Universidade Federal de Lavras

Departamento de Automática

GUIA DE AULAS PRÁTICAS DE REDES NEURAIS ARTIFICIAIS

v. 1.3

Prof. Wilian Soares Lacerda

Lavras, fevereiro de 2019

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos da Biblioteca Universitária da UFLA

Lacerda, Wilian Soares.

Guia de aulas práticas de redes neurais artificiais : versão 1.3 / Wilian Soares Lacerda. – Lavras : UFLA, 2019.

70 p. : il.

Guia elaborado para orientação da Disciplina Redes Neurais Artificiais do Departamento de Automática da Universidade Federal de Lavras.

1. Redes neurais. 2. Aprendizado de máquina. 3. Sistemas inteligentes. I. Universidade Federal de Lavras, Departamento de Automática. II. Título.

CDD – 006.3

Prefácio

Este guia de aulas práticas foi elaborado para auxiliar os alunos da disciplina de Redes Neurais Artificiais ofertada pela Universidade Federal de Lavras no desenvolvimento de atividades práticas dos assuntos pertinentes vistos em aulas teóricas. Este guia é disponibilizado em pdf e pode ser impresso no formato frente/verso, em papel A4 para melhor aproveitamento.

São apresentadas 10 experiências didáticas a serem desenvolvidas utilizando o software Matlab (*http://www.mathworks.com*). É considerado que o aluno já domine o software para a implementação das experiências. As experiências iniciam com implementações simples do neurônio artificial do tipo Perceptron utilizando comandos básicos da linguagem de programação do Matlab. Estas têm o objetivo principal de familiarizar os alunos com o laboratório e com o software utilizado, além de ensinar os princípios básicos do assunto tratado.

Na sequência, são apresentadas experiências com algoritmos de treinamento do Perceptron e outros tipos de neurônios artificiais como o Adaline. Ao final são indicadas algumas experiências que utilizam Redes de Neurônios Artificiais na solução de problemas de classificação e aproximação de funções não lineares. Algumas técnicas de pré-processamento de dados e análise dos resultados também são utilizadas.

Desejamos a todos um bom aproveitamento do curso.
Prof. Wilian Soares Lacerda

Normas do Laboratório

Algumas normas deverão ser seguidas pelos usuários do Laboratório da UFLA no desenvolvimento das aulas práticas.

A experiência prática é realizada pelos alunos de forma individual, entretanto o desenvolvimento das práticas pode ser realizado em grupo para troca de ideias e conhecimento. Cada aluno deverá ter em mãos este guia de aulas práticas e usá-lo como roteiro.

Deve ser apresentado um relatório por aluno para cada experiência. O relatório deverá ser entregue (postado) ao professor conforme combinado no início das aulas (veja modelo no Apêndice), incluindo:

- o *script* desenvolvido para implementação da prática, com os comentários pertinentes;
- os resultados obtidos da implementação, em forma numérica e/ou gráfica, com discussão;
- as conclusões com a síntese da experiência, principais resultados, e dificuldades encontradas.

Comportamento na aula

Não é permitido fumar, comer ou beber dentro do Laboratório. Deve-se manter a bancada limpa e organizada. Durante a aula deve-se evitar falar alto. Não é permitido instalar ou desinstalar algum software, assim como mudar alguma configuração do computador. Desligar o computador adequadamente ao final da aula.

Sumário

1ª EXPERIÊNCIA: Perceptron

Título: Perceptron

Objetivos:

- Conhecer o neurônio artificial tipo Perceptron;
- Implementar o Perceptron no Matlab;
- Utilizar o Perceptron na classificação de dados (função lógica AND).

Teoria:

Um neurônio artificial *k* é composto, conforme Figura 1, por:

- As sinapses (entradas) onde cada entrada x_j é um valor numérico do tipo binário, inteiro ou real. A entrada x_0 é uma entrada especial com valor constante igual a +1.
- Os pesos w_{kj} que representam a importância de cada entrada x_j associada ao respectivo peso. Cada peso é um valor numérico real e o treinamento do neurônio é justamente a busca do melhor valor de cada peso.
- A junção somadora que soma todos produtos obtidos entre a entrada x_j e o peso w_{kj} correspondente.
- A função de ativação $f(u_k)$ que transforma o resultado numérico do somatório, podendo saturá-lo entre dois valores limites. A função pode ser: degrau, sigmoidal, linear, semi-linear, tangente hiperbólica, etc.

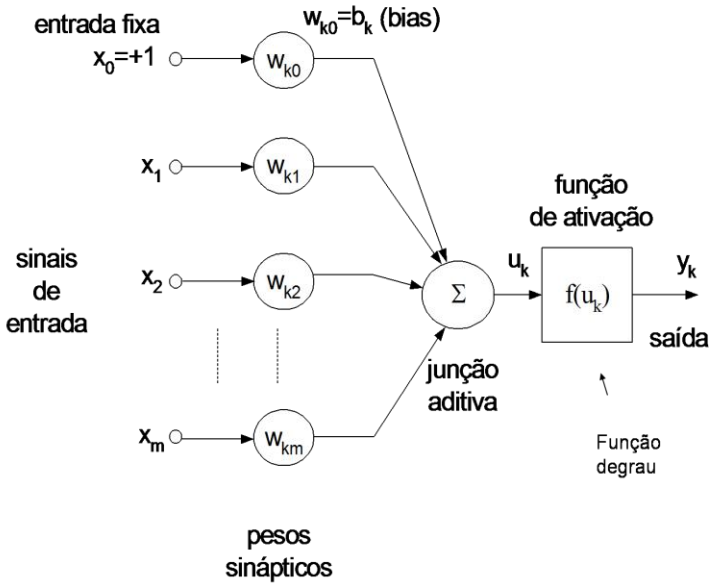


Figura 1: Diagrama do Perceptron

A operação de um neurônio artificial se resume em:

- Sinais (valores) são apresentados às entradas (x_1 à x_m);
- Cada sinal (entrada x_j) é multiplicado por um peso (w_{kj}) que indica sua influência na saída y_k do neurônio;
- É feita a soma ponderada dos sinais de entrada que produz um nível de atividade (u_k);
- A função de ativação $f(u_k)$ transforma o valor de u_k, limitando o valor da saída entre máximo e mínimo e introduzindo não-linearidade ao modelo;
- O bias (b_k) tem o papel de aumentar ou diminuir a influência do valor das entradas;
- É possível considerar o bias como uma entrada de valor constante 1, multiplicado por um peso igual a b_k.

Matematicamente, a operação do neurônio *k* pode ser representado pela seguinte equação:

$$y_k = f(u_k) = f\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

onde $b_k = w_0$.

A função de ativação do tipo limiar (ou *threshold*, ou degrau), é descrita como:

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

e possui o seguinte gráfico:

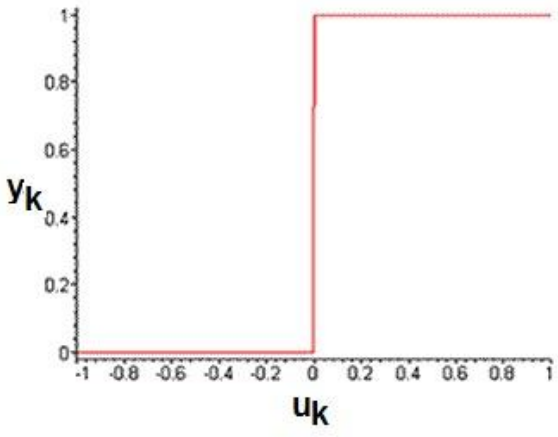


Figura 2: Função degrau (ou limiar)

Um Perceptron de duas entradas com função de ativação degrau pode implementar a função lógica AND desde que os pesos e bias do neurônio tenham seus valores ajustados para se comportar de acordo com a Tabela 1 e Figura 3.

Tabela 1: Função lógica AND

x_2	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1

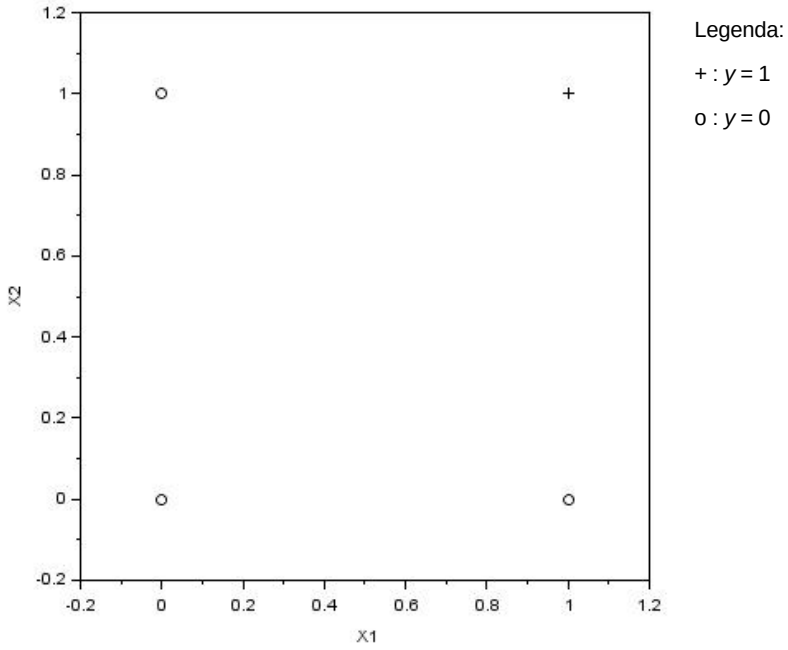


Figura 3: Gráfico da representação da função lógica AND

Uma solução possível para a implementação da função lógica AND utilizando um Perceptron seria:

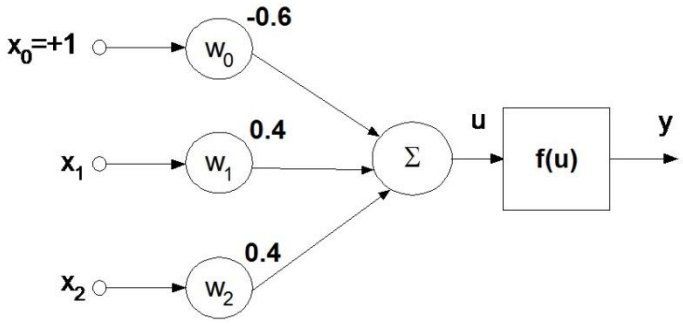


Figura 4: Perceptron que implementa a função lógica AND

onde f(u) é a função de ativação degrau (ou limiar).

Prática:

1 - Desenvolver e implementar o Perceptron como uma função no Matlab. A função de ativação do Perceptron é a limiar (ou degrau, com saída 0 ou 1).
Declaração da função:

***function* y = *yperceptron*(W,b,X)**

Onde:

- X (m x N) é uma matriz de números reais. Cada linha (m) de X é um dos M atributos (ou características) das amostras de dados. Cada coluna de X é uma das N instâncias (ou amostras) dos dados;
- W (1 x m) é um vetor linha de números reais com o valor dos pesos de cada entrada m (atributo) do Perceptron;
- b é um escalar com o valor em real do bias do Perceptron para a entrada que está sempre em +1;
- y (1 x N) é um vetor linha com os valores da saída no neurônio (0 ou 1) para cada amostra de X.
- m e N podem ser qualquer valor inteiro maior que zero;

Obs.: a função deve ser criada de forma genérica, isto é, capaz de aceitar qualquer número de entradas (linhas) e amostras (colunas) para X.

2 - Desenvolver o programa (*script*) que utiliza e testa a função criada no item 1, com os seguintes valores que implementam a função lógica AND:

```
X = [0 1 0 1 ; 0 0 1 1];
W = [0.4 0.4];
b = -0.6;
```

A saída do Perceptron (função lógica AND) deverá ser:

```
y = [0 0 0 1]
```

O fluxograma do *script* pode ser visualizado na Figura 5.

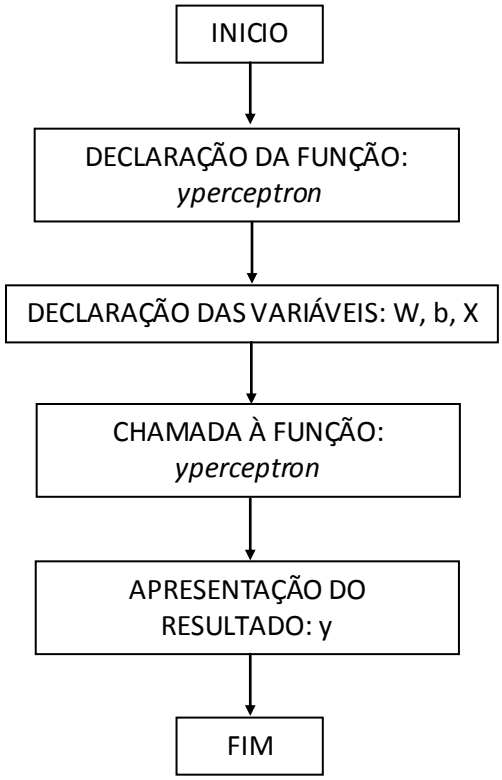


Figura 5: Fluxograma geral de um *script* de implementação de um Perceptron

3 - Repetir o item anterior para a função lógica OR. Para isto, modifique os valores de pesos e bias se necessário. A Tabela 2 apresenta a função lógica OR.

Tabela 2: Função lógica OR

x_2	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

A saída do Perceptron (função lógica OR) deverá ser:

y = [0 1 1 1]

4 - Postar na sala virtual da disciplina o relatório referente a prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

2ª EXPERIÊNCIA: Algoritmo do Perceptron

Título: Algoritmo de treinamento do *Perceptron*

Objetivos:

- Conhecer o algoritmo de treinamento do Perceptron;
- Implementar a Regra Delta de treinamento do Perceptron;
- Utilizar o algoritmo de treinamento do Perceptron na classificação de dados (função lógica AND).

Teoria:

Algoritmo de aprendizagem de Perceptrons

O algoritmo de treinamento do Perceptron executa as seguintes tarefas:

- Executa os exemplos (amostras ou padrões de entrada) de treinamento através do neurônio artificial;
- Ajusta (modifica) os pesos depois de cada exemplo para reduzir o erro;
- Completa uma época de treinamento a cada ciclo de passagem das amostras através do neurônio artificial;
- Repete-se as épocas até que se alcance algum critério de parada;
- Atinge-se um critérios de parada: quando um número máximo de épocas de treinamento é alcançado, ou quando o somatório dos erros quadráticos (SEQ) atinge um valor mínimo tolerável.

No pseudocódigo abaixo se encontra um exemplo de um algoritmo de treinamento do Perceptron:

```
function [W, b, VetorSEQ] = perceptron (W, b, X, yd, alfa, max_epocas, tolerancia)
   inicializar N                          // N = número de amostras de X
   SEQ ← tolerancia                       // SEQ = somatorio dos erros quadraticos
   Epoca ← 1                              // começa na Epoca 1
   while (Epoca ≤ max_epocas) & (SEQ ≥ tolerancia) do
      SEQ ← 0                             // inicia SEQ da Epoca
      for i from 1 to N do                // para cada padrão i de entrada
         y_i ← f(W.x_i + b)               // determinar a saída do neurônio para amostra i
         erro_i ← yd_i – y_i              // determinar o erro
         W ← W + alfa.erro_i.x_i          // atualizar o vetor peso
         b ← b + alfa.erro_i              // atualizar o bias
         SEQ ← SEQ + erro_i^2             // acumula SEQ
      end for
      VetorSEQ ← [VetorSEQ  SEQ]          // salva SEQ da Epoca
      Epoca ← Epoca + 1
   end while
end function
```

A ideia do algoritmo é ajustar repetidamente os pesos do neurônio para minimizar alguma medida de erro no conjunto de treinamento, geralmente o somatório dos erros quadráticos (SEQ) de cada amostra de treinamento durante uma época, ou erro médio quadrático (EMQ) de todas as amostras durante uma época.

Se os padrões de entrada forem linearmente separáveis, o algoritmo de treinamento possui convergência garantida, isto é, tem capacidade para encontrar um conjunto de pesos que classifica corretamente os dados, e o algoritmo se encerrará com erro zero.

Se os padrões de entrada não foram linearmente separáveis, então o algoritmo se encerrará pelo número máximo de épocas de treinamento. O erro médio quadrático final não será igual a zero.

Prática:

1 - Implementar uma função no Matlab para treinamento do Perceptron considerando qualquer número de entradas e quantidade de amostras.

