando cruzamento entre indivíduos muito semelhantes. Podemos permitir o cruzamento somente entre indivíduos cuja **distância de Hamming** seja grande, onde a distância de Hamming mede a diversidade entre dois cromossomos: é o número de alelos (valores que os genes podem assumir) diferentes para as mesmas posições relativas [Lopes, 1999].

Dificuldades na utilização de AGs

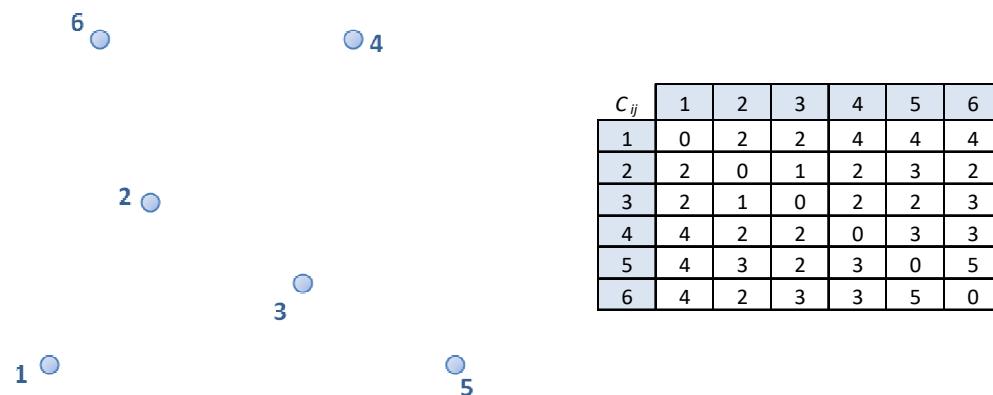


Utilização de AG para resolver o problema do Caixeiro Viajante

Os indivíduos representam uma sequência factível de rota, e quando são gerados filhos infactíveis através de cruzamentos, os mesmos podem tornar-se factíveis utilizando-se mutação.

Exercícios:

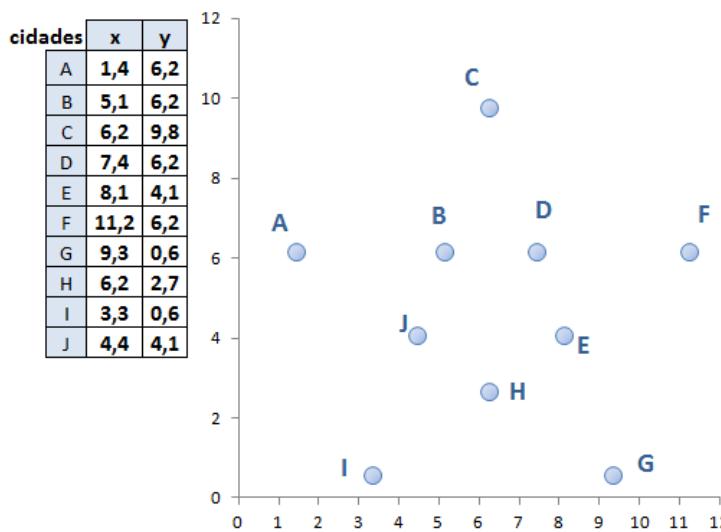
1. Utilize o AG para encontrar rotas factíveis para o problema do Caixeiro Viajante com matriz de custos dada abaixo.



Use os quatro indivíduos dados abaixo para construir uma população inicial.

