Redes Neurais para Inferência Estatística

-FEA/USP Jun/2002

Renato Vicente*

Programa

1. Introdução: Redes Neurais, Matlab e Netlab
2. Algoritmos de Otimização
3. Estimação de Densidades de Probabilidade
4. Redes Neurais com uma Camada
5. Redes Neurais Multicamada
6. Amostragem
7. Técnicas Bayesianas

Opcionalmente:

8. Radial Basis Functions
9. Redução Dimensional e Visualização de dados
10. Processos Gaussianos

Referências Básicas:

- Using Matlab, The MathWorks Inc. (2000);
- Bishop, C.M., Neural Networks for Pattern Recognition, Oxford University Press, Oxford, 1995;
- Nabney, I.T., Netlab: Algorithms for Pattern Recognition, Springer, London, 2002.

* rvicente@usp.br

Introdução: Redes Neurais, Matlab e Netlab

Redes Neurais

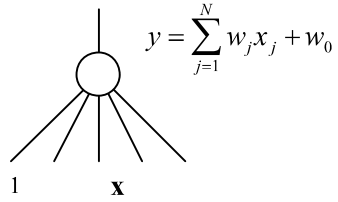
Rede Neural é um nome fantasia para modelos de inferência multidimensionais e não-lineares. O grande apelo destes modelos está em sua capacidade de "aprender", generalizar ou extrair regras automaticamente de conjuntos de dados complexos.

No caso mais simples, temos um conjunto de pares entrada-saída $D = \{(\mathbf{x}^{(n)}, \mathbf{t}^{(n)})\}_{n=1}^{p}$ e queremos modelar a função $\hat{t} = y = y(\mathbf{x}, \mathbf{w})$ que produz melhores estimativas para pares fora do conjunto *D,* dados vetores **x**. Este problema é denominado *regressão.* A Rede Neural consiste simplesmente de uma escolha particular para a família de funções $y(\mathbf{x}, \mathbf{w})$ parametrizada por **w**, denominados *pesos sinápticos.*

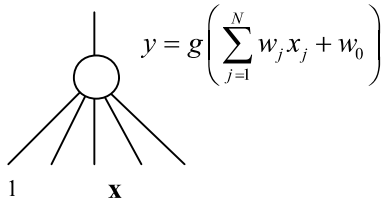
A importância das Redes Neurais neste contexto está no fato delas representarem um esquema bastante genérico para a representação de famílias de funções não-lineares com várias variáveis de entrada e saída e controladas por um certo número de parâmetros ajustáveis.

Há uma série de famílias, ou arquiteturas, clássicas para Redes Neurais que são representadas graficamente.

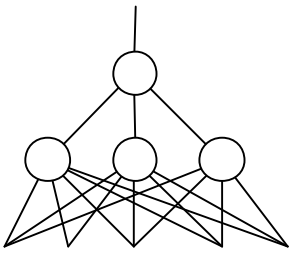
Perceptron Linear



Perceptron Não-linear

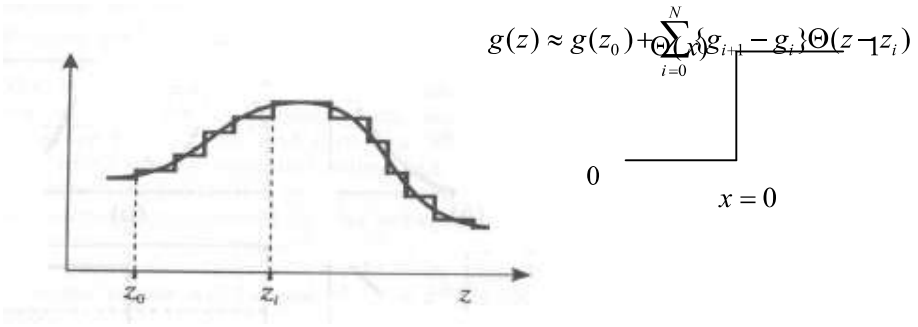


Rede Multicamada



$$y = g\left(\sum_{l=1}^{n} u_l g_1\left(\sum_{j=1}^{N} w_j x_j + w_0\right)\right)$$

É possível[1] mostrar que Redes Neurais Multicamada com um número N suficientemente grande de unidades na mamada interna (ou escondida) podem representar qualquer função com erro menor que $\varepsilon(N)$. Uma maneira simples e direta para percebermos este fato é ilustrada abaixo:



Na figura uma função arbitrária g(z) é representada por uma soma de funções degrau $\Theta(z - z_i)$ sobre uma partição da reta $P = \{z_0, z_1, ..., z_N\}$. A medida que refinamos a partição fazendo $|z_{i+1} - z_i| < \delta(N)$, reduzimos o erro $\varepsilon(N)$.