Declaração:

***function* [W,b,VetorSEQ] = *treina_perceptron*(W,b,X,yd,alfa,maxepocas,tol)**

Onde:

- W = matriz (k x m) de pesos; para apenas um neurônio k = 1
- b = vetor (k x 1) com valor do bias de cada neurônio
- X = matriz (m x N) com as amostras dos dados em colunas
- yd = matriz (k x N) com as saídas desejadas para cada amostra de dados
- k = número de neurônios
- m = número de entradas de cada neurônio (dimensão dos dados)
- N = número de amostras dos dados
- alfa = taxa de correção do peso; taxa de aprendizagem
- maxepocas = valor máximo de épocas de treinamento
- tol = erro máximo tolerável
- VetorSEQ = matriz (k x maxepocas) com todos os somatórios dos erros quadráticos por Época obtidos durante aprendizado de cada neurônio

Um exemplo de fluxograma da função *treina_perceptron* é apresentado na Figura 6.

Obs.: a função *treina_perceptron* possui uma chamada a função *yperceptron*.

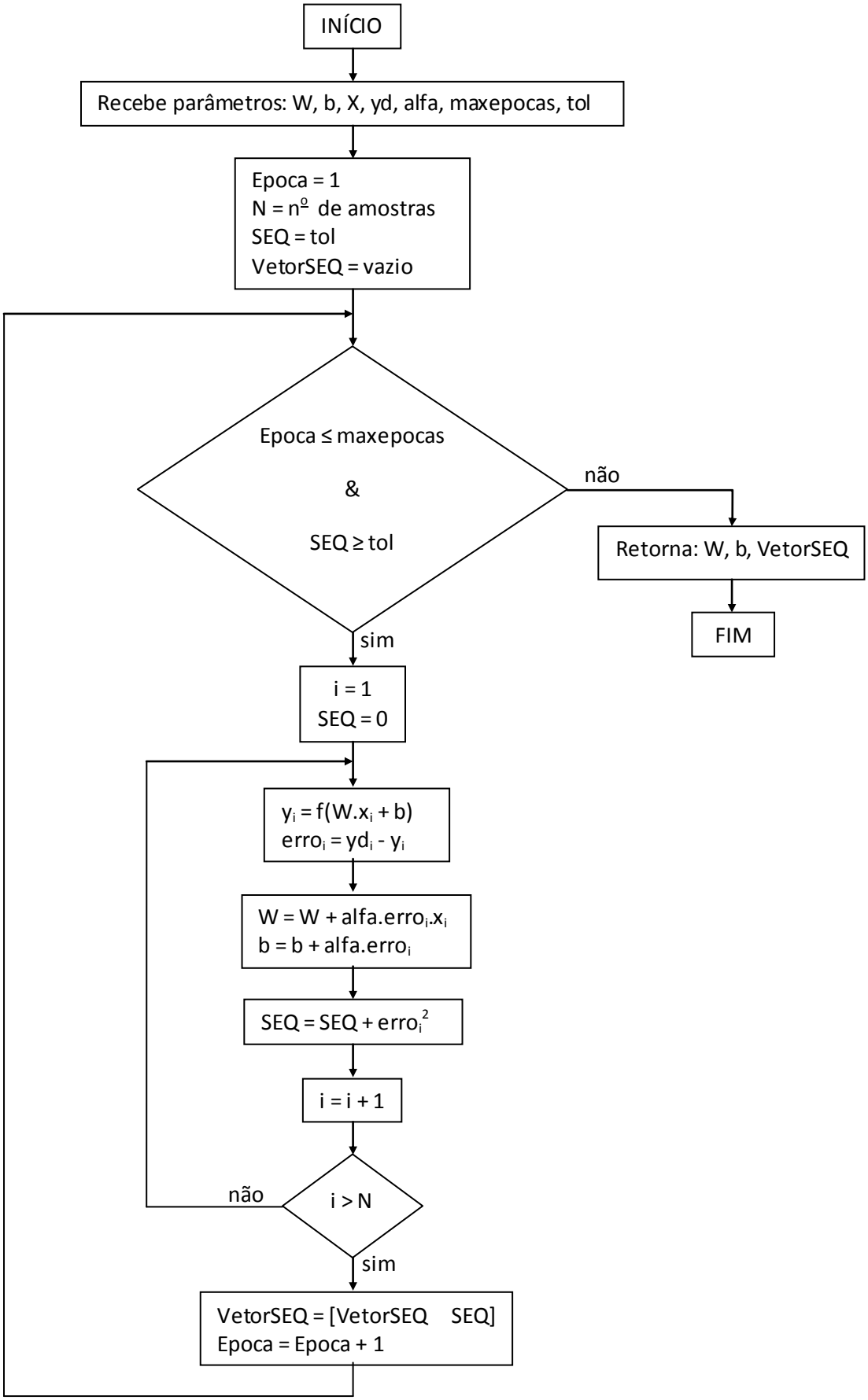


Figura 6: Fluxograma do algoritmo de treinamento do Perceptron

2 - Implementar um script para testar a função acima, em conjunto com a função *yperceptron* criada em exercício anterior, para treinar o Perceptron na solução do problema da função lógica AND de duas entradas binárias. Colocar valor inicial aleatório para os pesos e bias, entre -1 e +1 . Após treinamento, plotar o gráfico da evolução do erro quadrático durante treinamento (dica: usar comando *plot*). Usar:

```
X =  [0 1 0 1 ;
      0 0 1 1];
yd = [0 0 0 1];
alfa = 1.2;
maxepocas = 10;
tol = 0.001;
```

Um exemplo do *script* pode ser visto na Figura 7.

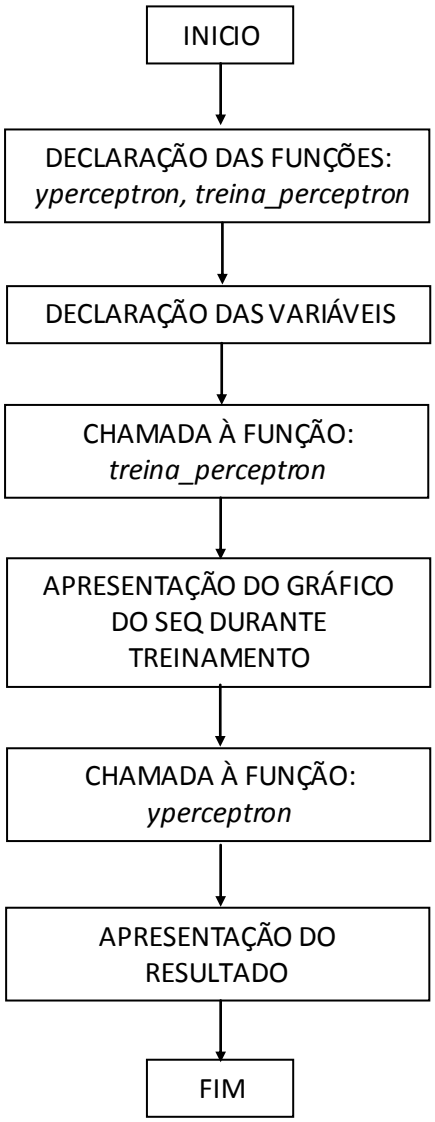


Figura 7: Fluxograma geral de um *script* de treinamento de um Perceptron

Obs.: a chamada à função *yperceptron* (antes de apresentar o resultado) deve ser feita passando os pesos do neurônio treinado.
Um gráfico esperado para o SEQ durante treinamento é mostrado na Figura 8.

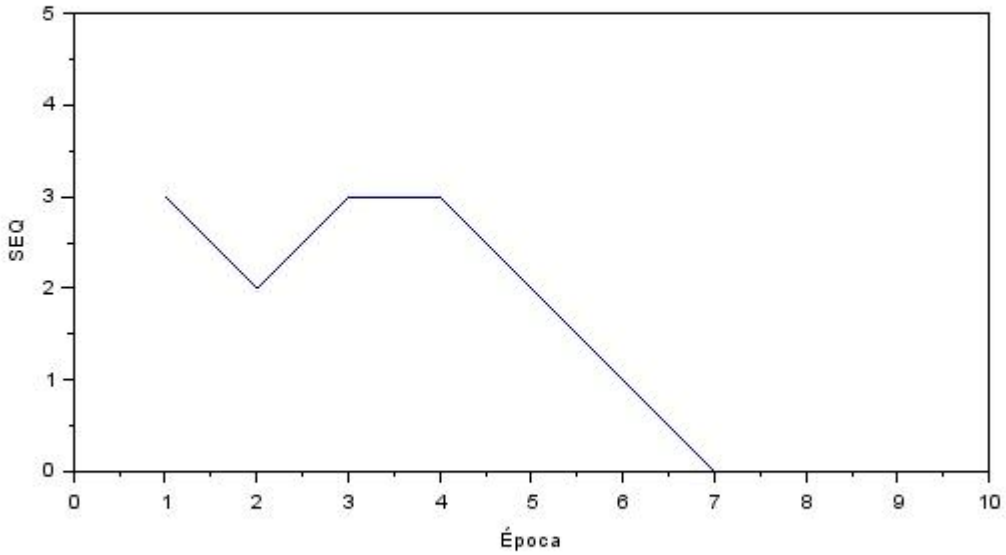


Figura 8: Gráfico da evolução do SEQ (Somatório dos Erros Quadráticos) durante treinamento do Perceptron

3 - Repetir o item 2 para a função lógica OR. Neste caso, usar yd = [0 1 1 1].

4 - Postar na sala virtual da disciplina o relatório referente a prática (**veja modelo no Apêndice**) em arquivo formato pdf sem compactar até a data combinada.

3ª EXPERIÊNCIA: Plota dados e reta

Título: Plota dados e reta de duas dimensões

Objetivos:

- Criar um algoritmo para plotar dados de duas dimensões;
- Criar um algoritmo para plotar reta de duas dimensões;
- Utilizar as 2 funções criadas para visualizar os dados e a reta obtida com o treinamento do Perceptron na classificação de dados.

Teoria:

Considerando o Perceptron de duas entradas mostrado na Figura 9.

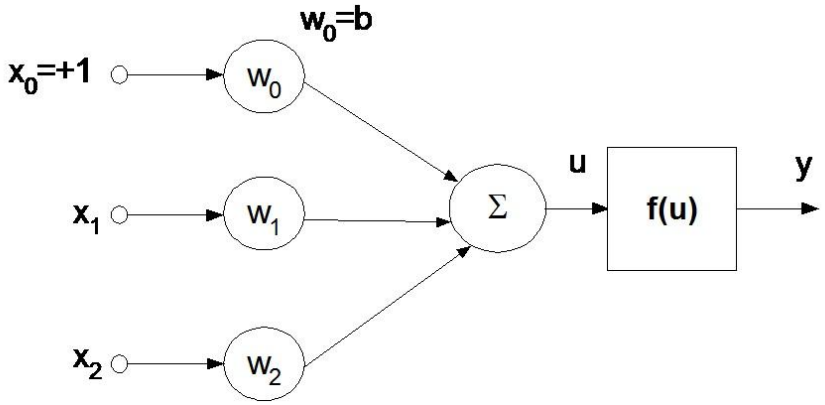


Figura 9: Perceptron de duas entradas

Considerando os valores dos parâmetros w_0, w_1 e w_2, a função $f(u)$ separa o espaço de entradas (x_1, x_2) em duas regiões, usando uma linha reta dada pela equação:

$$w_1x_1 + w_2x_2 + w_0 = 0$$

ou

$$x_2 = -(w_1x_1 + w_0)/w_2$$

obtidas fazendo $u = 0$, ou seja:

$$\sum_{j=0}^{m} w_{kj} x_j = 0$$

Para:

$w_1 = 0.4$

$w_2 = 0.4$

$b = -0.6$

então, a reta de separação tem o aspecto mostrado no gráfico da Figura 10.

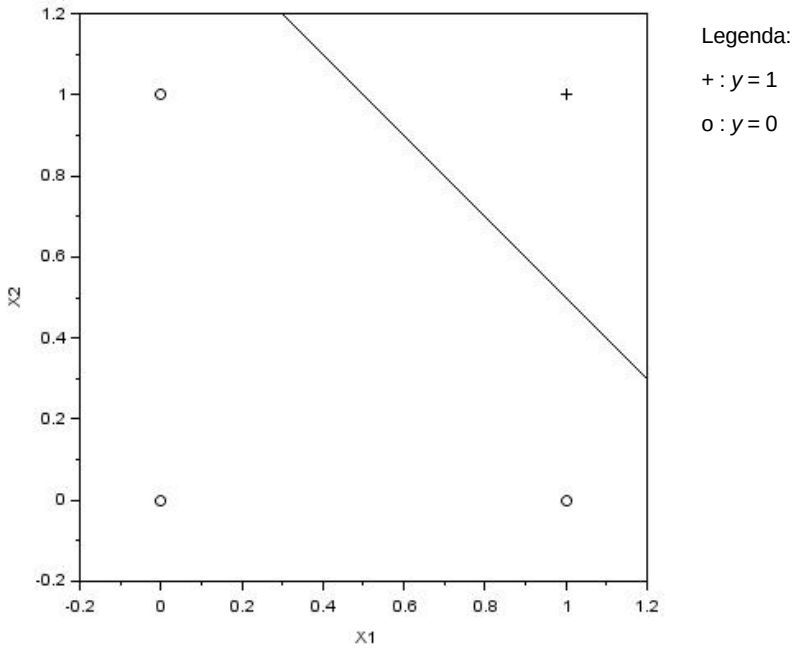


Figura 10: Reta de separação (classificação) dos dados obtida pelo Perceptron de duas entradas treinado para resolver a função lógica AND

Prática:

Criar duas funções no Matlab:

1 - Criar uma função (*plotadc2d*) que recebe:

- uma matriz X (tamanho 2 x N: duas linhas por N colunas) de dados numéricos reais de duas dimensões (x_1 - na linha 1, x_2 - na linha 2) onde cada coluna representa uma amostra de dados; e
- um vetor yd (tamanho 1 x N) com a classificação numérica (números inteiros) de cada amostra de dados dentre C classes possíveis (a classe varia de 0 à C-1).

Em seguida, a função plota o gráfico (em 2 dimensões) das amostras utilizando um símbolo diferente para cada classe de dados.
Cada coluna da matriz X é uma amostra (exemplo) de duas dimensões. Cada elemento de yd (número inteiro) é a classificação da amostra de X da mesma coluna.

Nome da função: *plotadc2d* (plota amostras de diferentes classes em 2d)
Chamada da função: ***plotadc2d*(X,yd)**
A função não retorna nada.
Dica: utilizar função *"plot"*, *"rand"*, comando *"for"*, e qualquer outra função pré-definida no MATLAB (**por exemplo: *"find"*).**
Para facilitar a implementação, a quantidade máxima de classes (C) pode ser limitado a 10 (dez).

2 - Criar uma função (*plotareta*) que plota a reta de separação dos dados obtida com um Perceptron de duas entradas (dados de duas dimensões) em um gráfico de coordenadas x_2 e x_1.
Chamada da função: ***plotareta*(W, *b*, intervalo)**
onde:

- W : vetor de pesos, [w_1 w_2]
- *b* : valor do bias
- intervalo : vetor que contém os valores mínimo e máximo para x_1, [x_{1min} x_{1max}]

A função não retorna nada.

3 - Testar as duas funções com um script que utiliza o resultado do exercício anterior: treinamento do Perceptron para implementar a função lógica AND e OR. Plotar os dados classificados pelo Perceptron treinado e a reta de separação das classes de dados no mesmo gráfico. Plotar também o desenvolvimento do somatório do erro quadrático (SEQ) durante treinamento.
Um possível fluxograma é mostrado na Figura 11. Um exemplo de gráfico obtido é apresentado na Figura 10.

4 - Postar na sala virtual da disciplina o relatório referente a prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

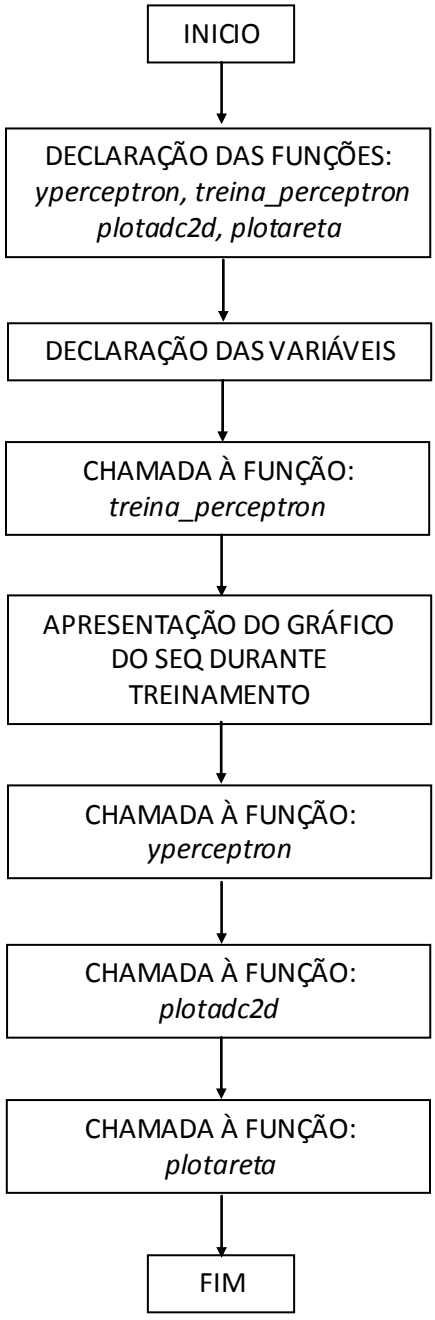


Figura 11: Fluxograma para treinamento do Perceptron e apresentação do resultado dos dados reclassificados pelo neurônio treinado com a reta de separação dos dados.