i ₁	1	2	3	4	5	6	1
i ₂	2	4	6	1	3	5	2
i ₃	1	5	6	3	4	2	1
i ₄	6	5	1	4	2	3	6

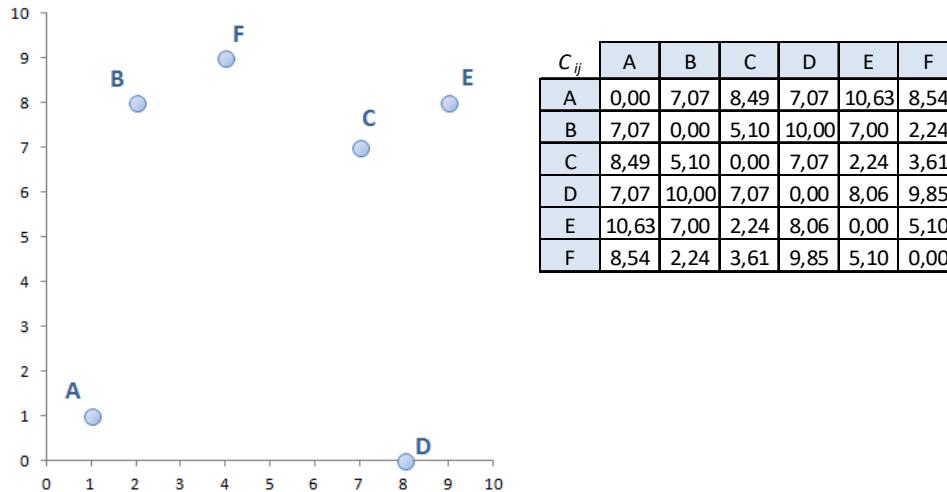
2. Resolva o problema do Caixeiro Viajante com 10 cidades dado abaixo, usando os 4 indivíduos dados para uma população inicial.



C_{ij}	A	B	C	D	E	F	G	H	I	J
A	0,00	3,70	6,00	6,00	7,02	9,80	9,68	5,94	5,91	3,66
B	3,70	0,00	3,76	2,30	3,66	6,10	7,00	3,67	5,88	2,21
C	6,00	3,76	0,00	3,79	6,01	6,16	9,71	7,10	9,65	5,98
D	6,00	2,30	3,79	0,00	2,21	3,80	5,91	3,70	6,94	3,66
E	7,02	3,66	6,01	2,21	0,00	3,74	3,70	2,36	5,94	3,70
F	9,80	6,10	6,16	3,80	3,74	0,00	5,91	6,10	9,68	7,12
G	9,68	7,00	9,71	5,91	3,70	5,91	0,00	3,74	6,00	6,02
H	5,94	3,67	7,10	3,70	2,36	6,10	3,74	0,00	3,58	2,28
I	5,91	5,88	9,65	6,94	5,94	9,68	6,00	3,58	0,00	3,67
J	3,66	2,21	5,98	3,66	3,70	7,12	6,02	2,28	3,67	0,00

i ₁	A	B	J	D	H	E	C	J	G	I
i ₂	A	I	H	J	G	F	C	D	E	B
i ₃	H	A	E	B	J	D	C	F	I	G
i ₃	A	I	C	G	F	B	H	D	J	E

3. Utilize o AG para encontrar rotas factíveis para o problema do Caixeiro Viajante com matriz de custos dada abaixo. Crie 4 indivíduos factíveis para iniciar a população.



C_{ij}	A	B	C	D	E	F
A	0,00	7,07	8,49	7,07	10,63	8,54
B	7,07	0,00	5,10	10,00	7,00	2,24
C	8,49	5,10	0,00	7,07	2,24	3,61
D	7,07	10,00	7,07	0,00	8,06	9,85
E	10,63	7,00	2,24	8,06	0,00	5,10
F	8,54	2,24	3,61	9,85	5,10	0,00

Utilização de AG para resolver o problema de p -medianas

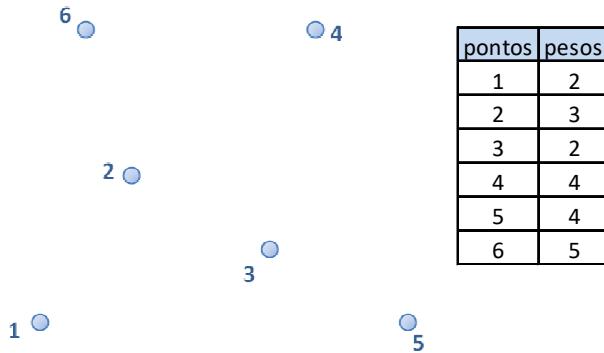
Considere um conjunto de n pontos, cada um com um peso w_i , onde $i = 1, 2, \dots, n$. O problema de p medianas tem o objetivo de definir os p pontos que representam as melhores escolhas para designar facilidades (por exemplo, as melhores localizações para construções de novas escolas, novos postos de saúde, entre outros).