Quando as saídas consistem de classes discretas $t \in \{1, ..., c\}$ o problema de inferência é denominado *classificação*.

[1] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks* ***4*** (2), 251-257.

Matlab

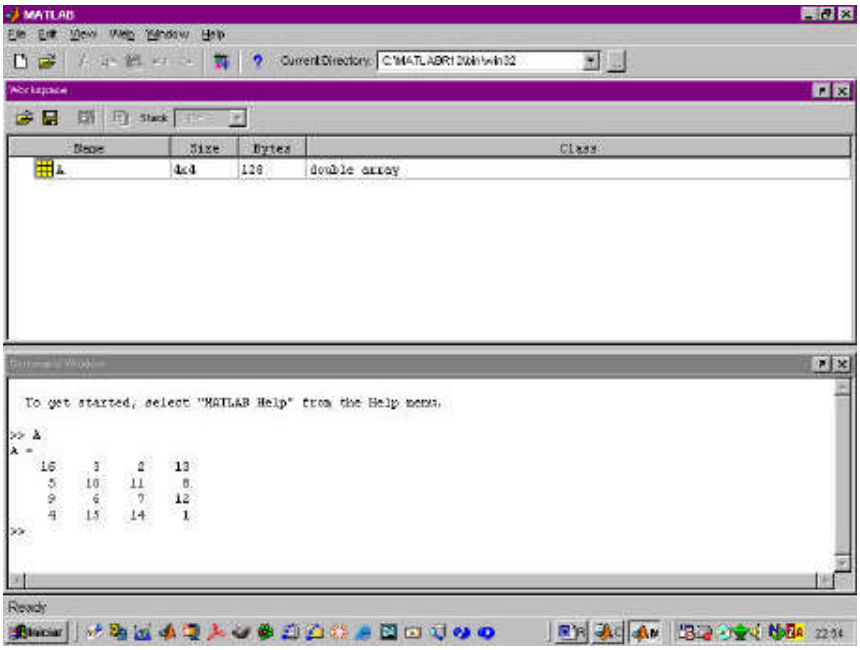
Matlab (*Matrix Laboratory*) é uma linguagem para computação científica com estrutura otimizada para a realização de operações com matrizes. O ambiente Matlab é composto por uma série de funções pré-definidas para cálculo, leitura e escrita de arquivos e visualização. Este conjunto de funções pode ser facilmente extendido por *toolboxes* dedicadas. Há *toolboxes* para finanças, tratamento de sinais, econometria e redes neurais.

Para ilustrar a forma como matrizes podem ser tratadas pelo Matlab utilizamos o quadrado mágico de Dührer que é utilizado como exemplo no *Get Start* que acompanha o software. Na linha de comando do Matlab entramos:

```
> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]

A =
   16    3    2   13
    5   10   11    8
    9    6    7   12
    4   15   14    1
```

Após este comando a matriz A fica armazenada no *Workspace* do Matlab conforme ilustrado abaixo.



As propriedades do quadadro mágico podem ser analisadas utilizando funções do Matlab. Por exemplo, a função abaixo soma cada uma das linhas da matriz A.

```
> sum (A)

ans =
   34    34    34    34
```

Na ausência de definição de variável de saída o Matlab aloca o resultado no *Workspace* sob o nome *ans*. Assim ao entrarmos

```
> ans

ans =
   34    34    34    34
```

Se quisermos somar as colunas ao invés das linhas temos que transpor a matriz A, isso feito simplesmente pelo comando

```
>A'

ans =
   16     5     9     4
    3    10     6    15
    2    11     7    14
   13     8    12     1
```

Assim podemos realizar a soma das colunas entrando

```
>sum(A')'

ans =
   34
   34
   34
   34
```

Novamente o resultado é 34. Podemos também calcular a soma da diagonal, ou traço, de duas formas:

```
> diag (A)

ans =
   16
   10
    7
    1
```

```
> sum (diag(A))

ans =
   34
```

Ou

```
> trace(A)

ans =
   34
```

Curiosamente continuamos obtendo 34. Somemos agora a antidiagonal da matriz A, para isso podemos utilizar a função *fliplr* para enviarmos cada coluna da esquerda para a direita da matriz A.

```
> fliplr (A)

ans =
   13    2    3   16
    8   11   10    5
   12    7    6    9
    1   14   15    4
```

E a seguir

```
> sum(diag(fliplr(A)))

ans =
   34
```

As somas são sempre iguais a 34 pois quando dividimos números inteiros de 1 a 16 em quatro grupos com somas iguais temos:

```
> t=1:1:16

t =
    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16

> sum(t)/4

ans =
   34
```

Aqui utilizamos ":" para gerarmos números de 1 a 16 em passos de 1.