4ª EXPERIÊNCIA: Classificação de dados gaussianos

Título: Geração e classificação de dados aleatórios com distribuição gaussiana

Objetivos:

- Gerar dados de duas dimensões rotulados em classes para treinamento do Perceptron;
- Implementar um algoritmo para misturar os dados gerados;
- Utilizar as funções criadas para treinar o Perceptron utilizando a Regra Delta.

Teoria:

Uma variável real aleatória *X* é chamada gaussiana (ou normal) se sua densidade de probabilidade de ocorrência (*P*) for dada pela equação:

$$P(X) = \frac{1}{\sqrt{2.\pi.\sigma^2}}.e^{-\frac{(x-\bar{x})^2}{2.\sigma^2}}$$

onde:

x = um valor real de ocorrência da variável aleatória *X*;

$\bar{x}$ = valor médio de ocorrência da variável aleatória *X*;

σ = desvio padrão da variável aleatória *X*.

Esta densidade de probabilidade pode ser representada pelo gráfico indicado na Figura 12.

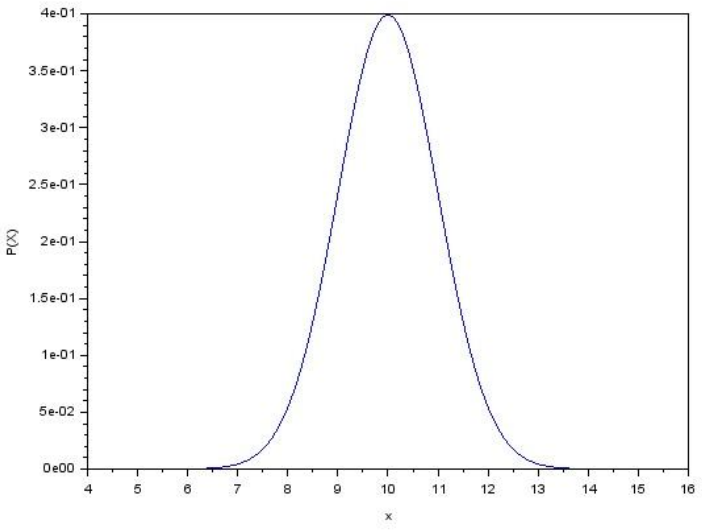


Figura 12: Gráfico da densidade de probabilidade de uma variável aleatória gaussiana (normal)

Duas variáveis aleatórias independentes (X_1 *e* X_2) são consideradas gaussianas se o gráfico de sua densidade de probabilidade conjunta P(X_1,X_2) puder ser representada pelo gráfico 3D mostrado na Figura 13.

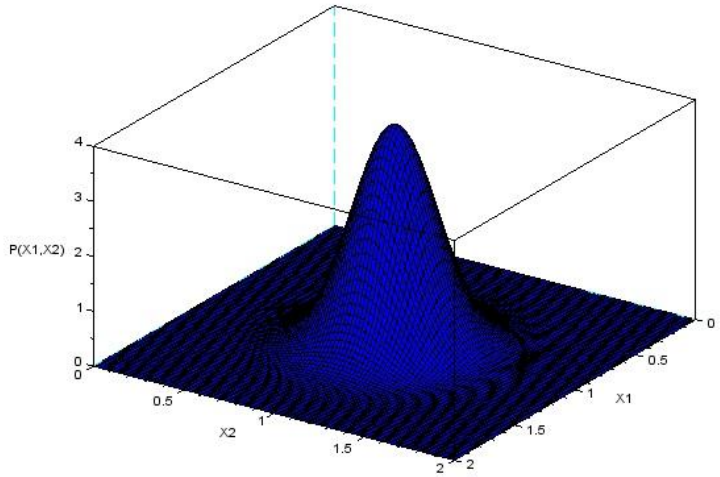


Figura 13: Gráfico da densidade de probabilidade conjunta de duas variáveis aleatórias gaussianas.

Prática:

1 - Criar uma função no MATLAB para gerar dados de duas dimensões, aleatórios, com distribuição gaussiana, e a correspondente classificação para cada amostra.
Declaração da função:

function **[X,yd] =** ***geragauss*****(nc,npc,mc,varc)**

onde:

- nc = tipo escalar que identifica o número total de classes dos dados
- npc = vetor linha com quantidade de amostras para cada classe de dados
- mc = matriz (2 x nc) com o valor médio de cada dimensão para cada classe
- varc = matriz (2 x nc) com a variância de cada dimensão para cada classe
- X = matriz (2 x N) com as amostras (por coluna) de todas as classes

  $X = [\ x_{11}\ x_{12}\ x_{13}\ x_{14}\ \ldots\ x_{1N}$

  $x_{21}\ x_{22}\ x_{23}\ x_{24}\ \ldots\ x_{2N}\ ]$
- N = total do número de amostras
- yd = vetor linha (1 x N) com a classificação das amostras de X, valores entre 0 e (nc-1). O valor da classe começa em zero.

  $yd = [yd_1\ \ yd_2\ \ yd_3\ \ yd_4\ \ \ldots\ yd_{npt}\ ]$ onde: $0 \leq yd_i < (nc-1)$

Dica: use a função *"rand"* da biblioteca do MATLAB com a opção *"normal"*.

2 - Criar uma função que mistura os dados da matriz X do exercício anterior, e correspondentemente também a classificação de cada amostra no vetor yd.

Declaração da função:

***function* [xp,yp] = *mistura*(X,yd)**

onde:

- X = matriz (2 x N) com as amostras (por coluna) de todas as classes
- yd = vetor linha (1 x N) com a classificação numérica dos dados
- xp = matriz (2 x N) com as amostras (por coluna) de todas as classes misturadas
- yp = vetor linha (1 x N) com a classificação numérica dos dados misturada
- N = número total de amostras (soma de todas as classes)

Dica: use a função *grand*.

3 - Criar um script no MATLAB que gera dados de duas classes e duas dimensões linearmente separáveis e mistura a ordem (utilizando as funções dos itens 1 e 2), e utiliza-os para treinar um Perceptron (utilizando as funções criadas em exercícios anteriores). Ao final do treinamento, o Perceptron deve reclassificar os dados de treinamento e plotar os resultados obtidos da reclassificação. Plotar também o desenvolvimento do somatório do erro quadrático (SEQ) de treinamento por época e a reta de separação das classes obtida pelo Perceptron treinado. Um exemplo de fluxograma do script é apresentado na Figura 14.

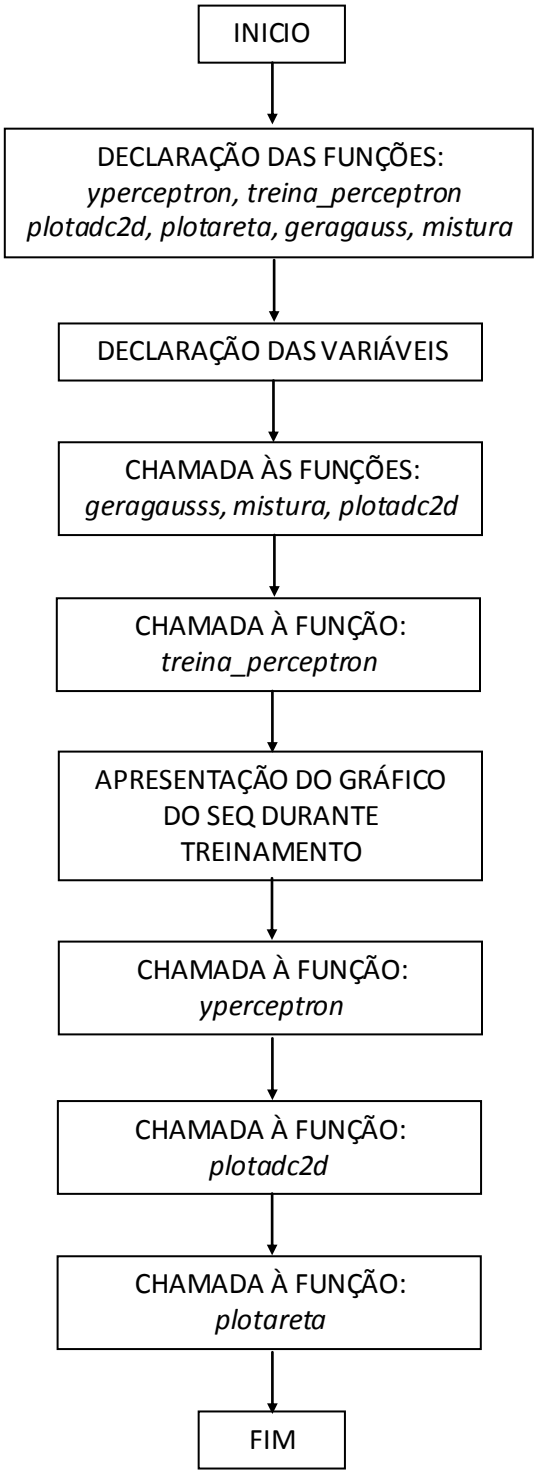


Figura 14: Fluxograma do script para treinamento do Perceptron na tarefa de classificação de dados de duas classes e duas dimensões com distribuição gaussiana e linearmente separáveis.

Exemplos de resultados obtidos para dados de duas classes e duas dimensões linearmente separáveis são mostrados nas Figuras de 15 a 17.

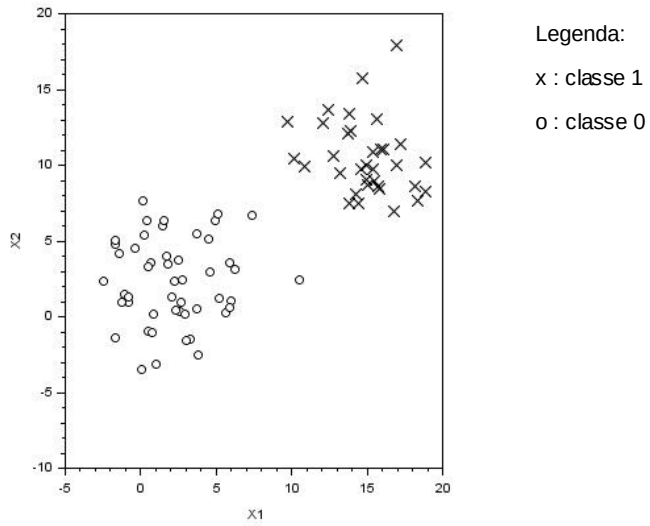


Figura 15: Gráfico dos dados (duas dimensões e duas classes) gerados aleatoriamente com distribuição gaussiana e linearmente separáveis.

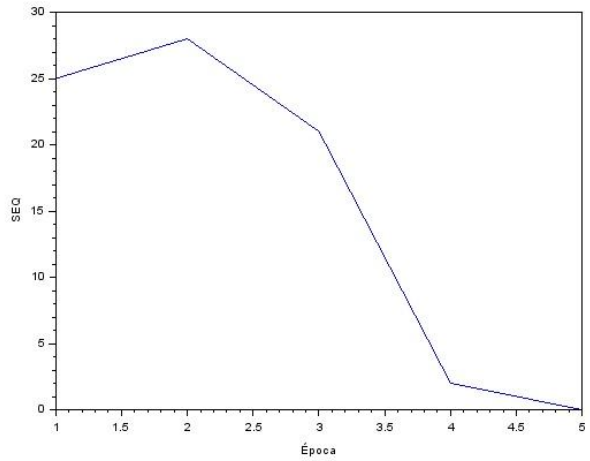


Figura 16: Gráfico do Somatório dos Erros Quadráticos por Época de treinamento do Perceptron.

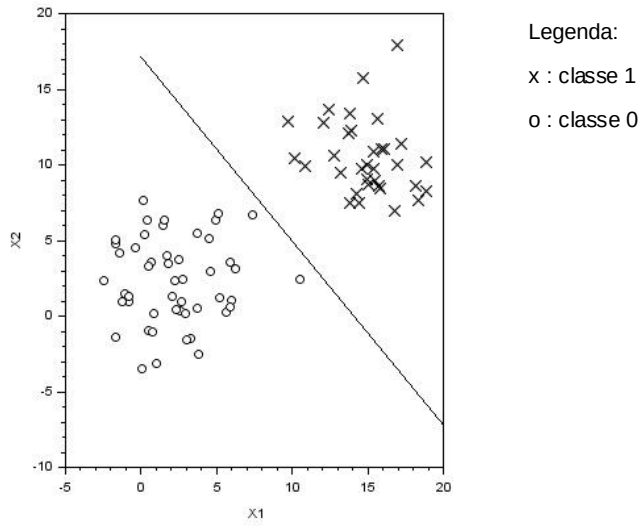


Figura 17: Gráfico dos dados reclassificados com o Perceptron treinado e reta de separação das classes obtida. Observe que as amostras continuam com a mesma classificação original.

4 - Repetir o item 3 para dados não linearmente separáveis.

Exemplos de resultados obtidos para dados de duas classes e duas dimensões não linearmente separáveis são mostrados nas Figuras de 18 a 20.

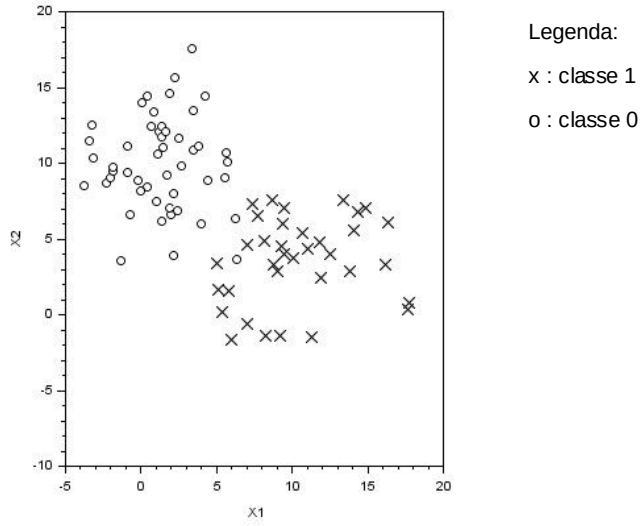


Figura 18: Gráfico dos dados (duas dimensões e duas classes) gerados aleatoriamente com distribuição gaussiana e não linearmente separáveis.

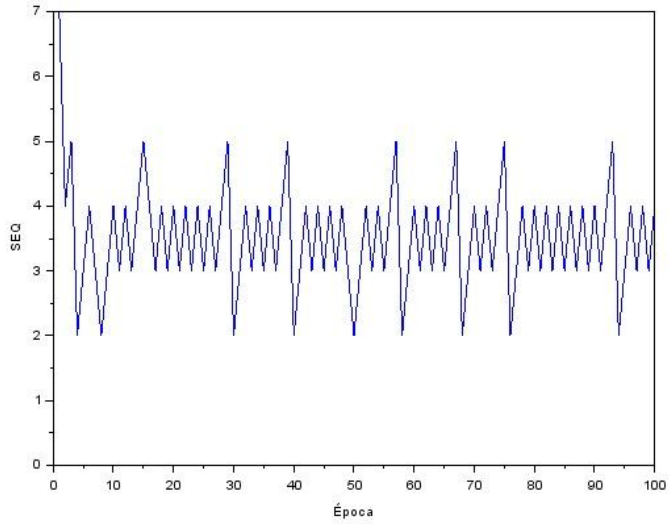


Figura 19: Gráfico do Somatório dos Erros Quadráticos por Época de treinamento do Perceptron.