Para aplicar o AG, escolhe-se aleatoriamente p vértices dos n possíveis, criando os indivíduos como vetores $r_q = (v_1, v_2, \dots, v_p)$. O custo da escolha de um conjunto r_q é dado por:

$$\text{custo}_q = \sum_{j=1}^n w_j \cdot \min_{v_k \in r_q, k \neq j} d(v_k, v_j).$$

Exercícios:

1. Utilize o AG para encontrar 4 medianas para o conjunto de 6 vértices dado abaixo com os 5 indivíduos da população inicial indicados.



<i>d</i>	1	2	3	4	5	6
1	0	2	2	4	4	4
2	2	0	1	2	3	2
3	2	1	0	2	2	3
4	4	2	2	0	3	3
5	4	3	2	3	0	5
6	4	2	3	3	5	0

População inicial:

<i>r</i> ₁	1	3	5	6
<i>r</i> ₂	2	4	6	1
<i>r</i> ₃	3	5	6	2
<i>r</i> ₄	1	4	5	6
<i>r</i> ₅	6	3	2	4

2. Utilize o AG para encontrar 4 medianas para o conjunto de 11 vértices dado abaixo com os 5 indivíduos da população inicial indicados.

<i>d</i>	1	2	3	4	5	6	7	8	9	10	11
1	0	3	2	4	8	8	11	16	14	9	13
2	3	0	2	5	9	8	11	16	13	7	10
3	2	2	0	2	7	6	9	14	12	7	10
4	4	5	2	0	4	4	7	12	12	9	11
5	8	9	7	4	0	4	3	8	12	11	13
6	8	8	6	4	4	0	4	8	8	8	9
7	11	11	9	7	3	4	0	5	11	12	12
8	16	16	14	12	8	8	5	0	10	14	13
9	14	13	12	12	12	8	11	10	0	7	4
10	9	7	7	9	11	8	12	14	7	0	3
11	13	10	10	11	13	9	12	13	4	3	0

	pontos	pesos
1	3	
2	1	
3	8	
4	5	
5	2	
6	6	
7	10	
8	4	
9	8	
10	10	
11	20	

População inicial:

<i>r</i> ₁	1	2	7	4
<i>r</i> ₂	4	5	7	11
<i>r</i> ₃	2	5	9	8
<i>r</i> ₄	6	9	11	3
<i>r</i> ₅	5	4	8	10

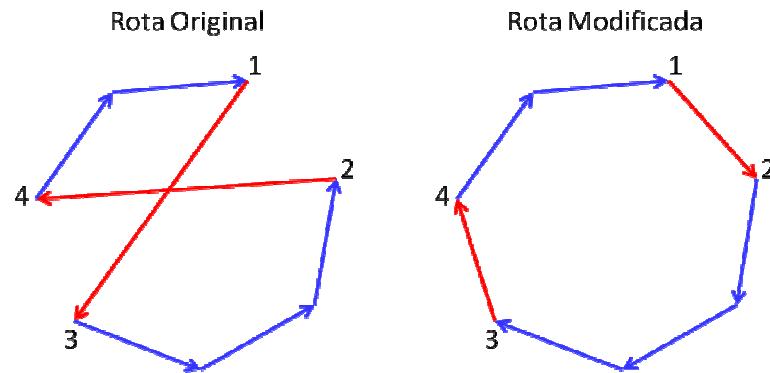
3.9 BUSCA LOCAL k-opt

Os métodos de melhoria de soluções mais utilizados são denominados *busca local*, e os mais comuns são do tipo **k-opt** [Lin-Kernighan, 1973].