O Matlab possui uma função específica para a criação de quadrados mágicos de qualquer dimensão, esta função é

```
> B=magic(4)

B =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
```

É possível reordenar as colunas de B para reobtermos A utilizando o comando

```
> B(:,[1 3 2 4])

ans =
   16    3    2   13
    5   10   11    8
    9    6    7   12
    4   15   14    1
```

Netlab

O Netlab é uma *toolbox* para Redes Neurais para Matlab disponibilizada gratuitamente no site do Neural Computing Research Group da Univerisdade de Aston (www.ncrg.aston.ac.uk). Esta *toolbox* toran a implementação de modelos baseados em Redes Neurais muito simples. Os códigos produzidos em Matlab podem então ser convertidos em executáveis utilizando o *Matlab Compiler*. Como um primeiro contato com o Netlab escreveremos sua versão para o clássico "*Hello World*".

Começamos por gerar de um conjunto de dados fictícios para treinamento de nossa Rede Neural. Para isso vamos supor que a função que queremos inferir é $f(x) = sin(2\pi x)$. No entanto, só temos acesso a uma versão corrompida por ruído gaussiando desta função assim, nosso conjunto de dados é "gerado" da seguinte forma:

$$t_n = \sin(2\pi x_n) + \sigma\varepsilon_n$$
$$\varepsilon_n \sim N(0,1) \quad ,$$

onde N(0,1) é uma distribuição normal com média nula e variância unitária.

No Matlab teremos

```
1       x= [0:1/19:1]';
2       ndata= size(x,1);
3       t= sin(2*pi*x) + 0.15*randn(ndata,1);
```

Aqui utilizamos alguns recursos novos. Utilizamos ao final de cada linha ";" , esta instrução faz com que o Matlab omita resultados intermediários dos cálculos sendo executados. A primeira instrução gera um vetor coluna x com 20 dimensões contendo números de 0 a 1 em intervalos de 1/19. Na linha 2 utilizamos a função *size(x,1)* para obtermos o tamnho em número de linhas do vetor coluna x, assim, se entrarmos:

```
> size(x)

ans =
   20    1
```

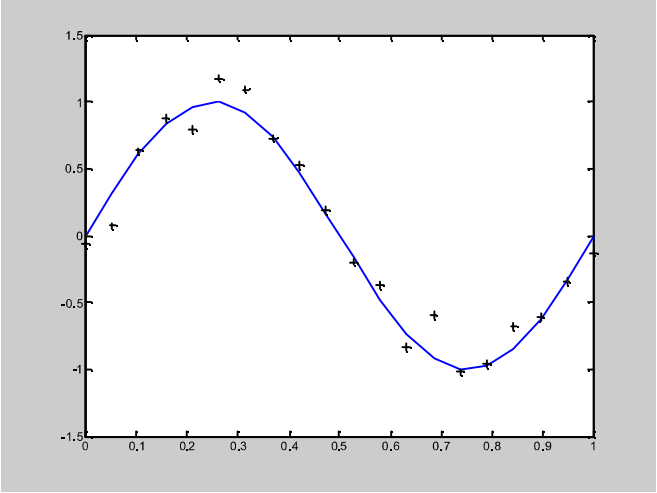
Ou seja, o vetor x possui 20 linhas e 1 coluna, *size(x,1)* nos dá o número de linhas, enquanto *size(x,2)* nos dá o número de colunas. Na linha 3 utilizamos a função *randn(linhas, cols)* para gerarmos um vetor de números aleatórios independentes com distribuição gaussiana com média nula e variância unitária. De forma genérica esta instrução pode ser empregada pra gerar matrizes aleatórias de qualquer tamanho.

```
> randn(5,5)

ans =
   1.3808    0.3908   -1.3745   -1.3454   -0.7826
   1.3198    0.0203   -0.8393    1.4819   -0.7673
  -0.9094   -0.4060   -0.2086    0.0327   -0.1072
  -2.3056   -1.5349    0.7559    1.8705   -0.9771
   1.7887    0.2214    0.3757   -1.2090   -0.9640
```