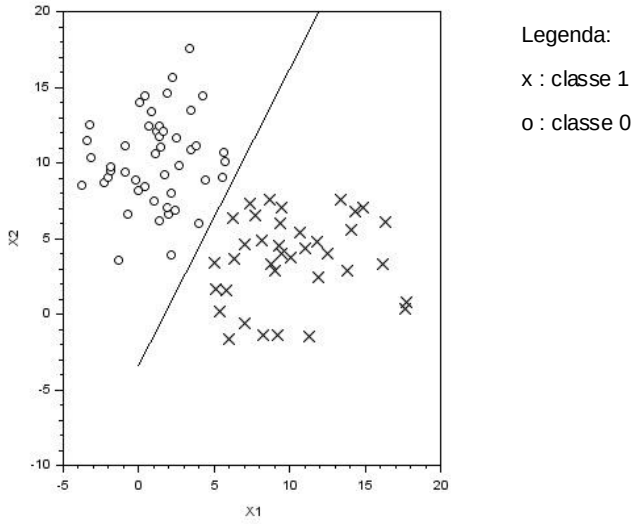


Figura 20: Gráfico dos dados reclassificados com o Perceptron treinado e reta de separação das classes obtida. Observe as amostras que tiveram a classificação trocada perto da reta de separação.

5 - Postar na sala virtual da disciplina o relatório referente a prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

5ª EXPERIÊNCIA: Adaline

Título: Adaline

Objetivos:

- Conhecer o neurônio artificial tipo Adaline;
- Implementar o Adaline no MATLAB;
- Utilizar o Adaline para aproximar uma reta (sem e com ruído nos dados de treinamento).

Teoria:

Widrow e Hoff propuseram o algoritmo dos mínimos quadrados (Regra Delta) para o neurônio denominado Adaline (Adaptive Linear Element), que era similar ao Perceptron (Figura 1) porém com função de ativação linear ao invés de função degrau (limiar). Veja Figura 21.

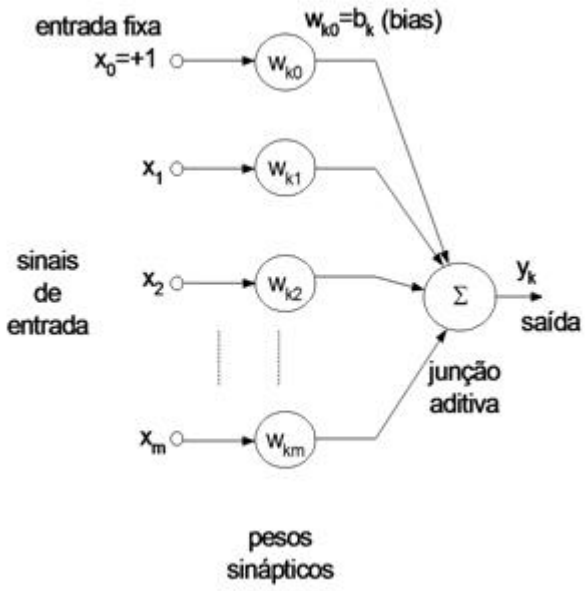


Figura 21: Adaline.

O objetivo do algoritmo de treinamento do Adaline é minimizar o somatório do erro quadrático (SEQ) entre a saída do neurônio e a saída desejada. A soma dos erros quadráticos para um determinado padrão *i* é dada por:

$$E = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (d_i - y_i)^2$$

A direção oposta ao gradiente de *E* é a direção de maior decrescimento do erro. Assim, o erro pode ser reduzido ajustando-se os pesos da rede de acordo com:

$$w_{kj} = w_{kj} - \alpha \frac{\partial E}{\partial w_{kj}}$$

onde w_{kj} é o peso específico para o neurônio *k*, da entrada *j*, e α é a taxa de aprendizagem.

Como o gradiente do erro pode ser calculado para cada amostra *i*, então:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_{i=1}^{n} (d_i - y_i)^2 = \frac{\partial}{\partial w_{kj}} (d_i - y_i)^2$$

Como no Adaline a função de ativação é linear, e aplicando a regra da cadeia, então:

$$y_i = f(\mathbf{w}_k \mathbf{.x}) = f(\sum_j w_{kj} x_j) = \sum_j w_{kj} x_j$$

$$\frac{\partial E}{\partial w_{kj}} = -2(d_i - y_i) \frac{\partial y_i}{\partial w_{kj}} = -2(d_i - y_i) x_j$$

Assim:

$$w_{kj} = w_{kj} + \alpha (d_i - y_i) x_j$$
$$b_k = b_k + \alpha (d_i - y_i)$$

que é denominado **Regra Delta**.

Prática:

Implementar o neurônio Adaline (função de ativação linear) no MATLAB e aplicá-lo na regressão linear de dados de uma dimensão. Seguir os seguintes passos:

1 - Criar a função *"yadaline"* que calcula a saída do neurônio Adaline.

Declaração:

***function* y = *yadaline*(W,b,X)**

onde:

- X = matriz (*m* x N) de dados com N amostras de *m* características (*m* entradas);
- W = vetor linha (1 x *m*) de pesos das entradas;
- b = polarização do neurônio;
- y = vetor linha (1 x N) com os valores da saída no neurônio para cada amostra.

Dica: aproveite a função *yperceptron* já implementada em prática anterior; basta retirar a função de ativação do Perceptron.

2 - Criar a função *"treina_adaline"* que treina o neurônio Adaline (função idêntica a função *"treina_perceptron"* desenvolvida em exercício anterior, exceto pela chamada a função *yperceptron*). Declaração:

***function* [W,b,VetorSEQ] = *treina_adaline*(W,b,X,yd,alfa,maxepocas,tol)**

3 - Gerar um vetor *X* (1 x N) de dados de entrada (uma entrada apenas com N amostras) e um vetor *Yd* (1 x N) de dados de saída correspondente, segundo uma reta crescente. O aluno define os parâmetros da reta. Os dados deverão ser misturados no vetor *X* da mesma forma no vetor *Yd* utilizando a função mistura já criada em prática anterior. Plotar o gráfico dos dados gerados para conferência. Um exemplo dos dados gerados pode ser visto na Figura 22.

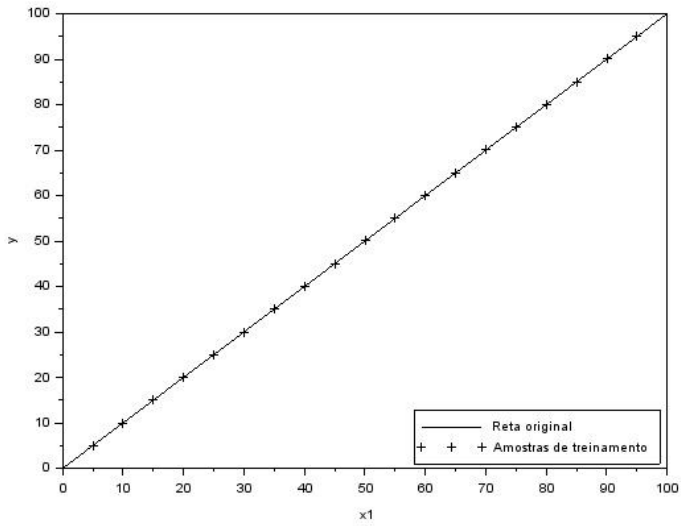


Figura 22: Gráfico apresentando a reta de origem e as amostras (símbolo +) de treinamento do Adaline.

4 - Treinar um neurônio do tipo Adaline com os dados gerados no item 3 para fazer a regressão linear utilizando a função *treina_Adaline*. Sugere-se os seguintes valores dos parâmetros de treinamento:

maxepocas = 10000

tol = $1x10^{-9}$

alfa = $1x10^{-4}$

Plotar a evolução do somatório dos erros quadráticos de treinamento do Adaline em função da época. Plotar também um outro gráfico com os dados gerados junto com a reta encontrada pelo Adaline treinado (utilizar a função "*yadaline*" para obter a resposta com outros dados de entrada). Um fluxograma possível do *script* é mostrado na Figura 23, e nas Figuras 24 a 26 são apresentados os resultados gráficos esperados.

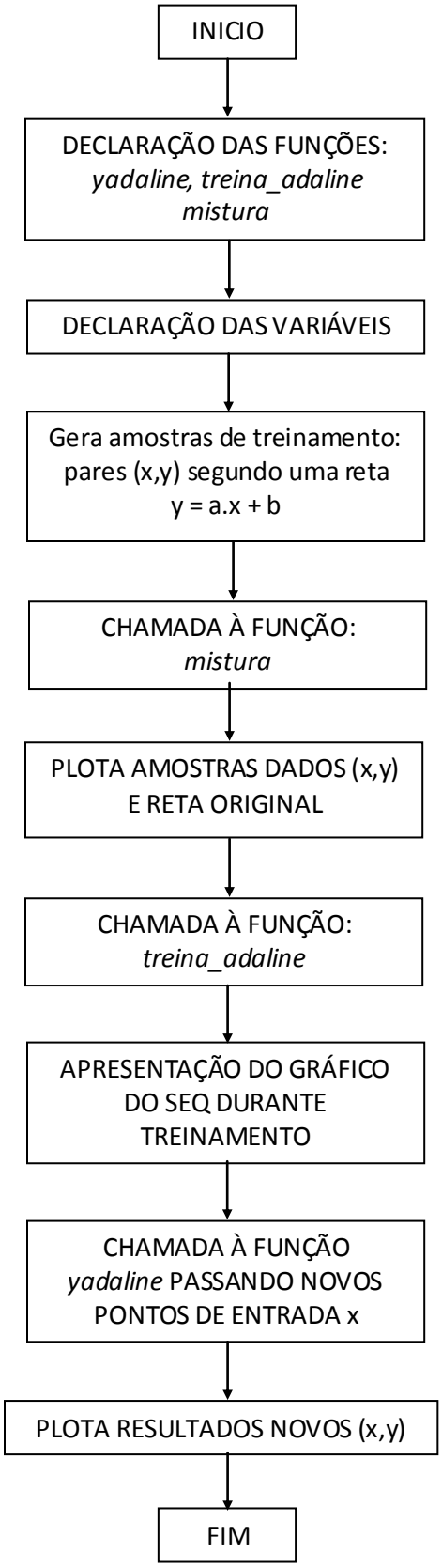


Figura 23: Fluxograma do *script* que treina o Adaline para a tarefa de regressão linear.

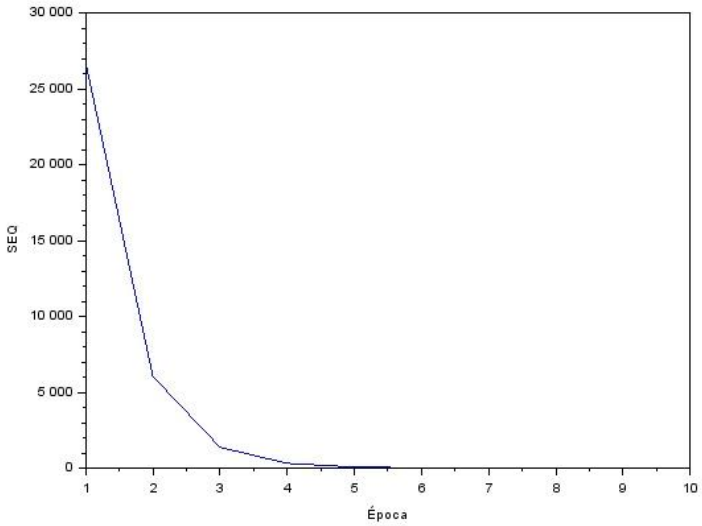


Figura 24: Gráfico da evolução do SEQ durante o treinamento do Adaline. Repare que o erro chega a zero.

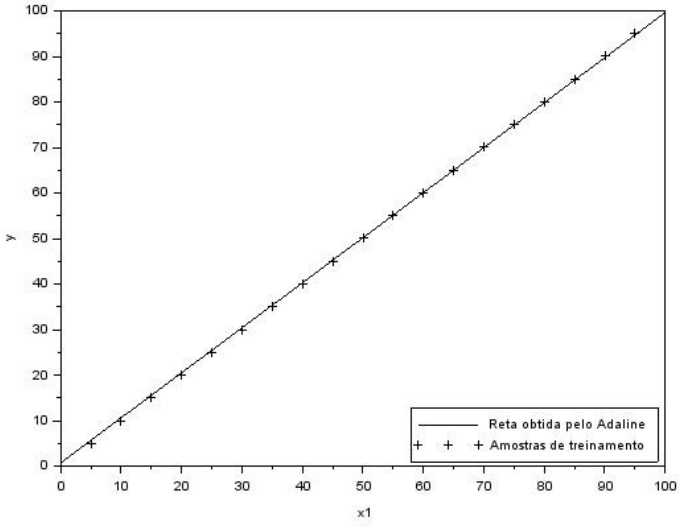


Figura 25: Gráfico apresentando as amostras (símbolo +) de treinamento e a reta obtida pelo Adaline. Repare que coincidem.

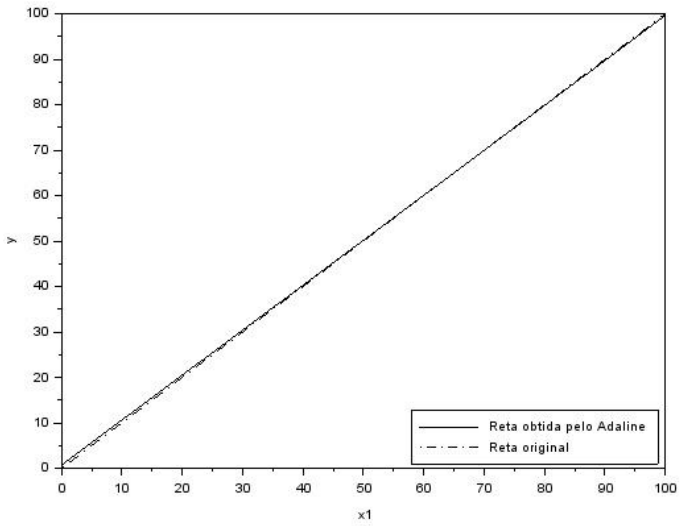


Figura 26: Gráfico apresentando a reta original (que gerou as amostras de treinamento) e a reta obtida com o Adaline treinado. Repare que coincidem.

5 - Acrescentar ruído aos valores de saída (utilize a função *"randn"* do Matlab), e repetir o item anterior.

Um fluxograma possível para implementação do *script* está apresentado na Figura 27. Nas Figuras 28 a 31 são apresentados os resultados esperados.

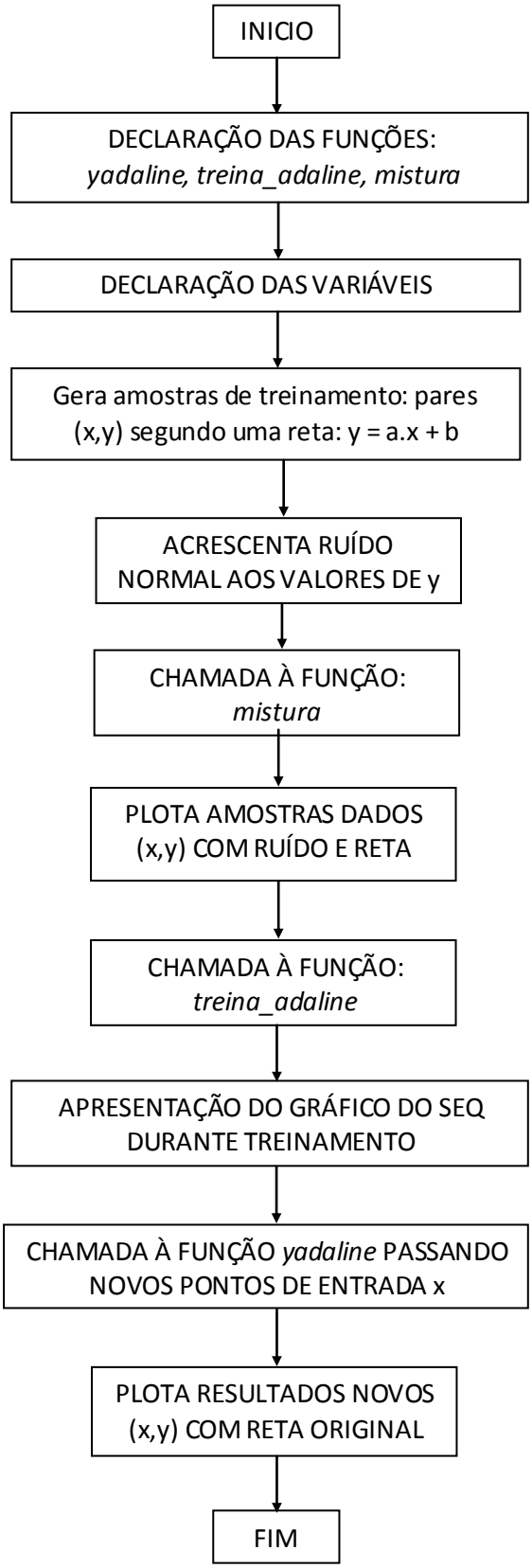


Figura 27: Fluxograma do *script* que treina o Adaline para tarefa de regressão linear com dados com ruído.