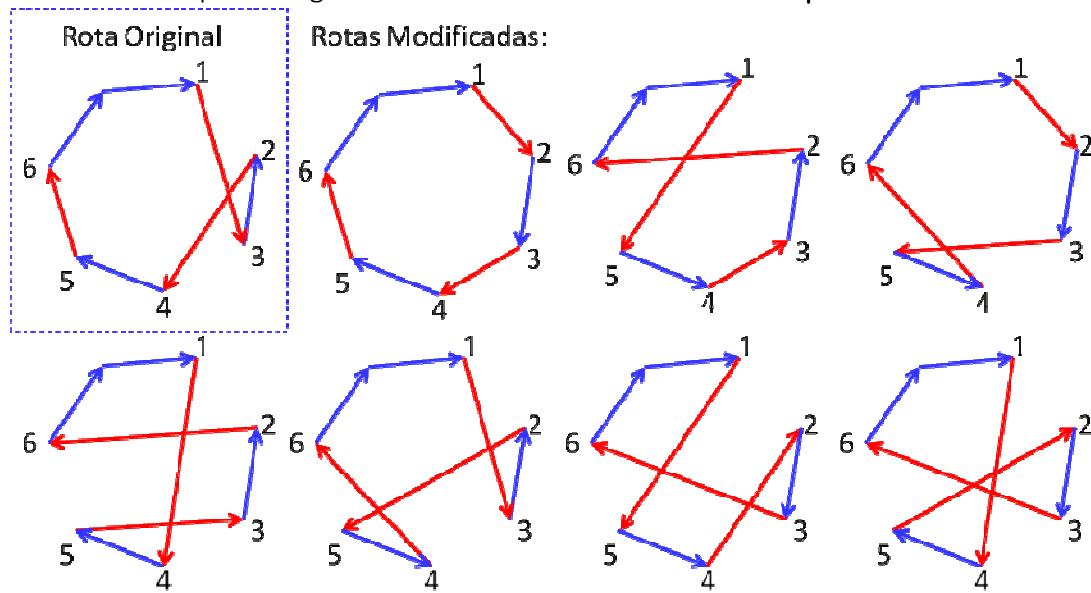
Nestes métodos, *k* arcos são removidos de uma rota e substituídos por outros *k* arcos, e o objetivo é de melhorar a solução encontrada por outro método. Na prática são mais utilizados os métodos **2-opt** e **3-opt**, pois o tempo computacional dos demais é bem elevado.

Se a troca de arcos for vantajosa, o método considera a nova solução; caso contrário, testa-se uma nova troca.

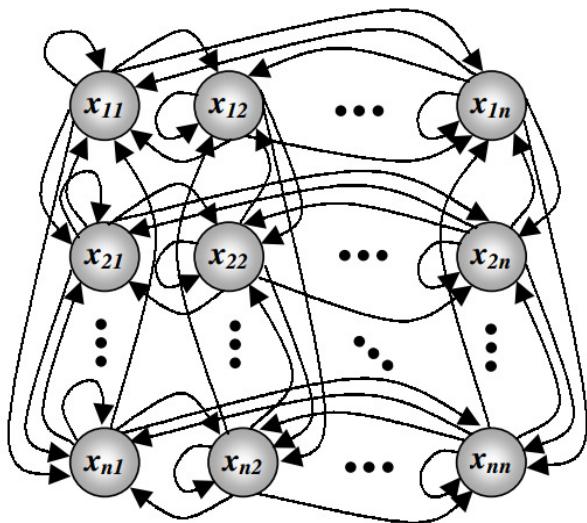
Considere os exemplos abaixo de rotas do PCV com melhoria 2-opt:



Considere os exemplos a seguir de rotas do PCV com melhoria 3-opt:



Com a aplicação de uma RNA recorrente de Wang [Siqueira, Steiner, Scheer, 2010], temos os resultados de rotas do PCV mostrados a seguir. A estrutura da rede recorrente de Wang é a seguinte:



$$\frac{du_{ij}(t)}{dt} = -\eta \sum_{k=1}^n x_{ik}(t) - \eta \sum_{l=1}^n x_{lj}(t) + \eta \theta_{ij} - \lambda c_{ij} e^{-\frac{t}{\tau}},$$

$$x_{ij} = g(u_{ij}(t)),$$

Aplicando a técnica Winner Takes All (WTA) o número de iterações diminui drasticamente (temos exemplos que utilizavam mais de 15.000 iterações, e com o princípio WTA utilizam somente 15 iterações).