No Matlab, podemos visualizar o vetor *t* e a função original utilizando:

```
> plot(x,sin(2*pi*x),x,t,'k+');
```



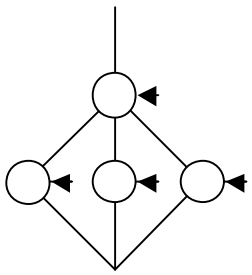
Gerado nosso conjunto fictício de dados, podemos definir uma arquitetura de rede para utilizarmos na inferência. Assumiremos aqui que uma Rede Neural Multicamada com 3

unidades na camada escondida seja adequada. No Netlab podemos definir uma arquitetura de rede em uma única linha de comando.

```
4       net= mlp (1, 3, 1, 'linear')

net =
    type: 'mlp'
    nin: 1
    nhidden: 3
    nout: 1
    nwts: 10
    actfn: 'linear'
    w1: [-1.6823 -0.5927 0.1820]
    b1: [-0.1300 -0.1185 -0.0827]
    w2: [3x1 double]
    b2: 0.2541
```

A função *mlp* do Netlab cria uma estrutura que especifica completamente a arquitetura e os parâmetros iniciais da Rede Neural. Assim a rede possui *nin=1* entrada, *nhidden=3* unidades na camada interna, *nout=1* saída e *nwts=10* parâmetros entre pesos sinápticos e limiares. Os parâmetros são iniciados em valores aleatórios. Graficamente teremos:



Cada unidade possui uma entrada extra denotada por uma seta que regula o *limiar de disparo* de cada um dos neurônios. Cada neurônio da camada escondida gerado pela função *mlp* implementa uma *função de transferência sigmoidal* do tipo $\tanh(b + \sum_j w_j x_j)$.

Já o neurônio da camada de saída implementa a função de transferência especificada na função *mlp*, no caso, uma função linear. Assim, a arquitetura da Rede Neural que utilizaremos representa a seguinte família de funções:

$$y(x, \mathbf{w}, \mathbf{b}) = b_2 + \sum_{l=1}^{3} w_{2l} \tanh(b_{1l} + w_{1l} x) .$$

O terceiro passo consiste no treinamento desta rede utilizando os dados fictícios contidos no conjunto de treinamento. Para isso o Netlab utiliza a infra-estrutura de algoritmos de otimização do Matlab. Primeiro é necessário armazenar as configurações padrão dos otimizadores do Matlab:

```
5       options= foptions;
```

A seguir personalizamos algumas destas opções:

```
6       options(1)  = 0;
```

Este comando evita que os algoritmos de otimização exibam na tela a evolução passo a passo da função custo sendo minimizada.

```
7       options(14) =100;
```

Esta instrução limita em 100 passos os ciclos de otimização. Agora podemos treinar nossa rede utilizando a função *netopt* do Netlab:

```
8       [net, options] = netopt(net, options, x, t, 'scg');

Warning: Maximum number of iterations has been exceeded
```

A função *netopt* otimiza os parâmetros da rede definida pela estrutura *net*, utilizando as opções de otimizador defindas em *options* e utilizando os dados de entrada contidos no vetor *x* e de saída contidos no vetor *t*. O último argumento em *netopt* especifica o tipo de algoritmo a ser utilizado na otimização, no exemplo, '*scg*' significa *scaled gradient*. A função retorna uma mensagem de aviso indicando que a otimização foi terminada devido ao limite de 100 passos de otimização.

Neste momento já obtivemos no *Workspace* do Matlab uma estrutura *net* com parâmetros otimizados assim:

```
> net.w1

ans =
  -5.2017    0.8701    1.5318

> net.w2

ans =
   2.6033
   1.0716
   3.4209

> net.b1

ans =
   2.4785   -0.6179   -0.3766

> net.b2

ans =
  -0.7588
```

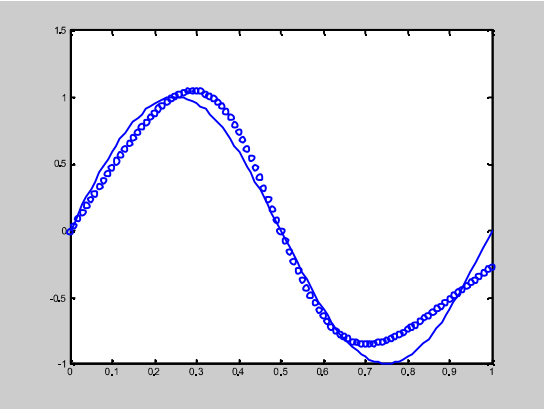
O quarto passo consiste em utilizarmos a rede recém treinada. Para isso utilizamos a função *mlpfwd* do Netlab, assim se quisermos saber qual seria a saída para a entrada *x=0. 565*, calculamos:

```
>mlpfwd(net,0.565)

ans =
  -0.4632
```