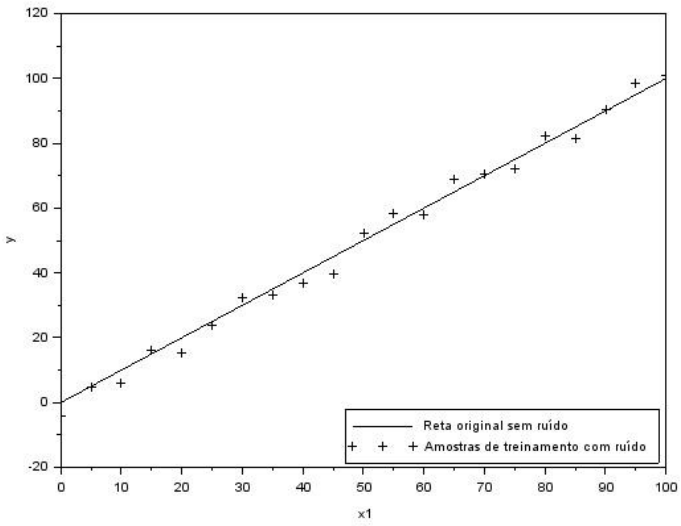


Figura 28: Gráfico apresentando a reta de origem e as amostras (símbolo "+") de treinamento do Adaline com ruído.

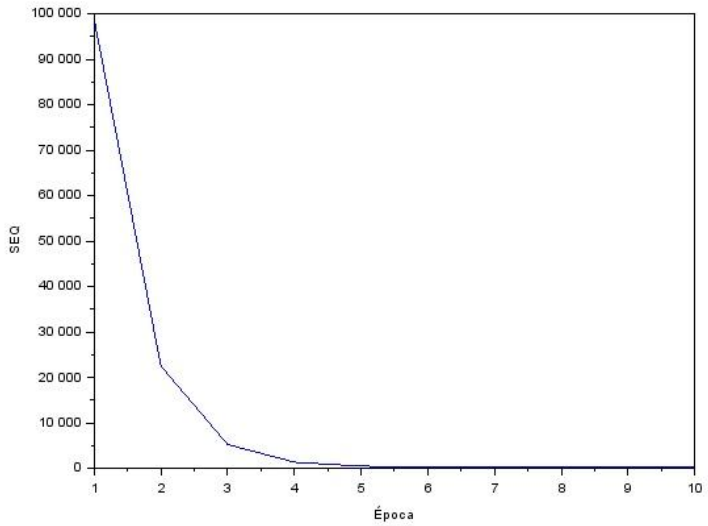


Figura 29: Gráfico da evolução do SEQ durante o treinamento do Adaline. Repare que o erro chega quase a zero.

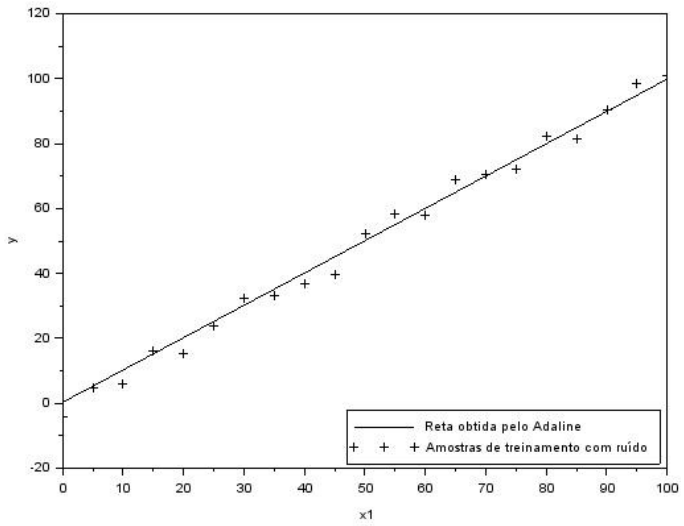


Figura 30: Gráfico apresentando as amostras de treinamento (símbolo "+") e a reta obtida pelo Adaline. Repare que o Adaline encontrou uma reta que passa entre as amostras.

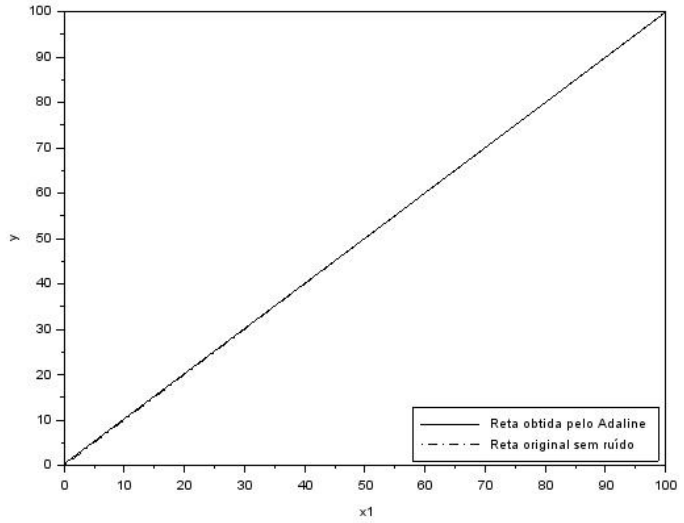


Figura 31: Gráfico apresentando a reta original (que gerou as amostras de treinamento) e a reta obtida com o Adaline treinado com as amostras com ruído.

6 - Postar na sala virtual da disciplina o relatório referente a prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

6ª EXPERIÊNCIA: Momentum

Título: Treinamento do Adaline com momentum

Objetivos:

- Implementar o algoritmo de treinamento do Adaline com momentum;
- Testar se a técnica de treinamento com momentum agiliza o treinamento do Adaline.

Teoria:

Aprendizagem com momento (ou *momentum*) usa uma memória (incremento anterior) para aumentar a velocidade e estabilizar a convergência do treinamento do neurônio artificial. Para isto, a equação de correção dos pesos (regra Delta) fica:

$$w(t+1) = w(t) + \alpha.e_i(t).x'_i + \beta.[w(t) - w(t-1)]$$

onde:

- w: é o vetor (linha) de pesos a ser ajustado do neurônio;
- t: é a iteração atual;
- $t+1$: é a iteração posterior (futuro);
- $t-1$: é a iteração anterior (passado);
- α: é a taxa de aprendizagem;
- β: é a constante de momento, normalmente β é ajustada entre 0,5 e 0,9;
- e_i: é o erro produzido pela amostra i de entrada;
- x_i: é o vetor (coluna) da amostra de entrada que gerou o erro.

Obs.: o mesmo vale para o bias, onde $x_i = 1$, ou seja:

$$b(t+1) = b(t) + \alpha.e_i(t) + \beta.[b(t) - b(t-1)]$$

Prática:

1 - Implementar o Adaline do exercício anterior com *momentum,* para regressão linear de dados de uma dimensão com ruído. Modificar a função *treina_adaline* para receber a taxa de momento como parâmetro de entrada da função. A declaração da função fica como:

***function* [W,b,VetorSEQ] = *treina_adaline*(W,b,X,yd,alfa,beta,maxepocas,tol)**

Sugere-se os seguintes valores para os parâmetros de treinamento:

$\alpha = 1x10^{-4}$

$\beta = 0,01$

maxepocas = 10000

$tol = 1x10^{-9}$

2 - Verificar se o treinamento do neurônio torna-se mais rápido com *momentum*. Um exemplo dos resultados esperados (mas não todos) é apresentado nas Figuras 32 e 33.

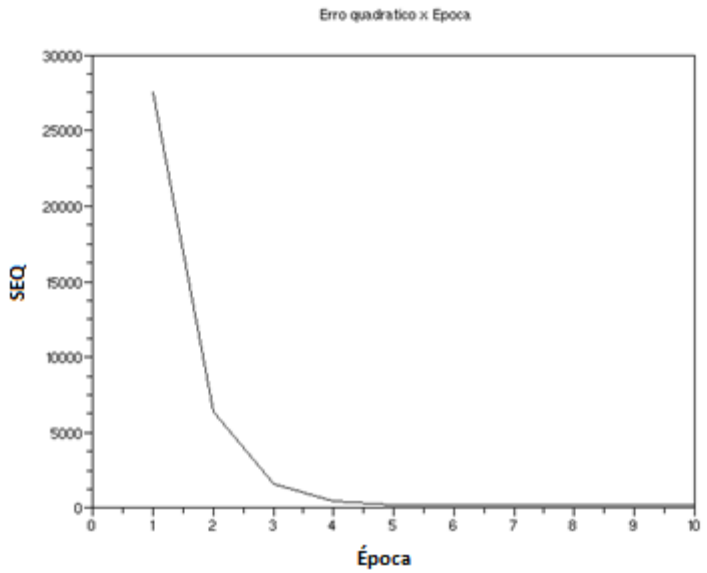


Figura 32: Gráfico da evolução do SEQ durante o treinamento do Adaline sem momento. Repare que o erro chega quase a zero na época 5.

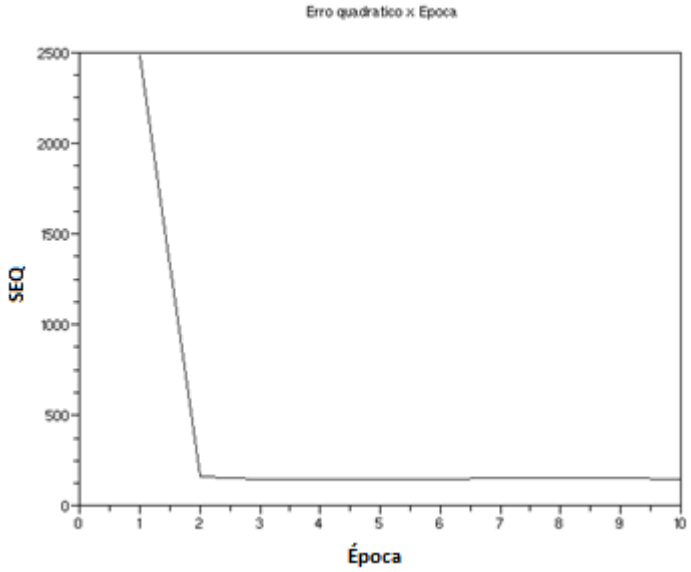


Figura 33: Gráfico da evolução do SEQ durante o treinamento do Adaline com momento. Repare que o erro chega quase a zero na época 2.

3 - Postar na sala virtual da disciplina o relatório referente à prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

7ª EXPERIÊNCIA: Rede de camada simples

Título: Rede Perceptron de camada simples

Objetivos:

- Conhecer a Rede Neural Artificial de camada simples (rede Perceptron de camada única);
- Implementar o algoritmo de treinamento de uma rede de camada simples;
- Utilizar a rede de camada simples para resolver um problema de classificação de dados com mais de duas classes.

Teoria:

Arquitetura das RNAs

- Uma rede neural artificial é composta por várias unidades de processamento (neurônios), cujo funcionamento é bastante simples.
- Essas unidades, geralmente são conectadas por canais de comunicação que estão associados a determinado peso.
- As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões.
- O comportamento inteligente de uma Rede Neural Artificial vem das interações entre as unidades de processamento da rede.

Tipos de Redes Neurais Artificiais

Existem basicamente 3 tipos básicos de arquitetura de RNAs:

- *Feedforward* de uma única camada;
- *Feedforward* de múltiplas camadas; e
- Redes recorrentes.

Rede *feedforward de* uma única camada

- Os neurônios da camada de entrada correspondem aos neurônios sensoriais que possibilitam a entrada de sinais na rede (não fazem processamento).
- Os neurônios da camada de saída fazem processamento (vide Figura 34).

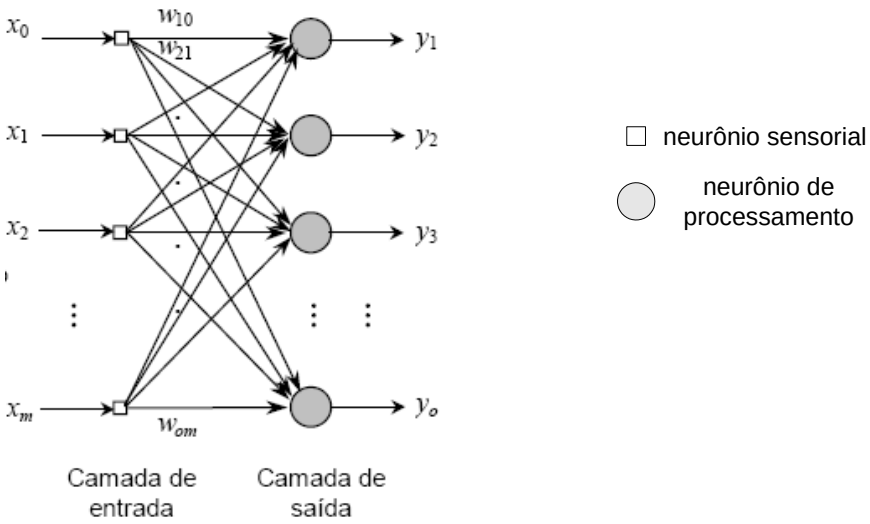


Figura 34: Arquitetura de uma Rede Neural Artificial tipo Perceptron de camada única.

Prática:

Implementar e treinar uma rede de neurônios tipo Perceptron de camada única para classificar dados de mais de duas classes, usando o MATLAB:

1 - Gerar dados aleatórios de duas dimensões e três classes com distribuição gaussiana, sem interseção de classes (linearmente separáveis), utilizando as funções "*geragauss*" e "*mistura*" já criadas em exercício anterior. Plotar os dados utilizando a função "*plotadc2d*" já desenvolvida em experiência anterior.

2 - Criar uma função no MATLAB que converta o vetor de classificação (código decimal) das amostras dos dados gerados no item 1, para uma matriz de classificação com formato binário. Exemplo:

Vetor original:

Yd = [1 3 0 1 0 ...]

Vetor convertido:

```
Ydc  = [0 0 1 0 1 ...;     -> classe 0
        1 0 0 1 0 ...;     -> classe 1
        0 0 0 0 0 ...;     -> classe 2
        0 1 0 0 0 ...;     -> classe 3
        etc....        ]
```

Onde:

- cada linha de Ydc corresponde a saída desejada de um neurônio tipo Perceptron da rede de neurônios. O valor 1 significa que a amostra do dado correspondente (na mesma coluna da matriz X de dados) pertence a classe identificada pela linha com 1. O valor 0 significa que a amostra do dado correspondente não pertence a classe identificada pela linha;
- cada coluna corresponde as saídas de todos os neurônios da rede Perceptron para uma dada amostra de entrada.

Resumindo, crie uma função (*converte_dec_bin*) que converta o vetor de números decimais (classificação das amostras) para uma matriz de números binários.

3 - Treinar 3 Perceptrons (função de ativação limiar) com os dados gerados usando as funções "*yperceptron*" e "*treina_perceptron*" já desenvolvidas em exercício anterior e modificadas para treinar vários neurônios com os mesmos dados de entrada, mas com saídas desejadas diferentes (cada linha da matriz binária).

4 - Plotar os dados gerados e reclassificados pela rede Perceptron (separados em classes usando a função *"plotadc2d"*) e as retas de separação das classes obtidas com o treinamento dos Perceptrons (função *"plotareta"*), e o gráfico da evolução do somatório dos erros quadráticos durante treinamento (SEQ x Época) para cada neurônio.

Para usar a função *plotadc2d*, será necessário criar uma função que converta a classificação de binário para decimal (*converte_bin_dec*), ou seja, faça a conversão inversa do item 2. Um exemplo de fluxograma do script pode ser visto na Figura 35. Nas Figuras 36 a 38 são apresentados os resultados esperados.