Exemplo:

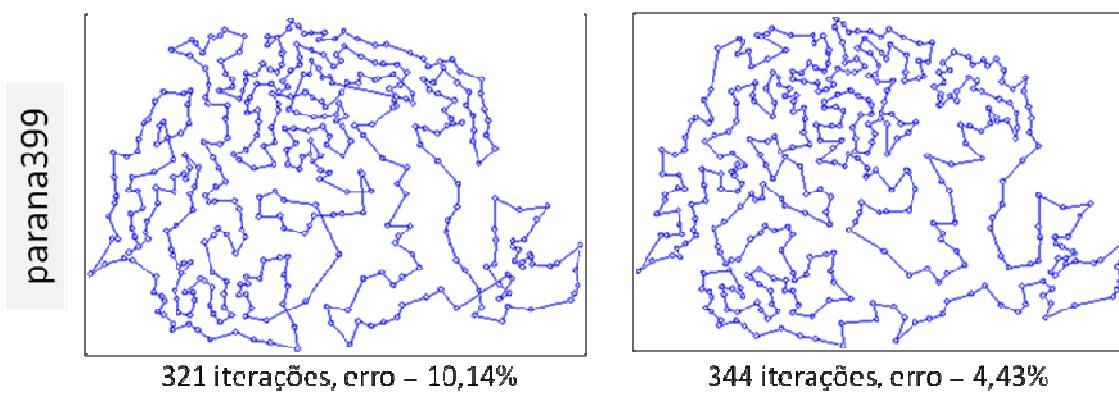
$$\bar{x} = \begin{pmatrix} 0,0808 & 0,0011 & 0,0168 & 0,1083 & 0,0514 & 0,0033 & *0,422 & 0,3551 \\ 0,0056 & 0,2827 & 0,0168 & 0,1525 & 0,1484 & 0,1648 & 0,0866 & 0 \\ 0,1754 & 0,0709 & 0,0688 & 0,3449 & 0 & 0,3438 & 0,0425 & 0,0024 \\ 0,1456 & 0,2412 & 0,2184 & 0,0521 & 0,1131 & 0,0747 & 0,0598 & 0,1571 \\ 0,0711 & 0 & 0,2674 & 0,272 & 0,3931 & 0,0024 & 0,0306 & 0,0061 \\ 0,2037 & 0,2823 & 0,2956 & 0,0366 & 0 & 0,0025 & 0,0136 & 0,2186 \\ 0,1681 & 0,174 & 0,1562 & 0 & 0,3053 & 0,2016 & 0,0369 & 0,0144 \\ 0,1142 & 0,0031 & 0,0138 & 0,0829 & 0,0353 & 0,2592 & 0,251 & 0,2907 \end{pmatrix}$$