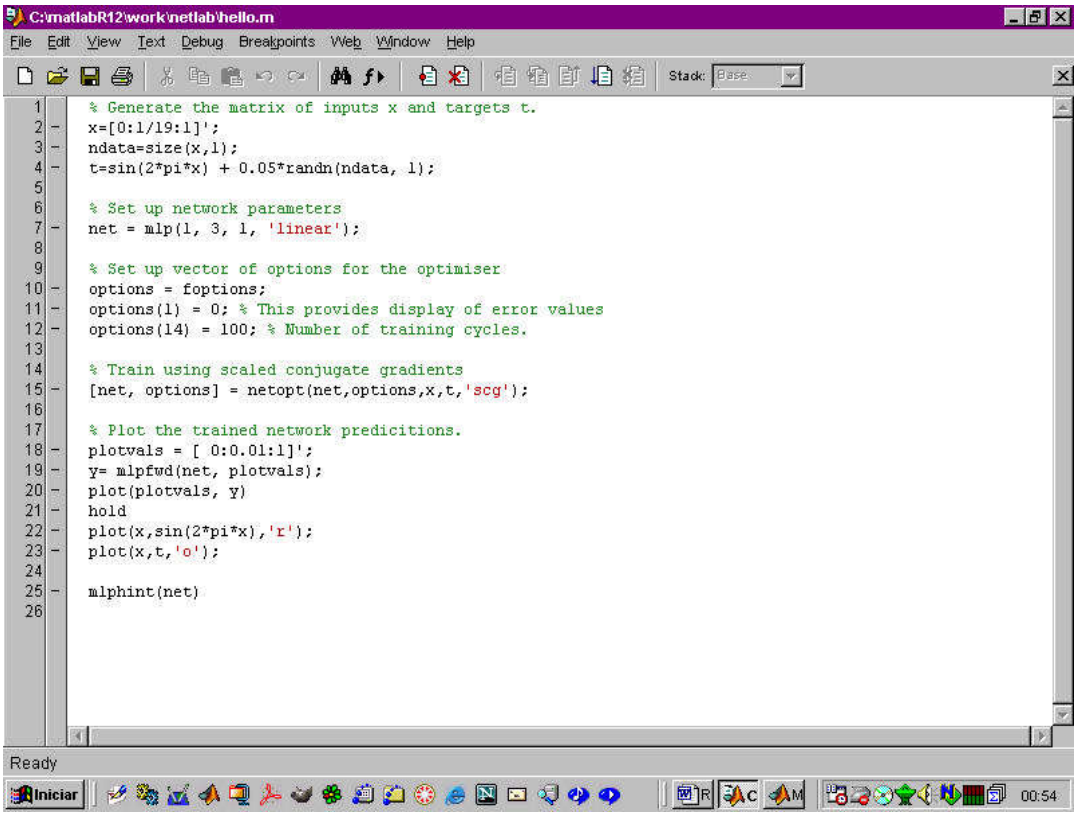
Para analisarmos visualmente a qualidade de nosso modelo para os dados representamso em um gráfico previsões para várias entradas e o valor "real":

```
9       plotvals=[0:0.01:1]';
10      y=mlpfwd(net,plotvals);
11      plot(plotvals, y, 'ob',plotvals,sin(2*pi*plotvals));
```



Na figura acima as previsões aparecem como circunferências e o valor "real" como uma linha cheia. A qualidade da previsão irá depender da Rede Neural empregada e do nível de ruído dos dados.

Toda a seção de Matlab, utilizada neste problema pode ser armazenada na forma de um arquivo .m como o exibido a seguir:



Exercícios

(1) Escreva uma rotina ou função em Matlab que calcule o erro quadrático de predição da Rede Neural Multicamada utilizada no exemplo acima no conjunto *plotval*.

$$E(\mathbf{x}) = \sum_j \left[f(x_j) - y(x_j; \mathbf{w}, \mathbf{b}) \right]^2 .$$

(2) Treine Redes Multicamada com número de unidades na camada interna indo de $nhidden = 1$ a $nhidden = 30$ e construa um gráfico do erro de predição acima contra o número de unidades na camada escondida. Qual é a melhor escolha de arquitetura ?