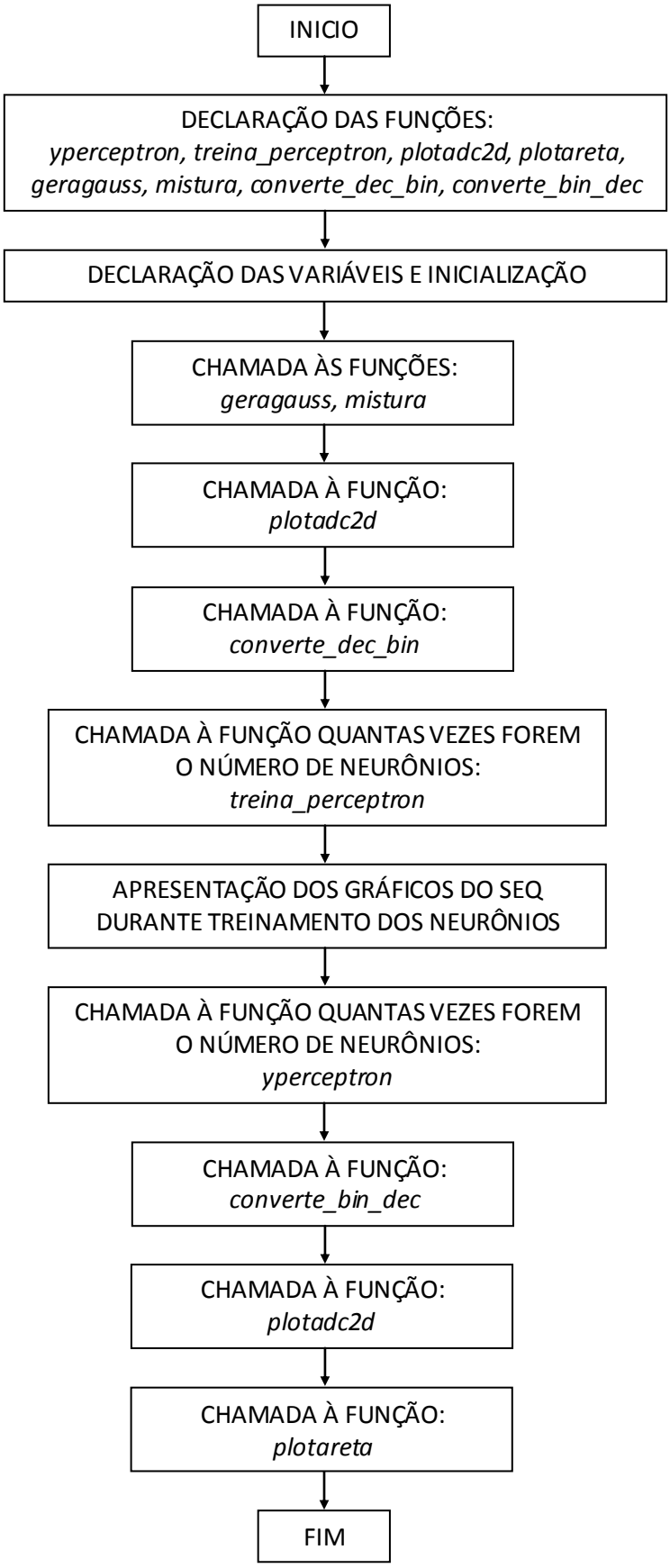


Figura 35: Fluxograma do *script* que implementa uma Rede Neural Artificial do tipo Perceptron de camada única para classificação de dados.

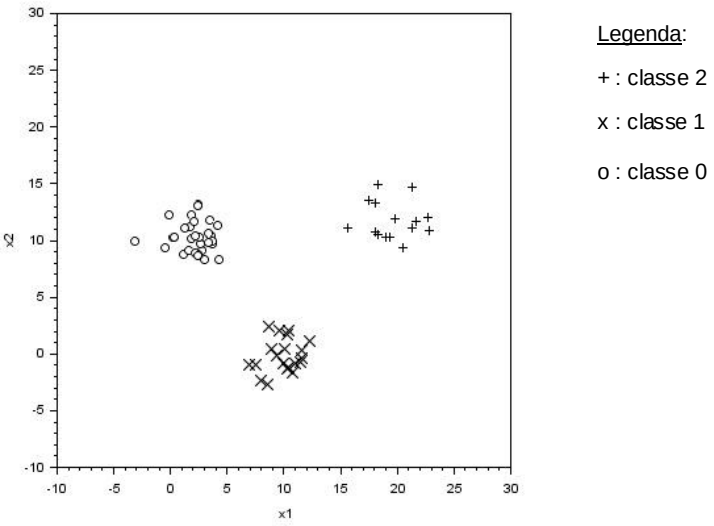


Figura 36: Gráfico dos dados (duas dimensões e três classes) gerados aleatoriamente com distribuição gaussiana e linearmente separáveis.

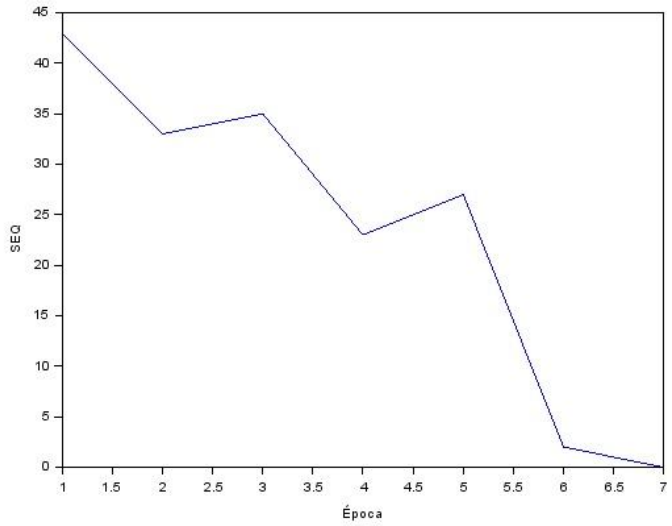


Figura 37: Gráfico do Somatório dos Erros Quadráticos por Época de treinamento de um dos neurônios da rede Perceptron de camada simples. Observe que o erro chegou a zero na época 7 e o treinamento foi interrompido.

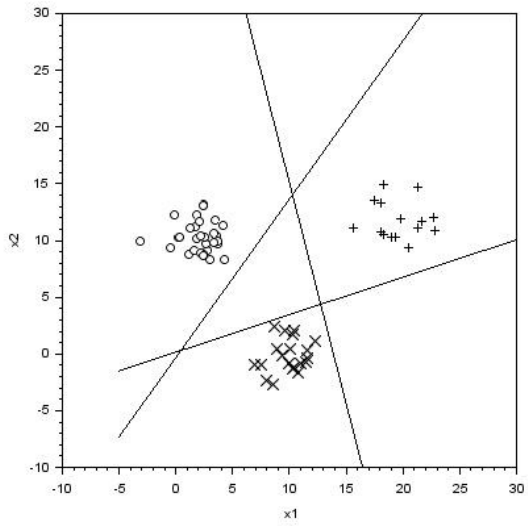


Figura 38: Gráfico dos dados reclassificados com a rede Perceptron treinada e as retas de separação das classes obtidas. Observe que as amostras de dados continuam com a mesma classificação original.

5 - Repetir o treinamento usando dados com sobreposição de classes (dados não linearmente separáveis) e gerar os resultados gráficos. Nas Figuras 39 a 41 são apresentados os resultados esperados.

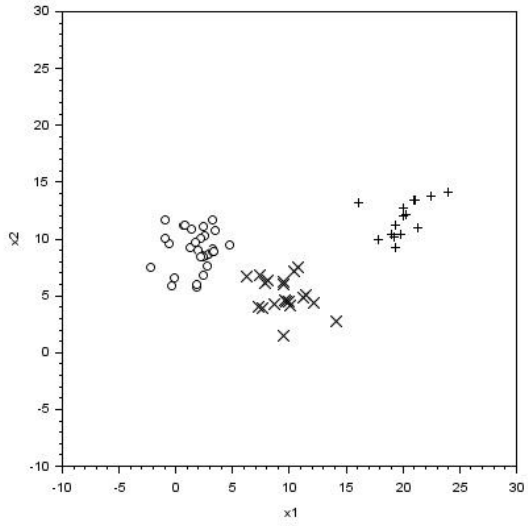


Figura 39: Gráfico dos dados (duas dimensões e três classes) gerados aleatoriamente com distribuição gaussiana e não linearmente separáveis.

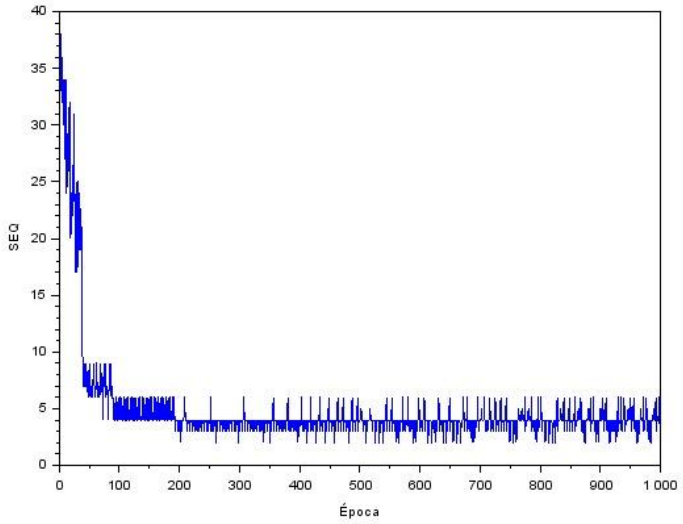
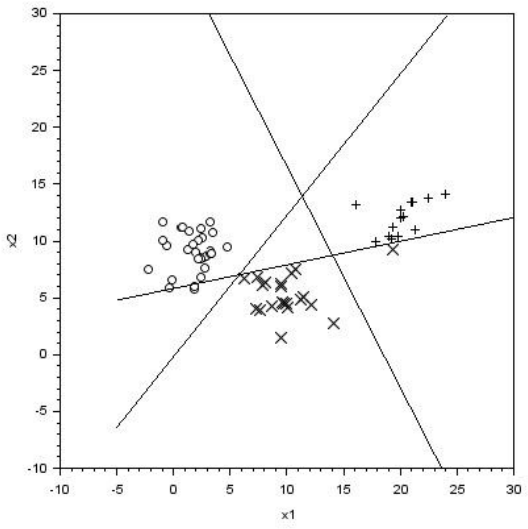


Figura 40: Gráfico do Somatório dos Erros Quadráticos por Época de treinamento de um dos neurônios da rede Perceptron de camada única. Repare que o erro não chega a zero e o treinamento é interrompido pelo número máximo de épocas.



Legenda:

+ : classe 2

x : classe 1

o : classe 0

Figura 41: Gráfico dos dados reclassificados com a rede Perceptron treinada e as retas de separação das classes obtidas. Observe que uma das amostras teve a classificação trocada.

6 - Postar na sala virtual da disciplina o relatório referente a prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

8ª EXPERIÊNCIA: Rede MLP

Título:

Implementação e treinamento de uma Rede Perceptron Multicamadas (MLP) para solução do problema XOR

Objetivos:

- Conhecer a Rede Neural tipo Perceptron com multicamadas;
- Implementar a MLP no Matlab;
- Utilizar a rede MLP na classificação de dados não linearmente separáveis.

Teoria:

Rede *feedforward* de Múltiplas Camadas (Multilayer Perceptron - MLP)

A partir da Figura 42, observa-se que as camadas de uma rede tipo MLP são classificadas em três grupos:

- Camada de Entrada: onde os padrões são apresentados à rede;
- Camadas Intermediárias ou Ocultas: onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características; Essas redes tem uma ou mais camadas intermediárias ou escondidas.
- Camada de Saída: onde o resultado final é concluído e apresentado.

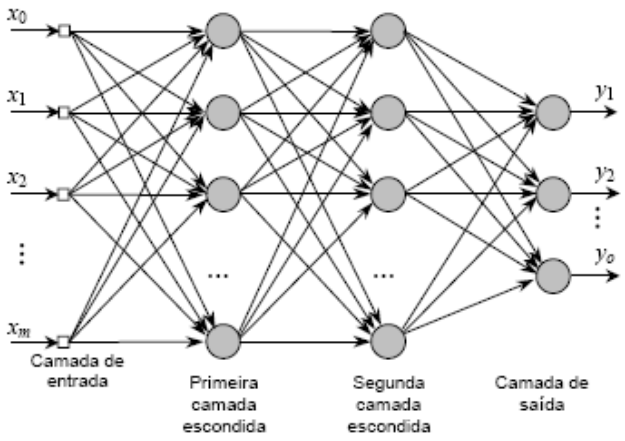


Figura 42: Diagrama de uma Rede Neural Artificial tipo Perceptron de múltiplas camadas (MLP - *Multi Layer Perceptron*) .

Denomina-se algoritmo de aprendizado a um conjunto de regras bem definidas para a solução de um problema de aprendizado. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais. Estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados.

A quantidade de neurônios nas camadas de entrada e saída é dada pelo problema a ser abordado. No entanto, a quantidade de neurônios nas camadas de processamento (camadas ocultas) são características do projeto. Aumentando-se o número de neurônios na camada escondida aumenta-se a capacidade de mapeamento não-linear da rede. No entanto, quando esse número for muito grande, o modelo pode se sobre-ajustar aos dados, na presença de ruído nas amostras de treinamento. Diz-se que a rede está sujeito ao sobre-treinamento (*overfitting*).

Por outro lado, uma rede com poucos neurônios na camada escondida pode não ser capaz de realizar o mapeamento desejado, o que é denominado de *underfitting*. O *underfitting* também pode ser causado quando o treinamento é interrompido de forma prematura.

Algoritmo de aprendizagem – *backpropagation*

- Regra de aprendizagem baseada na correção do erro pelo método do Gradiente
- O algoritmo de aprendizagem é composto de duas fases:
    - Cálculo do erro (fase *forward*)
    - Correção dos pesos sinápticos (fase *backward*)

O erro no k-ésimo neurônio da camada de saída para a amostra *i* é dado por:

$$e_k(i) = d_k(i) - y_k(i)$$

onde: *i = i-ésimo padrão de treinamento apresentado à rede*.

O ajuste dos pesos é realizado por:

$$w_{kj}(i) = w_{kj}(i-1) - \alpha \cdot \delta_k(i) \cdot x_j(i)$$

Onde:

Para a camada de saída:

$$\delta_k(i) = f_k'[u_k(i)].e_k(i)$$

Para a camada oculta:

$$\delta_k(i) = f_k^{'}[u_k(i)] \cdot \sum_{l=1}^{q} \left[\delta_l(i).w_{lj} \right]$$

Durante o treinamento com o algoritmo *backpropagation*, a rede opera em uma sequência de dois passos:

- Primeiro, um padrão é apresentado à camada de entrada da rede. A atividade resultante flui através da rede, camada por camada, até que a resposta seja produzida pela camada de saída.
- No segundo passo, a saída obtida é comparada à saída desejada para esse padrão particular. Se esta não estiver correta, o erro é calculado. O erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificados conforme o erro é retropropagado.

É recomendável que a ordem de apresentação das amostras seja aleatória de uma época para outra. Isso tende a fazer com que o ajuste de pesos tenha um caráter estocástico ao longo do treinamento.

Prática:

1 - Estudar as funções do Toolbox ANN do Matlab:

- feedforwardnet
- configure
- init
- train
- sim
- perform
- getwb
- separatewb
- cell2mat

Dica: use o *help* do Matlab

2 - Implementar o *script* do Matlab a seguir para criar, treinar e executar uma rede neural de duas entradas, dois neurônios na camada escondida, e um neurônio de saída. O *script* utiliza os seguintes dados de treinamento correspondentes à função lógica XOR mostrada na Figura 43:

```
x = [-1 ,  1 , -1 , 1 ;                    y = [-1 ,  1 ,  1 , -1];
     -1 , -1 ,  1 , 1 ];
```

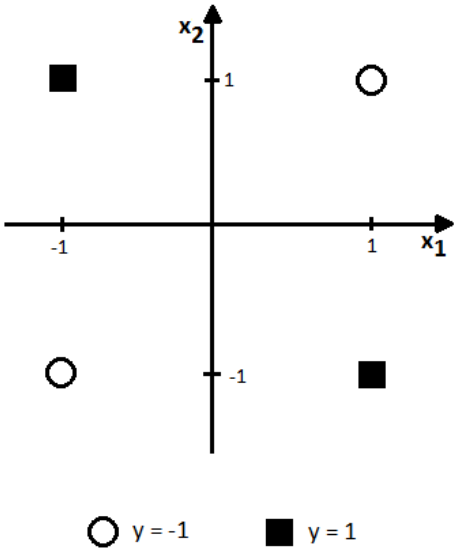


Figura 43: Gráfico dos dados que representam a função lógica XOR.

Um gráfico esperado do resultado da execução do *script* é o apresentado na Figura 44.