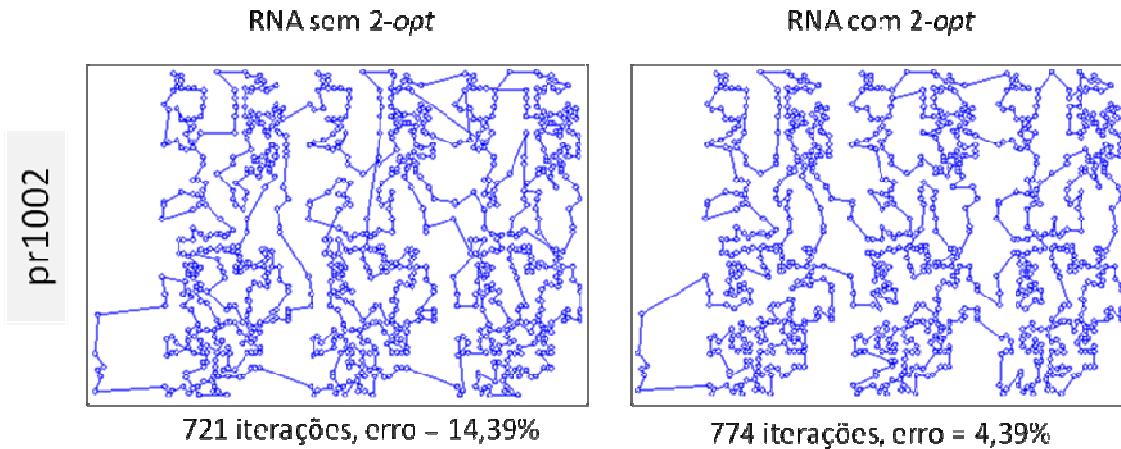
$$\bar{x} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1,0412 & 0 \\ 0,0056 & 0,2827 & 0,0168 & 0,1525 & 0,1484 & 0,1648 & 0 & 0 \\ 0,1754 & 0,0709 & 0,0688 & 0,3449 & 0 & 0,3438 & 0 & 0,0024 \\ 0,1456 & 0,2412 & 0,2184 & 0,0521 & 0,1131 & 0,0747 & 0 & 0,1571 \\ 0,0711 & 0 & 0,2674 & 0,272 & *0,393 & 0,0024 & 0 & 0,0061 \\ 0,2037 & 0,2823 & 0,2956 & 0,0366 & 0 & 0,0025 & 0 & 0,2186 \\ 0,1681 & 0,174 & 0,1562 & 0 & 0,3053 & 0,2016 & 0 & 0,0144 \\ 0,1142 & 0,0031 & 0,0138 & 0,0829 & 0,0353 & 0,2592 & 0 & 0,2907 \end{pmatrix}$$

Solução aproximada:

$$\bar{x} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1,0412 & 0 \\ 0 & 1,0564 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1,0491 & 0 & 0 & 0 & 0 \\ 1,0632 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1,0446 & 0 & 0 & 0 \\ 0 & 0 & 1,0533 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1,0544 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1,0473 \end{pmatrix}$$

Nos grafos a seguir estão os resultados sem melhoria (erro de 10,14%) e com melhoria 2-opt (erro de 4,43%).





REFERÊNCIAS

- DORIGO, M., GARAMBARDELLA, L.M. Ant Colonies for the Traveling Salesman Problem. *Biosystems*, v. 43, n. 2, p. 73-81, 1997.
- ENGELBRECHT, A. P., Computational Intelligence, John Wiley & Sons, 2007.
- EBERHART, R. C., SHI, Y., Comparison between genetic algorithms and particle swarm optimization, *Evolutionary Programming VII: Lecture Notes in Computer Science*, v. 1447, p. 611-616, 1998.
- LIN, S., KERNIGHAN, B. W. An effective heuristic algorithm for the traveling-salesman problem, *Operations research*, v. 21, n. 2, p. 498-516, 1973.
- HU, X. PSO Tutorial. Disponível em: <<http://www.swarmintelligence.org/tutorials.php>>
- GLOVER, F. Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Graduate School of Business, University of Colorado, 1991.
- LOPES, H. S. Algoritmos genéticos em projetos de engenharia: aplicações e perspectivas futuras. *Anais do IV Simpósio Brasileiro de Automação Inteligente*, p. 64-74, 1999.
- MICHALEWICZ, Z., SCHOENAUER, M. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, v. 4, n. 1, p. 1-32, 1999.
- VAN LAARHOVEN, P. J., AARTS, E. H. Simulated annealing. In *Simulated annealing: Theory and applications* p. 7-15. Springer, Dordrecht, 1987.
- MITRA, D., ROMEO, F., SANGIOVANNI-VINCENTELLI, A. Convergence and finite-time behavior of simulated annealing. *Advances in applied probability*, v. 18, n. 3, p. 747-771, 1986.
- KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P. Optimization by simulated annealing. *science*, v. 220, n. 4598, p. 671-680, 1983.
- ARAGON, C. R., JOHNSON, D. S., McGEOCH, L. A., SCHEVON, C. Optimization by simulated annealing: an experimental evaluation. In: *Workshop on Statistical Physics in Engineering and Biology*. Yorktown Heights, 1984.