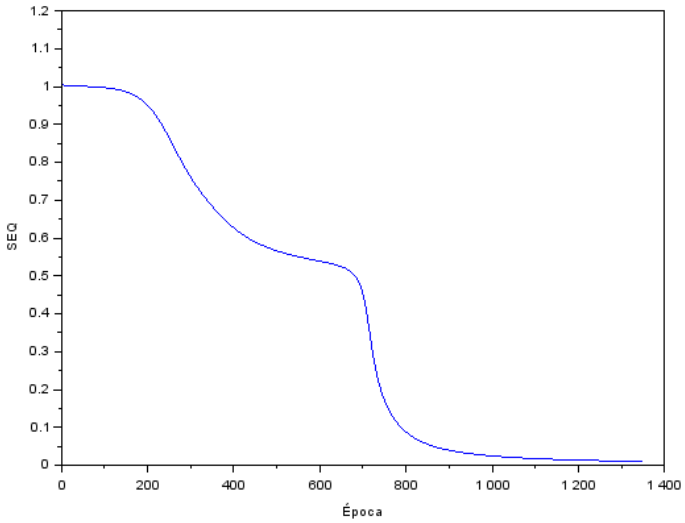


Figura 44: Gráfico do somatório dos erros quadráticos de treinamento por época.

```
close all;
clear all;
clc;

disp('Programa MLP para XOR 2 entradas');

% Entradas
X = [ -1 ,  1 , -1 ,  1 ;
      -1 , -1 ,  1 ,  1 ];

% Saida desejada: funcao logica XOR
Yd =[ -1 ,  1 ,  1 , -1 ];

% Quantidade de neuronios na camada escondida
neuronios_camada_escondida = 2;

disp('Criando a rede MLP...');
net = feedforwardnet(neuronios_camada_escondida);

disp('Configurando a rede...');
net = configure(net,X,Yd);

% divisao dos dados entre treinamento, teste, validacao
net.divideParam.trainRatio = 1;   % training set [%]
net.divideParam.valRatio   = 0;   % validation set [%]
net.divideParam.testRatio  = 0;   % test set [%]

% Ajusta parametros para treinamento:

% metodos de treinamento: backpropagation (traingd), backpropagation com
% momentum (traingdm), Levenberg-Marquardt (trainlm), RPROP (trainrp)
net.trainFcn = 'traingd';

% funcao a ser minimizada: mse, mae, sse
net.performFcn = 'mse';    % erro medio quadratico

% numero maximo de epocas para treinamento
net.trainParam.epochs = 5000;

% taxa de aprendizado
net.trainParam.lr = 0.01;

% taxa de momento
net.trainParam.mc = 0;

% erro maximo permitido
net.trainParam.goal = 0.01;

% Ajusta funcao de ativacao de cada camada da rede: tansig, logsig
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'tansig';

% Inicializa a rede neural com valores de pesos e bias aleatorios
disp('Inicializando a rede neural....');
net = init(net);

% Treina a rede neural
disp('Treinando a rede neural...');
[net, tr] = train(net,X,Yd);

% Plota o grafico do erro durante treinamento
plotperform(tr);
```

```
% Simula a rede neural
disp('Simulando a rede neural treinada...');
Ysaida = sim(net,X);
disp('Resultado: ');
disp(Ysaida);

% Calcula o desempenho obtido
disp('Calculando o erro da rede neural...');
perf = perform(net,Yd,Ysaida);
disp('Erro: ');
disp(perf);

% Obtem os valores de pesos e bias em um unico vetor
wb = getwb(net);

% Separa os valores dos pesos e bias
[b,IW,LW] = separatewb(net,wb);

% Converte de celula para vetor
b = cell2mat(b);
disp('Bias de todas camadas =');
disp(b);

% Converte de celula para vetor
Wescondida = cell2mat(IW);
disp('Pesos da camada escondida =');
disp(Wescondida);

% Converte de celula para vetor
Wsaida = cell2mat(LW);
disp('Pesos da camada de saida =');
disp(Wsaida);
```

Observe os valores finais dos pesos e bias dos neurônios da MLP treinada. Observe o resultado do processo de treinamento e identifique se o treinamento foi interrompido pelo número máximo de épocas ou pelo erro máximo tolerável.

5 - Modifique o *script* para implementar outros algoritmos de treinamento, como por exemplo: Levenberg-Marquardt, *backpropagation* com momentum, RPROP, etc. Plotar o gráfico dos erros quadráticos em função da época e verificar se o treinamento agilizou. Um exemplo de gráfico esperado é mostrado na Figura 45. Repare que o treinamento agilizou quando comparado com o resultado da Figura 44.

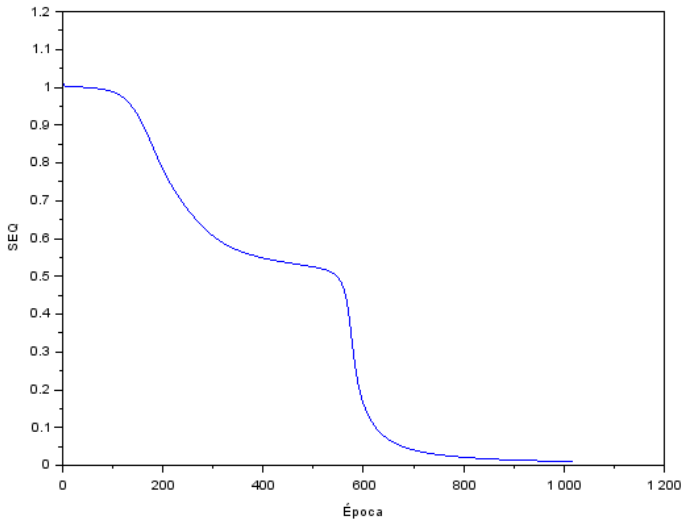


Figura 45: Gráfico do somatório do erro quadrático de treinamento por época utilizando momento.

6 - Postar na sala virtual da disciplina o relatório referente à prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

9ª EXPERIÊNCIA: MLP com multiclasses

Título: Classificação de dados multiclasses com MLP

Objetivos:

- Gerar amostras de dados com mais de 2 classes;
- Implementar uma rede MLP para classificar dados de mais de duas classes;
- Avaliar o desempenho da MLP e verificar a confiabilidade dos resultados.

Teoria:

Inicialização dos vetores de pesos e *bias*

A atualização de um peso entre dois neurônios de uma rede MLP depende da derivada da função de ativação da unidade posterior e função de ativação da unidade anterior, pelo algoritmo *backpropagation*. Por esta razão, é importante evitar escolhas de pesos iniciais que tornem as funções de ativação ou suas derivadas iguais a zero.

Os valores para os pesos iniciais não devem ser muito grandes, tal que as derivadas das funções de ativação tenham valores muito pequenos (região de saturação). Por outro lado, se os pesos iniciais são muito pequenos, a soma pode cair perto de zero, onde o aprendizado é muito lento.

Um procedimento comum é inicializar os pesos e *bias* a valores aleatórios entre -0.5 e 0.5, ou entre -1 e 1. Os valores podem ser positivos ou negativos porque os pesos finais após o treinamento também podem ser positivos ou negativos.

Critérios de parada de treinamento

Um critério de parada, que pode ser usado em conjunto com o número máximo de épocas de treinamento e erro máximo tolerável, é a avaliação da capacidade de generalização da rede após cada época de treinamento. Para isto, são necessários dois conjuntos de dados separados: treinamento e teste, conforme Figura 46. Assim, o processo de treinamento é interrompido antes que a capacidade de generalização da rede piore, ou seja, quando o SEQ dos dados de teste começa a aumentar.

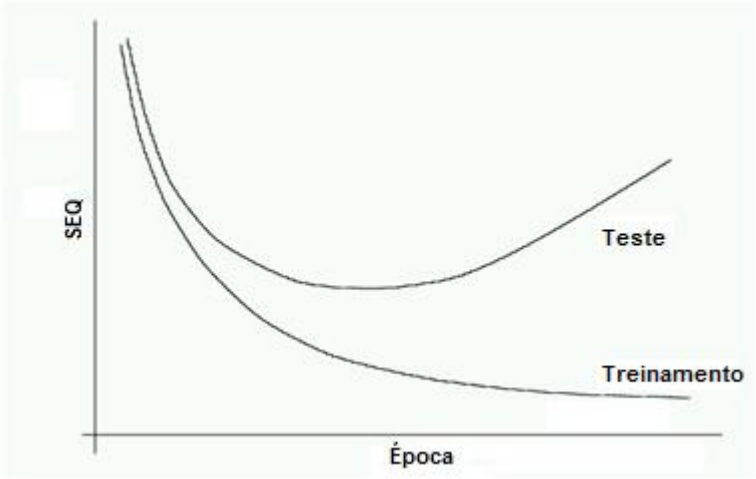


Figura 46: Gráfico do SEQ para dados de teste e dados de treinamento em função da época.

Parada com validação cruzada

É a avaliação da capacidade de generalização da RNA durante treinamento. Assim:

- separa-se os dados em dois conjuntos: treinamento e validação (teste).
- tamanho do conjunto de validação: 10-25% do total das amostras de dados.
- com certa frequência (5 a 10 épocas) verifica-se o desempenho da RNA para o conjunto de validação.
- quando $erro_{valid}$ aumenta, pára-se o treinamento.

O objetivo da aprendizagem não deve ser o erro igual a zero no conjunto de treinamento, mas o menor erro para os dados de validação ou teste.

Prática:

1 - Estudar as funções do Toolbox ANN do Matlab. Dica: use o "*help*".

2 - Elaborar um *script* que:

a - crie várias amostras (mais de 100 por classe) de dados de duas dimensões divididas em 3 classes linearmente separáveis; utilize a função *geragauss* já criada;

b - embaralhe os dados criados; utilize a função *mistura* já criada;

c - plote os dados criados para verificação; utilize a função *plotadc2d* já criada;

d - separe os dados aleatoriamente em dois conjuntos: treinamento (80%) e teste (20%) (ou validação);

e - plote os dados separados em treinamento e teste (2 gráficos separados) para verificação; utilize a função *plotadc2d* já desenvolvida;

f - crie a matriz de classificação binária (saída desejada); utilize a função *conv_dec_bin* já desenvolvida;

g - crie uma rede neural tipo MLP e treine com o conjunto de dados de treinamento; utilize as funções do toolbox ANN do Matlab; sugestões de parâmetros:

- número de entradas: 2
- número de camadas escondidas: 1
- número de neurônios na camada escondida: 9
- número de saídas: 3
- função de ativação: sigmoidal
- taxa de aprendizado: 0.05
- taxa de momento: 0.2
- número máximo de épocas: 400
- tolerância do erro: 0.00001

h - plote o somatório dos erros quadráticos de treinamento por época para verificar se o treinamento convergiu;

i - reclassifique os dados do conjunto de treinamento e teste utilizando a RNA treinada (utilize as funções do Toolbox ANN do Matlab) e contabilize o número de acertos (resposta em porcentagem de acertos separado para treinamento e teste); mostre o resultado e armazene;

j - repita 10 vezes o procedimento a partir do item b, de modo a calcular a média e desvio padrão do resultado final (número de acertos dos dados de treinamento e teste); complete a tabela abaixo:

Execução	Nº de acertos dos dados treinamento	Nº de acertos dos dados de teste
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
Média:		
Desvio:		

l - plote os dados de treinamento e teste reclassificados pela RNA e compare, graficamente, com a classificação original (utilize a melhor rede criada).

Exemplos de alguns gráficos esperados são mostrados nas Figuras 47 a 49.

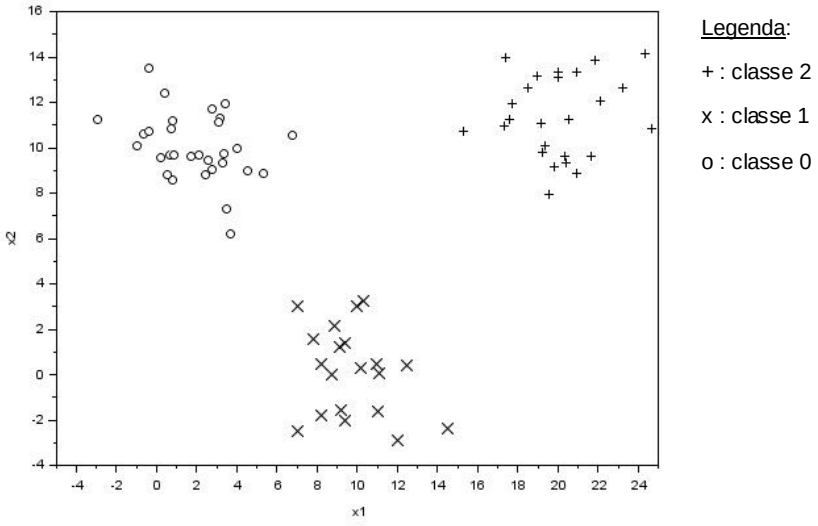


Figura 47: Gráfico dos dados gerados de duas dimensões e três classes linearmente separáveis.

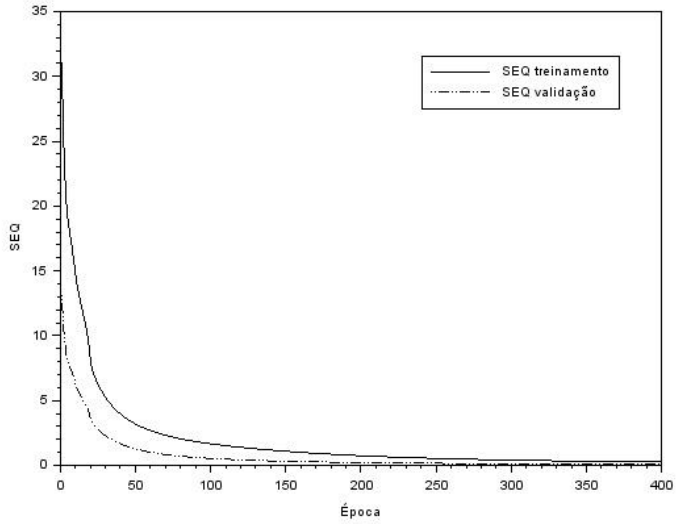
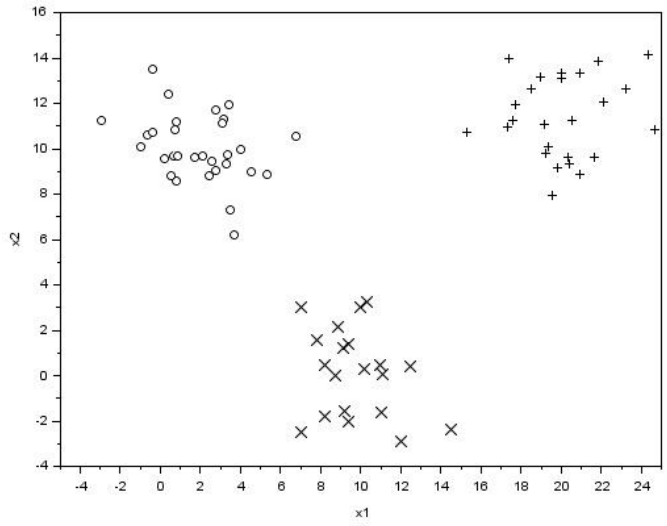


Figura 48: Gráfico do somatório dos erros quadráticos.



Legenda:

+ : classe 2

x : classe 1

o : classe 0

Figura 49: Gráfico dos dados reclassificados pela rede neural tipo MLP treinada. Repare que todas as amostras (treinamento e teste) são classificadas corretamente.

3 - Repita o item 2 utilizando dados não linearmente separáveis. Sugestões de parâmetros:

- número de entradas: 2
- número de camadas escondidas: 2
- número de neurônios na primeira camada escondida: 27

- número de neurônios na segunda camada escondida: 9
- número de saídas: 3
- função de ativação: sigmoidal
- taxa de aprendizado: 0.02
- taxa de momento: 0.4
- número máximo de épocas: 1200
- tolerância do erro: 0.00001

Complete a tabela a seguir:

Execução	**Nº de acertos dos dados treinamento**	**Nº de acertos dos dados de teste**
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
Média:		
Desvio:		

Exemplos de alguns gráficos esperados são mostrados nas Figuras 50 a 52.

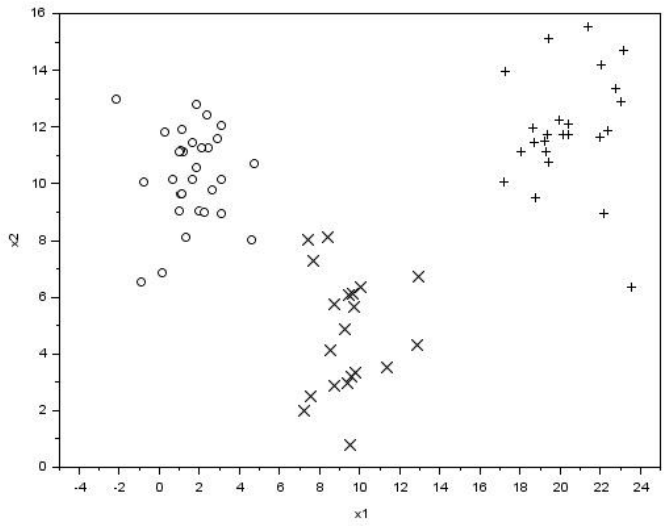


Figura 50: Gráfico dos dados gerados de duas dimensões e três classes não linearmente separáveis.

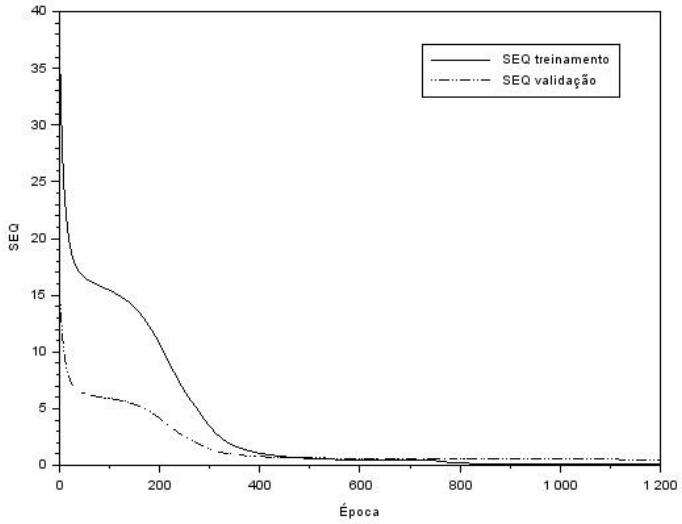


Figura 51: Gráfico do somatório dos erros quadráticos.

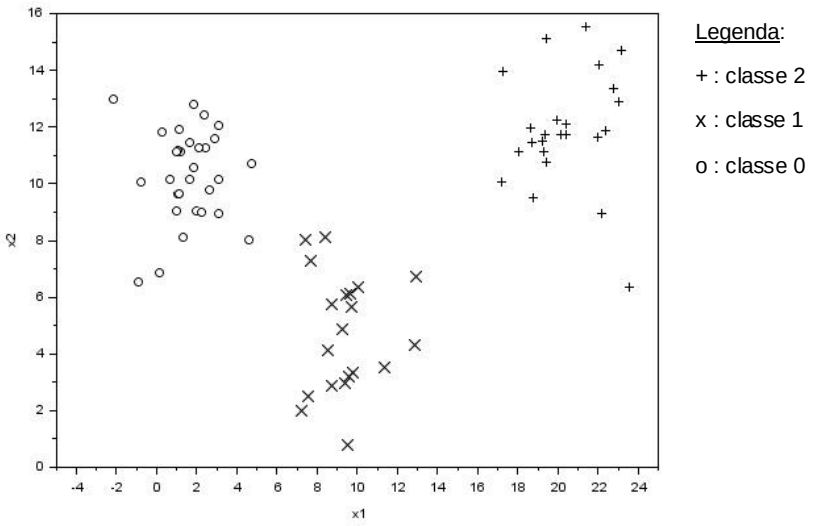


Figura 52: Gráfico dos dados reclassificados pela rede neural tipo MLP treinada. Repare que todas as amostras (treinamento e teste) são classificadas corretamente.

4 - Postar na sala virtual da disciplina o relatório referente a prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

10ª EXPERIÊNCIA: Aproximação de funções com MLP

Título: Aproximação de função não linear utilizando MLP

Objetivos:

- Conhecer o método de aproximação de função utilizando uma Rede Perceptron Multicamadas (MLP);
- Implementar a MLP para aproximação da função seno;
- Verificar a tarefa de aproximação de função não linear com uma MLP treinada com dados ruidosos.
- Avaliar estatisticamente os resultados obtidos.

Teoria:

Normalização dos dados de entrada e saída

Uma característica das funções sigmoidais é a saturação, ou seja, para valores grandes de argumento, a função opera numa região de saturação (Figura 53).

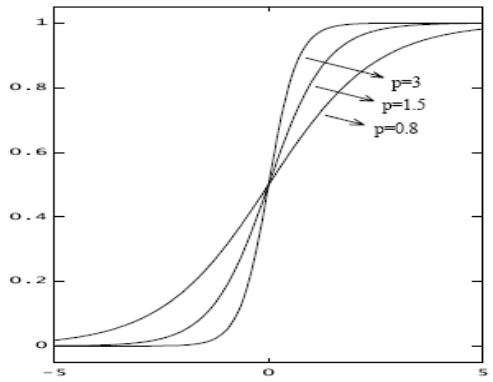


Figura 53: Gráfico da função sigmoidal.

Por isto, é importante trabalhar com valores de entrada que estejam contidos num intervalo que não atinjam a saturação, por exemplo: [0,1]. Para isto, uma forma de normalizar os valores dos dados de entrada e saída desejada é dividir todos os valores de uma entrada (ou saída) pelo maior valor de ocorrência.

Os valores da saída desejada para cada amostra de entrada também devem ser normalizados entre 0 e 1, pois a função de ativação sigmoidal na saída de cada neurônio da última camada satura entre 0 e 1.

Prática:

1 - Estudar as funções do Toolbox ANN do Matlab. Dica: use o "*help*". Teste o *script* abaixo que exemplifica uma implementação de uma MLP para aproximação de função a partir de amostras de dados de entrada e saída desejada.

```
clear all;
close all;
clc;

% Cria os dados de entrada e saida
[X,Yd] = simplefit_create;
figure(1);
plot(X,Yd);
title('Dados de treinamento');

% Cria a rede neural
net = feedforwardnet(10);

% Treina a rede neural
net = train(net,X,Yd);

% Simula a rede neural
Ys = sim(net,X);

% Plota resultado da aproximacao
figure(2);
plot(X,Ys);
title('Resultado');
```

2 - Elaborar um programa que crie, treine e execute uma rede neural artificial tipo MLP para aproximar a função seno (intervalo de $-\pi$ a $+\pi$), usando o Matlab. Utilizar o Toolbox ANN. Implemente as seguintes etapas:

a - crie várias amostras de dados (pares x,y) onde $y = sen(x)$; normalize os valores de y para ficar entre 0 e 1;

b - embaralhe os dados criados; utilize a função *mistura* já criada;

c - plote os dados criados e normalizados para verificação;

d - separe os dados normalizados em dois conjuntos: treinamento (70%) e teste (30%);

e - crie uma rede neural tipo MLP e treine **apenas** com o conjunto de dados de treinamento; utilize as funções do toolbox ANN do MATLAB; sugestão:

- número de camadas escondidas: 1
- número de neurônios na camada escondida: 5
- taxa de aprendizado: 1

- número máximo de épocas de treinamento: 1000
- função de ativação: sigmoidal

f - plote o somatório dos erros quadráticos dos dados de treinamento e teste (separado) por época de treinamento;

g – após treinamento, execute os dados do conjunto de treinamento e teste utilizando a RNA treinada (utilize o Toolbox ANN) e contabilize o somatório dos erros quadráticos (separado para dados de treinamento e teste);

h - plote todos os pares de dados (*x*,*y*) de treinamento e teste aproximados pela RNA e compare, graficamente, com a função seno original (utilize a melhor rede criada).

i - repita 10 vezes o procedimento a partir do item b, de modo a calcular a média e desvio padrão dos resultados; complete a tabela de resultados:

Execução	**Resultados SEQ dos dados de treinamento**	**Resultados SEQ dos dados de teste**
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
Média:		
Desvio:		

Os gráficos esperados dos resultados são mostrados nas Figuras 54 a 56.

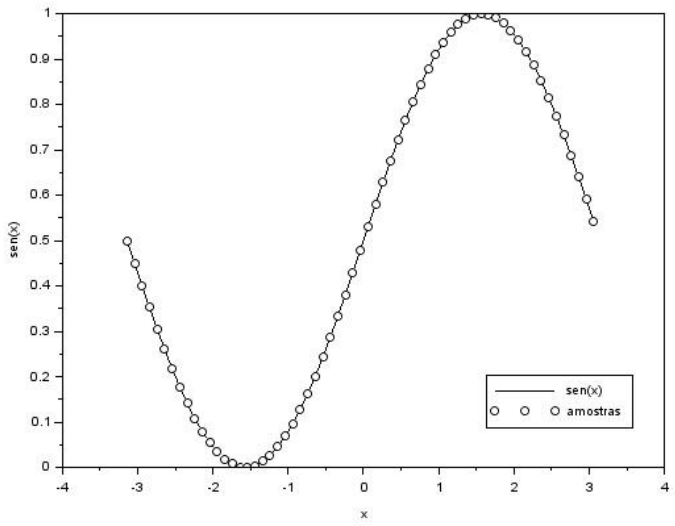


Figura 54: Gráfico da função seno e as amostras de dados gerados (símbolo “o”).

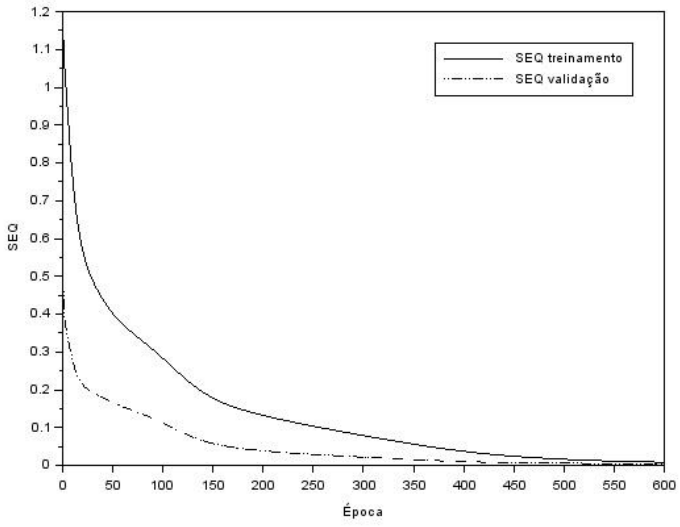


Figura 55: Gráfico do somatório dos erros quadráticos por época.

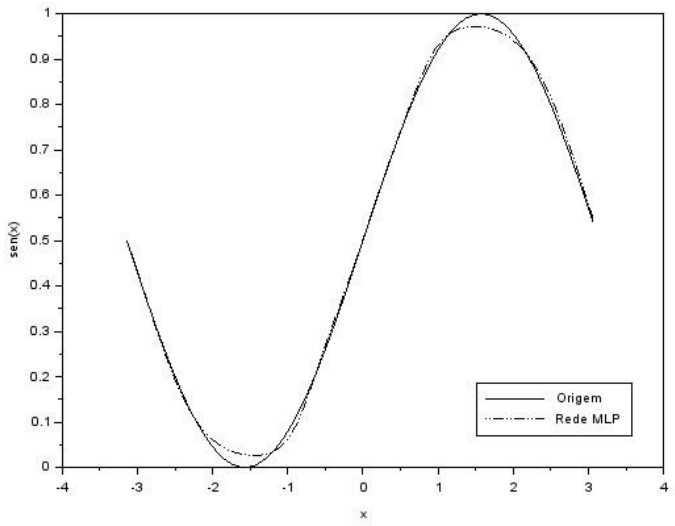


Figura 56: Gráfico da função seno original e o resultado da aproximação pela rede MLP.

3 - Acrescentar ruído aos dados e repetir o item 2. Sugestão:

- número de camadas escondidas: 1
- número de neurônios na camada escondida: 10
- taxa de aprendizado: 1
- taxa de momento: 0,6
- número máximo de épocas de treinamento: 600
- função de ativação: sigmoidal

Os gráficos esperados dos resultados obtidos são mostrados nas Figuras 57 a 59.

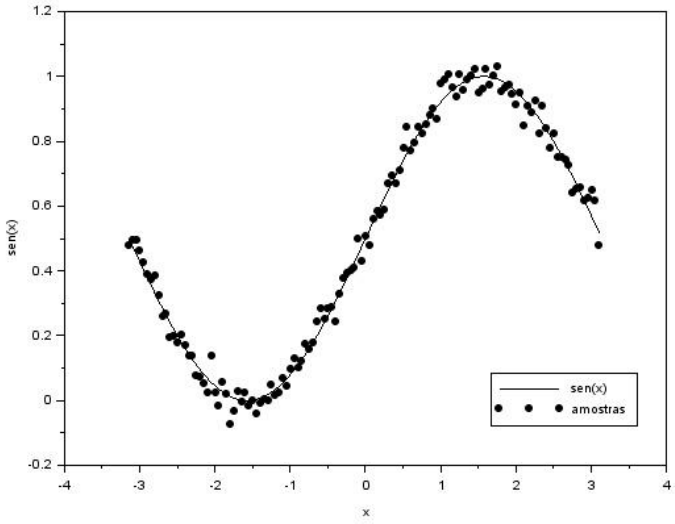


Figura 57: Gráfico da função seno e as amostras de dados gerados (símbolo "•") com ruído.

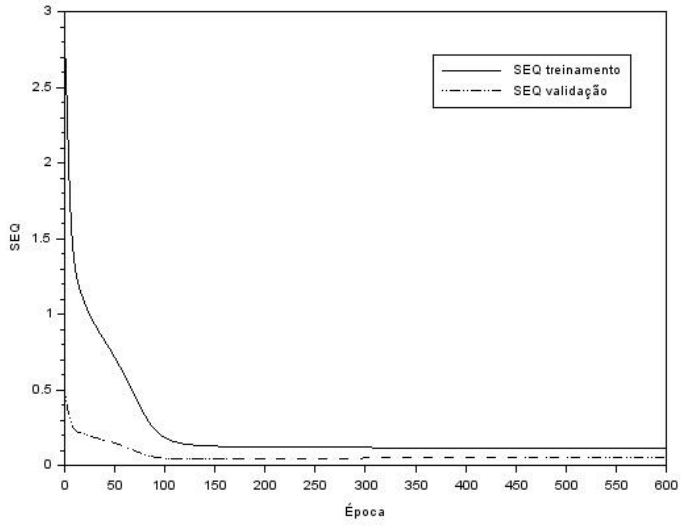


Figura 58: Gráfico do somatório dos erros quadráticos (SEQ) por época para os dados de treinamento e validação.

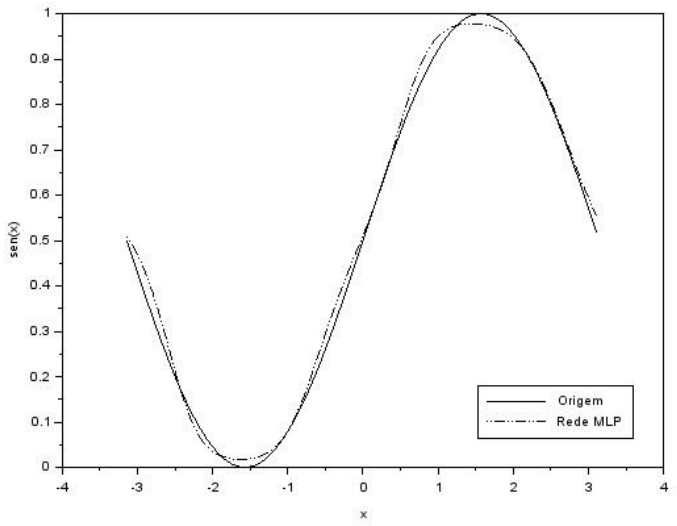


Figura 59: Gráfico da função seno original e o resultado da aproximação pela rede MLP treinada com dados ruidosos.

Contabilize os resultados na tabela abaixo:

Execução	Resultados SEQ dos dados de treinamento	Resultados SEQ dos dados de teste
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
Média:		
Desvio:		

4 **-** Postar na sala virtual da disciplina o relatório referente à prática (veja modelo no Apêndice) em arquivo formato pdf sem compactar até a data combinada.

Referências Bibliográficas

BISHOP, Christopher M. **Neural Networks for Pattern Recognition**. New York: Oxford University Press, 1997, 482 p.

BRAGA, Antônio P., LUDERMIR, Teresa B., e CARVALHO, André C. P. L. F. **Redes Neurais Artificiais: Teoria e Aplicações**. Rio de Janeiro: Livros Técnicos e Científicos, 2000, 262 p.

DUDA, Richard O., HART, Peter E., e STORK, David G. **Pattern Classification**. New York: John Wiley & Sons, 2000, 654 p.

HAYKIN, Simon. **Redes Neurais: Princípios e Prática**. Porto Alegre: Bookman, 2ª edição, 2001, 900 p.

APÊNDICE

Modelo de Relatório

Relatório Prático Nº <coloque aqui o número>

Redes Neurais Artificiais

Nome: <coloque aqui seu nome> **Data:** <coloque aqui a data: DD:MM:AA>

Curso: <coloque aqui seu curso>

Título: <coloque aqui o título da experiência prática>

Objetivo: <coloque aqui o principal objetivo da experiência prática>

Script:

<coloque aqui todas as funções criadas e utilizadas para desenvolver a experiência prática, incluindo o *script* principal com todos comentários de programa>

Resultados:

<coloque aqui todos os resultados obtidos em forma de texto, gráfico e/ou tabela com comentários pertinentes>

Conclusão:

<coloque aqui as principais conclusões e discussões>